PASCAL/UBC User's Guide

by

Bary W. Pollack
Robert A. Fraley

Technical Manual TM-2

Nov 8, 1976

Revised Nov 22, 1977

Department of Computer Science
The University of British Columbia
Vancouver, British Columbia V6T 1W5

Abstract

PASCAL/UBC is a PASCAL compiler for the IBM 360/370 computers running under the MTS operating system. It processes a version of PASCAL producing standard OS object modules. The language accepted contains Standard Revised PASCAL as a proper subset.

This work has been supported, in part, by the National Research Council of Canada, grant number A3606, and by the Department of Computer Science, University of British Columbia.

TABLE OF CONTENTS

PASCAL/UBC USER'S GUIDE

0. Introduction

PASCAL/UBC is a PASCAL [1,2] compiler for the IBM 360/370 computers running under the MTS operating system. It was originally developed at Stanford University [3] and then partially rewritten at the University of British Columbia. It processes a version of PASCAL producing standard OS object modules. These modules may be executed under the supervision of a run time monitor. PASCAL may be run in batch or from a terminal.

The most recent version of this manual usually will be found (in TN-chain ready form) in the file PASC:WRITEUP. Revisions to the last published version of this manual are indicated by vertical bars in the right-hand margin such as those in the margin of this sentence. Current news regarding the state of the PASCAL system may be found in the file PASC:NEWS.

This work has been supported, in part, by the National Research Council of Canada, grant number A3606, and by the Department of Computer Science, University of British Columbia.

1. Compiling and Running a PASCAL Program

PASCAL/UBC uses standard MTS conventions: the source program and/or data are read from SCARDS; the compilation listing, error diagnostics, and execution output are written on SPRINT; and the object code (if any) is sent to SPUNCH. The PAR field is used for the specification of various options and to make file assignments. If SPRINT is different from *SINK* the number of syntax errors is sent to both places. The error count always is reported on SPRINT.

A semicolon (;) may (optionally) terminate the PAR field and anything to the right of the semicolon is ignored by the system. If a semicolon is present, it must be preceded by at least one space. The entire PAR field may be retrieved from within a program using a standard MTS library routine.

The PASCAL programmer may:

a. Compile a PASCAL program, saving (or discarding) the object code;

b. Run a previously compiled PASCAL program;

c. Compile and run a PASCAL program in a "compile, load, and go" fashion;

d. Compile and run a PASCAL program in a "compile, link, load, and go" fashion.

1.1 Compiling a PASCAL Program

The PASCAL translator may be invoked by using the following command:

```
$RUN CS:PASCAL SCARDS=fdname SPRINT=fdname SPUNCH=fdname
               PAR=options ; comments
```

If no SPUNCH file is specified the compiler will send the object code to file -P.OBJ.

The options pertinent during compilation are: EX, NERRS, NEW, and TIME.

1.2 Running a Previously Compiled PASCAL Program

PASCAL object modules may be executed directly by using the following command:

```
$RUN object SCARDS=fdname SPRINT=fdname SPUNCH=fdname
     PAR=options file-assignments ; comments
```

If SCARDS is not specified it defaults to *SOURCE*; if SPRINT is not specified it defaults to *SINK*.

The options pertinent during execution are: BATCH, EX, NERRS, NEW, NOPMD, and TIME.

The object modules may be a single file or a concatenation of files (e.g., a main program and possibly several support routines).

Whenever a main program is compiled the last line of the object file always will contain

```
$CONTINUE WITH CS:PASCALLIB
```

Thus the user need not explicitly concatentate the run-time library when running the object file.

If NOPMD is specified the user may wish to concatenate the "dummy" PMD routines to his object modules:

```
$RUN PASC:DPMD+object ...
```

This will save the memory charge for the full PMD package which otherwise would have been loaded.

1.3 Compiling and Running with LOADNGO

The PASCAL system will operate in a "compile, load, and go" fashion if the following command is issued:

```
$RUN CS:PASCAL SCARDS=fdname SPRINT=fdname SPUNCH=fdname
     PAR=...,LOADNGO,...  file-assignments ; comments
```

Under LOADNGO no object modules will be generated unless the user specifies the compiler option $OBJFILE+$ (See below). The object code is loaded directly into memory, and after the compilation phase is finished the resulting code is executed.

The only restriction under LOADNGO is that no reference may be made to a routine not present in the standard PASCAL library. This implies that a program will not execute correctly under LOADNGO if it declares procedures FORTRAN or EXTERNAL, or if has any unresolved FORWARD reference. This restriction is necessary because the MTS loader is not used.

The options pertinent during LOADNGO execution are: BATCH, EX, NEW, NOPMD, SIZE, and TIME.

1.4 Compiling and Running with LINK

The PASCAL system will operate in a "compile, link, load, and go" fashion if the following command is issued:

```
$RUN CS:PASCAL SCARDS=fdname SPRINT=fdname SPUNCH=fdname
     PAR=...,LINK,...  file-assignments ; comments
```

Under LINK object code is emitted as usual. Following code generation a dynamic link (actually an XCTL) is performed. The standard PASCAL library CS:PASCALLIB is referenced as well as *LIBRARY and any other libraries the user has set up with MTS.

If you wish to explicitly specify a library to be included in the linkage process, you may issue the following command:

```
$RUN CS:PASCAL SCARDS=fdname SPRINT=fdname SPUNCH=fdname
     PAR=...,LINK,...,LI=library,...
     file-assignments ; comments
```

The order of LINK and LI is irrelevant. The specified library may be a concatenation of files. CS:PASCALLIB will always be accessed to resolve any open references after accessing the specified library.

The options applicable under LINK execution are: BATCH, EX, NEW, NOPMD, and TIME.

1.5 Options

The first field after the PAR= is for specification of the options: BATCH, EX, LI, LINK, LOADNGO, NERRS, NEW, NOPMD, SIZE, and TIME. The options are separated by commas, and the options field is terminated by one or more blanks. No blanks may appear within the options field.

Options occur in two forms -- those taking values and those whose presence/absence is significant. The syntax for those options taking values is <option>=<value>.

The BATCH option disables all interactive features. The snapshot and post mortem dump packages are forced to act in batch mode.

EX=<value> specifies the maximum number of memory pages to be acquired for the execution stack. If the user specifies a value for EX, this value will be used both during program compilation and execution. The default value is 10 pages.

LI=<value> (Submonitor only) specifies the name of a user's object file (a user's library file) which is to be concatenated with the standard PASCAL monitor PASC:MON prior to execution of the translated program.

LINK specifies that the user wishes to compile, link, load, and execute. A dynamic link (actually an XCTL) will be performed. This option must be present if the user wishes to use the LI= option.

LOADNGO specifies that the user wishes to compile, load, and execute without punching object code onto SPUNCH. No FORTRAN or EXTERNAL routines may be accessed. The compiler assumes $OBJFILE-$.

NERRS=<value> specifies the maximum number of run time errors allowed before the run is terminated. The default value is taken to be 4.

NEW=<value> specifies the maximum number of memory pages to be acquired for the NEW stack. If the user specifies a value for NEW, this value will be used both during program compilation and execution. The default value is 20 pages.

NOPMD specifies that in the event of a run error, no post mortem dump is to be generated. This option has no effect on the execution of the standard SNAPshot procedure.

SIZE=<value> specifies the number of pages of memory to be used for the object code when running under LOADNGO. The default value is 10 pages.

TIME=<value> specifies the maximum allowed execution time for the running program in seconds. The usual MTS variants are allowed (e.g., TIME=1.5S, T=.057, TIME=1.25M, etc.). If TIME is not specified, one minute is taken as the default.

1.6 File Assignments

Subsequent fields of the PAR= field (after the options) are used for file assignments. Each such assignment must be bounded by one or more spaces. The pattern is PASCAL_NAME=MTS_NAME, where PASCAL_NAME is the name given in a FILE declaration in the source program, and MTS_NAME is the name of an MTS file or device.

1.7 Examples

Examples of valid PAR fields are:

```
PAR=NEW=25,EX=50 ; THIS IS A COMMENT                    (1)

PAR= PFILE=MTSFILE                                      (2)

PAR=NOPMD,NEW=15 P1FILE=MTSFILE1 P2FILE=*SINK*          (3)
```

(1) indicates that the program is to be executed with a maximum of 25 pages allocated for the NEW stack and 50 pages for the execution stack.

(2) indicates that the PASCAL file PFILE is to be associated with the MTS file MTSFILE during execution.

(3) indicates that no post mortem dump is to be generated if a run error occurs, allocate 15 NEW pages, associate P1FILE with the MTS file MTSFILE1, and associate P2FILE with *SINK*.

2. Running PASCAL under the Student Terminal System

The PASCAL translator is invoked under the Student Terminal System (STS) by use of the $PASCAL control card. The rest of this card (currently) is ignored. When running under STS, one is unable to set any of the submonitor options or make file assignments. These are all set by the STS supervisor. It is impossible to link to user-supplied external routines under STS, although many of the more commonly used MTS system routines are available in the standard library.

The defaults effective under STS are BATCH, EX=40, LOADNGO, NERRS=0, NEW=40, PAGES=12, SIZE=25, and TIME=5.0S.

Under STS no $DATA card separates the PASCAL source program from the following data. Thus it is extremely important to remember that all PASCAL programs end "END.". If a $DATA card is included in the source, it will be read by your program during execution.

3. Compiler Options

Compiler options are delimited by '$' and are separated by commas or semicolons. They consist of a keyword followed by a plus or a minus sign indicating 'on' or 'off', respectively. One need supply only as much of the keyword as is necessary to distinguish it uniquely from among the other options. If the '+' or '-' is omitted, a '+' is assumed. Extraneous blanks within the options field are ignored.

Option	Default	Meaning
ALIGN	- off	Allows nonalignment of data. Normally the compiler forces correct alignment of all data (half-, full-, or double-word, as required). This option allows the compiler to ignore "correct" alignment, conserving some amount of storage in the process. Execution speed may be slowed somewhat as a result.
BAR	+ on	Permits underbar (_) to be used as an alphabetic character.
BYTEALLOC	- off	Allows byte allocation. Normally the smallest unit of storage allocated by the compiler is the half-word. This option allows the use of byte storage for all objects having the range 0..255.
CASENEXT	+ on	Forces an error if a CASE index is out of range.
DEBUG	+ on	Forces generation of debugging information. The DEBUG option is equivalent to setting all of CASENEXT, INDEXCHECK, RANGECHECK, and DUMPTABLE.
DUMPTABLE	+ on	Produce debug tables for snapshots and post mortem dump. This option must be turned on/off before the first declaration of a procedure (or the main program) for it to be effective.
EJECT	- off	Forces a page eject: the current line will begin a new page. This option automatically resets itself to off whenever used.

Option	Default	Meaning
FULLXREF	- off	Forces the generation of a cross-reference listing of all symbols (including those within options fields. This option forces XREF+ and XPREDEF+.
IBM370	+ on	Compile code for IBM 370 if on; otherwise IBM 360.
ILIST	- off	Print object code as each statement is processed. (Intralist)
INDEXCHECK	+ on	Check index range in subscripts.
LIST	+ on	List source program; lines containing syntax errors always are listed.
MCCARTHY	+ on	McCarthy evaluation. This feature forces optimal evaluation of Boolean expressions.
OBJFILE	+ on	Produce object code on SPUNCH (even if PAR=LOADNGO).
OLIST	- off	Print object code after each procedure or function is processed and all fix-ups have been made.
PRECEDENCE	+ on	Normal arithmetic precedence. If on, the "usual" rules for arithmetic and logical precedence are used in evaluating expressions; if off, the PASCAL precedence is used. If R is on, the "extra" parentheses that PASCAL sometimes requires in order to correctly interpret relational expressions are not needed. (E.g., IF A+B>C-D THEN ...).
RANGECHECK	+ on	Perform subrange checking on assignments.
SEQUENCE	- off	Sequence number mode: if on, only columns 1-72 are read by the compiler; if off, columns 1-100 are read.
STANDARD	- off	If on, forces Standard PASCAL; if off, allows PASCAL/UBC extensions. This option forces $ DEBUG+, MCCARTHY-, PRECEDENCE-, BAR- $

Option	Default	Meaning
SUBTITLE	- off	This option accepts a string delimited by quotes and uses it for the subtitle line on all subsequent pages. E.g., SUBTITLE='THIS IS A SUBTITLE'
TITLE	- off	This option accepts a string delimited by quotes and uses it for page titling on all subsequent pages. E.g., TITLE='THIS IS A TITLE'
UNDERLINE	± on/off	Forces automatic underlining of all PASCAL reserved words. The default is off if running from a terminal. The default is on if running in batch.
XPREDEFS	- off	Forces the generation of a cross-reference listing of all predefined PASCAL/UBC symbols.
XREF	- off	Forces the generation of a cross-reference listing of all user-defined symbols.

The compiler implicitly uses the defaults

```
$ ALIGNED+, DUMPTABLE+, RANGECHECK+, INDEXCHECK+, CASENEXT+,
  IBM370+, LIST+, PRECEDENCE+, MCCARTHY+, OBJFILE $
```

at the beginning of a compilation.

4. Input/Output

This section describes the I/O facilities of PASCAL/UBC. It duplicates some portions of the PASCAL User Manual and Report, but clarifies a number of points. It also describes the I/O interface with MTS and a number of built-in procedures which have been added for file handling.

There are basically two levels of I/O in PASCAL: READ/WRITE and GET/PUT. While the READ/WRITE procedures are the most useful, GET and PUT are described first since they provide the basis for the READ/WRITE level.

4.1 GET, PUT, and PASCAL Files

A PASCAL file is defined using the FILE type generator:

```
VAR F : FILE OF TY;
```

There are two basic file classes in PASCAL -- TEXT files and all others. A file is a TEXT file when TY=CHAR. For non-TEXT files, each line is one element of type TY. For TEXT files, each line is an ARRAY OF CHAR, and reading progresses along one line before advancing to the next.

Each file F has a buffer variable F@, which is that component of the file which is currently accessible. The function GET(F) advances the buffer to the next component of the file. Successive calls to GET allow F@ to reference successive lines of the file (or successive characters in a TEXT file). When no components remain in the file, the function EOF(F) becomes TRUE.

Whenever EOF(F) is TRUE it is possible to add new components to a file. This is accomplished by placing the component in the buffer F@ and calling PUT(F) to append it to the file. This may be repeated any number of times. Note: if you wish to transport your programs to another system, you must assume that the contents of F@ are destroyed during the execution of PUT(F).

Text files have a number of peculiar properties. Firstly, all blanks are removed from the end of input lines. If F@ is positioned after the last character of a line, F@ will be set to a special character: EOL. At the same time, the built-in function EOLN(F) becomes TRUE. The next call to GET(F) advances F@ to the first character of the next line and returns EOLN(F) to its FALSE state.

Programming Notes

1. EOF(F) may become TRUE only when F@ contains EOL and a call is made to GET(F).

2. In Standard PASCAL, EOL doesn't exist; a blank is in F@ when EOLN(F) is TRUE.

3. In PASCAL/UBC, a blank input line contains a single blank (rather than no blanks).

4. ECLN is short for EOLN(INPUT). INPUT is the standard file INPUT, not a user-defined file of that name.

5. GET(F) is only allowed while EOF(F) is FALSE; PUT(F) is only allowed while EOF(F) is TRUE.

6. Due to the operation of MTS files, if a file is extended by reading until EOF(F) becomes TRUE and then writing on the file using PUT(F), one line number will be skipped.

An MTS file must be associated with each PASCAL file. The association is made via the PASCAL name. The following default associations are provided by PASCAL/UBC:

PASCAL name	MTS file
INPUT	logical unit SCARDS
OUTPUT	logical unit SPRINT
GUSER	logical unit GUSER
SERCOM	logical unit SERCOM
other	file name specified in PAR field of the $RUN command

Effectively all PASCAL/UBC files are external and must be specified. (But see the extensions to RESET and REWRITE described below.)

If declarations are provided for INPUT or OUTPUT they will completely override the standard assignments. (Use of READ and/or WRITE without a file name will refer to the standard file, not the declared file.)

4.2 I/O Using READ and WRITE

The READ and WRITE functions are the most convenient I/O mechanisms in PASCAL. There are, however, a few "glitches" because the GET/PUT mechanism historically was designed first. This section describes how to best use these functions.

The basic READ accepts two arguments:

```
READ(F,X)
```

where F is a TEXT file and X is a character. This call is equivalent to the code segment:

```
X := F@;  GET(F).
```

Note that X is assigned before the GET, so that X will not be the same as F@ following READ(F,X).

Programming Notes

1. F@ can be used as a look-ahead character.

2. When F@=EOL, the call READ(F,X) will set X to a blank. This aspect of READ is compatible with Standard PASCAL. Note that when X is set to blank at the end of a line:

   a. The EOLN(F) flag has just been turned off.

   b. The next input line has already been read. This is significant when F@ is associated with a physical terminal.

PASCAL/UBC includes a number of extensions to the basic READ function:

1. READ(F,I) with I an integer: reads an integer in free format. An arbitrary number of blanks (and lines) are skipped until the integer is found. The integer value replaces I, and F@ contains the next character following the integer. (If a "," immediately follows the number, it is skipped as well.)

2. READ(F,R) with R a real: same as READ(F,I) except that a real number is read.

3. READ(F,S) with S an ARRAY (...) OF CHAR: reads characters (starting with F@) and fills S. If EOLN(F) becomes TRUE before S is full, the remainder of S is filled with blanks. (If EOLN(F) is TRUE when READ(F,S) is called, the next line is read.)

4. READ(F,X1,X2,...Xn) means READ(F,X1); READ(F,X2); ... READ(F,Xn).

5. READ(INPUT,X,...) may be abbreviated READ(X,...).

6. READLN(F) skips to the next input line (i.e., F@ will be the first character of the next line).

7. READLN(F,X...) is an abbreviation for READ(F,X,...); READLN(F). READLN is an abbreviation for READLN(INPUT).

Programming Notes

1. Fixed format input of numbers is not (yet) available.

2. A library routine, SKIPBLANKS, may be used to skip blanks before reading a character or string.

3. If an EOF(F) condition occurs within a READ(F,X), the variable X will not be changed (except when X is of type CHAR). The EOF condition should be checked before the next READ, READLN, or GET is attempted. If EOF(F) occurs when reading field X in READ(F,X) or READLN(F,X) an error will occur (since READLN(F,X) is equivalent to READ(F,X); READLN(F), and READ(F,X,Y) is equivalent to READ(F,X); READ(F,Y)).

4. When no file is specified in READ or READLN, the standard INPUT file (rather than a user-defined file named INPUT is used.

5. The table below summarizes the synchronization between the input, the buffer variable F@, and the variable CH on successive calls READ(F,CH): The input consists of the characters A, B, EOL, C, D, EOL, EOF :

Effect of READ(F,CH)

Input	*	A	B	EOL	C	D	ECL	EOF
EOF(F)	F	F	F	F	F	F	F	T
ECLN(F)	T	F	F	T	F	F	T	UND
F@	EOL	A	B	EOL	C	D	EOL	UND
CH	UND	**	A	B	SP	C	D	SP

* - this column is prior to the first GET, READ, or RESET
** - this column is internal to READ(F,CH), so the value of CH is undefined.
T - TRUE
F - FALSE
SP - space (a blank character)
UND - undefined

Note that EOF(F) is defined as FALSE before the first GET or READ. PASCAL/UBC differs from standard PASCAL by the presence of the EOL character, and the existence of the first column in the above table. If READ is used instead of GET, the only difference is that PASCAL/UBC allows a line to be printed before the first input line is read.

The procedure WRITE is analagous to READ, but is used for output. It has a number of features for controlling the output format.

The basic character form WRITE(F,CH) is equivalent to:

```
F@ := CH;  PUT(F).
```

It writes a single character into file F. WRITE(F,X) will convert variable X to character format and write it into a fixed width field. The table below shows the available data types and their default field widths:

Type	Default Field-Width
CHAR	1
ALFA	10
BOOLEAN	10
INTEGER	10
REAL	22
ARRAY(M..N) OF CHAR	ORD(N)-ORD(M)+1

In situations where the default field width is not adequate, an explicit field width may be given:

```
READ(F,X:W)
```

W may be any expression which yields an integer value. Numeric values are right-justified in the field, while characters, strings, and Booleans are left-justified. If a numeric value can't fit in a field, the field width is expanded to accommodate the value. If a string, character, or Boolean doesn't fit, the string is truncated (on the right).

Some additional formatting details, with other formatting options, are described below:

1. If W is 0 for integers or reals, the number is printed in the minimum number of columns required for the value. This is useful when printing text to avoid unwanted blanks. Be sure to provide spacing around the numbers.

2. Hexadecimal output of integers may be achieved by making W negative. ABS(W) columns will be used to print the value.

3. Real numbers are normally printed in "guess" format. The number of significant digits is always maximized. In a field of width W, the first column is always blank, one column is reserved for the decimal point, and one more is used for the sign of the number if it is negative. One digit is always printed on <u>each</u> side of the decimal point. Let R be the number of columns remaining. If -4<=log(value)<R, the value is printed in fractional format (e.g., 123.45); otherwise it is printed in exponential format (e.g., 1.2345E-17). Reals are always rounded before

output. If the numeric value was exactly zero, no fractional part is printed. Otherwise, the precision is maximized to the limit of the field width or the numeric precision of the value, whichever is less.

4. Explict control over the precision and format of reals may be achieved using the call:

```
WRITE(F,R:W:D)
```

The "D" value gives the number of digits to the right of the decimal point. D also has some non-obvious effects on the output format:

a. If D is positive, the number is always output in fractional form.

b. If D is zero, no decimal point is printed (an integer is output).

c. If D is negative, the value is output in exponential format. ABS(D) is the number of digits to the right of the decimal point.

Programming Notes

1. Some day we hope to improve the notation for formatting so that discontinuous events don't occur, truncation rules are more flexible, and the notation is more transparent.

2. All formatting rules, even the justification and truncation rules, may differ in other implementations of PASCAL. The PASCAL standard is very vague in this area.

The WRITE function has a number of variations which correspond to the READ variations:

1. WRITE(F,X1,X2,...,Xn) is equivalent to WRITE(F,X1); WRITE(F,X2); ... WRITE(F,Xn).

2. WRITELN(F) starts a new output line. WRITELN(F,X,...) is an abbreviation for WRITE(F,X,...); WRITELN(F).

3. WRITE(OUTPUT,X) and WRITELN(OUTPUT) may be abbreviated by WRITE(X) and WRITELN, respectively, when OUTPUT is the standard PASCAL output file.

In PASCAL/UBC and certain older PASCAL compilers, there is a character, EOL, which will terminate the output line. This character may be used as a parameter to WRITE instead of making several WRITELN calls. Note that EOL is no longer part of the standard PASCAL language.

At UBC and many other PASCAL implementations, the first character of each line which is sent to the printer is a "carriage control" character. It controls the line spacing on the printed page. This must be provided by the programmer (since the PASCAL compiler doesn't know (or care) which files eventually will be printed). While any MTS carriage control character may be used, those standard at a number of installations are:

' '	Single space (before printing)
'0'	Double space (before printing)
'-'	Triple space (before printing)
'1'	Skip to new page (before printing)
'+'	No space (overprint)

In an attempt to avoid the use of charriage control characters, Standard PASCAL has introduced the function, PAGE(F). In PASCAL/UBC, this is equivalent to WRITE(F,EOL,'1',EOL).

Programming Notes

1. EOF(F) may become TRUE only when F@ contains EOL and a call is made to GET(F).

2. Because PASCAL has no way to specify the maximum line length for a TEXT file, the programmer is responsible for starting new lines when needed. PASCAL/UBC will automatically start a new line only when 255 characters have been placed on a single line. The formatting routines do not currently check to see if ample space remains on a line for an entire number or string. (See the LINELENGTH function below.)

4.3 Using PASCAL Interactively

The standard PASCAL READ and WRITE procedures described above and in [1,2] are designed more for a batch environment than an interactive one. The following paragraphs describe how PASCAL/UBC may be used interactively under MTS.

READLN(...) has three effects: 1) it copies information from the system's internal input buffer into the variables specified in its argument list; 2) it flushes the buffer; 3) it refills the buffer from the file (device) specified. If READLN is used from a terminal it will ask for a new line before the program prompts the user. The sequence: READLN; READ(...) ignores all input currently in the input buffer, and begins reading after retrieving a new input line.

This may be undesirable if several data items are to be input and the programmer desires to be notified if one or more items are missing. Instead of READLN; READ(X,Y) the following sequence may be prefered:

```
READLN;
READ(X);
WHILE INPUT@=' ' DO GET(INPUT);
IF INPUT@=EOL THEN WRITELN('ENTER Y');
READ(Y);
```

Since READLN discards unread input in the input buffer, loss of information may occur. You may wish to solve this problem by using a sequence such as:

```
WHILE INPUT@=' ' DO GET(INPUT);
IF INPUT@¬=EOL THEN WRITELN('EXTRA DATA SUPPLIED');
READLN;
```

If you wish to enter data on the same line as the prompt message, use the carriage control character specified in UBC TERMINALS [5]. Currently the appropriate control character is the ampersand (&). For instance:

```
WRITELN('&ENTER X: ');
READLN; READ(X);
```

It is important to remember that PASCAL will display output only after an EOL has been transmitted. This may be accomplished either by WRITE(...,EOL,...) or by WRITELN(...). Otherwise, the "written" information is held in a system output buffer until an EOL is sent.

In some applications it may be desirable to read an entire line at once, rather than item by item or character by character. INPUT may be redefined as

```
VAR INPUT : FILE OF ARRAY (1..100) OF CHAR;
```

(or any other appropriate size). You may now issue a GET and the entire line will be read (with blank fill on the right, if necessary). Note that any numeric conversions must now be done manually by the programmer.

4.4 Extension to non-TEXT Files

In the corrections to the PASCAL User Manual and Report, the READ and WRITE procedures are extended to non-TEXT files as follows: Given

```
VAR
    X  :  TY;
    N  :  FILE OF TY;
```

the call	means
READ(N,X)	X := N@; GET(N)
WRITE(N,X)	N := X; PUT(N)

No other forms of READ and WRITE are allowed.

In the future it is expected that we will expand the non-TEXT READ and WRITE definitions to be entirely compatible with the definitions for the TEXT versions of these routines.

4.5 RESET and REWRITE

PASCAL provides two functions for positioning a file. RESET(F) positions a file at the beginning, reads the first line, and places its contents in F@. If the file was empty, EOF(F) becomes TRUE, otherwise it is FALSE.

REWRITE(F) positions the file at the beginning, and sets EOF(F) to TRUE. This allows the file to be written. Any previous information in the file will be lost.

Programming Notes

1. Due to the nature of MTS line files, a call to REWRITE will perform an MTS $EMPTY command to empty the file. If a line number range is specified, however, the file is not emptied. When line number ranges are used it is the responsibility of the programmer to see that unwanted lines will not remain in the file.

2. Standard PASCAL has no provision for extending a file except to read all lines of the file.

3. Standard PASCAL requires that all files except INPUT be RESET or REWRITEn before use. PASCAL/UBC makes no such requirement.

PASCAL/UBC offers extended forms of RESET and REWRITE which accept a second parameter. The parameter may be:

a. An integer value in the range 0..19 representing an MTS logical unit.

b. The name of an MTS logical unit: 'SERCOM ', 'GUSER ', 'INPUT ', 'OUTPUT '.
Note:

i. The trailing blank is required.

ii. INPUT and OUTPUT replace SCARDS and SPRINT, respectively.

iii. SPUNCH is not currently available.

c. An MTS file name:
e.g.,

```
'-ABC '
'FILE1 '
'USER:DATAFILE '
'-XYZ(55,128)+QRS¬IC '
```

Note:

i. The trailing blank is required.

ii. The MTS logical unit names listed above cannot be used as a file name unless a line number range or modifier is specified.

Programming Note

1. REWRITE(F,'FILE(*L+1) ') may be used to extend file FILE without first reading through the entire file. However, a later RESET(F) will position F at the first added line, not the first line of the file.

4.6 PASCAL/UBC I/O Functions

The functions described in this section have been added to facilitate the interface with MTS line files. None of them are standard.

DELETELINE(F,line) first POSITIONs F to the specified line, then deletes the line from the file (if it exists), and leaves the file POSITIONed at line. After a DELETELINE either a GET or a PUT is valid. Reading/writing will begin at the specified line.

LINELENGTH(F) returns the length of the last line read from file F. This is the length of the line as it appeared in the file. If F is a TEXT file being used for output, LINELENGTH(F) is the length of the current line (not including any character still in F@ which has not yet been PUT to F).

LINENO(F) returns the integer line number of the last line read or written to file F. This line number is the MTS line number times 1000.

OPENED(F) returns TRUE if a RESET, REWRITE, GET, PUT, or POSITION has been issued to this file, and it returns FALSE otherwise.

POSITION(F,line) is a procedure which positions file F to the specified line. Following a POSITION, EOF(F) remains unchanged, but either a GET or a PUT is valid. If the specified line does not exist, a GET will retrieve the next line of the file, or will set EOF(F) if "line" is beyond the end of the file. In all other cases, EOF(F) will be FALSE following a GET, and LINENO may be called to determine whether "line" actually existed. After PUT is called following a POSITION, EOF(F) always will be TRUE.

When POSITION is used with TEXT files, several additonal actions occur:

1. If the file has been used for output prior to the POSITION, the current buffer is written out if LINELENGTH(F)>0.

2. After the POSITION, EOLN(F) is TRUE.

Programming Note

1. If OPENED(F) is FALSE (the file has not been used), a call to LINENO(F) or LINELENGTH(F) will cause an addressing exception.

5. PASCAL Libraries, Standard Functions and Procedures

Users may reference functions in the standard PASCAL library PASC:LIB, libraries of their own, and functions in any of the system libraries.

5.1 The Standard PASCAL Library: CS:PASCALLIB

The PASCAL library includes system routines for perfcrming input/output plus various other procedures and functions. The source programs for CS:PASCALLIB reside in files PASC:LIB.S and PASC:MON.S and these files should be examined to determine the library's precise contents. Currently PASC:LIB.S includes two random number generators: RAND, RANDU; three exponentiation functions: PWR (integer raised to an integer power), RPWR (real raised to an integer power), and RRPWR (real raised to a real power); a routine (SKIPBLANKS) to skip over blanks in the standard input; and the SNAPshot and post mortem dump packages.

All routines in CS:PASCALLIB have names beginning with the three characters "PA#". To avoid possible conflicts, the user should avoid using these three characters as the initial segments of any external routine's name.

You may include FORWARD declarations for all the standard PASCAL library routines by saying

```
$CONTINUE WITH PASC:LIB.S(100,199) RETURN
```

In addition to the run-time support routines mentioned above, the standard library contains the assembly language PASCAL monitor PSCLMON#. The monitor performs various I/O tasks, acquires and releases NEW space, interfaces with MTS, etc. The source for the monitor resides in PASC:MON.S.

5.2 Constructing a User Library

Users may wish to construct their own libraries of PASCAL programs in object form to save the cost of repeated compilations. Object modules may be accumulated from several compilations and placed in a single file. After a recompilation of several routines, the MTS program *ROBJ may be used to replace obsolete modules. For selective loading of routines the MTS routine *SGEN may be used to process the file, forming an MTS library. *ROBJ and *SGEN documentation may be found in UBC LOADER [6].

To create a PASCAL library given a source program in MYSOURCE one should run the compiler as follows:

```
$RUN CS:PASCAL SCARDS=MYSOURCE SPUNCH=MYLIBRARY
```

where MYLIBRARY is the name of the library file. After the "END;" of the final procedure, a period "." indicates the end of the source program. (A main program may be present if desired.) Data, if any, may begin on the first line following the ".".

One then might use the newly created library as follows:

```
$RUN CS:PASCAL SCARDS=MYPROGRAM+DATA PAR=LINK,LI=MYLIBRARY
```

Each procedure (function) included in a library (or those compiled separately and later linked together) must satisfy the following restriction:

> "Each procedure (function) should be compiled in the presence of identical declarations (LABEL, CONST, TYPE, VAR)."

This restriction may be relaxed somewhat -- CONST and TYPE declarations which are not used in the current compilation need not be present. However, it most often will be simplest to maintain a file containing all requisite global declarations and $CONTINUE WITH it prior to each compilation. This will ensure that the above restriction always is satisfied.

If the global VAR section is totally absent, the resulting library may be used with any PASCAL program.

Due to restrictions imposed by MTS the names of every external procedure or function which is to be separately compiled must be unique in their first seven characters. No warning is issued if this restriction is not heeded.

5.3 Standard Functions and Procedures

The following tables list those routines whose actions are modified from the corresponding routines in Standard Revised PASCAL and those routines which are unique to PASCAL/UBC.

Modified Routines

HALT INSERT PACK READ READLN RESET REWRITE UNPACK WRITE WRITELN

Unique Routines

DECR DELETELINE INCR LINELENGTH LINENO MARK MAX MIN OPENED POSITION RELEASE SNAP SUBSTR

Refer to the Index for descriptions of the actions of all of the above routines.

6. Language Differences and Extensions

There are numerous differences between standard revised PASCAL [1,2] and the PASCAL/UBC language processed by this compiler. These are described below, in section-by-section correspondence with the PASCAL User Manual and Report [1]. In the discussion below:

Code	Meaning
R	(Restriction). A violation or conflict with Standard Revised PASCAL. Programs may require modification to work under PASCAL/UBC.
E	(Extension). An upwards compatible extension to Standard Revised PASCAL.
D	(Deviation). A deviation from Standard Revised PASCAL which may or may not require modification to work under PASCAL/UBC.
C	(Clarification). The specification of something the User Manual or Report leaves undefined or unspecified. Such items may or may not require modification to work under PASCAL/UBC.

6.1 Differences and Extensions from the User Manual

Section 1 - Notation and Vocabulary

E Names may be composed of any number of upper or lower case alphabetic characters, the digits 0 - 9, and underbar (if the BAR option is '+'). The first character of a name must be alphabetic (or _).

C PASCAL/UBC only retains the first 10 characters of names; thus the user must ensure that he forms unique names within the first 10 characters. Names of external functions and procedures must be unique within their first 7 characters due to restrictions imposed by MTS.

E ASCII braces are now supported as comment delimiters. Comments also may be delimited by pairs of double quotes " <comment> ". (* *) are the standard comment symbols.

E The dollar sign ($) is used to delimit compiler options. The text between pairs of $'s is ignored by the compiler proper.

R The biliteral ¬= and the word VALUE are special symbols, as are |, &, and ¬.

E Square brackets to define array subscript expressions may be replaced by parentheses. Alternatively, they may be replaced by the biliterals (. and .) .

D An uparrow to denote pointer and file references is replaced by the symbol @ .

E Brackets to define powersets may be replaced by the biliterals (. and .) .

C The compiler reads columns 1 to 100 of each input line; the rest of the line is ignored. The compiler may be instructed to read only columns 1 to 72 through use of a compiler option.

E The symbols &, |, and ¬ may be used in place of AND, OR, and NOT, respectively. The biliteral ¬= may be used in place of <> .

Section 2 - The Concept of Data

R Integers have the range $2^{-31}..2^{31}-1$.

R Reals are defined according to the IBM 360/370 long real floating point format. A 54-bit matissa is used providing a precision of approximately 16 decimal digits.

E Hexadecimal quantities may be specified (e.g., #4C represents the decimal value 76). Hexadecimal numbers are treated as being of type INTEGER.

Section 3 - The Program Heading and the Declaration Part

D Program headings, i.e., 'PROGRAM <identifier> ; ', are not currently used. PROGRAM is not a reserved word in PASCAL/UBC.

E Initial values may be declared for simple global variables and one-dimensional global arrays. SETs may not be initialized using VALUE. This facility is relatively untested and is NOT recommended. Almost no type checking or bounds checking is done. Caveat emptor! The values are declared after the global VAR declarations. Any procedure or function may have a VALUE section as well as the main routines. Initialization occurs each time the procedure or function is entered. The syntax in BNF is:

```
<value-part> ::= <value-part> <value-assigment> ;
                 | VALUE <value-assigment> ;

<value-assigment> ::= <identifier> = <constant>
                      | <identifier> = ( <constant-list> )

<constant-list> ::= <constant-list> , <dup-const>
                    | <dup-const>

<dup-const> ::= <constant> | <integer> * <constant>
```

Section 4 - The Concept of Action

See below: section 9.2, subpart "Section 9, Statements".

Section 5 - Scalar and Subrange Types

No changes.

Section 6 - Structured Types in General--the Array in Particular

E The word PACKED is ignored.

R The UNPACK and PACK procedures may be used for their data transfering functions between all (length-) compatible pairs of arrays. No actual (un)packing occurs as data is implicitly "packed" on byte addressable machines such as the IBM 360/370.

Section 7 - Record Types

C PASCAL/UBC does not detect duplicate labels in the variant portions of record declarations.

Section 8 - The Set Types

R Restrictions on sets are: the base type can have a maximum of 256 values. Subranges of integers must be between 0 and 255. CHAR is now allowed as a base type.

Section 9 - File Types

E The standard functions RESET and REWRITE may be applied to any files (including INPUT and OUTPUT).

R Files may not be components of ARRAYs, RECORDs, or FILEs.

R The NEW function cannot (yet) create a file.

Section 10 - Pointer Types

R The function DISPOSE in the old standard does not exist.

E Standard procedures MARK and RELEASE have been added to maintain the 'NEW' stack. Each takes an integer variable as an argument. Execution of MARK stores the current stack pointer in the argument. Execution of RELEASE restores the stack pointer to the location indicated by the argument. The argument must not be used for any other purpose in the program. These are dangerous functions since pointers may no longer be valid following a RELEASE.

Section 11 - Procedures and Functions

E PACKED is ignored in PASCAL/UBC.

R Any procedure/function which is used before it is defined must be declared with all its parameters as 'FORWARD' before its first use. Then when its body is defined the parameter list and result type must be omitted.

E Separate compilation of global procedures/functions is allowed. If the compiled procedure(s) is (are) to be combined with other procedures, then the global declarations for all compilations must be identical. Procedure and function declarations for any global level procedures or functions which are not included in the compilation must be declared FORWARD in order to generate the proper argument lists and to allow the external references to be resolved. There must be a "." at the end of the program. (The main program -- from BEGIN to END -- may be omitted.)

E A procedure or function may be declared 'FORTRAN' in which case the compiler will produce the correct calling sequence to the named FORTRAN subprogram. Thus the entire FORTRAN library is available to the user, as are Assembly Language and other routines written using standard FORTRAN calling conventions. An optional string may follow FORTRAN, (e.g., PROCEDURE PROC1; FORTRAN 'FNAME';), in which case the FORTRAN routine FNAME will be invoked as a result of any PASCAL reference to PROC1.

E A procedure or function may be declared 'EXTERNAL'. At the top level, this has the same effect as 'FORWARD'. But used in an inner procedure EXTERNAL will force the compiler to reference the routine as if it were declared at the top level (as an outermost procedure) so that an inner procedure may refer to a separately compiled top level procedure which is not present during the current compilation.

Section 12 - Input and Output

E PASCAL/UBC has a special character, EOL, designating the end of a line.

D If F is a TEXT file, then F@=EOL initially.

C Note that even when F@=EOL, READ(F,C) will set C to blank as in standard PASCAL. WRITE(F,EOL) may be used instead of WRITELN(F).

E PASCAL/UBC recognizes READ(F,S), where S is of type ARRAY (i..j) OF CHAR for any i and j. This reads characters from the current line and places them into S(i), S(succ(i)), ..., until (j-i+1) characters have been read. If, however, an EOL character is encountered before input is complete, the remainder of S is filled with blanks. In this case F@ is the EOL character.

E WRITE(F,C:W), where C is of type CHAR, will left justify C. Similarly, WRITE will left justify strings.

E READ(F,X) and WRITE(F,X), where F is not TEXT are both recognized.

E RESET(F,G) and REWRITE(F,G) both accept an optional second argument, G, an ARRAY(..) OF CHAR which contains the name of an MTS file to be dynamically associated with the PASCAL name F. The file name must be left justified and have at least one trailing blank. In both cases, remaining output is sent to file F, F is closed, F is associated with the file G, and then F (G) is opened for reading or writing, respectively. Alternatively, G may be an integer value in the range 0..19, in which case G refers to the corresponding MTS logical unit.

Section 13 - PASCAL 6000-3.4

R Segmented files do not exist, nor do the associated standard functions EOS, PUTSEG, GETSEG.

E PASCAL/UBC provides access to "external procedures" through use of the FORWARD, EXTERNAL, and FORTRAN declarations. EXTERN does not exist in PASCAL/UBC; however, its effect may be achieved by using a FORWARD or EXTERNAL declaration.

R No program headings are allowed.

D The standard identifier MAXINT is defined as

```
CONST MAXINT = 2147483647; (* = 2^31-1 *)
```

R The statement IF ABS(I) > MAXINT THEN WRITE(' TOO BIG') is essentially useless because no integer may be larger than MAXINT. The statement IF I < -MAXINT THEN ... may be useful since the smallest integer possible in the PASCAL/UBC implementation is -MAXINT-1.

D The type REAL is defined according to the IBM 360/370 long real floating point format. A mantissa of 54 bits is provided, corresponding to approximately 16 decimal digits. The maximum absolute value is approximately 10^{+75}, and the minimum non-zero absolute value is approximately 10^{-75}.

D A value of type CHAR is an element in the EBCDIC character set. 256 distinct values exist, although many are not printable characters. EOL is defined as CHR(0).

R The word SEGMENTED has no special significance in PASCAL/UBC.

E The maximum cardinality of the base type of a set is 256.

C The standard types ALFA and TEXT are predefined as

```
TYPE ALFA = ARRAY (1..10) OF CHAR;
     TEXT = FILE OF CHAR;
```

D The entire MTS library is available to the user via the FORTRAN declaration. Thus the predefined procedures and functions defined in this section are available as follows:

PASCAL 6000-3.4	PASCAL/UBC
DATE	UBC DATE
HALT	HALT
LINELIMIT	Use $RUN xxx P=yyy
MESSAGE	No corresponding function exists
TIME	UBC TIME
CARD	Not (yet) implemented.
CLOCK	UBC TIME
EXPO	Not (yet) implemented.
UNDEFINED	Not implemented.
EOS	Not implemented.

PUTSEG, GETSEG, and the corresponding extensions to REWRITE and RESET do not exist in PASCAL/UBC.

Section 14 - How to use the PASCAL 6000-3.4 System

R Since DISPOSE has been removed from the list of "standard functions", it is not supported at UBC.

C A jump into the range of a FOR or WITH statement from outside its range is undefined; no error message is produced.

E Function LINENO(file) returns the line number of the most recently read line. This function must not be used before the first read from the file. The line number is the MTS line number times 1000.

E Integer and real valued maximum and minimum functions have been implemented: MAX(a1,a2,...) and MIN(a1,a2,...). These functions accept an arbitrary number of integer and/or real arguments, and will return an integer value if all arguments are integers or a real value if not.

E A limited substring function has been added: SUBSTR(V,I,L) where V is a constant or variable, I is an integer valued expression, and L is a constant. V must be an array of characters. The result is the substring of V which starts at character I and is L characters long. If V has bounds A and B, then A<=I<=(I+L-1)<=B .

E The standard variable RCODE is predefined to be of type -32768..32767. It functions as a "return code" and is available as a "global variable" to all PASCAL routines. Routines declared FORTRAN will automatically set RCODE to the appropriate return code value prior to return to their invoking PASCAL procedure.

6.2 Differences and Extensions from the Report

Section 3 - Notation, Terminology, and Vocabulary

D The character set used by PASCAL/UBC is the IBM EBCDIC character set, with one modification: There is the character EOL=CHR(0). This character causes a new line to be started on output to a TEXT file. It follows the last character on an input from a TEXT file.

C PASCAL/UBC <letter>'s now include lower case.

E Alternate symbols

..	may	be	used	for	:
¬=	"	"	"	"	<>
(.	"	"	"	"	[
.)	"	"	"	"	]
&	"	"	"	"	AND
|	"	"	"	"	OR
¬	"	"	"	"	NOT
@	replaces				↑

R All reserved words must appear in upper case.
PROGRAM is not reserved.
VALUE is reserved.

E Comments may be delimited by braces, double quotes ("), or the biliterals (* and *).

E Dollar signs ($) are used to delimit compiler options.

Section 4 - Identifiers, Numbers, and Strings

E "_" may appear as a <letter> in an identifier.

R Only 10 characters of an identifier are significant. Procedure and function identifiers must differ in their first 7 characters for proper program execution.

D INTEGERs are restricted to the subrange $-2^{31}..(2^{31}-1)$. REALs are restricted in magnitude to approximately $\pm 10^{+75}..\pm 10^{-75}$ and 0. Approximately 16 digits of precision are retained.

R String constants must fit on one source line.

E Hexadecimal numbers may be referenced by preceding them with "#". These numbers are treated as if they had integer type.

Section 5 - Constant Definitions

E The definition CONST NULL=NIL; is permitted in PASCAL/UBC.

Section 6 - Data Type Definitions

E The word PACKED is ignored.

R ARRAYs, FILEs, and RECORDs may not have FILEs as components.

R SETs are restricted to a subrange 0..255, and to scalar types having at most 256 elements.

E Parentheses may be used instead of brackets in array declarations.

C A type identifier may only be referenced after (not inside) its declaration. The only exception is a type identifier in a pointer declaration (@typeid). This identifier can be defined after such a use.

E There is the standard type ALFA:

```
TYPE ALFA = ARRAY (1..10) OF CHAR;
```

Section 7 - Declarations and Denotations of Variables

E Parentheses may be used instead of brackets in array references.

E There is a standard variable RCODE. It will contain the return code upon return from FORTRAN routines. It also may be used within a PASCAL program.

E A pointer will be dereferenced automatically if it is followed by a "." and a field name.

Section 8 - Expressions

E The square brackets "[" and "]" in <set> may be replaced by "(." and ".)".

R The set <element> form "<expression>..<expression>" is not implemented.

C No range check is made on intermediate results of expression computations.

C No range check is made on set elements.

Section 9 - Statements

R A label may have at most four digits. (This adopts the suggested restriction at the end of the Report.)

E An expression having type ARRAY ... OF CHAR may be assigned to any ARRAY ... OF CHAR variable which is at least as long as the expression. Blank extension occurs on the right if necessary.

C A VAR actual parameter must have the identical type as the corresponding formal parameter. One cannot be a subrange of the other.

C A GOTO into the range of a FOR or WITH from outside the range has unpredictable results. No warning message is given.

E A ";" may optionally precede "ELSE" in an IF statement.

E The symbol "<>" may appear as a label in a CASE statement. The corresponding statement is executed if the value of the expression does not appear in any other label of the statement.

C If a CASE statement is executed and the expression has no corresponding constant value an error occurs (unless there is a <> label). If the CASENEXT option is off, the next statement is executed with no message. If the CASENEXT and RANGECHECK options are both off and there is no <> label, no range check is made on the expression. Label>

R CASE labels must lie in the range -32768..32767.

C In PASCAL/UBC the method of statement selection for CASE labels depends on the density of labels within their numeric range. Thus it is possible to have only labels 1: and 1000: without undue memory overhead.

C In a FOR statement, if the initial value is greater than the TO value (less than the DOWNTO value), no assignment is made to the control variable. Otherwise, its value is the limit value. The control variable thus is accessible outside of the loop after normal loop termination.

E A WITH statement may contain a pointer to a record. It will be dereferenced automatically.

Sections 10 and 11 - Procedure Function Declaration

R A procedure or function declaration must appear before the first reference to that procedure or function. Recursive routines still may be defined by separating the procedure (function) heading from the procedure (function) body. When this is done, an alternative form of <procedure declaration> is used:

<procedure heading> FORWARD

(That is, the word FORWARD replaces the code block.) An analogous extension is made for functions. The code block is presented later, preceded by a procedure (function) introduction:

PROCEDURE <identifier> ; <block> or
FUNCTION <identifier> ; <block>

Note that the parameter list is not repeated with the code body.

E FORTRAN-compatible routines may be called using the declaration:

<procedure heading> FORTRAN or
<procedure heading> FORTRAN '<name>'

VAR parameters should be used when the routine returns a value in a parameter. If such a result is irrelevant, a value parameter may be used. An optional string may follow FORTRAN designating the name to be used in the ESD entry for the specified routine. For example, PROCEDURE FOO; FORTRAN 'ALIAS';

E Separate compilation of procedures and functions is possible. Such a procedure (function) may be invoked by including a FORWARD declaration in the main program block. The user is responsible for seeing that the procedure (function) declaration used while compiling the program is identical to that used in referencing the program. If a separately compiled procedure refers to any main program variables, the CONST, TYPE, and VAR sections must be identical with those in the calling program.

E If an inner procedure refers to a procedure which is not present in the current compilation the procedure should be declared EXTERNAL rather than FORWARD.

E The RESET and REWRITE procedures opticnally may take a seccnd parameter.

R The NEW(P) procedure will not work properly when P is a pointer to an object of type FILE.

R The DISPOSE procedure found in some editicns of the Manual and/or Report is not implemented.

E No argument of PACK or UNPACK needs the word PACKED in its declaration.

E PUT(F) is valid when EOF(F) is false if there has been no prior use of F, or when POSITION has been called pricr tc the PUT.

E Additional functions:

INSERT(I,J,K:INTEGER) : INTEGER
The result is (I*2**J OR K). J must be non-negative. A full-word logical OR is performed.

LINELENGTH(F:FILE) : INTEGER
The integer result is the length of the last line which was obtained via GET or READ.

LINENO(F:FILE) : INTEGER
The integer result is the MTS internal line number (printed line number * 1000) of the last line read from file F.

MAX(a1,a2,...) and MIN(a1,a2,...).
These functions accept an arbitrary number of integer and real arguments, and will return an integer value if all arguments are integers or a real value ctherwise.

OPENED(F:FILE) : BOOLEAN
Returns TRUE if F has been opened and FALSE otherwise.

SUBSTR(S,F,L) : ARRAY (1..L) OF CHAR
S is a character string, F a scalar, and L an integer constant. The result is the substring of S starting with the character at position F and containing L characters.

E Additional procedures:

DECR(V:variable)
V is a variable of any simple scalar type. DECR(V) is equivalent to: V := PRED(V).

DELETELINE(F:FILE; N:INTEGER)
F is a file and N is an MTS line number. The file is positioned at the specified line and the line is deleted (if it was present). The file is left positioned so that the next GET, READ, PUT or WRITE will occur beginning at the specified line.

HALT
Stops execution and returns to the system.

INCR(V:variable)
V is a variable of any simple scalar type. INCR(V) is equivalent to: V := SUCC(V).

MARK(VAR N:INTEGER)
Saves the current NEW stack position in N.

POSITION(F:FILE; N:INTEGER)
F is a file and N is an MTS line number. The file is positioned so that the next GET, READ, PUT, or WRITE will occur beginning at line N. If line N doesn't exist in the file, the next line will be used. LINENO may be used to determine the actual line which was accessed.

RELEASE(N:INTEGER)
Restores the NEW stack to the state indicated by N. All allocated storage made subsequent to the MARK(N) is now inaccessible. Note that this procedure is extremely dangerous because pointers may now refer to inaccessible memory.

SNAP(N:INTEGER)
Invokes the snapshot package and requests that the last N activations be displayed. N should be a positive integer. If no arguments are given, the default value is N=1. A display of all activations back to the PASCAL monitor may be obtained by saying SNAP(MAXINT).

Section 12 - Input and Output

C READ and WRITE now may be used on both TEXT and non-TEXT files.

E READ may be used to read a character string. Warning: the input line will be extended with spaces to fill any number of string variables. READLN should be used to start a new line.

E A WRITE of a REAL number uses exponential format when :W:D is specified and D is negative. The absolute value of D is the number of digits to be printed to the right of the decimal point. If :D is not specified, fractional or exponential format is selected based on the size of the number.

C The output generated by a WRITE is right justified for reals and integers, and is left justified for all other types.

D When EOLN(F) becomes TRUE, F@ will have the value EOL instead of ' '. Execution of READ(F,C) where C has the type CHAR, will set C to ' ' as in standard PASCAL.

C All blanks are eliminated from the end of input lines of TEXT files.

Section 13 - Programs

R Program headings are not implemented. A program is "<block> ." .

E Procedures and functions may be separately compiled by eliminating the <statement part> from the program <block>. The period is still required.

E A <value part> may follow the <variable declaration part> to provide initial values for simple variables and one-dimensional arrays. SETs may not be initialized with VALUE. This feature is relatively error-prone, and its use should be avoided. The syntax is:

```
<value-part> ::= <value-part> <value-assigment> ;
                 | VALUE <value-assigment> ;

<value-assigment> ::= <identifier> = <constant>
                      | <identifier> = ( <constant-list> )

<constant-list> ::= <constant-list> , <dup-const>
                    | <dup-const>

<dup-const> ::= <constant> | <integer> * <constant>
```

6.3 The STANDARD Option

A check for compatibility with standard PASCAL may be made by specifying the option $STANDARD+$. Use of the extensions described above are generally marked as errors under this option. Differences are not marked, and restrictions are still in effect. A few extensions are not flagged:

Alternate symbols, such as & and " still may be used.

10 characters are still used to distinguish identifiers, instead of 8 as suggested at the end of the Report.

The use of PACKED variables is not distinguished from the use of unpacked variables.

A call to WRITE having a :D expression will not check for the use of a negative value to produce exponential format.

The use of the RCODE variable is not flagged.

Some built-in procedures found in other implementations also are not flagged.

7. Miscellaneous Implementation Notes

A complete description of the PASCAL/UBC implementation may be found in the Implementation Guide [7]. These notes are designed to aid the casually interested user.

7.1 Communication with FORTRAN

This section is designed to aid the user who wishes to write PASCAL programs which communicate with routines written in other languages (e.g., FORTRAN, Assembler).

PASCAL/UBC uses the following storage allocations:

Type	No. Bytes
CHAR	1
BOOLEAN	2
INTEGER	4
SET	2-32
REAL	8
scalar and subrange types	2,4
'character string'	length of string

Note that if the BYTALLOC option is on, scalar types may have length 1. Scalar, subrange, and set types are aligned on half/full/double-word boundaries unless the ALIGN option is on.

The constant FALSE is represented internally by the halfword #0000; the constant TRUE is represented internally by the halfword #0001.

PASCAL files are not compatible with FORTRAN or MTS files. They should not be used as parameters. Their internal representation is subject to change.

PASCAL will generate a standard FORTRAN calling sequence if the word "FORWARD" is replaced with "FORTRAN". In other respects the procedure heading is standard. Either value or VAR parameters may be passed to FORTRAN (Assembly Language, etc.) and will work correctly. I.e., if the called program modifies a value parameter, the corresponding actual parameter in the PASCAL program will not be changed. Procedure parameters should not be passed, although predictable results will occur if the procedure lies in an outer nesting level. Note that an interface is generated for all routines declared FORTRAN, so these routines should not be passed out as parameters. (However, they may be passed as parameters to other PASCAL procedures.)

When a FORTRAN routine returns to PASCAL the predefined variable RCODE is set to the value of the FORTRAN return code. Thus PASCAL may distinguish between a RETURN and a RETURN i. (And similarly, PASCAL may obtain the return code value set by any system routine which is accessed via the FORTRAN mechanism.)

The names of all FORTRAN routines and all external routines declared FORWARD must be unique within their first 7 characters due to restrictions imposed by MTS.

7.2 Communication with Assembly Language

An Assembler program which has been called by a PASCAL program can, in turn, call another external PASCAL procedure so long as certain rules are followed. The calling routine must:

1. Restore registers 2 and 12 from its calling program.

2. Use register 2 as a base register for a DSECT. The DSECT contains the following fields:

```
SAVE    DS  18F
RESULT  DS  F       only present for functions
PARAM1  DS  ...
...     DS  ...
```

   a. Store register 12 in SAVE.

   b. If the routine is a function, place the address of the result field in RESULT. Results from PASCAL functions are placed directly in memory.

   c. Each parameter, in the order declared. For a VAR parameter, use DS F and insert the address of the actual parameter. For a value parameter, use a DS for the variable itself, and place the value of the variable in this field. All 2-, 4-, and 8-byte scalar, set, and pointer fields are half-, full-, and double-word aligned, respectively, unless the N option was on when the program was compiled. The parameters appear in the order in which they are declared.

3. Use registers 13, 14, and 15 as usual. The external name consists of the first 7 characters of the PASCAL routine name, followed by enough "$"'s to make the total name length 8. (E.g., FN becomes FN$$$$$$, and PASCALPROG becomes PASCALP$.)

4. On return, register 15 will not contain a return code. A function result will be in the result field, not in register 0.

To call a PASCAL main program, invoke the PASCAL monitor at entry point PSCLMON#. An alternate (completely equivalent) entry point is PSCLMN.

The names of all Assembly Language routines must be unique within their first 7 characters due to restrictions imposed by MTS.

An Assembly Language routine may send back a return code to its parent PASCAL program by setting register 15 in the usual way during the exit sequence. The parent PASCAL program may retrieve the value of the return code via the RCODE standard variable.

8. Snapshot and Post Mortem Dump Packages

This section contains a preliminary description of the snapshot and post mortem dump packages. The packages are currently under development, and are subject to change with little notice. No interactive facilities are currently available (even though the documentation below implies that they are).

Usually when a run error occurs the PASCAL monitor is invoked and it transfers control to a special run error supervisor. This supervisor allows NERR run errors to occur, after which it calls the standard HALT procedure. If a compilation is done with the Debug option turned on (this is the default) special tables are produced which allow PASCAL to print an informative display of all currently active variables with their associated values each time the run error supervisor is activated.

If one is running interactively and Debug is on, the run error monitor will instead of producing its standard display and continuing execution, enter an interactive loop. It then will process user requests for the display and/or modification of any active variable(s), after which the user may continue (or terminate) the execution. This interactive feature may be disabled by running with PAR=BATCH.

The snapshot package may be invoked directly by the user via the standard procedure SNAP. The values of all current variables will be displayed and execution will continue if running in batch mode, or the special interactive loop will be entered if running interactively.

SNAP takes an optional integer argument specifying the number of levels back in the execution stack which are to be displayed. The default value is one. Display of all levels back as far as the monitor may be achieved by saying SNAP(MAXINT). The integer argument should be positive; negative values are reserved for future use.

9. Warning and Error Messages

Source errors are flagged by the compiler as they occur and are summarized at the end of the compilation. Each error is flagged by a vertical bar (|) under the last character of the offending word or symbol. Several errors may be detected at the same position in the input leading to a sequence of two or more vertical bars in a row. Each bar corresponds to its respective error number printed on the right of the same line.

The text corresponding to each error number is shown below. Not all error conditions have been thoroughly tested. The error messages are sent to SPRINT. In many cases PASCAL is able to generate correct code even though an error has occurred. However, correct code cannot be guaranteed unless the source program is error free.

```
 1 Expecting '.'
 2 Number out of range
 3 Identifier expected
 4 Expecting '='
 5 Field already defined
 6 Illegal subrange bounds
 7 Tag must be integer or enumeration
 8 Identifier already defined
 9 Expecting ')'
10 Expecting ':'
11 Procedure/function illegal
12 Identifier not defined
13 Subrange error
14 Expecting 'OF'
15 Expecting '.)'
16 More than 9 errors on a line
17 Variable not of record type
18 Type declaration error
19 Error in code generation
20 Expecting ',' or ')'
21 Division by zero
22 Only variables defined in this procedure may be initialized
23 Ignoring parameter list of FORWARD-declared proc/function
24 Procedure body must start with BEGIN
25 Statement expected
26 Unpacking illegal types
27 Variable not ARRAY type
29 Expecting '('
30 File type illegal
31 Range error
32 Incorrect data type
```

```
 33 Expression too complicated -- all registers full!
 34 Identifier not ARRAY type
 35 Expecting constant
 36 Incorrect index type
 37 Non-standard PASCAL feature used
 38 Variable in 'WITH' clause not of type record
 39 Record field undefined
 40 'ELSE' has no preceding 'IF', or extra ';' used
 42 Expecting factor
 43 Label not defined
 44 File error
 45 Error in expression
 47 Incorrect argument in standard procedure/function
 48 Label value illegal
 49 Closing string quote not found
 50 Illegal data types for previous operation
 51 Illegal data types for this operation
 52 Expecting ':='
 53 Illegal assignment
 54 End of statement expected
 55 Illegal use of symbol
 56 Expecting 'THEN'
 57 Variable required for VAR parameters
 58 Expecting ';'
 59 Expecting 'DO'
 60 Parameter error
 61 Expecting label
 62 Illegal set elements
 63 External procedures may not be forward declared
 64 Illegal function type
 65 Too many files
 66 Illegal arguments in 'NEW'
 67 Expecting 'UNTIL'
 68 Expecting 'END'
 69 Illegal control variable
 70 Expecting 'TO' or 'DOWNTO'
 72 Bad repetition constant for VALUE
 73 Too many array elements
 74 Too many labels
 75 Illegal option name
 76 Expecting ',' or '$'
 77 Label redefined
 78 Code area exhausted
 80 Expecting ','
 81 Too many procedures for 'load-and-go'
 82 Load-and-go code area exhausted
 83 Missing 'FORWARD' or 'EXTERNAL' procedures
 84 Object file not allowed at Student Terminal System
 85 Compiler error
 88 Expecting digit
 89 Undeclared type(s)
```

```
 90 Expecting type identifier
 91 '@' does not follow pointer or file variable
 92 Top-level procedure names are not unique in first 7
       characters
 93 Error in case label
 94 'FORTRAN' not allowed at student terminal system
 95 Illegal use of :W or :D
 96 Label did not appear in a LABEL declaration
 97 Input line too long
 98 Unexpected end of file encountered
 99 Unimplemented feature
```

Runtime warning and error messages are printed on SPRINT as soon as they occur. Generally recovery will be attempted NERR times, after which the run will be terminated. Each message is preceded by '****' or '$***'. If the latter form occurs, no recovery is possible and the run will be terminated immediately. These messages are indicated by '$' below. The texts of the run messages are relatively self-explanatory.

$ Keyword error in parameter list

An illegal option has been used in the PAR field.

$ Error in file assignments

An erroneous file assignment has been attempted in the PAR field.

$ Too many files

Currently, at most 16 file assignments are allowed in the PAR field.

$ PASCAL error return

An error has occurred in the PASCAL/UBC monitor during the exit sequence. Please show your program to Bary Pollack, Department of Computer Science.

$ Operation exception

An attempt to execute an unknown operation code has occurred.

$ Priviledged operation exception

An attempt to execute a priviledged (system) operation code has occurred.

$ Execute exception

An attempt to execute an illegal execute instruction has occurred.

* Protection exception

An attempt to branch to or change a memory location outside of your program area has occurred. This usually means that a pointer is NIL. If the X and K options are off, this also could mean a bad array reference or CASE index.

* Addressing exception

An attempt to access memory outside of your data area has occurred. Possibly an attempt to use NIL as a pointer. If the X and K options are off, it also could mean an out-of-bounds array access or CASE index.

$ Specification exception

Illegal use of a general or floating point register was attempted.

* Data exception

A decimal instruction had invalid data fields.

* Fixed overflow exception

An integer has been computed which does not fit into one full word. This condition is not normally checked. The result is truncated. Sign inversion may occur.

* Fixed division exception

An attempt to divide an integer by zero has occurred.

* Decimal overflow

A decimal number has been computed which does not fit into the specified field.

* Decimal division exception

An attempt to divide a decimal number by zero has occurred.

* Exponent overflow exception

During a floating point operation, an exponent has been developed which is greater than is allowed.

* Exponent underflow exception

  During a floating point operation, an exponent has been developed which is smaller than is allowed. This condition is not normally checked; a zero result is used.

* Significance exception

  Loss of significance has occurred during a floating pcint operation. This condition is not normally checked; a zero result is used.

* Floating division exception

  An attempt to divide a floating point number by zero has occurred.

$ Stack overflow

  PASCAL's execution stack is full. Rerun the program with EX= a larger number.

* File not assigned

  Reference has been made tc a PASCAL file which has no corresponding MTS file assignment.

* Unable to open file

  An illegal operation was attempted on a file before the file was opened.

$ Get on EOF=TRUE

  A GET or READ has occurred while the file was empty, or while no data remains in the file.

$ Input too long

  A source program input line is too long. Currently the limit for PASCAL source programs is 100 characters/line. At run time, a line lcnger than the declared line length will have been read. The line is truncated to the buffer size.

* Put on EOF=FALSE

  A PUT or WRITE has been attempted on a file while pointing somewhere other than the end of the file without a prior call to POSITION.

$ 'NEW' space overflow

The NEW stack is out of space. Rerun your program with NEW= a larger number.

* Reset file failure

An attempt to RESET a file has occurred and the file name in question is improper or has no associated MTS file name.

$ Local time limit exceeded

Your program has taken longer than was specified in the PAR field. Choose a longer time and rerun your program.

$ Local page limit exceeded

A Student Terminal System job has tried to print more execution output than is allowed. Use a more concise output format.

* Assignment value out of range

An attempt to assign a value to a variable when the value is outside the range declared for the variable.

* Index value out of range

An index to an array is outside the range permitted in the declaration for the array.

* Case value out of range

A CASE expression has resulted in a value for which there is no corresponding constant label.

* Rewrite file failure

An attempt to REWRITE a file has occurred and the file name in question is improper or has no associated MTS file name.

* Call on a formal procedure doesn't match actual parameters

The actual arguments to a function or procedure which has been passed as a parameter do not match the types declared for them in the formal argument declaration.

$ Compiler object file unavailable

The compiler is unable to write on the file specified for its object code (binary) output.

* Elementary function error

A (FORTRAN) elementary function has been invoked with an illegal argument. E.g., SQRT(-1.0).

10. References

[1] Jensen, K., and Wirth, N.
PASCAL User Manual and Report
Lecture Notes in Computer Science, No. 18.
Springer-Verlag, New York, 1974.

[2] Wirth, N.
Systematic Programming
Prentice-Hall, New York, 1973.

[3] Russell, D.L., and Sue, J.Y.
"Stanford PASCAL 360 Implementation Guide"
SLAC CGTM No. 89
Stanford University
Stanford, California, November, 1974.

[4] Computing Centre
"UBC BATCH"
University of British Columbia
Vancouver, British Cclumbia, August, 1975.

[5] Computing Centre
"UBC TERMINALS"
University of British Cclumbia
Vancouver, British Columbia, April, 1974.

[6] Computing Centre
"UBC LOADER"
University of British Cclumbia
Vancouver, British Columbia, October, 1976.

[7] Pollack, B.W.
"PASCAL/UBC Implementation Guide"
Technical Manual TM ??
Department of Computer Science
University of British Columbia
Vancouver, British Cclumbia, forthcoming.

INDEX