PASCAL/UBC User's Guide

by

Bary W. Pollack

and

Robert A. Fraley

Technical Manual 76-00

Sep 24, 1976

Department of Computer Science
University of British Columbia
Vancouver, B. C.

TABLE OF CONTENTS

PASCAL/UBC USER'S GUIDE

0. Introduction

PASCAL/UBC is a PASCAL [1,2] compiler for the IBM 360/370 computers 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 always will be found (in TN-chain ready form) in the file PASC:WRITEUP.

Current news regarding the state of the PASCAL system may be found in the file PASC:NEWS.

1. Running Under the PASCAL Submonitor

Most PASCAL users will wish to run under the submonitor. It provides extended error diagnostics and eliminates the necessity for separate compilation and execution steps. To run PASCAL under the submonitor, the user should issue the following MTS command:

```
$RUN PASC:GO  SCARDS=fdname  SPRINT=fdname  SPUNCH=fdname
              PAR=options  file-assignments  ;  comments
```

A semicolon (;) may (optionally) terminate the run command and anything to the right of the semicolon is ignored by the submonitor. If a semicolon is present, it must be preceded by at least one space.

PASCAL/UBC takes its input from SCARDS, sends its printed output to SPRINT, and sends its binary output (the translated program) to SPUNCH. If no SPUNCH file is given explicitly, it defaults to -P.OBJ.

Programs need not be translated nor run under the submonitor (see below); however, it usually is much easier to do so.

1.1 Options

The first field after the PAR= is for specification of the options: BATCH, DUMP, EX, GO, LI, NERRS, NEW, NOGO, NOPMD, PAGES, TIME, and TR. The options are separated by commas, and the option field is terminated by one or more blanks. No blanks may appear within an option.

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> with no spaces allowed on either side of the equal sign.

The BATCH option disables all interactive features. The snapshot/post mortem dump package is forced to act in batch mode.

The DUMP option requests a dump of the PSW, registers, and execution stack when a run time error occurs. It defaults to <absent> -- no dumping. This flag is primarily of use to systems programmers. A partial description of the format of the dump is given below in Section 9, Miscellaneous Implementation Notes.

EX=<value> specifies the maximum number of 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 20 pages.

GO specifies that the default object file (-P.OBJ) is to be loaded and executed immediately with no prior compilation step.

GO=<value> specifies the name of an object file with is to be loaded and executed immediately with no prior compilation step.

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

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 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.

NOGO specifies that the PASCAL translator is to be invoked, but that the object program is not to be executed.

NOPMD specifies that in the event of a run error, no post mortem dump is to be generated.

PAGES=<value> specifies the maximum number of pages of execution output allowed if running at the Student Terminal System. It has no effect otherwise.

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.

TR=<value> specifies which translator is to be used. If absent, the default translator is PASC:COMP (which always is the current stable translator). PASC:PASC may be selected instead. It is the current experimental translator (and it is not guaranteed always to produce the results expected).

1.2 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. Examples are:

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

PAR= PFILE=MTSFILE                                      (2)

PAR=DUMP,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 the program is to dump the execution stack if a run error occurs, allocate 15 NEW pages, associate P1FILE with the MTS file MTSFILE1, and associate P2FILE with *SINK*.

2. Compiling a PASCAL Program Directly

The PASCAL translator may be invoked directly without the submonitor by using the following command:

```
$RUN PASC:COMP PAR=options file-assignments ; comments
```

The formats of the options and file-assignments fields are as described above. The options valid during compilation are DUMP, EX, NEW, and TIME.

3. Running a Compiled Program Directly

PASCAL object modules may be executed directly without the submonitor by use of the following command:

```
$RUN object+PASC:MON PAR=options file-assigments ; comments
```

The formats of the options and file-assignments fields are as described above. The options valid during execution are BATCH, DUMP, EX, NERRS, NEW, NOPMD, PAGES, and TIME.

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

4. 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, no DUMP, EX=20, LI=PASC:STUD.LIB, NERRS=4, NEW=20, PAGES=4, and TIME=0.5S.

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.

5. Compiler Options

A comment whose first character is a dollar sign indicates a compiler option request. The options are separated by commas and consist of a letter followed by a plus or a minus sign indicating 'on' or 'off', respectively. A blank will terminate the option list, separating it from the rest of the comment.

Option	Default	Meaning
A	+ on	Perform subrange checking on Assignments.
B	- 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.
C	- off	Print object Code as each statement is processed.
D	- off	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.
E	- off	Forces a page Eject: the current line will begin a new page. This option automatically resets itself to off whenever used.
K	+ on	Forces an error if a CASE index is out of range.
L	+ on	List source program; lines containing syntax errors always are listed.
N	- 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.
P	- off	Print object code after each procedure or function is processed and all fix-ups have been made.

Q	- off	SeQuence number mode: if on, only columns 1-72 are read by the compiler; if off, columns 1-100 are read.
S	- off	If on, forces Standard PASCAL; if off, allows PASCAL/UBC extensions.
T	+ on	Forces Testing; the T option is equivalent to setting all of A, K, and X.
U	± 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.
X	+ on	Check indeX range in subscripts.
7	+ on	Compile code for IBM 370 if on; otherwise IBM 360.
_	+ on	Permits underbar (_) to be used as an alphabetic character.

The compiler implicitly uses the defaults

```
(*$A+,B-,C-,D-,E-,K+,L+,N-,P-,Q-,S-,T+,U±,X+,7+,_+*)
```

at the beginning a compilation.

6. Input/Output

This section describes the manner in which input and output work in PASCAL/UBC. Of particular importance is the existence of the special character, EOL, which designates end-of-line. EOL may be used in the same manner as any other character. It has the side-effect, on output, of terminating the output line.

Assume F is a PASCAL file (or is absent, in which case F defaults to INPUT or OUTPUT for READ and WRITE, respectively); CH is declared CHAR; and X, Y, Z are any collection of CHAR, INTEGER, or REAL on input, or CHAR, INTEGER, REAL, or ALFA on output; then,

```
READ(F,CH) is equivalent to the sequence
CH := F@; GET(F)

WRITE(F,CH) is equivalent to the sequence
F@ := CH; PUT(F)
```

```
READ(F,X,Y,Z, ...)  is equivalent to the sequence
READ(F,X); READ(F,Y); READ(F,Z); ...

WRITE(F,X,Y,Z, ...)  is equivalent to the sequence
WRITE(F,X); WRITE(F,Y); WRITE(F,Z); ...

READLN(F) is equivalent to the sequence
WHILE F@¬=EOL DO GET(F);  GET(F)

READLN(F,X,Y, ..., Z) is equivalent to the sequence
READ(F,X); READ(F,Y); ...  READ(F,Z); READLN(F)

WRITELN(F) is equivalent to WRITE(F,EOL)

WRITELN(F,X,Y, ..., Z) is equivalent to the sequence
WRITE(F,X); WRITE(F,Y); ...  WRITE(F,Z); WRITELN(F)
```

6.1 GET and READ

The following example will help to explain the precise manner in which READ operates. Remember that PASCAL has two standard functions, EOF and EOLN, which are TRUE just after they have read an EOF and EOL, respectively, and are FALSE otherwise. The input consists of the characters A, B, EOL, C, D, EOL, EOF :

	INPUT							
AFTER READ(F,CH)	*	A	B	EOL	C	D	EOL	EOF
EOF(F)	F	F	F	F	F	F	F	T
EOLN(F)	T	F	F	T	F	F	T	UND
F@	EOL	A	B	EOL	C	D	EOL	UND
CH, after READ(F,CH)	UND	SP	A	B	SP	C	D	SP

* - this column is prior to the first GET, READ, or RESET
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. Note also that commas and multiple blanks disappear when READing integers or reals. The built-in numeric read routines must read one character beyond the end of the number in order to tell when the number ends. A number must be followed by a blank, a comma, or the end of the line. If a comma follows a number, F@ is the character beyond the comma; otherwise F@ is the character after the number.

6.2 PUT and WRITE

If a file has not been used at all, PUT(F) is valid even though EOF(F) is FALSE. Once the file is in use, PUT(F) is valid only if EOF(F) is TRUE.

The arguments of WRITE may appear with or without a field-width specification. If no field-width is given explicitly, the following values are used:

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

An explicit field-width specification consists of a constant, integer, or integer valued expression preceded by a colon (:W or :W:D). Assume that N is a datum or expression (which may be CHAR, ALFA, INTEGER, REAL, BOOLEAN, or a suitable subrange of the aforementioned), then a WRITE argument may appear in one of two forms:

```
N:W     or
N:W:D
```

where W is the total number of spaces to be used in printing, and D is the number of digits to the right of the decimal point.

The following rules govern the operation of WRITE:

The :W form is valid regardless of the type of N.

Unless N is of type REAL, the :W:D form is invalid.

If N is a number and W is negative, the number will be printed in hexadecimal in -W columns.

If N is of type CHAR, it will be printed left justified, with W-1 trailing spaces. If W is zero a field width of zero will be used, and no character will be printed.

If N is of type ARRAY OF CHAR it will be printed left justified with trailing blanks. If W is too small, truncation occurs on the right.

If N is of type INTEGER it will be printed right justified. If W is too small, the value will be printed in as many columns as necessary to represent it accurately.

If N is of type BOOLEAN it will print as TRUE or FALSE, right justified in a field of W columns. Truncation occurs on the right if W is too small.

If N is of type REAL it will print right justified in W columns. REAL numbers always are preceded by at least one blank.

In the following paragraphs assume that there is a :W specification. If no :W is present, the following paragraphs apply assuming the default value.

Real numbers always have at least one digit on each side of the decimal point. Fractions will force a minimum field size of 4 (5 if negative), while scientific notation forces a minimum field size of 8 (9). If D=0 the minimum size is 2 and no decimal point is printed.

If there is no :D specification, an appropriate value is selected based on W and the number size. E format is used if more significance can be printed that way.

The :D specification determines the precise format of the number. If D is zero, N will be rounded to the nearest integer. If D is positive, N will be rounded so that D digits are correctly printed to the right of the decimal point. D digits always will be printed after the decimal point when D is positive, even if the value then occupies more than W columns. If D is negative, scientific (exponential) notation will be used to print the value in W columns.

6.3 RESET and REWRITE

In PASCAL/UBC both RESET and REWRITE accept an optional second argument -- the name of an MTS file. This name must be a character string (ARRAY(..) OF CHAR). The file name must be left justified within the string, and it must have at least one trailing blank.

If F is a FILE and G is a filename, RESET(F,G) has the following effects:

1. Flushes out any output remaining to be sent to file F,
2. Rewinds F,
3. Associates the MTS file G with the PASCAL name F,
4. Opens F (G) for reading.

If F and G are as above, REWRITE(F,G) has the following effects:

1. Flushes out any output remaining to be sent to file F,
2. Rewinds F,
3. Associates the MTS file G with the PASCAL name F,
4. Opens F (G) for writing.

Note that the MTS files whose names are in G may be computed dynamically. And in particular, note that these names do not need to appear in the file assignments field of the $RUN command's PAR field.

6.4 Standard Files

PASCAL/UBC provides several standard files and file associations. These may be overridden in the file-assignments field of the $RUN command if desired.

PASCAL Name	MTS Name	Declaration
INPUT	SCARDS	No further declaration required
OUTPUT	SPRINT	No further declaration required
GUSER	GUSER	VAR GUSER : TEXT;
SERCOM	SERCOM	VAR SERCOM : TEXT;

If declarations are provided for INPUT or OUTPUT they will completely override the standard assignments. GUSER and SERCOM may be declared TEXT or FILE OF X, where X is any type.

6.5 Carriage Control

PASCAL does not automatically insert carriage control information into column one of output lines -- this is the responsibility of the programmer. MTS demands that all files sent directly to printing devices must have carriage control characters inserted before the first data character as per UBC BATCH [4]. Thus a typical program fragment doing output to a printing device might look like:

```
CONST
   O = '+';      (* OVERPRINT THE LAST LINE *)
   S = ' ';      (* BLANK FOR SINGLE SPACE  *)
   D = '0';      (* ZERO  FOR DOUBLE SPACE  *)
   T = '-';      (* MINUS FOR TRIPLE SPACE  *)
   P = '1';      (* ONE   FOR NEW PAGE      *)

     ....................................

WRITELN (P,'START ON A NEW PAGE');
WRITELN (S,'SINGLE SPACE THIS LINE');
WRITELN (D,'DOUBLE SPACE THIS ONE');
WRITELN (T,'TRIPLE SPACE THIS LINE');
WRITELN (O,'                          OVERPRINT THE LAST LINE');
WRITELN (S,'DEMONSTRATION',EOL,D,'OF',EOL,S,'EOL');
```

which would produce the following output:

```
START ON A NEW PAGE
SINGLE SPACE THIS LINE

DOUBLE SPACE THIS ONE


TRIPLE SPACE THIS LINE OVERPRINT THE LAST LINE
DEMONSTRATION

OF
EOL
```

The effect of the PAGE(F) standard function is:

```
F@ := EOL;  PUT(F);
F@ := '1';  PUT(F);
F@ := EOL;  PUT(F);
```

Note PAGE does not allow one to print on the first line of the new page. To achieve this effect, you should use:

```
WRITELN('1MESSAGE');
```

6.6 Using PASCAL Interactively

The standard PASCAL READ and WRITE procedures described 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 preferable:

```
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 converstions must now be done manually by the programmer.

7. PASCAL Libraries

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

7.1 The Standard PASCAL Library: PASC:LIB

The PASCAL library includes system routines for performing input/output plus various other procedures and functions. The source programs for PASC:LIB reside in file PASC:LIB.S1 and this file should be examined to determine the library's precise contents. Currently it includes two random number generators: RAND, RANDU; an exponential function EXP; integer valued MAX and MIN functions; 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). Of course the user always has access to the entire contents of *LIBRARY and any additional libraries he wishes to specify. You may include FORWARD declarations for all the standard PASCAL library routines by saying

```
$CONTINUE WITH PASC:LIB.S1(100,199) RETURN .
```

7.2 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. Any collection of PASCAL object modules may serve as a library regardless of whether it has been created with *SGEN (although one normally will want to use *SGEN in order to speed loading and provide for selective module inclusion). *SGEN documentation may be found in UBC LOADER [6].

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

```
$RUN PASC:GO SCARDS=MYSOURCE SPUNCH=MYLIBRARY PAR=NOGO
```

where MYLIBRARY is the name of the library file. A null main program (specified with a single period rather than BEGIN END.) normally should be used.

One then might use the newly created library as follows:

```
$RUN PASC:GO SCARDS=MYPROGRAM PAR=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 PASCAL warning is generated if this restriction is not heeded.

8. 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 required 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.

8.1 Differences and Extensions from the User Manual

Section 1 - Notation and Vocabulary

E Names may be composed of any number of upper case alphabetic characters, the digits 0 - 9, and underbar (if the _ option is on). 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.

D Braces do not exist in the EBCDIC character set. Comments may be delimited by pairs of quotes (" <comment> "), or by the the biliterals (* <comment> *) .

R The biliteral ¬= and the word VALUE are special symbols.

D Square brackets to define array subscript expressions are 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 @ .

D Brackets to define powersets are replaced by the notation SET(...) . Alternatively, they 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 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. 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. 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

No changes.

Section 5 - Scalar and Subrange Types

No changes.

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

R 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 of characters. No actual (un)packing occurs as data is implicitly "packed" on byte addressable machines such as the IBM 360/370.

R The standard procedures PACK and UNPACK only transfer data between objects of type ARRAY OF CHAR.

Section 7 - Record Types

No changes.

Section 8 - The Set Types

R Restrictions on sets are: the base type can have a maximum of 32 values. Subranges of integers must be between 0 and 31. CHAR is not an allowed 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.

Section 10 - Pointer Types

R The standard function DISPOSE 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 as pointers may no longer be valid following a RELEASE.

Section 11 - Procedures and Functions

R 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.

Section 12 - Input and Output

E PASCAL/UBC has a special character, EOL, designating the end of a line. If F is a TEXT file, then F@ = EOL initially. F@ will be the first character of the file before performing GET(F) or RESET(F). 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. To avoid infinite loops, use READLN(F,S) instead of READ(F,S). READLN will skip 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 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 F, F is closed, F is associated with the file G, and F (G) is opened for reading or writing, respectively.

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 and FORTRAN declarations. EXTERN does not exist in PASCAL/UBC; however, its effect may be achieved by using a FORWARD declaration.

R No program headings are allowed.

D The standard identifier MAXINT is defined as

```
CONST MAXINT = 2147483647; (* = 2**32-1 *)
```

R The statement IF ABS(I) > MAXINT THEN WRITE(' TOO BIG') will not execute correctly because no integer may be larger than MAXINT. The statement IF I < -MAXINT ... will execute correctly, as 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. Provided is a mantissa of 54 bits, corresponding to approximately 16 decimal digits. The maximum absolute value is .... BARY ...., and the minimum absolute value is .... BARY .....

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

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

R The maximum cardinality of the base type of a set is 32.

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 or PAR=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 DISPOSE is the only "standard function" not supported by PASCAL/UBC, and since now has been removed from the list of "standard functions", it will not be 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 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 -32767..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.

8.2 Differences and Extensions from the Revised 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.

R In PASCAL/UBC <letter> does not 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.
Comments may be delimited by quotes (") or (* and *).

R Left and right braces may not be used as comment delimiters.

Section 4 - Identifiers, Numbers, and Strings

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

D INTEGERs are restricted to the subrange $-2^{31}..(2^{31}-1)$. REALs are restricted to $\pm 10^{+75}..\pm 10^{-75}$ 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

No changes.

Section 6 - Data Type Definitions

R The word PACKED is ignored.

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

R SETs are restricted to a subrange 0..31, and to scalar types having at most 32 elements.

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

R 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 are the standard types ALFA and TEXT:

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

Section 7 - Declarations and Denotations of Variables

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

Section 8 - Expressions

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

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

Section 9 - Statements

R A label may have at most four digits.

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.

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 may be done. However, no warning message is given, and the results are unpredictable.

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 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.

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.) 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

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.

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 The RESET and REWRITE procedures optionally may take a second parameter. This is a string expression which is the

external file name for the file. The external name must end with a blank. The names 'INPUT ', 'OUTPUT ', 'GUSER ', and 'SERCOM ' refer to MTS logical units instead of external files.

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 editions of the Manual and/or Report is not implemented.

E Neither argument of PACK or UNPACK needs the word PACKED in its declaration.

R PACK and UNPACK only work on ARRAYs OF CHAR.

E PUT(F) is valid when EOF(F) is false if there has been no prior use of F.

E Additional functions:

INSERT(I,J,K:INTEGER):INTEGER
Result is I*2**J+K. J must be non-negative.

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.

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

E Additional procedures:

HALT
Stops execution and returns to the system.

Section 12 - Input and Output

R READ and WRITE may be used only on 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 <statment part> from the program <block>. The period is still required.

E A <value part> may follow the <variable declaration part> to provide initial values. 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>
```

9. 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.

9.1 DUMP Format

The dump routine is invoked after an interrupt (real or simulated) has occurred. Output is produced by calling the system SDUMP routine giving it the address of a 72-byte region containing the PSW and general registers 0-15 in that order. Following this is the dump of the execution stack associated with each active procedure.

DUMP output is normally of use only to systems programmers maintaining the PASCAL system.

9.2 Communication with FORTRAN

The following notes are 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	4
REAL	8
'character string'	length of string

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

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.

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.

9.3 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 fields are half-, full-, and double-word aligned, respectively. The parameters appear in the order in which they are declared.

3. Use registers 13, 14, and 15 as usual.

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.

10. Snapshot and Post Mortem Dump Packages

This section contains a preliminary description of the snapshot/post mortem dump package. The package is currently under development, and it is subject to change with little notice. None of the snapshot 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 invoked.

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.

The following commands are understood by the interactive snapshot package:

To be continued ...

11. 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 ')'
22 Value clause at wrong level
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
 52 Expecting ':='
 53 Illegal assignment
 54 End of statement expected
 55 Illegal use of symbol
 56 Expecting 'THEN'
 58 Expecting ';'
 59 Expecting 'DO'
 60 Parameter error
 61 Expecting label
 62 Illegal set elements
 63 Not a constant
 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'
 71 Error in CASE statement
 74 Too many labels
 76 Procedures too deeply nested
 77 Label redefined
 78 Code area exhausted
 80 Expecting ','
 82 WITH/procedure nest too deep
 85 Compiler error
 88 Expecting digit
 89 Undeclared type(s)
 90 Expecting type identifier
 91 '@' does not follow pointer or file variable
 92 Too many forward defined types
 93 Error in case label
 95 Illegal use of :W or :D
 96 Label did not appear in a LABEL declaration
 98 Unexpected end of file encountered
 99 Unimplemented feature
```

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

```
$ Keyword error in parameter list
$ Error in file assignments
$ Too many files
$ Operation exception
$ Priviledged operation exception
$ Execute exception
  Protection exception
  Addressing exception
$ Specification exception
  Data exception
  Fixed overflow exception
  Fixed division exception
  Decimal overflow
  Decimal division exception
  Exponent overflow exception
  Exponent underflow exception
  Significance exception
  Floating division exception
$ Stack overflow
  File not assigned
  File not opened
  Get on EOF=TRUE
  Input too long
  Put on EOF=FALSE
$ 'NEW' space overflow
  Reset file failure
$ Local time limit exceeded
$ Local page limit exceeded
  Assignment value out of range
  Index value out of range
  Case value out of range
  Rewrite file failure
  Parameters to a procedure do not match actual parameters
```

12. 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 Columbia, August, 1975.

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

[6] Computing Centre
"UBC LOADER"
University of British Columbia
Vancouver, British Columbia, June, 1974.

[7] Pollack, B.W., and Fraley, R.A.
"PASCAL/UBC Implementation Guide"
Technical Manual TM 76-??
Department of Computer Science
University of British Columbia
Vancouver, British Columbia, forthcoming, 1976.

INDEX