AN INTRODUCTION TO PASCAL2

by

Wilhelm Burger

TR-22A September 1975
Revised March 1976

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

Introduction

A new compiler for PASCAL called PASCAL2 is available. This compiler generates relocatable code. It is possible to call subroutines written in FORTRAN or COMPASS. Also subroutines written in PASCAL can be compiled separately and made available to a PASCAL program. The following sections describe new features of the language, and some differences between the old and new PASCAL are pointed out. Further a description on how to use the new system is given. The new definition of PASCAL and a thorough description of the compiler may be found in: PASCAL, User Manual and Report, by Kathleen Jensen and Niklaus Wirth, Lecture Notes in Computer Science, Number 18, Springer Verlag, 1974 (available in the CO-OP).

Program Heading

The name of the program and external files are defined in the program heading.

Example: PROGRAM TEST(INPUT,OUTPUT,TAPE1);

The file names may be considered as formal parameters for which actual parameters are provided on the control card (see page 13). The program heading must always contain the file OUTPUT. The names INPUT and OUTPUT may be extended with + in the program heading. This predeclares these files as SEGMENTED FILE OF CHAR.

Example: PROGRAM TEST(INPUT+,OUTPUT+);

File Declaration

Files are declared as in the old version. Files which appear in the program heading are referred to as external files. The other files are local files. The system generates a unique name for local files so that a name conflict with files attached to a job, or with files declared in different declaration parts is avoided.

All external files must be declared in the declaration part of the main program except the standard files INPUT and OUTPUT. They are already predeclared to be of type TEXT (FILE OF CHAR).

PASCAL2 introduces the segmented file. Segmented files provide a new level of file structuring. File segments correspond to UT-2D logical records. The declaration for a segmented file is:

Example: TAPE1: SEGMENTED FILE OF INTEGER;

Special operations are provided for these files (see page 4).

Arrays and Records

Arrays may be declared packed in order to save storage.

```
Example:      ERRBITS: PACKED ARRAY[1..20] OF BOOLEAN;
```

The predefined type ALFA is equivalent to

```
ALFA = PACKED ARRAY[1..10] OF CHAR;
```

Constants of type ALFA are strings with exactly 10 characters. Other strings may be considered constants of packed arrays of the appropriate size. Comparison between strings of the same length is possible. The result is determined by the ordering of the character set.

The standard procedures PACK and UNPACK can be applied to arrays of any type.

Variants of records may be declared without having to specify a tagfield part (so-called non-discriminating variants).

```
Example:      X: RECORD
                  CASE COLOR OF
                    RED: (A: INTEGER);
                    GREEN: (B: PACKED ARRAY[1..10] OF 0..77B);
                    BLUE: (C: REAL);
                 END;
```

Sets

Sets are restricted to values in the range of 0 to 58. The syntax for set operations has changed:

```
X + Y          union
X * Y          intersection
X - Y          set difference
```

Constants of type set can be defined by using a subrange notation:

```
Example:      X := [1,2..5,7,9..12];
```

This is equivalent to X:=[1,2,3,4,5,7,9,10,11,12];

Pointer Declaration

The type CLASS does not exist in PASCAL2. Only pointer types may be declared.

```
Example:        TYPE  LINK = ^LIST;
                      LIST = RECORD
                                VAL: INTEGER; NEXT: LINK
                             END;
```

Space for the objects pointed to is provided on the heap with the standard procedure NEW. NEW is used in the same way as in the old PASCAL. No memory manager is provided in PASCAL2 to collect unused space of the heap. However, if space is allocated on the heap in a stack-like manner, then it is possible to reuse the space by resetting the top of the stack to a known location with the procedure RELEASE.

Procedures and Functions

Functions without parameters are permitted in PASCAL2.

```
Example:        FUNCTION NEXTTIME: REAL;
                BEGIN ...; NEXTTIME:=X END;
```

Procedures and functions which are used as actual parameters for formal parameters of kind procedure or function may only have parameters which are called by value.

Label Declaration

All labels must be declared in PASCAL2. The scope of labels is similar to the scope of names. The Goto statement has only the following form: GOTO <number>.

File Operations

A file is in write condition if the function EOF applied to the file returns true. A file is in read condition if the function EOF returns false.

A file is brought into the read condition by the procedures RESET and GETSEG. RESET rewinds the file and loads the file buffer, thus making the first component of the file available. If the file is empty then the end-of-file condition is set, and the procedure EOF will return true if applied to the file. If the file is a segmented file then the procedure GETSEG is used to position the file, and to load the file buffer. GETSEG has two arguments: the file name and the desired position relative to the current position of the file.

```
Example:        GETSEG(TAPE,3);
                GETSEG(TAPE,-2);
```

GETSEG(TAPE) initiates the reading of the next segment. It is equivalent to GETSEG(TAPE,1). If an empty record is read then the end-of-segment condition is set, and the function EOS will return true if applied to the file.

The procedure GET may not be called when either EOS or EOF returns true. The following is an example of how to read a record:

```
Example:        WHILE NOT EOS(TAPE) DO
                BEGIN READ(TAPE,CH); ... END;
```

The procedures RESET and GETSEG do not fill the file buffer if the file is assigned to an interactive terminal (see page 7).

A file is brought into the write condition by the procedure REWRITE. REWRITE rewinds the file and sets the end-of-file condition. REWRITE may also be applied to segmented files, thereby positioning the file to the record where writing begins. (Note that this is not a random rewrite in place). REWRITE then has two arguments: file name and relative position. Note that REWRITE(TAPE) is not equivalent to REWRITE(TAPE,1).

An end-of-record may be written on a segmented file with the procedure PUTSEG.

```
Example:        PUTSEG(TAPE1);
```

Files made available to the program are normally in write condition. The standard file INPUT is opened with an implicit call to RESET. If the actual file name is INPUT or OUTPUT then the procedures RESET and REWRITE have no effect.

Input/Output Routines

The EOL character has been eliminated in PASCAL2. However, the end of line is still indicated by an end-of-line character, which when read will set the end-of-line condition and return a blank as its value. Special routines are provided to deal with the end-of-line condition. A variable EOL of type CHAR can be initialized by EOL:=CHR(0) for purposes in which the EOL character was used previously other than terminating a line.

The first parameter of the input/output routines may be a file. Then the operation is performed on the file thus specified.

EOLN

This function returns the value true if the file pointer is positioned to the end-of-line character. EOLN is equivalent to EOLN(INPUT).

READ

Example: READ(K,L,M);

If the first parameter is not a file then the file INPUT is assumed. READ is defined by the sequence

```
CH:=INPUT^; GET(INPUT); .... GET(INPUT);
```

The consequences of this definition are: The file buffer must be loaded before READ is called (see page 7). The end-of-line condition becomes true when the last character of the line is read. (In order to get to the first character of the next line the end-of-line character may be read, or the procedure READLN may be called).

If READ is applied to text files then the types of the parameters are the same as in the old PASCAL. However, READ can be applied to files of other types. READ(TAPE,KK) is then equivalent to KK:=TAPE^; GET(TAPE);

READLN

READLN is only applicable to text files. The parameter list is the same as for READ. The file pointer, however, is automatically positioned to the first character of the next line after the last parameter is read. READLN may also be called without parameters, thus skipping the remainder of the line and positioning the file pointer to the first character of the next line. READLN is equivalent to READLN(INPUT).

WRITE

Example: WRITE(TAPE,K:5,"ABC",X:7:3);

Formats are the same as for the old PASCAL. (Strings with less than 10 characters are not treated as type ALFA but written with the appropriate number of characters enclosed in the quote symbols).

WRITE can also be applied to files of other types. WRITE(TAPE,LL) is then equivalent to TAPE^:=LL; PUT(TAPE);

WRITELN

WRITELN is only applicable to text files. The parameter list of WRITELN is the same as for WRITE. After the last parameter is written then the line is finished by producing an end-of-line character. WRITELN may also be called without parameters. The call

WRITELN is equivalent to WRITELN(OUTPUT).

Interactive Files

Files which are assigned to an interactive terminal require some special handling. The function ISTTY returns true if the file is used interactively. The procedure RESET does not load the file buffer for interactive files but only sets the read condition. Input is requested at the appropriate time by

```
Example:      IF ISTTY(INPUT) THEN GET(INPUT);
```

The call ISTTY is equivalent to ISTTY(INPUT).

Whenever appropriate, output may be sent to a terminal with the procedure SENDTTY. This procedure has no effect if the file is not interactive. SENDTTY is applicable to regular as well as segmented files. The call SENDTTY is equivalent to SENDTTY(OUTPUT).

For truly interactive use it is often desirable for the user to respond on the line just displayed. This effect can be achieved if the line written to the terminal consists of a multiple of 10 characters, and the procedure WRITE is used.

The procedure GETSEG has no effect when the file is interactive, however, it expects input from the terminal.

It is also possible to send or receive data in binary mode (e.g. upper and lower case). An external procedure must be called which sets the appropriate bit in the extended FET.

External Procedures and Functions

The procedure or function heading must be fully declared for parameter passing purposes. The procedure body consists only of the word EXTERN or FORTRAN to indicate that the routine is provided from somewhere else. When the procedure or function is called then either the FORTRAN calling sequence, or the PASCAL calling sequence is generated.

```
Example:      PROCEDURE UNFLAG; EXTERN;
              PROCEDURE INVERT(VAR A,B: ARRAYTYPE; N: INTEGER);
                   FORTRAN;
```

A FORTRAN procedure may be called with fewer parameters than specified but never with more. Note that PASCAL makes a distinction if a parameter is called by reference or by value. In the first case the address of the parameter is passed to the FORTRAN routine. In the second case the value of the parameter is copied and the address to the copy passed to the FORTRAN routine.

It is possible to use formatted FORTRAN I/O together with PASCAL I/O if the interface conditions are observed. In order to switch from PASCAL input to formatted FORTRAN input the file pointer must be positioned to the first character of the line to be read in FORTRAN. This is achieved by a call to READLN. In order to switch back to PASCAL input the interface routine GETFN must be called with the appropriate file as parameter. This routine reloads the character buffer used by PASCAL and positions the file pointer properly. Note that formatted FORTRAN input ignores end-of-record marks unless the file name is INPUT. In the following two examples the procedure FORTRANINPUT and FORTRANOUTPUT are assumed to be FORTRAN subroutines which perform formatted I/O.

```
Example:    READLN;
            FORTRANINPUT(X);
            GETFN(INPUT);
```

In order to switch from PASCAL output to FORTRAN output it is necessary that the character buffer used by PASCAL be emptied. This is achieved by a call to the procedure WRITELN.

```
Example:    WRITELN;
            FORTRANOUTPUT(X);
```

The following correspondence of parameter types in FORTRAN and PASCAL must be observed:

```
FORTRAN              PASCAL

INTEGER              INTEGER
REAL                 REAL
COMPLEX,DOUBLE       RECORD
                        R1: REAL; R2: REAL;
                     END;
LOGICAL              INTEGER
ARRAY                ARRAY
```

FORTRAN functions may not return values of type COMPLEX or DOUBLE. As PASCAL does not handle minus zero, a zero must be added to those parameters which might return a minus zero immediately after returning from the FORTRAN routine. Multi-dimensional arrays are stored differently in FORTRAN. The arrays must be either transposed before (and eventually after) calling the FORTRAN routine, or the algorithm must take the different storage method into consideration. A LOGICAL value represents false if it is positive, and true if it is negative. External routines or FORTRAN routines may use blank common only if the size of blank common is known and it can be guaranteed that no reference outside the given size is made.

Some system routines expect different types of parameters on the same parameter position. The use of a non-discriminating variant is suggested to overcome this problem. The following example is tailored for the system routine SYMBOL of the PLOT package.

Example:

```
VAR AUX: RECORD
          CASE INTEGER OF
         1: (A: CHAR);
         2: (B: INTEGER);
         3: (C: ALFA);
         END;
   STRNG: ARRAY[1..10] OF ALFA;
   XPAGE,YPAGE,HEIGHT,ANGLE: REAL;

PROCEDURE SYMBOL(XPAGE,YPAGE,HEIGHT:REAL;
                 VAR IBCD:ALFA; ANGLE: REAL;
                 NCHAR: INTEGER); FORTRAN;
BEGIN
   (* DRAW ONE CHARACTER *)
   AUX.A:="X";
   SYMBOL(XPAGE,YPAGE,HEIGHT,AUX.C,ANGLE,0);
   (* DRAW A STRING *)
   SYMBOL(XPAGE,YPAGE,HEIGHT,STRNG[1],ANGLE,15);
   (* DRAW A SPECIAL SYMBOL *)
   AUX.B:=13;
   SYMBOL(XPAGE,YPAGE,HEIGHT,AUX.C,ANGLE,-2);
END.
```

The current version of PASCAL2 does not have a value part. However, data statements can be used in a FORTRAN subroutine to initialize a storage area. This area then is copied to the appropriate place in the PASCAL program. (Notice that this PASCAL2 implementation allocates space in the reverse order for variables which are declared together for a type. Thus in the following example, array C is allocated before array B).

Example:

```
PROGRAM TEST(OUTPUT);
TYPE LIST=ARRAY[1..10] OF INTEGER;
VAR A: LIST;
    B,C: LIST;
PROCEDURE INITIALIZE(VAR X:LIST); FORTRAN;
BEGIN
  INITIALIZE(A);
  (*THIS CALL INITIALIZES THE ARRAYS A, B, AND C*)
END.
```

```
      SUBROUTINE INITIAL(X)
      INTEGER X(1)
      COMMON/VALUES/A(10),C(10),B(10)
      INTEGER A,B,C
      DATA A/3*5,1,2,5*7/
      DATA B/10*4/
      DATA C/-12,-15,3*0,5*1/
      DO 10 I=1,30
   10 X(I)=A(I)
      RETURN
      END
```

Additional Standard Procedures and Functions

ROUND(X)	The real argument X is rounded to the next integer.
EXPO(X)	yields the integer value equal to the exponent of the floating point representation of the real value X.
UNDEFINED(X)	returns true if the real value X is out of range or indefinite, otherwise false.
CARD(X)	returns the number of elements contained in the set X.
CLOCK	returns the TM time expressed in milliseconds.
DATE(X)	Assign to the variable X of type ALFA the current date.
TIME(X)	Assign to the variable X of type ALFA the current time.
PAGE(X)	Start a new page on file X. X must be a text file.
LINELIMIT(X,Y)	X must be a text file. Y gives the limit of lines to be written to file X.
MESSAGE(X)	X is a string which is written to the dayfile.
HALT	Abort the program and produce a post-mortem dump.

Summary of Standard Identifiers

Constants:	TRUE FALSE
Types:	INTEGER BOOLEAN REAL CHAR ALFA TEXT
Variables:	INPUT OUTPUT
Functions:	EOF EOS EOLN ODD PRED SUCC ROUND TRUNC EXPO ABS SQR ORD CHR UNDEFINED CARD CLOCK ISTTY SIN COS EXP SQRT LN ARCTAN RANF
Procedures:	GET PUT RESET REWRITE GETSEG PUTSEG READ READLN WRITE WRITELN PAGE LINELIMIT MESSAGE PACK UNPACK TIME DATE HALT NEW DISPOSE RELEASE GETFN SENDTTY

Compiler Options

Compiler options can be made available through the control card or by a special type of comment.

Example: (*$T+,X4,P+ *)

An option is turned on if it is followed by +, and turned off if it is followed by -. Some options may be followed by a digit.

The following options are available using the special form of a comment:

T — Run-time test.
The tests include array bound checks, division by zero, check of assignment, and range of case selector. Also the transformation of integers to reals is supervised.

P — Generate the necessary information during compilation to allow a post-mortem dump.

E — The compiler generates entry point names equal to the procedure or function identifier shortened to the first seven characters. (This is automatically done for procedures or functions declared FORTRAN or EXTERN).

L — Listing control.

B — The digit following B is used to compute the buffersize of files. The buffersize is a multiple of 200B words.

X — The digit following X may be between 0 and 6. The digit gives the number of X-registers to be used starting with X0 for parameter passing. The rest of the parameters is transmitted in the locations from B6+3 onward.

Q — Generate code for measuring heap and stack size when a routine is entered or an element is allocated on the heap.

W — The digit sequence following W must be octal. The number defines the workspace for the available program in 1000B words. Only the first definition of W holds. The W parameter on the control card overrides the W compiler option.

If no options are specified on the control card then the compiler options are set equivalent to

(*$T-,P-,E-,L+,B4,X4,Q+,W2*)

Equivalent Symbols in PASCAL2

<=	@
>=	?
<>	'
NOT	\
AND	&
OR	!
(*	#
*)	%

Control Cards

Compilation of a Program

PASCAL2,<input>,<output>,<lgo>.<parameters>

<input> input file, by default INPUT. The file is rewound unless it is INPUT.

<output> output file, by default OUTPUT. This file contains the program listing and error messages.

<lgo> file which contains the relocatable code. By default it is the file LGO. This file is not rewound by the compiler.

<parameters> This part is optional. when present it must immediately follow the period. It has the form

<options>,<compilersize>,<workspace>,<textsize>

<options>,<compilersize>,<workspace>, and <textsize> may appear in any order, or any or all may be missing.

<options> OPT=<letter sequence>
The following options are accepted by the compiler:

F generate the FTN calling sequence instead of the RUN calling sequence.
N no listing
U input is 80 columns instead of 72 columns.
Q do not generate code for measuring heap and stack size.

The letters E P T are also accepted and described in the section on compiler options.

<compilersize> C=n
n must be an octal number which gives the workspace for the compiler in 1000B words. Default work space is 7K.

<workspace> W=n
n must be an octal number giving the work space for the translated program in 1000B words. The default work space for the translated program is 2K.

<textsize> B=n
n must be octal and in the range 1 to 10B. This value sets the buffersize for files of type TEXT. Default value is 4.

Example: PASCAL2,TEST.OPT=N,C=10,W=4.

In this example it is assumed that the program is on file TEST. No listing of the program is made. Any output, e.g. error messages, is written to file OUTPUT. The translated program is on file LGO. 10K of work space are available for the compiler. The work space for the translated program is 4K.

Execution of a Program

```
LOAD,<lgo>,PASCLIB.
EXECUTE,,<actual files>.<parameters>
```

<lgo> File which contains the relocatable code. By default it is the file LGO.

<actual files> The actual files are mapped to the formal files on a one-to-one basis. If a file name is omitted then the corresponding name of the formal parameter is used.

<parameters> This part is optional. When present it must immediately follow the period. It has the form

```
<options>,<workspace>
```

<options> and <workspace> may appear in any order, any or all may be missing.

<options> OPT=<letter sequence>
The options can be used by the program as described on page 17.

<workspace> W=n
n must be an octal number giving the work space for the program in 1000B. This value overrides the work space value given at compilation. The work space used is recorded in the dayfile if the program was translated with the option to measure stack and heap. This allows choosing an optimal value for the W parameter (and thus for the fieldlength used by the program).

Example:
```
LOAD,LGO,PASCLIB.
EXECUTE,,X1,X2.w=5.
```

In this example it is assumed that the translated program is on LGO. The files X1 and X2 are the actual files. The work space for the execution is 5K.

Generation of an Absolute Program

```
LINK,F=<lgo>,P=PASCLIB,B=<program>.
```

This absolute program can now be executed by

```
<program>,<actual files>.<parameters>
```

Creation of a Library

A PASCAL program with a dummy main program part is used to create a file of relocatable PASCAL routines. The program must be translated with the E option to create entry names, and the last record (the main program) must be discarded. If a routine uses a global variable then the main program declaration part must be the same as the one in which the routine is declared external. Other relocatable routines which are generated by FORTRAN or COMPASS may be added to this file.

Routines written in COMPASS may either observe the FORTRAN calling sequence or the PASCAL calling sequence. In the PASCAL calling sequence the parameters are passed beginning with X0 and/or B6+3. The return address is in X7. The COMPASS routine may not alter the registers B2, B4, B5, or B6, and the register B1 must have the value 1 upon exit. A simple jump to the address found in X7 terminates the routine.

The file of relocatable routines is turned into a library by

```
LIBGEN,F=<rel file>,P=<library file>.
```

The library can be augmented with other relocatable routines by

```
LIBEDIT,P=<library file>,B=<more>,UL=<library file>,F.
```

Additional features of LIBEDIT are described in TPB-153, January 1975, Computation Center.

How to Use Libraries

Assume the libraries LIB, LIB1 and LIB2 are needed for the execution of the relocatable program on file LGO. The program is then loaded by

```
LOAD,LGO,LIB,LIB1,LIB2,PASCLIB.
```

An absolute version of the program can be created by

```
LINK,F=LGO,P=LIB,P1=LIB1,P2=LIB2,P3=PASCLIB,B=<program>
```

Utility Programs

Programs are available to produce a cross-reference listing of a PASCAL program, to format a PASCAL program, and to print the generated code in symbolic form. (The control cards following may also have a <workspace> parameter).

Cross-reference listing

```
EXECPF,6482,XREF,<input>,<output>.<options>
```

<input> program file, default INPUT.

<output> output file, default OUTPUT.

<options> OPT=<letter sequence>

- N print only the cross-reference table.
- U process 80 columns instead of 72 columns.

Cross-reference listing with formatting of program text

```
EXECPF,6482,XXREF,<input>,<output>,<newf>.<options>
```

<input> program file, default INPUT.

<output> output file, default OUTPUT.

<newf> file containing formatted output when option F is used.

<options> OPT=<letter sequence>

- F produce a formatted program file on <newf>. (Default file name is NEWF).
- L the lines on file <newf> are 80 characters long. (Default is 72 characters).
- N no listing of input.
- O input is a program for the old PASCAL compiler. (Default is a PASCAL2 program).
- U process 80 columns instead of 72 columns.

Symbolic form of generated code

```
EXECPF,6482,DECODE,<lgo>,<output>.
```

<lgo> file which contains relocatable code. Default is LGO.

<output> listing of code in symbolic form. Default is OUTPUT.

How to Make the Control Card Options Available to a Program

The following example shows how the control card options can be made available to a program:

Example:

```
TYPE OPTIONS=SET OF "A".."Z";
VAR CNTRL: RECORD
             CASE INTEGER OF
           1: (ADDR: INTEGER);
           2: (OPT: ^OPTIONS);
           END;
BEGIN CNTRL.ADDR:=100B;
  (*OPTIONS ARE AT LOCATION 100B*)
  WITH CNTRL DO BEGIN
    IF "X" IN OPT^ THEN ....
  END;
END.
```

Trap Labels

Control can be given back to a PASCAL program after a fatal error occurred if a trap label is specified. A label is declared as a trap label when it is followed by an asterisk in the label declaration.

The scope of trap labels may be nested. If a fatal error occurs then control is transferred to the current trap label, thereby making the trap label of the next lower level the current trap label. This avoids the possibility that the same trap label regains control after another fatal error occurs in the same context.

warning: The trap label is not updated by a jump out of the scope of a trap label into the scope of another trap label. Thus, jumps of this kind must be avoided.

Example:

```
PROGRAM TEST (OUTPUT);
LABEL 1*,2;

PROCEDURE ABC;
LABEL 5*,6;
BEGIN (* FATAL ERROR GIVES CONTROL TO 5 *)
   GOTO 6;
5: (* FATAL ERROR GIVES CONTROL TO 1 *)
6: END;

BEGIN (* FATAL ERROR GIVES CONTROL TO 1 *)
   ABC;(*WARNING: ABC MAY NOT BE LEFT BY A JUMP TO 2*)
   GOTO 2;
1: (* FATAL ERROR GIVES CONTROL TO POST-MORTEM DUMP
    IF AVAILABLE OR PROGRAM ABORTS *)
2: END.
```