12. Program Examples

12.1 Creation of a Data File

```
↱CREATE A STUDENT FILE.
 INPUT : 1) ANY NUMBER OF CARDS CONTAINING EACH
              A) A NAME IN COLUMNS 1..N    (N ≤ 10)
              B) A FIRST NAME IN COLUMNS N+P..N+P+Q  (P ≥ 2, Q ≤ 9)
              C) OTHER INFORMATION
         2) A CARD CONTAINING ≡$≡ IN COLUMN 1    ↓

TYPE SCHOOLYEAR = (FRESHMAN,SOPHOMORE,JUNIOR,SENIOR);
     STUDFILE = FILE OF
         PACKED RECORD NAME, FIRSTNAME : ALFA;
                       YEAR : SCHOOLYEAR;
                       AGE : 10..30;
                       ↱......↓
                END;
VAR STUDENTS[OUT] : STUDFILE;
    NAMEBUF : ARRAY[1..10] OF CHAR;
    I : 0..10; CH : CHAR;
    MAP : ARRAY[0..3] OF SCHOOLYEAR;
VALUE MAP = (FRESHMAN,SOPHOMORE,JUNIOR,SENIOR);

BEGIN READ(CH);
   REPEAT
      WITH STUDENTS↑ DO
      BEGIN
      ↱READ NAME↓
         I := 0;
         WHILE CH ≠ ≡ ≡ DO
         BEGIN I := I+1; NAMEBUF[I] := CH; READ(CH);
         END;
         FOR I := I+1 TO 10 DO NAMEBUF[I] := ≡ ≡;
         PACK(NAMEBUF,1,NAME);
         WRITE(NAME:12);

      ↱READ FIRST NAME↓
         REPEAT READ(CH) UNTIL CH ≠ ≡ ≡;
         I := 0;
         WHILE ¬(CH IN [≡ ≡,EOL]) DO
         BEGIN I := I+1; NAMEBUF[I] := CH; READ(CH) END;
         FOR I := I+1 TO 10 DO NAMEBUF[I] := ≡ ≡;
         PACK(NAMEBUF,1,FIRSTNAME);
         WRITE(FIRSTNAME:12);

      ↱ASSIGN YEAR↓
         YEAR := MAP[ TRUNC(SQRT(ORD(FIRSTNAME) DIV 4096)) MOD 4];
         WRITE(INT(YEAR),EOL);
         WHILE CH ≠ EOL DO READ(CH); READ(CH)
      END;
      PUT(STUDENTS);
   UNTIL CH = ≡$≡
END.
```

Data example:

```
WIRTH NIKLAUS
SCHILD RUEDI
AMMANN URS
MARMIER EDI
JENSEN KATHLEEN
$
```

12.2 Sort of a Data File

```
↱CLASSIFICATION OF STUDENTS↓
TYPE SCHOOLYEAR = (FRESHMAN,SOPHOMORE,JUNIOR,SENIOR);
     STUDFILE = FILE OF PACKED RECORD
                                  NAME, FIRSTNAME : ALFA;
                                  YEAR : SCHOOLYEAR;
                                  AGE : 10..30;
                                  ↱.....↓
                               END;

VAR STUDENTS[IN],
    FRESHMEN[OUT], SOPHOMORES[OUT],
    JUNIORS[OUT], SENIORS[OUT] : STUDFILE;
    OUTALF : ARRAY[SCHOOLYEAR] OF ALFA;
VALUE OUTALF = (≡FRESHMAN≡,≡SOPHOMORE≡,≡JUNIOR≡,≡SENIOR≡);

PROCEDURE PRINTFILE(VAR F: STUDFILE);
   ↱IT IS ASSUMED THAT F IS NOT EMPTY.↓
   VAR I : 1..6;
   BEGIN RESET(F); GET(F);
      REPEAT
         WITH F↑ DO WRITE(NAME:12,FIRSTNAME:12,OUTALF[YEAR]:12,EOL);
         GET(F)
      UNTIL EOF(F);
      WRITE(≡ ≡);
      FOR I := 1 TO 6 DO WRITE(≡----------≡);
      WRITE(EOL);
   END;

BEGIN RESET(STUDENTS); GET(STUDENTS);
   REPEAT
   CASE STUDENTS↑.YEAR OF
         FRESHMAN : BEGIN FRESHMEN↑ := STUDENTS↑; PUT(FRESHMEN) END;
         SOPHOMORE: BEGIN SOPHOMORES↑ := STUDENTS↑; PUT(SOPHOMORES)END;
         JUNIOR : BEGIN JUNIORS↑ := STUDENTS↑; PUT(JUNIORS) END;
         SENIOR : BEGIN SENIORS↑ := STUDENTS↑; PUT(SENIORS) END
   END;
   GET(STUDENTS)
   UNTIL EOF(STUDENTS);

   PRINTFILE(FRESHMEN); PRINTFILE(SOPHOMORES);
   PRINTFILE(JUNIORS); PRINTFILE(SENIORS);
END.
```

12.3 Tree Traversal

```
↱INPUT: 1 CARD CONTAINING THE DESCRIPTION OF A TREE, WHERE

 <TREE> ::=  <NON-BLANK CHARACTER>  (  <TREE>  ,  <TREE>  )
        ::=  <NON-BLANK CHARACTER>
        ::=  <ASTERISK>

EXAMPLE:   A(*,B(C(D(*,E),F(G(*,H),*)),I(*,J(K,*))))

        A
     (    )
  (          )
(               )
*               B
            (       )
        (               )
        C               I
     (    )           (   )
   (        )       (       )
   D        F       *       J
(    )   (    )          (     )
*    E   G    *          K     *
        ( )
        * H                                 ↓

TYPE POINTER = ↑TREE;
     ORDTYP = (PREORDER,POSTORDER,ENDORDER);

VAR TREE : CLASS 15 OF
       RECORD INFO : CHAR;
             LLINK,RLINK : POINTER
       END;
    T : POINTER;
    ORDER : ORDTYP; CH : CHAR;

PROCEDURE NEXTCH;
   BEGIN READ(CH); WRITE(CH) END;

PROCEDURE ENTERTREE(VAR P : POINTER);
   BEGIN
      IF CH = Ξ*Ξ ↱NO SON↓
      THEN BEGIN P := NIL; NEXTCH END
      ELSE ↱PROCESS SON↓
      BEGIN
         NEW(P);
         WITH P↑ DO
         BEGIN INFO := CH;
            NEXTCH;
            IF CH = Ξ(Ξ THEN ↱NON-TERMINAL NODE↓
            BEGIN NEXTCH;
               ENTERTREE(LLINK); NEXTCH;
               ENTERTREE(RLINK); NEXTCH
            END ELSE ↱TERMINAL NODE↓
            BEGIN LLINK := NIL; RLINK := NIL
```

```
              END
           END
        END
     END;

  PROCEDURE TRAVERSE (P : POINTER; TYP : ORDTYP);

     PROCEDURE VISITNODE(P : POINTER);
        BEGIN WRITE(P↑.INFO) END;

     BEGIN
        IF P ≠ NIL THEN
        CASE TYP OF
           PREORDER:  BEGIN
                         VISITNODE(P);
                         TRAVERSE(P↑.LLINK,TYP);
                         TRAVERSE(P↑.RLINK,TYP);
                      END;
           POSTORDER: BEGIN
                         TRAVERSE(P↑.LLINK,TYP);
                         VISITNODE(P);
                         TRAVERSE(P↑.RLINK,TYP);
                      END;
           ENDORDER:  BEGIN
                         TRAVERSE(P↑.LLINK,TYP);
                         TRAVERSE(P↑.RLINK,TYP);
                         VISITNODE(P);
                      END;
        END;
     END;

  BEGIN WRITE(≡ ≡); REPEAT NEXTCH UNTIL CH ≠ ≡ ≡ ;
     IF CH ≠ EOL THEN
     BEGIN
        ENTERTREE(T); WRITE(EOL,EOL);
        FOR ORDER := PREORDER TO ENDORDER DO
        BEGIN WRITE(≡ ≡); TRAVERSE(T,ORDER); WRITE(EOL)
        END
     END ELSE WRITE(≡ EMPTY TREE≡)
  END.
```

The result for the example above is:

```
 A(*,B(C(D(*,E),F(G(*,H),*)),I(*,J(K,*))))

 ABCDEFGHIJK
 ADECGHFBIKJ
 EDHGFCKJIBA
```

PART IV

13. Additional Procedures, Functions and Operations

13.1 Procedures

APPEND (A,B,C)

A must be a variable of type INTEGER, B and C expressions of type INTEGER or subrange thereof. The contents of A is shifted left by B bits and C is "or"ed to it. The result is in A. B and C are unchanged.

INSERT (A,B,C)

C must be a variable of type INTEGER. A and B expressions of type INTEGER or subrange thereof. The contents of A is shifted left by B bits and "or"ed to C. The result is in C. A and B are unchanged.

PP (A,B,C)

A must be of type ALFA, B and C of type INTEGER. The PP function is called which is specified by the first three letters of the ALFA value. If the fourth letter is a P the PP function is called with auto-recall.

Example:

```
PP(≡RFLP≡,0,40000B);
PP(≡DMPP≡,60108,7000B);
```

13.2 Functions

LOC(X)

X must be a variable or a label. The value returned is the absolute address of the variable or label. The result is of type INTEGER. Only labels which are already defined in a program part can be used as parameter. If the variable is a file then the address returned is the address of the File Environment Table (FET)-2.

TIME(X)

X must be a number from 0 to 8. TIME is a call to the PP function TIM. The value returned is for

0 TM-Time in milliseconds,
1 Date,
2 Time of day,
4 TM-Time left for the job in milliseconds,
7 PP-Time in milliseconds,
8 CP-Time in milliseconds.

The value returned is of type INTEGER, for 1 and 2 of type ALFA.

13.1 Variables and Operations

A <> B

A and B must be of type INTEGER. The absolute value of B must be less than or equal to 60. A is shifted left by B bits if B is greater than 0. A is shifted right by B bits (with sign extension) if B is less than 0. The result is of type INTEGER.

MEM

The memory allocated to the program can be considered to be defined by

```
MEM: ARRAY[0..FIELDLENGTH-1] OF INTEGER
```

The index of MEM is of type INTEGER. The integer is the absolute address of a location. Absolute addresses of variables are obtained with the function LOC. The values of pointer variables are also absolute addresses; they are transformed to values of type INTEGER with the function ORD.

ALFALENG

This is a synonym for the constant 10. It shows that this PASCAL implementation uses words with 10 characters.

14. Restrictions

14.1 Procedures and Functions

Formal parameters of kind procedure or function are not specified with the number of their parameters or with their types. It is therefore the responsibility of the programmer to use procedures or functions as actual parameters which have the correct number of parameters and the correct types with which the formal parameter is used.

Procedures and functions which are formal parameters cannot be called with actual parameters which correspond to formal parameters of kind procedure or function.

Standard functions and procedures cannot be used as actual parameters for formal parameters of kind function or procedure. The exceptions are SIN, COS, EXP, LN, SQRT, ARCTAN, and RANF.

No check is made if the value of the expression on an actual parameter position lies within the range for which the formal parameter is specified.

A function or procedure which contains a local file definition should not be activated recursively.

Formal parameters of type FILE and CLASS cannot be of kind value.

14.2 Types, Identifiers

Two variables are not recognized as being of the same type unless they occur in the same variable list or belong to the same type which was explicitly named in a type definition.

Components of structured types cannot be of type FILE or CLASS.

The type definition SET OF BOOLEAN and SET OF CHAR is possible. If the value of an element of a SET type is greater than or equal to 59 then the element which corresponds to value mod 64 is assigned.

An identifier may be used only once within a declaration part for naming purposes. Exceptions are field names of records. They can appear also in other record declarations, however, only once in each record declaration. The following is also possible:

```
          VAR X: RECORD X: INTEGER END;
```

All identifiers which have a predefined meaning, e.g. the names of standard functions and standard procedures may be used in a declaration part and thus redefined. They are local names to this declaration part and the corresponding program part.

15. The PASCAL System

The PASCAL system consists of

(1) The PASCAL Operating System, which contains the run-time support routines to handle e.g. input and output.

(2) The compiler

(3) Some library routines, e.g. SIN, COS, etc.

The PASCAL Operating System is used by the compiler and by the translated programs. The compiler generates absolute code and writes it to a temporary file with name PASCLGO. In order to execute the translated program this file is read on top of the PASCAL Operating System, then if necessary the requested library routines are loaded from the PASCAL library and the external routines from the library file, and control is given to the main program.

The generated code is kept on a file if this is requested in the Control Card. After the PASCAL Operating System is loaded it can be executed in the same manner as before. (The compiler itself is a kept PASCAL program.)

The space starting at the last word read in from the file PASCLGO or loaded from the library and ending at the end of the fieldlength is used as stack. Whenever a procedure or function is entered its return address is written on top of the stack and space for the local defined variables is reserved. The stack is reset when the procedure or function is left. The length of the stack can be influenced by the FL parameter on the Control Card. Normally there are at least 1000B words available.

Forward references and global jumps are handled by the so-called jump table. The table is 100B words long and is at the end of the space reserved for the variables of the main program and before the first generated code.

The time spent in the various parts of a program can be monitored if requested on the Control Card. The result is printed when the program finishes.

A special program is also available with which a Cross Reference Listing of the identifiers used in a PASCAL program can be produced. The Control Card used for this program is described in Appendix B.3.

The tables in the compiler are of fixed length. This imposes some almost negligible restrictions on the programs to be translated. The tables are used dynamically, i.e. if the compiler finishes with a procedure or function definition the entries used for the local entities are released.

The tables and their sizes are:

Size	Table for
10	files, which are declared in the declaration part of procedures and function,
10	classes, which are declared in the declaration part of procedures or functions,
10	labels, which are declared in the label section of a declaration part,
20	labels, which are defined in a program part,
40	constants, with values > 2↑17 ,
500	identifiers,
1000	code (available for each procedure or function).

Other limits are:

7	nesting depth of procedure or function definitions,
10	nesting depth of WITH statements,
30	labels in a CASE statement.

16. Special Features

16.1 Partial Compilation

In order to avoid the recompilation of large programs the state of the compilation can be frozen at the level of procedure declarations. This is done in the following way:

```
Example:        VAR I: INTEGER;
                PROCEDURE XYZ;
                   BEGIN ... END;

                FREEZE.
```

The program so far translated together with the local and global variables (especially the used part of the symbol table) of the compiler are kept on a file whose name is defined in the Control Card by

```
                FK=<file name>
```

or by default on the file PASCLGO.

The compilation can be continued where it was left. The continuation of the program is expected to be on the file named in the P parameter of the Control Card. The file which contains the partial compiled program must be mentioned on the Control Card as follows:

```
                FB=<file name>
```

The file on which the frozen compilation is kept is copied to file PASCLGO (if the file name is not PASCLGO) and the local and global variables of the compiler obtain the values they had when the compilation was frozen. However, the option parameters on the Control Card are used for the continued compilation. (The file PASCLGO is always returned at the end of the compilation.)

16.2 PASCAL Overlays

PASCAL programs can be translated as PASCAL overlays when EX=OVL is specified on the Control Card. The program then may not contain a VALUE part. Further no code to open the files of the main program and to initialize the internal class pointers of the main program is generated. The K parameter must specify on what file the overlay is to be kept.

PASCAL overlays can be called by PASCAL programs. The overlay is loaded from the jumptable onward. The space occupied by the variables and file buffers of the main program must be the same as for the overlay. Further it is expected that the file buffers are at the same place in the program and in the overlay. When the overlay is loaded the values of the variables and the buffers of the main program remain unchanged. Only the buffer of the file OUTPUT is emptied as this buffer is used to load external and standard procedures. Then control is transfered to the entry point of the main program of the overlay.

An overlay can only be called from a main program. This is done in the following way:

Example:

```
        OVERLAY(P1,P2,P3);
```

The parameters are:

P1 is of type ALFA and used as file name of the file where the overlay resides. If P1 is ≡   ≡ the overlay is on the file mentioned in the B parameter of the Control Card.

P2 an integer which specifies the fieldlength needed for the overlay.

```
     0    fieldlength = max(current fieldlength, required fieldlength)
    -1    fieldlength = required fieldlength
   XXX    fieldlength = XXX
```

P3 an integer which specifies the position of the overlay on the file.

```
     0    take next overlay on the file.
   XXX    rewind the file and take the XXXth overlay on the file.
```

16.3 Low Core

Some values in the low core of the PASCAL Operating System may be of interest to the programmer. Location 312B contains 1 if the standard output is TTY, otherwise 0. This information can be used to write programs for conversational input. The programmer must also insure that all output is sent to the teletype before input is accepted. This is done by finishing each output sequence with WEOR(OUTPUT).

Example:

```
VAR TTY: BOOLEAN;  X: INTEGER;

BEGIN TTY := MEM[312B]=1;  . . . . .
      WRITE(≡ START≡,EOL);
      IF TTY THEN WEOR(OUTPUT);
      READ(X);               . . . . .
END.
```

The option parameters are only used by the compiler. They can also be used by a kept program. When the option is mentioned on the Control Card the corresponding location contains 1 otherwise 0.

Location	Option
300B	A
301B	X
302B	D
303B	O
304B	C
305B	R
316B	S
317B	N

16.4 External Routines

Procedures and functions can be declared EXTERNAL in the declaration part of the main program.

Example: **FUNCTION F(X: INTEGER): REAL; EXTERNAL;**

External routines are relocatable COMPASS programs which must correspond to the calling sequence of PASCAL routines. External procedures may be called with any number of parameters. No check is made on the type or kind of the parameters. Up to 10 external procedures can be declared. External procedures may not be used as actual parameters.

The addresses of the parameters begin at location B6+3. B7 contains the number of parameters with which the procedure was called, X7 the return address. In the case of a function the value of the result is expected in B6+2.

The restrictions on external procedures are quite severe. They must have only 1 entry point, no common blocks and no external references. It is to be observed that the PASCAL system uses the registers B4, B5, B6 and that B1 must have the value 1. The locations 150B to 167B can be used as communication area for the external procedures. The locations 145B to 147B are reserved for addresses of the PASCAL Operating System, e.g. the pointer to the PEX word.

The external procedures must be on a file in the same order as they were declared in the PASCAL program. The name of the file is specified in the Control Card by

LIB=<file name>

By default PASCLIB is used as file name.

16.5 File Buffers and File Environment Table

The File Environment Table (FET) consists of 8 words:

Word	Contents	
1	File name	(Up to 7 characters left justified with zero fill).
2	First	pointer to first word of buffer
3	In	
4	Out	
5	Limit	pointer to last word of buffer +1
6	0	
7	0	
8	0	

The FET is preceded in PASCAL by two words:

Word	Contents
-1	"file head"
0	file descriptor

The file descriptor contains information about the file. In the following the bits are counted from left to right, starting with 1:

Bit	
1	if on, the file is in end-of-file status.
2	if on, the file is of kind Input.
4	if on, the file is of kind Output.
5	if off, the file is of type CHAR.

The file is a scratch file if neither bit 2 nor bit 4 are set. The rest of the word is used to specify the length of a file component.

If the file is a file of type CHAR the extended FET is preceded by a 10 word buffer. The "file head" points according to the file type either to a component in the character buffer or to a component in the file buffer.

Appendix A

A.1 Character Set

	0	1	2	3	4	5	6	7
0		A	B	C	D	E	F	G
1	H	I	J	K	L	M	N	O
2	P	Q	R	S	T	U	V	W
3	X	Y	Z	0	1	2	3	4
4	5	6	7	8	9	+	-	*
5	/	(	)	$	=		.	.
6	≡	[	]	:	≠	↱	∨	∧
7	↑	↓	<	>	≤	≥	¬	;

The value 00B corresponds to EOL, the value 55B to blank.

A.2 Table of Standard Identifiers

Constants:	FALSE, TRUE, EOL, ALFALENG
Types:	INTEGER, BOOLEAN, REAL, CHAR, ALFA, TEXT
Variables:	INPUT, OUTPUT, MEM
Functions:	ABS, SQR, ODD, SUCC, PRED, ORD, CHR, ALF, EOF, TRUNC, TIME, LOC, SIN, COS, EXP, LN, SQRT, ARCTAN, RANF
Procedures:	GET, PUT, RESET, READ, WRITE, INFILE, OUTFILE, WEOR, PACK, UNPACK, NEW, PP, DUMP, TRACE, APPEND, INSERT

A.3 Printer Control Characters

Blank	next line
1	page eject before printing
2	advance to the last line of the form before printing
+	suppress line space before printing
/	suppress line space after printing
0	space one line before printing
-	space two lines before printing

Appendix B

The Control Card for translating a program, executing it and eventually keeping the translated version is described in B.1. The Control Card for executing a kept program is described in B.2. The Control Cards necessary to generate a Cross Reference Listing are to be found in B.3.

B.1 Translation and Execution of a PASCAL Program

```
PASCAL,P=<program>,L=<listing>,D=<data>,R=<result>,K=<keepfile>,
       FK=<freeze keepfile>,FB=<freeze loadfile>,LIB=<library>,
       OPT=<options>,LL=<line limit>,FL=<fieldlength>,
       EX=<execution mode>,FCL=<lower>,FCU=<upper>.
```

The parameters may appear in any order.

P=<program>

<program> is the file which contains the source program. By default it is INPUT.

L=<listing>

File to which the listing of the source program is written. By default it is OUTPUT.

D=<data>

File which is used as standard input. By default it is the file INPUT. The file is not rewound if it is the same as the program file or INPUT, otherwise it is rewound.

R=<result>

File which is used as standard output. By default it is the file OUTPUT.

K=<keepfile>

File to which the translated version of the program is written.

FK=<freeze keepfile>

File to which a partial compiled program is written. By default it is PASCLGO.

FB=<freeze loadfile>

File which contains a partial compiled program. The compilation is continued.

LIB=<library>

File which contains the external routines. By default it is PASCLIB.

OPT=<options>

<options> is a list of letters which may appear in any order, e.g. OPT=ASP. Options for the compiler are:

A Generate additional code to check assignments.
C Print the generated code in COMPASS assembler form.
D Generate additional code for divisions to check for a zero divisor.
N do not list the source program.
O Generate additional code to check for stack overflow.
R Generate code for true rounding.
S Save symbolic dump information.
X Generate code to check for array bounds and label values in CASE statements.

The options are activated when the option letter appears on the Control Card. The options may be changed within the program as described in 11.1.

Options for the program execution are:

P Produce a symbolic dump automatically when a run-time error occurs.

T Produce a symbolic and octal dump automatically when a run-time error occurs.

LL=<line limit>

Decimal number which specifies the maximum number of lines which may be written by the program. The default value is 1000, about 15 pages. There is no line limit if LL=0.

FL=<fieldlength>

An octal number which specifies the minimum fieldlength required for the execution of the program. The default value for the minimum fieldlength is computed as follows:
X = (address of the last generated word rounded to the next higher 1000B) +1000B.
The octal number specified on the Control Card may not be smaller than this value X.

The minimum fieldlength required is also recorded when the translated version of the program is kept. If the program is immediately executed after the compilation the fieldlength is automatically adjusted to the minimum fieldlength.

EX=<execution mode>

NOGO The program is translated (and kept if requested) but not executed.

FC The time spent in the various parts of the program is monitored. The parameters <lower> and <upper> are used to specify the range of addresses to be monitored. Jobs using the time sharing system cannot be executed in this mode.

OVL The program is translated as overlay.

FCL=<lower>

<lower> is an octal number which is less than the fieldlength and greater than or equal to 0. By default the beginning of the executable code of the translated program is taken.

FCU=<upper>

<upper> is an octal number which is less than the fieldlength and greater than or equal to <lower>. By default the end of the executable code of the translated program is taken.

Octal numbers on the Control Card are not followed by the letter B.

B.2 Execution of a Kept PASCAL Program

```
PASCAL,B=<load file>,D=<data>,R=<result>,LIB=library,
       OPT=<options>,LL=<line limit>,FL=<adjust>,
       EX=<execution mode>,FCL=<lower>.FCU=<upper>.
```

B=<loadfile>

File which was kept in a previous translation.

FL=<adjust>

By default the fieldlength used is the larger of current fieldlength and minimum required fieldlength as stated when the program was translated. The fieldlength for the execution of the program is adjusted to the minimum required fieldlength if FL=MIN is used as parameter.

All other parameters are described in B.1.

B.3 Control Cards for a Cross Reference Listing of a PASCAL Program

Assume the source program is on file ABC.

```
        READPF,2117,XREF.
        PASCAL,B=XREF,D=ABC,LL=0.
```

The program is listed and followed by a table of cross references of the identifiers. The program listing is suppressed if the parameter OPT=N is used.

References

Wirth, N. The Programming Language PASCAL,
Acta Informatica 1, 35-63, (1971)

Wirth, N. The Programming Language PASCAL,
(Revised Report), Bericht No. 5,
Berichte der Fachgruppe Computer-Wissenschaften,
Eidgenoessische Technische Hochschule, Zurich
(November 1972)

Wirth, N. The Design of a PASCAL Compiler,
Software - Practice and Experience,
1, 309-333, (1971)

Ammann, U., Schild R. Private communications,
Eidgenoessische Technische Hochschule, Zurich

Burger, W. Unpublished documents on the Programming
Language PASCAL,
University of Texas at Austin, (1972)

Index

Corrections and Additions to the PASCAL Manual, Version July 73

Page 7, replace line 12 and 13:

Example:

```
TYPE SHORT = 0 .. 3;
COLOR = (RED,GREEN,BLUE,WHITE,ORANGE);
FIRST = WHITE .. ORANGE;
```

(The use of subrange types is restricted in this PASCAL version, see 14.2).

Page 10, insert after line 40:

NOTE: In assignments of the form A := B the variables A and B may be of type RECORD or ARRAY. However, they must be of the same type.

Page 11, replace line 13

Example:

```
WRITE(≡ ≡,X+3,A,≡MILES≡,EOL);
```

Page 16, insert after line 35:

Equality and inequality may be applied to variables of type ARRAY and RECORD if the variables are of the same type. The elements of the two arrays or the two records are compared until all elements are compared or a disagreement is found. Equality and inequality may not be applied to variables of type CLASS and FILE (see 6.1 and 9.2).

Page 19, replace line 18:

```
A := ALF(01020355555555555555B);
```

Page 21, insert after line 42:

Note that the type of a parameter can only be specified by a type identifier.

Page 24, insert after line 33:

5.21 VAR parameters and value parameters

As mentioned earlier parameters of kind value are treated like local variables. Thus the value assigned to such a variable is strictly local and not known to the calling program part. The expression at the actual parameter position is evaluated when the procedure is called and assigned to the local variable. If this variable happens to be an array or record then the whole array or record is copied to the local array or record.

Only variables may be used as actual parameters for formal parameters of kind variable. An assignment to the formal variable in the procedure results in an assignment to the actual variable.

Example:

```
PROCEDURE TEST1(X: INTEGER);
BEGIN  X := 5  END;

PROCEDURE TEST (VAR X: INTEGER);
BEGIN  X := 5  END;
• • •

A := 0;
TEST1(A);      ↱THE VALUE OF A IS STILL 0↓
TEST2(A);      ↱THE VALUE OF A IS NOW 5↓
```

Page 32, insert after line 50:

The number of bits occupied by a packed record component can be found in 13.4.

Page 33, insert after line 36:

WARNING: No assignment may be made to the record-variable within the WITH statement.

Page 37, replace line 12:

when the value is present in the set. Counting the bits of a word from left to right beginning with 0 the value 58 corresponds to bit 1 and the value 0 to bit 59. The sign bit of the word is not used.

Page 39, replace line 10 to 12:

For parameters of other possible types it is defined by

```
CH := INPUT↑; GET(INPUT);
• • •
CH := INPUT↑; GET(INPUT);
```

The character immediately following the item read is then under the "file head".

Page 41, insert after line 49:

A line may contain 136 characters (including the carriage control character). A new line is started when the 137th character is not EOL. A blank is used as carriage control for the new line, and a $ sign indicates that an overflow of the previous line occurred.

Page 42, insert after line 45:

The value NIL may also be used to initialize pointer variables.

Page 43, insert after line 44:

```
   P      Page eject

          The next line of the listing starts on a new page.
```

Page 45, insert after line 26:

```
Example:       PROCEDURE ABC;
               ↱$T+↓
               BEGIN . . . . END;

               PROCEDURE XYZ;
               ↱$T-↓
               BEGIN . . . . END;
               . . . .
               TRACE(1); ABC; XYZ;
               ↱GIVES TRACE INFORMATION ABOUT THE PROCEDURE CALL ABC↓
```

Page 51, replace line 1

13.3 Variables and Operations

Page 51, insert after line 26

13.4 Packed Records

The number of bits occupied by a field of a packed record depends on the type of the field. It is

```
      18 bits   for pointer types,
       6 bits   for type CHAR and subrange thereof,
       1 bit    for type BOOLEAN.
```

For scalar types and subrange types with a lower bound greater than or equal to 0 the number of bits used is the number of bits with which the highest value of the set can be expressed. For types with a lower bound less than 0 the number of bits is given by the number of bits with which the larger of the absolute values of lower bound and upper bound can be expressed plus an additional bit for the sign. All other types require one or more words. The last field in a word is always right justified.

In the following example the bits of a word are numbered from left to right, beginning with 0:

```
Example:       R: PACKED RECORD
                     F1: CHAR;                    ↱WORD 1, BIT 0-5↓
```

```
                F2: BOOLEAN;                ↱BIT 6↓
                F3: (RED,GREEN,BLUE);       ↱BIT 7-8↓
                F4: 0..777B;                ↱BIT 9-17↓
                F5: -777B..+77B;            ↱BIT 18-27↓
                F6: 0..100;                 ↱BIT 28-59, LAST FIELD↓
                F7: INTEGER;                ↱WORD 2↓
                F8: ARRAY[1..5] OF CHAR;    ↱WORD 3-7↓
                CASE C: BOOLEAN OF          ↱WORD 8, BIT 1↓
                 TRUE:  (F91: 0..77B;       ↱WORD 8, BIT 2-7↓
                         F92: 0..77B);      ↱WORD 8, BIT 8-59↓
                 FALSE: (F10: CHAR);        ↱WORD 8, BIT 2-59↓
            END;
```

Page 52, replace line 36:

explicitly named in a type definition. A type declaration of the form A = B is not possible.

Subrange types of CHAR, BOOLEAN, and of scalar type cannot be used in type and variable declarations with the exeption of using them as index type of arrays or as base type of sets. (The example of 2.2.13 shows the only use of a scalar subrange type which is not permitted in this PASCAL version.) There are no restrictions for subrange types of INTEGER.

The type name declaration for SET types is limited to one name per base type (and subranges thereof). The following example is not permitted in this PASCAL version since the base type in both cases is a subrange of type CHAR:

```
            TYPE A = SET OF ≡A≡..≡Z≡;
                 B = SET OF ≡1≡..≡9≡;
```

Either type declaration would be acceptable. Another possibility is the type declaration:

```
            TYPE X = SET OF CHAR;
```

Page 58, insert after line 24:

Two additional words are used as counter: word -12 is used as column counter and word -13 as card counter.

Page 61, insert after line 33:

The data must immediately follow the last line of the program if data and program are on the same file. A 7-8-9 card may be used as separator of program and data.

Page 64: insert Appendix C and Appendix D

Appendix C

==========

Error Messages

```
  1: SCALAR TYPE EXPECTED.
  2: INTEGER TOO LARGE.
  3: ERROR IN CONSTANT.
  4: ≡=≡ EXPECTED.
  5: FIELD NAME DECLARED TWICE.
  6: BAD RANGE.
  7: TAG FIELD TYPE BAD.
  8: NAME DECLARED TWICE.
  9: ≡)≡ EXPECTED.
 10: ≡:≡ EXPECTED.
 11: IDENTIFIER EXPECTED.
 12: IDENTIFIER NOT DECLARED.
 13: INDEX MUST BE OF SCALAR TYPE.
 14: ≡OF≡ EXPECTED.
 15: VARIABLE TYPE IS NOT CLASS.
 16: PROCEDURE DECLARED TWICE.
 17: ≡END≡ EXPECTED.
 18: ERROR IN TYPE DECLARATION.
 19: ERROR IN VARIABLE DECLARATION.
 20: ERROR IN VALUE PART.
 21: ERROR IN PROCEDURE DECLARATION.
 22: VALUE PART IN PROCEDURE.
 23: PARAMETER LIST IGNORED.
 24: ERROR IN DECLARATION PART.
 25: LOWBOUND > HIGHBOUND.
 26: NOT A VARIABLE IDENTIFIER.
 27: DECREASING ADDRESSES IN VALUE PART.
 28: SYMBOLIC SUBRANGE TYPE NOT ALLOWED.
 29: PARAMETER MISSING IN FUNCTION DECLARATION.
 30: COMPONENT TYPE IS CLASS OR FILE.
 31: UNDECLARED IDENTIFIER.
 32: VARIABLE OR FIELD IDENTIFIER EXPECTED.
 33: EXPRESSION TOO COMPLICATED.
 34: TYPE OF VARIABLE SHOULD BE ARRAY.
 35: TYPE OF EXPRESSION MUST BE SCALAR.
 36: CONFLICT OF INDEX TYPE WITH DECLARATION.
 37: ≡]≡ EXPECTED.
 38: TYPE OF VARIABLE SHOULD BE RECORD.
 39: NO SUCH FIELD IN THIS RECORD.
 40: TYPE OF VARIABLE SHOULD BE POINTER OR FILE.
 41: FIELD NAME EXPECTED.
 42: ILLEGAL SYMBOL IN EXPRESSION.
 43: UNDEFINED LABEL.
 44: ILLEGAL TYPE OF PARAMETER IN STANDARD FUNCTION OR PROCEDURE.
 45: TYPE IDENTIFIER IN STATEMENT PART.
 46: PROCEDURE USED AS FUNCTION.
 47: TYPE OF STANDARD FUNCTION PARAMETER SHOULD BE INTEGER.
 48: ≡)≡ EXPECTED.
 49: IDENTIFIER EXPECTED.
```

```
50: ILLEGAL TYPE OF OPERAND.
51: ≡∨≡ CANNOT BE USED AS MONADIC OPERATOR.
52: ≡:=≡ EXPECTED.
53: ASSIGNMENT NOT ALLOWED.
54: ILLEGAL SYMBOL IN STATEMENT.
55: TYPE OR CONSTANT IDENTIFIER.
56: ≡THEN≡ EXPECTED.
57: TYPE OF EXPRESSION IS NOT BOOLEAN.
58: ≡;≡ EXPECTED.
59: ≡DO≡ EXPECTED.
60: ILLEGAL PARAMETER SUBSTITUTION.
61: LABEL EXPECTED.
62: ILLEGAL TYPE OF EXPRESSION.
63: CONSTANT EXPECTED.
64: ≡:≡ EXPECTED.
65: ≡OF≡ EXPECTED.
66: TAG FIELD MISSING FOR THIS VARIANT.
67: ≡UNTIL≡ EXPECTED.
68: ≡END≡ EXPECTED.
69: LOOP CONTROL VARIABLE MUST BE SIMPLE AND LOCAL OR GLOBAL.
70: ≡TO≡ OR ≡DOWNTO≡ EXPECTED.
71: TOO MANY CASES IN CASE STATEMENT.
72: NUMBER OF PARAMETERS DOES NOT AGREE WITH DECLARATION.
73: MIXED TYPES.
74: TOO MANY LABELS IN THIS PROCEDURE.
75: TOO MANY LONG CONSTANTS OR YET UNDEFINED LABELS IN THIS PROCEDURE.
76: DEPTH OF PROCEDURE NESTING TOO LARGE.
77: LABEL DEFINED MORE THAN ONCE.
78: TOO MANY EXIT LABELS.
79: ≡(≡ EXPECTED.
80: ≡,≡ EXPECTED.
81: ASSIGNMENT TO FORMAL FUNCTION IDENTIFIER ILLEGAL.
82: TOO MANY NESTED WITH-STATEMENTS.
83: STANDARD IN-LINE PROCEDURE OR FUNCTION USED AS ACTUAL PARAMETER.
84: TOO MANY LONG CONSTANTS IN THIS PROCEDURE.
85: ASSIGNMENT TO FUNCTION IDENTIFIER MUST OCCUR IN FUNCTION ITSELF.
86: ACTUAL PARAMETER MUST BE A VARIABLE.
87: PACKED FIELD NOT ALLOWED HERE.
88: OPERATORS ≡<≡ AND ≡>≡ ARE NOT DEFINED FOR POWERSETS.
89: REDUNDANT OPERATION ON POWERSETS.
90: PROCEDURE TOO LONG.
91: TOO MANY EXIT LABELS OR FORWARD PROCEDURES.
92: TOO MANY CLASS OR FILE VARIABLES.
93: BAD FUNCTION TYPE.
94: ONLY ≡=≡ AND ≡≠≡ ALLOWED HERE.
95: BAD FILE DECLARATION.
96: TYPE DECLARED TWICE.
97: ≡END.≡ ENCOUNTERED.
98: ≡[≡ EXPECTED.
99: INDEX OUT OF RANGE.
```

```
100: LABEL TOO LARGE.
101: VALUE IS OUT OF RANGE.
102: DIVISION BY ZERO.
103: PARAMETER PROCEDURE HAS MORE THAN 17 PARAMETERS.
104: TEN OR MORE ERRORS ON THIS LINE.
105: STRING TOO LONG.
106: TOO MANY EXTERNAL PROCEDURES.
107: EXTERNAL DECLARATION NOT IN MAIN PART.
108: EXTERNAL PROCEDURE USED AS ACTUAL PARAMETER.
109: OVERLAY CALL NOT IN MAIN PROGRAM.
110:
```

Appendix D

A graphical output package is available for the PASCAL version at the University of Texas at Austin. The package consists of the following routines:

PLT
BGNPLT
ENDPLT
MULPLT
WHERE
OFFSET
SYMBOL
SYMSET
NUMBER

These routines are described in more detail in Chapter 14 of the Users Manual, Computation Center, The University of Texas at Austin.

In order to use a routine of the graphics package, a PASCAL program must contain the declaration:

```
PROCEDURE GRAPH; EXTERNAL;
```

The first parameter in a call to GRAPH is a value of type ALFA which describes the specific routine of the package to be called. Then follow the parameters of this routine.

Example:
```
GRAPH(≡PLT≡,0.5,0.6,2);
```

It is the responsibility of the programmer to call the graphics routines with the appropriate number and type of parameters. ALFA values are used where Hollerith values would be required. A type conversion from integer to real in any parameter position must be done explicitly by multiplying by 1.0. Conversion from real to integer is done by the function ORD.

The graphics package is obtained by

```
READPF,2117,PLTLGO.
```

This file must be specified as value of the LIB parameter on the PASCAL control card or else be copied to the file containing the other external routines used by a particular PASCAL program.

2. SUPPLEMENT TO THE PASCAL MANUAL, VERSION JULY 73

THE PASCAL VERSION MAY 1974 MAKES SOME CHANGES TO THE PASCAL MANUAL NECESSARY. THEY ARE MOSTLY CONCERNED WITH THE AUTOMATIC INCREASE OF THE FIELDLENGTH WHEN A STACK OVERFLOW OCCURS.

PAGE 43, DELETE LINE 39 TO 43 (OVERFLOW OPTION)

PAGE 44, REPLACE LINE 7:

```
        →$A-,C-,D-,L+,R-,X-↓
```

PAGE 45, REPLACE LINE 2:

SYSTEM. IN THE CASE OF STACK OVERFLOW (I.E. WHEN THE MAXIMUM FIELDLENGTH IS EXCEEDED) ONLY THE MAIN PROGRAM VARIABLES ARE DUMPED. A PASCAL DUMP IS ALSO GIVEN FOR TIME LIMIT AND USER ABORT WITH DUMP IF A DUMP OPTION IS SPECIFIED ON THE CONTROL CARD. THE DUMP IS ALWAYS WRITTEN TO THE STANDARD FILE OUTPUT.

PAGE 50, REPLACE LINE 24 TO 26:

> FIRST THREE LETTERS OF THE ALFA VALUE. IF THE FOURTH LETTER IS NOT A BLANK THE PP FUNCTION IS CALLED WITH AUTO-RECALL. (A CALL TO THE PP FUNCTION RFL ALSO SETS REGISTER B4, WHICH IS USED FOR THE STACK OVERFLOW CHECK).

PAGE 53, INSERT AFTER LINE 33:

> IF THE STACK BECOMES TOO SMALL TO RESERVE SPACE FOR THE LOCAL VARIABLES, THE FIELDLENGTH IS INCREASED BY 2000B WORDS (OR MORE, IF MORE SPACE IS NEEDED). A MAXIMUM FIELDLENGTH, HOWEVER, WILL NOT BE EXCEEDED. THE MAXIMUM FIELDLENGTH MAY BE SPECIFIED BY THE FM PARAMETER.

PAGE 54, INSERT AFTER LINE 20:

```
   17        FUNCTION OR PROCEDURE PARAMETERS
```

PAGE 61, INSERT AFTER LINE 15:

```
        FM=<MAXIMUM FIELDLENGTH>,
```

PAGE 62, REPLACE LINE 15:

```
        0        (NOT USED)
```

PAGE 62, REPLACE LINE 44 TO 49:

FOR THE EXECUTION OF THE PROGRAM. THE DEFAULT VALUE FOR THE MINIMUM FIELDLENGTH IS COMPUTED BY THE COMPILER AND PRINTED ON THE LISTING TOGETHER WITH THE PROGRAM LENGTH. THE OCTAL NUMBER SPECIFIED ON THE CONTROL CARD MUST BE LARGER THAN THE PROGRAM LENGTH + 100B, OTHERWISE IT IS IGNORED AND THE DEFAULT VALUE IS USED.

PAGE 62, INSERT AFTER LINE 55:

FM=<MAXIMUM FIELDLENGTH>

AN OCTAL NUMBER WHICH SPECIFIES THE MAXIMUM FIELDLENGTH WHICH THE PROGRAM MAY OBTAIN. THE DEFAULT VALUE IS 70000B. THE MAXIMUM FIELDLENGTH SHOULD BE LARGER THAN THE FIELDLENGTH WITH WHICH THE PROGRAM IS LOADED. OTHERWISE A WARNING MESSAGE IS GIVEN AND NO DYNAMIC INCREASE OF THE FIELDLENGTH IS POSSIBLE.

PAGE 63, INSERT AFTER LINE 12:

A FILE ON WHICH THE OVERLAY IS TO BE KEPT MUST BE SPECIFIED WITH THE K PARAMETER.

PAGE 63. INSERT AFTER LINE 35:

FM=<MAXIMUM FIELDLENGTH>,

CHANGES TO APPENDIX C

 69: LOOP CONTROL VARIABLE MUST BE SIMPLE AND LOCAL OR IN MAIN PROGRAM.
103: PROCEDURE HAS MORE THAN 17 PARAMETERS.
110: CLASS NOT DECLARED OR ERRONEOUS.
111: STRING CROSSES CARD BOUNDARY.