STARTING PASCAL

on the Sharp microcomputer

R G MEADOWS

BSc, MSc, PhD, MIEE, CEng, MInstP, ARCS

SHARPSOFT LTD.
86-90 Paul Street,
London EC2A 4NE

Tel: 01-739 8559

First published 1983

ISBN 0 907690 03 3

ASSOCIATED BOOKS BY SAME AUTHOR

A beginners guide to using the Sharp microcomputers. (SHARPSOFT)

Microprocessors: essentials, components and systems. (PITMANS)

CONTENTS

PREFACE

This book has been written for beginners wishing to learn PASCAL so as to be able to write their own programs in this important and now widely used language.

No prior knowledge of any other programming language is assumed nor is it needed. The text is written with the underlying purpose of showing in the clearest and easiest possible way how to write programs in PASCAL.At every stage examples of complete programs are included to support the Pascal concepts being introduced. Good luck!

Richard Meadows
May 1983

DEDICATION

To Mike and Larry

CHAPTER 1
GETTING STARTED

1.1 INTRODUCTION

In this first chapter we start by loading the PASCAL INTERPRETER tape and then learn to use the basic edit commands, which you will find absolutely essential in order to be able to prepare, modify and run programs. The chapter concludes with some simple PASCAL programs designed to introduce you to this language, get you conversant with the edit commands and running PASCAL PROGRAMS.

1.2 LOADING THE PASCAL INTERPRETER TAPE SP-4015

1. First turn on your computer-the power switch is located at the back. The following will be displayed on your screen:

** MONITOR SP-1510 **	** MONITOR SP-1002 **
for the "A"	for the "K"

Note for "A" users only. The Sharp Interpreter SP-4015 can be used for both A and K computers. However, for the A you must now press

[CNTL] + [{ ε]

This operation essentially configures the A as a K and must always be done to avoid errors which would otherwise occur in entering and running programs.

2. Insert the PASCAL INTERPRETER tape SP-4015, and type in LOAD followed by CARRIAGE RETURN (the CR key), i.e.

```
LOAD  <CR>
```

← symbol used throughout text for "press CR key".

3. ⬇ PLAY will be displayed.

Press the PLAY key on the cassette control unit.

```
LOADING PASCAL SP-4015
```

will appear within 10 or so seconds, followed when loading is completed, after about 2 minutes, by:

```
* INTERPRETER PASCAL SP-4015
31208.BYTES
READY
```

WE ARE NOW READY TO GO!

3 PREPARING AND EDITING PASCAL PROGRAMS

We begin straightway by entering a simple PASCAL program so as to learn the basic edit commands associated with preparing and modifying programs for your future work.

Here is a simple PASCAL program.

```
0.%TO LIST SEASONS OF YEAR%
1.BEGIN
2.   WRITELN("************");
3.   WRITELN("   SPRING   ");
4.   WRITELN("   SUMMER   ");
5.   WRITELN("   WINTER   ");
6.   WRITELN("************")
7.END.
8.
```

We will first type it in and then see how we can list it, run it, insert extra lines, correct any errors, kill it ...etc.

The program writes out the seasons of the year. You will probably have seen I have, at this stage, missed out AUTUMN.

(a) TO ENTER A PROGRAM VIA THE KEYBOARD

1. Type in

```
B        <CR>
```

This command is always used on commencing entries.

```
0.▒▒
```

is displayed.

Line numbers are automatically displayed. 0 is the first, followed by 1,2,3 ... after entering the line with <CR>.

2. Now type in the first line, in our example

```
0.%TO LIST SEASONS OF YEAR%    <CR>
1.▒▒
```

automatically displayed
ready for next entry

3. Type in second line followed by <CR>,i.e.

```
1.BEGIN     <CR>
```

4. Type in subsequent lines of program, taking great care to omit (or add) nothing-the program must be exact otherwise it is unlikely to run. Don't forget the ; which terminate most lines nor the . after END.

5. Finally after the entering the "END." line press <CR> again. No further line numbers will be displayed, just the cursor A command entry is now awaited.

(b) TO EXECUTE THE PROGRAM

Type in

G <CR>	the "GO" command

This command runs the program. In our case you will obtain the display:

```
************
   SPRING
   SUMMER
   AUTUMN
   WINTER
************
```

In fact this line does not appear since we omitted it in our program - see later how we insert it.

(c) "LIST" COMMANDS

1. To list complete program: type in

P <CR>

The entire program will be displayed on the screen.

2. To list complete program on printer:

H <CR>

3. To list a specific line :

P <line number> <CR>	for screen
H <line number> < >	for printer

For example,

```
P3    <CR>
```

will display line 3, i.e. in our program

```
3.  WRITELN("    SPRING    ");
```

4. To list range of lines:

```
P <start line no.>-<end line no.>    <CR>
```

For example,

```
P2-5    <CR>
```

will display lines 2,3,4,5.

5. To list up to a specific line:

```
P-<specific line no.>    <CR>
```

For example,

```
P-4    <CR>
```

will display lines 0,1,2,3 and 4

(d) "DELETE" COMMANDS

Exactly similar to "list" commands but with D replacing P, i.e.

1. To delete one line:

```
D <line no.>    <CR>
```

2. To delete a group of lines:

```
D <start no.>-<end no.>  <CR>
```

3. To delete lines upto/after specific line:

D-<end no.> <CR>	D<start no.> <CR>

DELETES all lines upto/after specified line no. including also that line.

Examples

D3 <CR>	deletes line 3
D-12 <CR>	deletes lines 0,1...to 12
D65- <CR>	deletes lines from 65 upto the end of the program.

(e) "KILL" COMMAND

To erase the entire program, type in

K/ <CR>

(f) TO INSERT ADDITIONAL LINES IN PROGRAMS

1. To insert a single line:

To insert an additional line between lines (for example) 4 and 5 in our "seasons" program, type in:

```
5→ WRITELN("    AUTUMN    ");
```

line we wish to insert between 4 and 5 of old program

→ this symbol is obtained
for the K: BY PRESSING SHIFT + Z keys
for the A: BY PRESSING CTRL + Z keys

Check to see that the line is indeed inserted by using the "list" command, i.e.

P <CR>

```
0.%TO LIST SEASONS OF YEAR%
1.BEGIN
2.   WRITELN("************");
3.   WRITELN("    SPRING    ");
4.   WRITELN("    SUMMER    ");
5.   WRITELN("    AUTUMN    ");
6.   WRITELN("    WINTER    ");
7.   WRITELN("************")
8.END.
```

2. To insert more than one line:

   To insert a number of lines, for example, between lines 6 and 7, type in

7→ <CR>

   On pressing the carriage return subsequent line number (beginning at 7 in the above case) will be automatically dispayed-now carry on in with your entries.

3. To insert lines at beginning of program:

B <CR>

   (Note same command as to start entries).

4. To insert lines at end of program:

Z <CR>

(g) LOADING A PROGAM ON TAPE: APPEND COMMAND

To load a program contained on tape, type in

A <CR>

the LOAD command

FILENAME?▒-------

is displayed

type in here name of program followed by carriage return

↓PLAY	is displayed, asking us to press PLAY on cassette control
FOUND PAYOFF	is displayed when program found, followed immediately by
LOADING PAYOFF	when "ready" (with audible blip)

(h) SAVING A PROGRAM ON TAPE: SAVE COMMAND

```
S     <CR>
FILENAME?PAYOFF    <CR>
RECORD. PLAY
WRITING PAYOFF
```

that is, simply type in S followed by carriage return, FILENAME? is then displayed. Type in suitable name (as always, followed by carriage return) and your program will be stored on the tape cassette, when you action RECORD PLAY

1.4 GENERAL CORRECTIONS WITHIN THE PROGRAM

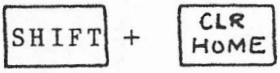
Use cursor control keys:  

and instant delete key: 

in exactly the same way as in correcting BASIC programs.

1.5 CLEARING THE DISPLAY COMPLETELY

To clear the display on the screen use



keys,

whilst to execute "clear display" as a program statement use statements of the form:

```
WRITE(" ");   WRITELN(" ");
```

1.6 SOME PASCAL PROGRAMS

Try entering and running these programs. They will provide you some practice in using the edit and other commands and also serve as an introduction to the form of program structure used in PASCAL.

1. A very simple program using the

   `WRITELN("*******");` statement.

   This statement is used to display on the screen the characters enclosed within the quotation marks.

```
 0.% NAME AND ADDRESS %
 1.BEGIN
 2.   WRITELN("**********************");
 3.   WRITELN("My name is:");
 4.   WRITELN("    PIERS STAPLETON");
 5.   WRITELN("I live at:");
 6.   WRITELN("    60 LYNTON SLOPE,");
 7.   WRITELN("       WEST ARTFORD,");
 8.   WRITELN("            YORKSHIRE");
 9.   WRITELN("**********************")
10.END.
```

```
G    <CR>
```


```
**********************
My name is:
   PIERS STAPLETON
I live at:
   60 LYNTON SLOPE,
      WEST ARTFORD,
           YORKSHIRE
**********************
```

We also use the comment statement:

% ..... % ;

This is used to add any comments we may wish to include in our program (e.g. what the program or what a certain section of lines does).
Comment statements are ignored by the computer when executing the program- they are to help our understanding.

2. To illustrate use of the WRITELN statement in both display and calculation.

```
0.% ARITHMETIC CALCULATIONS %
1.BEGIN
2.   WRITELN("4.12+5.36=",4.12+5.36);
3.   WRITELN("4.12-5.36=",4.12-5.36);
4.   WRITELN("4.12*5.36=",4.12*5.36);
5.   WRITELN("4.12/5.36=",4.12/5.36)
6.END.
```

```
G    <CR>
4.12+5.36=            9.48
4.12-5.36=           -1.24
4.12*5.36=         22.0832
4.12/5.36=      0.76865672
```

3. To "draw-a-line". A simple example of an inter-active program. The READLN statement asks us to input the LENGTH, the program draws the line.

```
0.% TO DRAW A LINE LENGTH 1 TO 40 UNITS %
1.VAR N,LENGTH:INTEGER;
2.BEGIN
3.   WRITE("Enter length of line");
4.   READLN(LENGTH);
5.     FOR N:=1 TO LENGTH DO
6.       WRITE("-")
7.END.
```

4. Finally a more complex program, Input your LOAN, the INTERESTRATE and REPAYMENT-the program shows how your loan is (or rather will be) paid off.

```
 0.% PAYOFF YOUR LOAN %
 1.VAR LOAN,INTERESTRATE,REPAYMENT:REAL;
 2.    MONTHNO:INTEGER;
 3.BEGIN
 4.   MONTHNO:=1;
 5.   WRITE("ENTER LOAN REQUIRED");
 6.   READLN(LOAN);
 7.   WRITE("INTEREST RATE %");
 8.   READLN(INTERESTRATE);
 9.   WRITE("AMOUNT OF REPAYMENT PER MONTH");
10.   READLN(REPAYMENT);
11.   WRITELN("MONTH   DEBT");
12.   WRITELN(" ");
13.     REPEAT
14.       LOAN:=LOAN*(1.0+INTERESTRATE/1200.0)
15.             -REPAYMENT;
16.       WRITELN(MONTHNO:4,LOAN:10:2);
17.       MONTHNO:=MONTHNO+1;
18.     UNTIL LOAN<=0.0;
19.   WRITELN(" ");
20.   WRITE("LOAN IS PAID OFF IN");
21.   WRITE(MONTHNO-1:3," MONTHS")
22.END.
```

CHAPTER 2

SOME PASCAL FUNDAMENTALS

VARIABLES, STATEMENTS AND PROGRAM STRUCTURE

2.1 INTRODUCTION

In this chapter we deal with some fundamentals concerning the definitions of terms, the rules and structure of PASCAL, and how we apply these to write programs.

A PASCAL program may be considered essentially as a series of statements-written according to the "grammar" or syntax of PASCAL-which the computer subsequently executes step by step. If the program does not follow these rules exactly it will not run and you will be told why-the computer automatically outputs an error statement (see list of errors in Appendix*). Thus it is essential to gain at least a basic understanding of the PASCAL syntax to enable you to begin to write your own "error-free" programs.

2.2 VARIABLES

The calculations and operations made in a PASCAL program are applied to variables. Each variable can be thought as a container or box to which can be assigned a given value. Once a value has been placed in a variable it remains there but may be altered when (or if) any subsequent program statements assign the variable a new value.

*Appendix of Sharp Pascal MZ-80 Manuel (pages 132-133)

In PASCAL there are four different types of variables:

INTEGER	(whole numbers, e.g. 0, 10, -87, 3059)
REAL	(decimal numbers, e.g. 0.0, 10.0, -12.76, 999.8)
CHAR	(characters, e.g. 'A','B','1','2','?',' ')
BOOLEAN	(used in decision making, Boolean variables can only take one of two values: TRUE or FALSE).

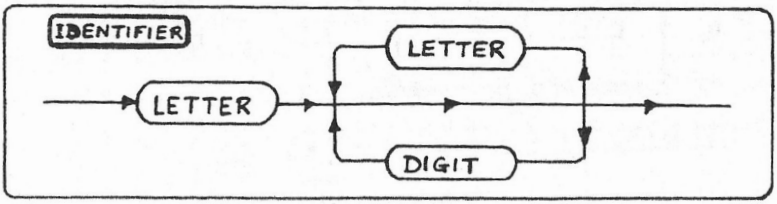
2.3 IDENTIFIERS

Variables (and, as we see later, procedures and functions) are all represented by identifiers. An identifier is the name we assign to a variable.

In PASCAL all identifiers must begin with a letter. They can then be followed by any sequence of letters or numbers.

You should always try to choose meaningful identifiers for the variables in your program. This helps in understanding what your program is about and does- especially useful for future reference and for other users.

The syntax diagram for an identifier is shown below.



Syntax diagrams will be used extensively in our explanations. They provide an easy pictorial means of illustrating PASCAL syntax-just follow the arrow directions-provided you always go in their direction you can pass round a loop as many times as you like.

Examples of legal identifiers:

```
NUMBER, JKFLIPFLOP, X11, A1X2, HIGHTEMP
```

Not allowed:

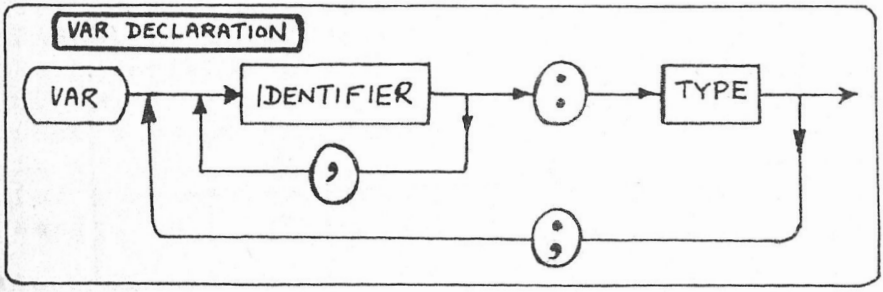
```
1NUMBER (starts with "1", should be a letter)
X+Y ("+" sign must not be used, only letters
     or numbers)
MIN SPEED ("space" used).
```

2.4 VARIABLE DECLARATION

In PASCAL we create variables by what is known as a declaration: the VAR declaration. This is normally done close to the beginning of our program.

In declaring variables we must specify both their identifiers (names) and their types (i.e. whether REAL, INTEGER, CHAR, or BOOLEAN).

The syntax diagram for the VAR declaration is given below:



Examples.

1. Suppose the identifiers for REAL variables we wish to use in a program are:

   VOLUME, SURFACEAREA, RADIUS, PI

   These are declared at the beginning of the program as follows:

```
VAR VOLUME,SURFACEAREA,RADIUS,PI:REAL;
```

at least one space between VAR and first identifier

each variable separated by comma

colon before defining type of variable

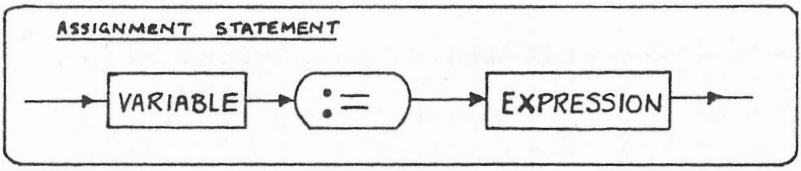
semicolon at end of each type in declaration.

2. If variables of more than one type are used they are declared as shown in the example below:

```
VAR LOAN,INVESTMENT,RATE:REAL;
    MONTH,YEAR:INTEGER;
    RED,QUESTION,ANSWER:CHAR;
    LLOYDS,BURNLEYBSOC,
    NATSAVE,PAIDOFF,INDEBT:BOOLEAN;
```

2.5 ASSIGNMENT STATEMENTS

An assignment statement "assigns" a value to a variable. The general form of assignment statement is given in the syntax diagram below:



(:=) means "is assigned" or "takes the value"

execute := by pressing : followed by = Keys

Some examples:

1. `X:= 5.2;` means the REAL variable identified by X is assigned the value 5.2

Note the statement is not a mathematical identity. The statement says:
"whatever value the variable had (if any was previously given) now give it 5.2"

2. `DAYOFWEEK:= 7;` means the INTEGER variable identified by DAYOFWEEK is given the value 7

3. `APPLE:='A';` means the CHAR variable identified by APPLE is assigned the value A

Note for character variables we can only assign a single character as its value and this character must be contained within the single quotation marks ' '.

4. `HITEMP:= T>=100;` means the BOOLEAN variable identified by HITEMP is assigned the value TRUE when the integer variable T is greater than or equal to 100 and FALSE when T is less than 100.

2.6 VALUES AND EXPRESSIONS

In the assignment statement a value or more generally an expression is assigned to a variable. We will be dealing with
REAL expressions, e.g. X-Y, X*Y , (X+Y)/34.62,
INTEGER expressions, e.g. A DIV B, A MOD B
BOOLEAN expressions, e.g. P AND Q, NOT(P OR Q)
in later chapters. No similar operations are available to perform "calculations" on CHAR

values, although they may be compared and used in READ and WRITE statements-again to be considered later.
However, at this stage it is worth while introducing some idea of expressions and now clarifying some important points concerning REAL, INTEGER, CHAR and BOOLEAN values.

(a) INTEGER AND REAL NUMBER VALUES

PASCAL distinguishes between INTEGERS (+ or - whole numbers) and REAL numbers (numbers with a decimal part even if this is zero, e.g. 10.0 MUST BE SPECIFIED in this way to represent the REAL value 10.

Examples of INTEGER values:
0 10 543 -5000 -43 +12000

Examples of REAL values:
0.0 12.78 -1000.0 +999.9
4.5E6 (= 4.5x1000000 or 4.5x10)
5E-3 (=5x0.001=0.005 or 5x10)

(b) CHARACTER VALUES

A character variable may be assigned the value of any character available on the keyboard,which include:

1. Upper and lower case letters
   A,B,C,D....X,Y,Z a,b,c.....x,y,z
2. The ten denary digits
   0,1,2,3,4,5,6,7,8,9
3. Punctuation marks
   ! , . ; ? :
4. Space and graphical symbols

Remember, the character value-known as the character constant- must always be entered in between the single quotation marks in the assignment statement, e.g.

```
QUESTIONMARK:='?'; SPACE:=' ';
```

(c) BOOLEAN VALUES

Boolean variables can only be assigned one of the two values: TRUE or FALSE. For example, if A and B are declared as BOOLEAN variables they may be assigned:

```
A:=TRUE;  B:=FALSE;
```

Boolean values very often arise from the result of a comparison. For example,

```
YOUNGAGE:=AGE<18
```

assigns the BOOLEAN variable the value TRUE when the integer variable AGE has the value of less than 18.

2.7 WRITE AND WRITELN STATEMENTS

The WRITE and WRITELN statements are the basic output display statements of PASCAL. They correspond to the PRINT statements used in BASIC.

(a) The WRITE statement

The WRITE statement is used
(i) to effect calculations and display the results
(ii) to display character strings, i.e. the characters enclosed within double quotation marks.

To effect and display the result of a calculation:

```
WRITE(<calculation expression>);
```

To display a character string:

```
WRITE("<character string>");
```

For example:

```
0.% SIMPLE CALCULATION%
1.%ILLUSTRATING USE OF WRITE STATEMENT%
2.VAR X,Y:REAL;
3.BEGIN
4.   X:=12.4;Y:=4.87;
5.   WRITE("X+Y=",X+Y)
6.END.
```

```
G        <CR>
```
← The "GO" command

```
X+Y=          17.27
```
← Result

(b) The WRITELN statement

The WRITELN statement performs the same function as the WRITE statement but in addition makes a carriage return after the statement has been obeyed. Any subsequent output will then be started on a new line. The WRITE statement does not make this carriage return.
The use of the WRITE and WRITELN statements is illustrated in the following program:

```
0.VAR X,Y:REAL;
1.BEGIN
2.   X:=98.0;Y:=56.7;
3.   WRITE("THE ANSWER TO X-Y IS");
4.   WRITELN(X-Y);
5.   WRITELN(" ");
6.   WRITE("THE ANSWER TO X/Y IS");
7.   WRITELN(X/Y)
8.END.
```

```
G        <CR>
```

```
THE ANSWER TO X-Y IS 41.3
THE ANSWER TO X/Y IS 1.728395
```

(c) The PWRITE and PWRITELN statements

These have the same action as WRITE and WRITELN but output the results, character strings..etc. to the printer.

(d) General comments on use of WRITE and WRITELN for display and cursor movement

```
WRITELN(" ");
```
produces a line feed

```
WRITE("■");
```
```
WRITELN("■");
```

use SHIFT + CLR HOME keys

used to clear screen completely

```
WRITE("■");
```
cursor is shifted to top left hand corner of screen without clearing display

```
WRITE("■");        WRITE("■");
WRITE("■");        WRITE("■");
```

used to shift cursor respectively
UP, DOWN, TO RIGHT, TO LEFT

2.8 FORMATTING THE WRITE AND WRITELN OUTPUTS

Formatting the output is very useful especially when the output display or copy is to be tabulated and also when the exact number of output digits or number of decimal places are to be controlled.
This may be accomplished by following each item in the WRITE or WRITELN statements by formatting information consisting of a comma, and one or two positive integer values separated by colons.

1. WRITE(A,B,C); and WRITELN(A,B,C);

These statements display the value of each vaiable or expression so the least significant digit (for REAL and INTEGER items) or character (for CHAR values) is 15 SPACES to the right of the current cursor position. For example:

```
0.VAR N1,N2:INTEGER;
1.BEGIN
2.N1:=3438;N2:=9562;
3.WRITE(N1,N2)
4.END.
```

```
G    <CR>
```

← The "GO" command

```
        3438                    9562
```

← 15 spaces → ← 15 spaces →

```
0.VAR LETTERA,LETTERZ:CHAR;
1.BEGIN
2.LETTERA:='A';LETTERZ:='Z';
3.WRITELN(LETTERA,LETTERZ)
4.END.
```

```
G    <CR>
```

```
        A                       Z
```

← 15 spaces → ← 15 spaces →

2. WRITE(<expression>:N);
and WRITELN(<expression:N);

These statements display the value of the expression so the least significant digit or character (in the case of a CHAR variable) is N (N=1,2,3...etc.) to the right of the current cursor position.
For example:

```
0.VAR A,B:CHAR;
1.BEGIN
2.A:='A';B:='B';
3.WRITELN(A:10,B:10)
4.END.
```

G <CR>

```
         A         B
```

← 10 spaces → ← 10 spaces →

```
0.VAR X,Y:REAL;
1.BEGIN
2.X:=23.54;Y:=9.834;
3.WRITELN(X:8,Y:8,X*Y:12)
4.END.
```

G <CR>

```
  23.54    9.834    231.49236
```

← 8 spaces → ← 8 sp. → ← 12 spaces →

3. WRITE(<expression:N:P>);
and WRITELN(<expression>:N:P>);

These statements display the value of the variable or the result of the expression so the least significant digit is N spaces to the right of the current cursor position whilst the second integer P specifies the number of decimal places given in the result. These form of statements are valid only for REAL values.
Examples:

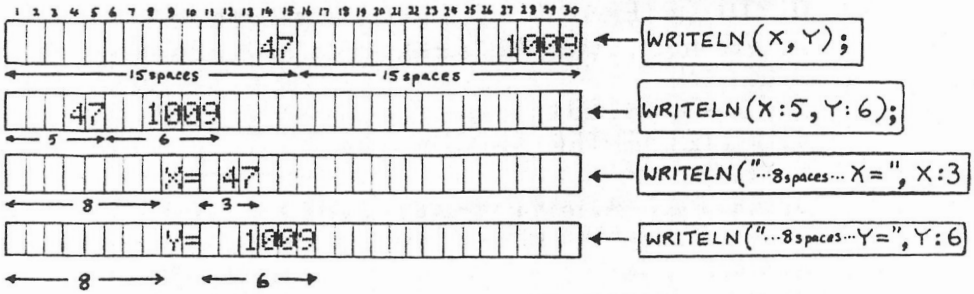
```
0.VAR X,Y:REAL;
1.BEGIN
2.X:=34.872;Y:=-6.683;
3.WRITELN(X:6:2,Y:8:2,X/Y:10:4)
4.END.
```

G <CR>

```
34.87   -6.68   -5.2180
```

```
 0.VAR X,Y:INTEGER;
 1.BEGIN
 2.X:=47;Y:=1009;
 3.WRITELN(X,Y);
 4.WRITELN(" ");
 5.WRITELN(X:5,Y:6);
 6.WRITELN(" ");
 7.WRITELN("          X=",X:3);
 8.WRITELN(" ");
 9.WRITELN("          Y=",Y:6)
10.END.
```

```
G    <CR
```



2.9 READ AND READLN STATEMENTS

The READ and READLN statements are the basic PASCAL statements requesting the input of data. They correspond to the INPUT statement used in BASIC.
REAL, INTEGER and CHAR values can be "READ", BOOLEAN values cannot.

(a) READ(<identifier>); and READLN(<identifier>);

READ(A); and READLN(A);

When this form of statement is executed a ? is displayed on the screen and an input of data (corresponding to the value of the identifier, A in our example) is awaited from the keyboard.
When the value is keyed in and carriage return pressed the program continues

The difference between READ and READLN is READLN makes a carriage return after the input (e.g. if the next program statement were WRITE(...); the output would commence on a new line), the READ does not make a carriage return.

Example. This program uses READLN statement (line 5) to request input of radius and then calculates and displays volume of sphere.

```
0.%TO DETERMINE VOLUME OF A SPHERE%
1.VAR R,PI,VOLUME:REAL;
2.BEGIN
3.PI:=3.14159;
4.WRITE("ENTER RADIUS ");
5.READLN(R);
6.VOLUME:=4.0*PI*R*R*R/3.0;
7.WRITELN("VOLUME= ",VOLUME:8:3)
8.END.
```

```
G        <CR>
ENTER RADIUS ? 5.62      <CR>
                    <-- VALUE OR RADIUS ENTERED FROM KEYBOARD

VOLUME = 743.528
                    <-- DISPLAY OF RESULT
```

(b) READ(A,B,C); and READLN(A,B,C);

When this form of statement is executed the first ? displayed awaits the value of A to be entered. Enter the value followed by a comma (or alternatively a carriage return).
A second ? is then displayed. Enter value of B followed by a comma (or <CR>)
The third ? displayed awaits value of C. Enter value of C and press <CR>.
Program execution then continues.

There is a practical limit to the number of characters that can be inputted using the READ(A,B,C,D...) statement. In our case it is not more than two lines (80 characters including the ? prompts).

The READLN statement overcomes this difficulty by making a carriage return after the last value in the READLN brackets has been entered. For example,

```
READ(A,B,C,D,E,F,G,H,I)
```

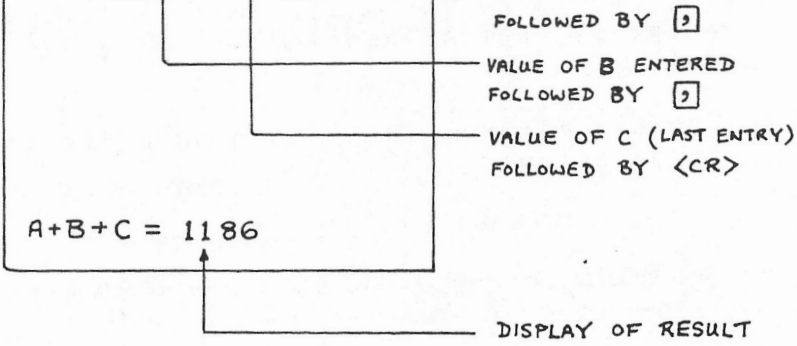
could easily exceed 80 charactres of data and is better programmed, for example, as

```
READLN(A,B,C,D);
READ(E,F,G,H,I);
```

Example

```
0.VAR A,B,C:INTEGER;
1.BEGIN
2.READLN(A,B,C);
3.WRITELN("A+B+C=",A+B+C:6)
4.END.
```

```
G        <CR>
? 365,487,334   <CR>
A+B+C =  1186
```



2.10 COMPOUND STATEMENTS

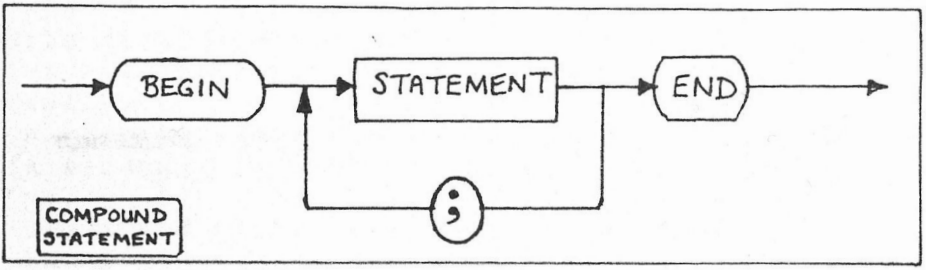
In PASCAL there are basically two types of statements:

simple statements...statements that cannot be grammatically divided; an assignment statement is an example of a simple statement

compound statements...statements which consist of a number of simple statements preceded by BEGIN and terminated by END

The executable sections of a PASCAL program normally consist of a combination of simple and compound statements.
The main objective of a compound statement is to cause a sequence of simple statements to be bracketed together and treated as a single statement for syntax purposes. This is done within the program by bracketing the required sequence of statements between BEGIN and END. It is good practice to indent a compound statement by starting each line making up the compound statement with 2or more spaces, although normally not more than 5.

The syntax diagram for a compond statement is given below:



Example.

The following program, which finds the average of any number of quantities entered via the keyboard.
The program contains a number of simple statements, e.g. the assignement statements on line 3; the WRITE and WRITELN statements on lines 4,6,7 and 15; and the READLN statement on line 5.
The WHILE...DO... statement (line 8) is one form of repetitive statement used in PASCAL and will be considered in detail in Chapter 4. In our example: "while the number entered is not equal to -9999.0", the COMPOUND statement from line 9 to 14 is obeyed. When the last value has been entered, the repetition is stopped by typing in -9999.0. The AVERAGE is then outputted by the WRITELN statement of line 15.

```
 0.% TO FIND AVERAGE%
 1.VAR SUM,NUMBER,N:REAL;
 2.BEGIN
 3.   N:=0.0;SUM:=0.0;
 4.   WRITE("ENTER FIRST NUMBER");
 5.   READLN(NUMBER);
 6.   WRITELN("*** AFTER LAST NUMBER ***");
 7.   WRITELN("*** TYPE IN -9999.0 ***");
 8.     WHILE NUMBER<>-9999.0 DO
 9.       BEGIN
10.         SUM:=SUM+NUMBER;
11.         N:=N+1.0;
12.         WRITE("ENTER NEXT NUMBER");
13.         READLN(NUMBER)
14.       END;
15.   WRITELN("AVERAGE= ",SUM/N:8:2)
16.END.
```

2.11 PASCAL PROGRAM STRUCTURE

By now you will be fairly familiar with the form PASCAL programs take. All PASCAL progams have a basic structure and an order that must be followed. The order for programs when using the SHARP PASCAL INTERPRETER SP-4015 (when no procedures or functions are declared-we consider these in Chapter 5) is as follows:

1. VAR section (variable declaration section)

   All VARIABLES used in your program are declared right at the beginning.
   A unique IDENTIFIER must be allocated to each variable. Identifiers of the same type are separated by a comma and their TYPE (REAL, INTEGER, CHAR, BOOLEAN) stated at the end of each type group, preceded by : and terminated by ;
   e.g.
```
VAR INCOME,EXPENSES:REAL;
    NOOFYEARS:INTEGER;
    QUESTIONMARK:CHAR;
    SHORT,TALL,OVERWEIGHT:BOOLEAN;
```

2. PROGRAM STATEMENTS

   After the VAR section we write the statements used to solve our problem, the executable statements. This section is always commenced with BEGIN
   The various simple and compound statements then follow in the order they are to be executed. Each individual statement must always be terminated by a ;
   except the very last one or the last statement of a compound statement (in this case the bracketing END is terminated in ; i.e. END;

   All progams must be "ended" by typing in END. (do not forget the full-stop after the END.

CHAPTER 3

PASCAL EXPRESSIONS AND STANDARD FUNCTIONS

3.1 INTRODUCTION

In this chapter we consider

* REAL and INTEGER arithmetic and comparison expressions and operations, e.g.+,-,*,/ DIV,MOD and <,>,=...etc. used, for example, in making calculations and comparing data.
* BOOLEAN expressions and operators, e.g. AND,OR,NOT,XOR used for making decisions.
* STANDARD FUNCTIONS availiable in PASCAL for performing prescribed tasks, e.g. SQRT(X),SIN(X),ARCTAN(X),RND(X),ODD(X), CHR(X),TRUNC(X),FLOAT(X),ABS(X)...etc.

3.2 REAL EXPRESSIONS: ARITHMETIC OPERATORS

The following are used with REAL data and variables:

ADDITION	+ key	e.g. W:=A+B;
SUBTRACTION	- key	e.g. X:=A-B;
MULTIPLICATION	* key	e.g. Y:=A*B;
DIVISION	/ key	e.g. Z:=A/B;

The usual rules of precedence apply, i.e. * and / before + and -.
Brackets may also be used in the normal way
e.g. G:=(A-B)/(A+B); H:=23.0*(2.5-5.6*A/B);

NOTE. In SHARP SP-4015 PASCAL REAL and INTEGER values and variables cannot be mixed. Thus, although +,-,* operators exist also for INTEGER variables the following expressions, for example, will lead to an ERROR message in program execution:

A+N A-N A*N

if variable A is REAL and N is INTEGER
It should be noted, however, in "standard" PASCAL INTEGER values may be used in a REAL expression without qualification. The integer value is automatically converted to the corresponding real value in program execution.

Examples

The following programs illustrate REAL expressions in some simple calculations. We use the WRITE and WRITELN statements to make the calculations and also fomat the results. COMMENT and READLN statements are also employed.

```
0.VAR A,B:REAL;
1.BEGIN
2.   A:=42.0;B:=15.0;
3.   WRITELN("A+B=",A+B:5:1);
4.   WRITELN("A-B=",A-B:5:1);
5.   WRITELN("A*B=",A*B:5);
6.   WRITELN("A/B=",A/B:5:1);
7.   WRITE("(A-B)/(A+B)=");
8.   WRITELN((A-B)/(A+B):5:3)
9.END.
```

```
G        <CR>
```

← The "GO" command

```
A+B= 57.0
A-B= 27.0
A*B=  630
A/B=  2.8
(A-B)/(A+B)= .473
```

← Results

```
  0.% TO DETERMINE RESISTANCE R %
  1.% OF R1 AND R2 IN PARALLEL  %
  2.VAR R,R1,R2:REAL;
  3.BEGIN
  4.   WRITE("ENTER R1 VALUE ");
  5.   READLN(R1);
  6.   WRITE("ENTER R2 VALUE ");
  7.   READLN(R2);
  8.   R:=(R1*R2)/(R1+R2);
  9.   WRITELN("****************");
 10.   WRITELN("R=",R:8:2);
 11.   WRITELN("****************")
 12.END.
```

```
G      <CR>
ENTER R1 VALUE ? 30
ENTER R2 VALUE ? 20
****************
R=    12.00
****************
```

30 entered by us

20 entered

Result

3.3 INTEGER EXPRESSIONS: ARITHMETIC OPERATORS

We have already noted that PASCAL distinguishes between REAL and INTEGER variables and data. There are also certain distinctions in dealing with arithmetic operations.
The addition, subtraction and multiplication operators + - * are identical for both REAL and INTEGER variables. Division, however, is different. The / is not used for INTEGER variables and data.

The DIV and MOD operators are used:

The DIV operator performs DIVISION WITH TRUNCATION, e.g.

X:=25 DIV 7 assigns X the value 3
i.e. 25 divided by 7 to the nearest whole number rounded downwards.

The MOD operator provides the REMAINDER,e.g.

XR:=25 MOD 7 assigns XR the value 4
i.e. 25-7=3 with a remainder of 4

Y:=100 DIV 20 assigns Y the value 5
YR:=100 MOD 20 assigns YR the value 0

Examples

```
0.VAR C,D:INTEGER;
1.BEGIN
2.   C:=42;D:=15;
3.   WRITELN("C+D=",C+D:7);
4.   WRITELN("C-D=",C-D:7);
5.   WRITELN("C*D=",C*D:7);
6.   WRITELN("C DIV D=",C DIV D:3);
7.   WRITELN("C MOD D=",C MOD D:3)
8.END.
```

```
G      <CR
```

```
C+D=      57
C-D=      27
C*D=     630
C DIV D=   2
C MOD D=  12
```

This program illustrates the use of the MOD operator in finding whether a number has a given factor, 7 in our example. We use the IF...THEN...ELSE statement (considered in the next chapter in more detail) to make the decision.

```
0.VAR N:INTEGER;
1.BEGIN
2.   WRITE("ENTER NUMBER ");
3.   READLN(N);
4.      IF N MOD 7=0 THEN
5.      WRITELN("7 IS A FACTOR OF",N:6)
6.      ELSE
7.      WRITELN("7 IS NOT A FACTOR OF",N:6)
8.END.
```

3.4 COMPARISON OPERATORS

The operators =,<>,<,>,<= and>= are used in comparing two data values. They can be used with REAL, INTEGER and CHAR data but not in a mixture, i.e. both members in a comparison expression must be of the same type. Comparison expressions always give a Boolean result, i.e. either TRUE or FALSE.

The comparison operators have the following meaning:

= equality; e.g. the expression A=B checks whether the left hand term A is equal to the right hand term B

<> inequality;e.g. A<>B checks whether A is not equal to B

<= less than or equal to; e.g. A<=B

>= greater than or equal to; e.g.A>=B

< less than; e.g. A<B checks whether A is less than B

> greater than; e.g. A>B checks whether A is greater than B

In the case of CHAR values you may wonder what meaning is given to their comparison. It is assumed, for example, that the alphabetical characters follow an increasing order so A<B, F>C ...etc.
The comparison operators must be used with care with REAL data. REAL values are stored in the computer to a limited number of significant figures (typically 7 to 8 in our case). Thus two real numbers cannot in general always be safely compared for equality, e.g. A=2.34178 and B=2.34172 would not be regarded as equal in the expression A=B.

Comparison expressions are extensively used in conjunction with the control type statements considered in the next chapter. The examples given below employ one of these,
the IF (...comparison expression...) THEN statement. The comparison expression result, i.e. TRUE or FALSE , is used to determine the course of action.

Examples

This simple program shows how CHAR data can be compared.

```
0.VAR A,B,C,X:CHAR;
1.BEGIN
2.  A:='A';B:='B';C:='C';
3.  READLN(X);
4.    IF X=A THEN WRITELN("1.",X:5);
5.    IF X>B THEN WRITELN("2.",X:5);
6.    IF X<C THEN WRITELN("3.",X:5)
7.END.
```

This program uses a comparison expression and the IF...THEN...ELSE statement to test whether a runner has reached a qualifying time of 24.0 seconds for, say, 200 metres.

```
 0.VAR TIME:REAL;
 1.BEGIN
 2.  WRITELN("ENTER RUNNER'S TIME");
 3.  READLN(TIME);
 4.    IF TIME<=24.0 THEN
 5.    BEGIN
 6.      WRITE("TIME IS ",24.0-TIME:4:1," SECS.");
 7.      WRITELN(" INSIDE QUALIFYING TIME")
 8.    END
 9.    ELSE
10.    BEGIN
11.      WRITE("TIME IS ",TIME-24.0:4:1," SECS.");
12.      WRITELN(" OUTSIDE QUALIFYING TIME")
13.    END;
14.END.
```

3.5 BOOLEAN OPERATORS AND EXPRESSIONS

Boolean expressions are used essentially for making decisions within a program. A Boolean expression can take only one of two values: TRUE or FALSE. In addition to the comparison operators, four Boolean or logic operators:

NOT, AND, OR, XOR

are used to create Boolean expressions. Their meaning is explained below:

NOT the logical NOT or logical negation
e.g. NOT A is TRUE if A is FALSE
is FALSE if A is TRUE.

AND the logical AND
e.g. A AND B is TRUE if and only if A and B are both TRUE; if A and/or B are FALSE, A AND B is assigned a FALSE value.

OR the logical OR
e.g. A OR B is TRUE if either or both A and B are TRUE; if A and B are both false A OR B is assigned a FALSE value.

XOR the logical EXCLUSIVE OR
e.g. A XOR B is TRUE if either A or B is TRUE; if A and B are both TRUE or both FALSE A XOR B is FALSE.

PREECEDENCE: the order of precedence of these operators in evaluating Boolean expressions is as follows:

highest NOT
AND
OR, XOR
=,<>,<=,>=,<,>

A simple Boolean expression consists of a series of Boolean values separated by AND, OR, XOR or preceded by NOT,

e.g. suppose A,B,C,D are declared as BOOLEAN variables and assigned either TRUE or FALSE values, then

A AND B AND C AND D is TRUE if and only if A,B,C,D are all assigned TRUE values

A AND NOT D is TRUE if A is TRUE and D is FALSE

A OR C OR D is TRUE if one or more of the variables is assigned TRUE

C XOR D is TRUE if either C or D is TRUE, otherwise it is FALSE.

When it is required to combine comparison expressions or indicate precedence brackets must be used,
e.g. if X and Y are INTEGER variables, then the Boolean expression:
(X>10) OR (Y<=100)
is assigned a TRUE value if the value of X is greater than 10 OR the value of Y is less than or equal to 100.
(Note the comparison expressions must be enclosed in brackets).

Examples

```
 0.% SIMULATION OF THERMOSTAT CONTROL %
 1.VAR LOWTEMP,HITEMP,TEMP:REAL;
 2.BEGIN
 3.   LOWTEMP:=15.0;HITEMP:=25.0;
 4.   READLN(TEMP);
 5.      IF TEMP<=LOWTEMP THEN
 6.     WRITELN("SWITCH ON HEATER+BOOST");
 7.     IF(TEMP>LOWTEMP)AND(TEMP<HITEMP)
 8.     THEN WRITELN("HEATER ON, SWITCH OFF BOOST");
 9.     IF TEMP>=HITEMP THEN
10.      BEGIN
11.      WRITE("SWITCH OFF HEATER, ");
12.      WRITELN("SWITCH ON FAN")
13.      END;
14.END.
```

This program simulates a simple thermostat control where a heater and/or boost and/or fan is turned on or off. It employs comparison operators, the logical AND and IF...THEN statements.

```
 0.% OPEN THE SAFE ! %
 1.VAR TIME:REAL;
 2.    COMBINATION:INTEGER;
 3.    UNLOCKAM,UNLOCKPM:BOOLEAN;
 4.BEGIN
 5.  WRITE("ENTER TIME ");
 6.  READLN(TIME);
 7.  WRITE("ENTER COMBINATION");
 8.  READLN(COMBINATION);
 9.  UNLOCKAM:=(TIME>8.0)AND(TIME<12.0)
10.             AND(COMBINATION=12332);
11.  UNLOCKPM:=(TIME>14.0)AND(TIME<16.3)
12.             AND(COMBINATION=21472);
13.    IF UNLOCKAM OR UNLOCKPM THEN
14.    WRITE("SAFE IS NOW OPEN")
15.    ELSE
16.    WRITE("SOUND THE ALARM")
17.END.
```

This program uses the logical OR to create a BOOLEAN variable "UNLOCK". When the value of this variable is TRUE the safe is opened if FALSE an alarm is sounded.

```
 0.VAR A,B,C,D,E,F:INTEGER;
 1.    SA,SB,SC,SD,SE,SF,Q:BOOLEAN;
 2.BEGIN
 3.  READLN(A,B,C,D,E,F);
 4.  SA:=A=1;SB:=B=1;SC:=C=1;
 5.  SD:=D=1;SE:=E=1;SF:=F=1;
 6.  Q:=SA AND SB AND SC AND(SD OR SE)AND NOT SF;
 7.   IF Q=TRUE THEN
 8.   WRITELN("RUN PROCESS")
 9.   ELSE WRITELN("CONDITS. NOT CORRECT")
10.END.
```

This program can be regarded as a simple example of a control program for an industrial process governed by the following conditions:
the process runs if
A the start button is "on"
AND B the flow of material is sufficient
AND C the temperature is high enough
AND D OR E either one or other of two other conditions are O.K.
AND NOT F the emergency stop button is not pressed
In the program execution these conditions are first "read", i.e. the INTEGER variables A,B,C,D,E,F simulate the process input information to the computer. We then define corresponding BOOLEAN variables SA,SB,SC,SD,SE,SF which take either TRUE or FALSE values on the basis of this input information, i.e. if A=1 then SA is assigned a TRUE value, if A=0 then SA is FALSE
...and so on.
The process runs if the "control" BOOLEAN variable Q (containing all the necessary requirements to be satisfied) has a TRUE value. Otherwise a warning is given that the run conditions are incorrect.

Try executing the program with various combinations of 1-0 values for A,B,C,D,E,F You will find that the process only "runs" when A=B=C=1, D and/or E=1, F=0.

3.6 STANDARD FUNCTIONS

Several standard functions are provided in PASCAL for performing prescribed tasks on REAL, INTEGER and CHAR DATA.
We describe the form and meaning of these in the following sections.

3.7 MATHEMATICAL FUNCTIONS

1. SQRT(X)

   This gives the square root of X for X REAL
e.g. A:=SQRT(81.0) assigns 9.0 to the
   REAL variable A

2. SIN(X)

   This gives the sine of X where X is REAL
   and expressed in RADIANS. Remember
   X(radians)=X(degrees)x /180
e.g. to find sin(42):
   A:=SIN(42.0*3.1415927/180.0)

3. COS(X)

   This gives cosine of X where X is REAL
   and assumed to be in radians.
e.g. A:=COS(45.3*3.1415927/180.0)
   assigns to the REAL variable
   A the value cos(45.3)

4. TAN(X)

   This gives tan(X), X must be REAL and
   is in radians.
e.g. TAN(56.0*3.1415927/180.0) gives tan(56)

5. ARCTAN(X)

   This gives the angle whose tan is X,
   the result being specified in radians
   between - /2 and + /2.X must be REAL;
e.g. A:= ARCTAN(1.0) assignes to A the
   value 0.7853982 radians,
   i.e. /2 or 45 since tan(45)=1.0

6. EXP(X)

   This gives e , the exponential function.
   X must be REAL.

7. LN(X)

This gives ln(X), the natural logarithm.
X must be REAL and greater than 0.0.

8. LOG(X)

This gives log(X) to base 10.
X must be REAL and greater than 0.0

9. ABS(X)

This gives the absolute value of X.
X may be either REAL or INTEGER;
e.g. A:=ABS(-89.3) assigns A the value 89.3
B:=ABS(-457) assigns B the value 457

10. TRUNC(X)

The TRUNC function converts the REAL value X to the nearest INTEGER value to X rounded "downwards";
e.g. A:=TRUNC(4.8) assigns A the value 4
B:=TRUNC(-7.8) assigns B the value -7
Note X must be REAL and TRUNC(X) gives an INTEGER value result.

11. FLOAT(X)

The FLOAT function converts an INTEGER value to a REAL value;
e.g. C:=FLOAT(20) assigns C the REAL value 20.0
D:=FLOAT(-8) assigns D the REAL value -8.0
Note X must be INTEGER and FLOAT(X) gives a REAL result.

12. ODD(X)

The parameter X must be INTEGER and ODD(X) gives a Boolean result, TRUE if X is odd, FALSE if X is even;
e.g. P:=ODD(5) assigns the BOOLEAN variable P the value TRUE
Q:=ODD(6) FALSE is assigned to Q

13. RND(X)

This function generates psuedo-random numbers. X must be REAL.
When X is greater than 0.0, then RND(X) gives the "random" number next to the one previously given in the group; when X is 0.0 or negative, RND(X) gives the initial value of the group.

3.8 STANDARD FUNCTIONS INVOLVING CHARACTERS

1. CHR(X)

X must be INTEGER and the character function CHR(X) gives the character value whose code is the number X.

X corresponds to the denary number equivalent to the ASCII code for the character.

e.g. A:=CHR(75) assigns to CHAR variable A the character K whose code is 75

Program example:

Note 63 is the code for ?
88 is the code for X;
the program displays X ?

```
0.VAR QUESTIONMARK,LETTERX:CHAR;
1.BEGIN
2.   QUESTIONMARK:=CHR(63);
3.   LETTERX:=CHR(88);
4.   WRITELN(LETTERX:3,QUESTIONMARK:3);
5.END.
```

```
G      <CR>
```
← THE "GO" COMMAND

```
X  ?
```
← DISPLAY

2. ORD(X)

X in this case must be a CHAR value. The order function ORD(X) gives the INTEGER value to the code for X;

e.g. if X:='?' then N:=ORD(X) gives 63, the code for the character ?
Note also the CHR and ORD are inverse functions.

3. PRED(X)

X must be a CHAR value. The predecessor function PRED(X) gives the character which has the code specified by X, minus 1;

e.g. if X:='M' then A:=PRED(X) gives L,
B:=PRED('Z') gives B the value Y

4. SUCC(X)

X must be a CHAR value. The successor function SUCC(X) gives the character which has the code specified by X, plus 1;

e.g. C:=SUCC('A') assigns C the value B
Note the PRED and SUCC are inverse functions.

CHAPTER 4
CONTROL STATEMENTS: CHOICE, SELECTION AND REPETITION

4.1 INTRODUCTION

PASCAL provides directly for three important requirements that are frequently needed in the solution of problems:

*** CHOICE of one or other course of action
*** SELECTION of one of many courses of action
*** REPETITION of a section of program.

In this Chapter we consider the PASCAL statements used to exercise these types of control. We explain and give program examples for the following "contol constructs":

CHOICE: the IF...THEN
and IF...THEN...ELSE statements.

SELECTION: the CASE statement.

REPETITION:
(1) WHILE some condition is satisfied, the WHILE...DO statement.
(2) UNTIL some codition is satisfied, the REPEAT...UNTIL statement
(3) FOR a given number of times, the FOR...TO...DO statement.

4.2 THE IF STATEMENTS

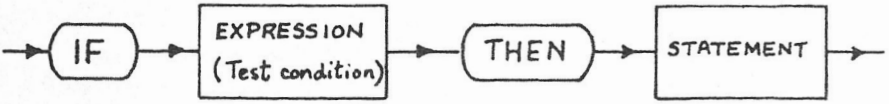
In PASCAL choice of one of two different courses of action can be made using the

IF... THEN

and IF... THEN... ELSE statements.

The syntax diagrams for these IF statements are given below.

IF statement (without ELSE):

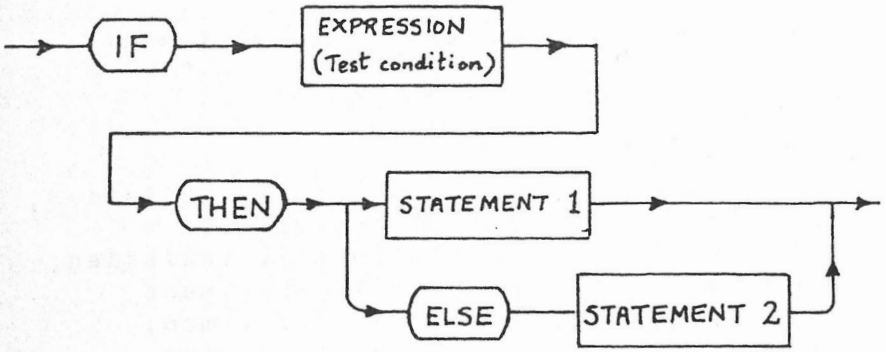


Note: 1. The expression following IF is a test condition, i.e. a Boolean type of expression.

2. The statement following THEN is "then" executed if the test condition yields a TRUE value.

3. If the test expression gives a FALSE value the statement is not executed and execution of the subsequent sections of the program is continued.

IF statement (with ELSE):



Note: 1. IF the test expression is satisfied, i.e. yields a TRUE value, THEN statement 1 is executed.
2. ELSE (if the test expression yields a FALSE value) statement 2 is executed.

The IF...THEN...ELSE statement enables alternative courses of action to be specified in a single statement. If either action consists of a number of statements, use BEGIN and END to bracket these into a compound statement.

Examples.

This program "tests" whether a candidate has passed or failed an exam. Enter "personsmark", the program displays "pass" or "fail".

```
 0.VAR PERSONSMARK,PASSMARK:INTEGER;
 1.BEGIN
 2.WRITELN("@PASS MARK FOR THIS EXAM IS 40%");
 3.PASSMARK:=40;
 4.WRITE("ENTER CANDIDATE'S MARK %");
 5.READLN(PERSONSMARK);
 6.   IF PERSONSMARK>=PASSMARK THEN
 7.     WRITELN("WELL DONE, YOU HAVE PASSED")
 8.     ELSE
 9.     WRITELN("SORRY, YOU HAVE FAILED")
10.END.
```

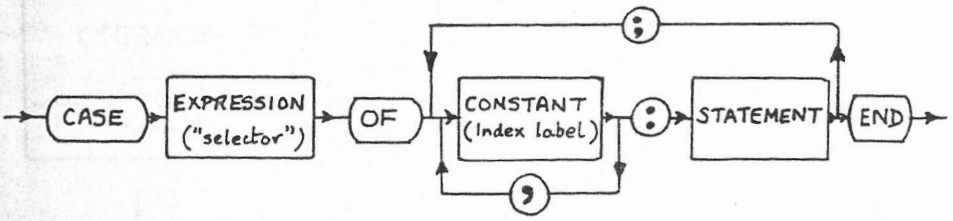
This program can be used to calculate your income tax (figures refer to 82/83 year). Enter your taxable income. The IF... THEN statements compute your tax. For example, IF your "taxincome" is less than £12800 the test expression on line 4 yields a TRUE value, THEN the statement on line

5 will be executed to compute "tax", followed by line 11 (all the other IF expressions are FALSE, so none of the other statements is executed).

```
 0.%TAX 82-83%
 1.VAR TAXINCOME,TAX:REAL;
 2.BEGIN
 3.  READLN(TAXINCOME);
 4.   IF TAXINCOME<=12800.0
 5.    THEN TAX:=0.3*TAXINCOME;
 6.   IF(TAXINCOME>=12801.0)AND(TAXINCOME<=15100.0)
 7.   THEN TAX:=0.3*12800.0
 8.              +(TAXINCOME-12800.0)*0.4;
 9.   IF(TAXINCOME>=15101.0)AND(TAXINCOME<=19100.0)
10.    THEN TAX:=3840.0+920.0+(TAXINCOME-15100.0)*0.45;
11.   IF(TAXINCOME>=19101.0)AND(TAXINCOME<=25300.0)
12.    THEN TAX:=6560.0+(TAXINCOME-19100.0)*0.5;
13.WRITELN(TAX)
14.END.
```

4.3 THE CASE STATEMENT

The IF statements allow only for the choice between two courses of action. Frequently we require the ability to select one of several alternative courses. For this PASCAL provides the CASE statement. The syntax diagram for this statement is given below.



The expression after CASE acts as the "selector" to determine which of the alternative courses of action to take. Each alternative is identified by a CONSTANT, known as the index label. The selector and constant values must be of the same type (INTEGER,CHAR or BOOLEAN but not REAL).

For example,

```
    CASE N OF 1:  statement 1 ;
              2:  statement 2 ;
              3:  statement 3 ;
```

"selector" ↑ — ↑ case label constant

the selector expression in this example is simply the INTEGER variable N. If N=1, THEN statement 1 following 1: is executed; if N=2 then statement 2 following 2: is executed...and so on.

```
    CASE LETTER OF 'A': statement A ;
                   'B': statement B ;
                   'C': statement C ;
                   'D': statement D ;
```

in this case the selector expression is the CHAR variable LETTER; when LETTER='A', then statement A is executed, if LETTER='B' statement B is executed...and so on.

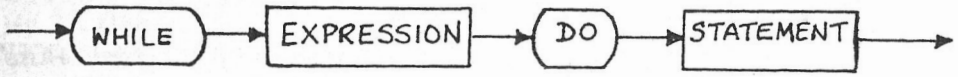
Program examples.

```
 0.%HOLIDAY TARIFF FOR CHILDREN%
 1.VAR AGE:INTEGER;
 2.  BEGIN
 3.      WRITELN("■■■■■■■■■■■■");
 4.    WRITELN("ENTER CHILD'S AGE AS");
 5.    WRITELN("AT LAST BIRTHDAY");
 6.    READLN(AGE);
 7.     WRITELN("■■■■■■■■■■");
 8.      CASE AGE OF
 9.         5,6,7:WRITELN("$40 PER WEEK");
10.         8,9,10:WRITELN("$50 PER WEEK");
11.         11,12,13,14:WRITELN("$60 PER WEEK")
12.      END;
13.      CASE(AGE>15)OR(AGE<4)OF
14.         TRUE:
15.          BEGIN
16.           WRITE("SORRY. WE DO NOT CATER");
17.           WRITE(" FOR CHILDREN OF THAT AGE")
18.          END
19.      END
20.END.
```

```
 0.%STAFF TELEPHOHE LIST%
 1.VAR NAME:CHAR;
 2.BEGIN
 3.WRITE("@");
 4.WRITELN("ENTER FIRST LETTER OF SURNAME ONLY");
 5.  READLN(NAME);
 6.   CASE NAME OF 'A':
 7.     BEGIN
 8.       WRITELN("ADAMS B N :342 8990");
 9.       WRITELN("AVIS W E :45 6767");
10.       WRITELN("AZUL M M :19441 45 65")
11.     END;
12.     'B':
13.     BEGIN
14.       WRITELN("BERNARD F J :89 4572");
15.       WRITELN("BROWN T T L :783 3361")
16.     END;
17.     'C':
18.      BEGIN
19.        WRITELN("CLARK H K :555 3941");
20.        WRITELN("COOK F B :78 12123")
21.      END
22.  END
23.% CONTINUE SAME FOR OTHER NAMES D TO Z%
24.END.
```

4.4 REPETITION 1: THE WHILE...DO STATEMENT

Frequently we require the computer to execute a section of a program repeatedly WHILE a given condition is satisfied. This may be accomplished using the WHILE...DO form of statement.
The syntax diagram for this statement is given below



The expression between WHILE and DO controls whether repetition occurs. This expression is evaluated at the beginning of each cycle, rather than at the end (as in the REPEAT statement). If it is TRUE "before", the statement (or compound statement) following DO is executed. The looping process continues so long as this expression yields TRUE, as soon as this gives FALSE, the statement is "skipped" and execution jumps to the next part of the program.

Program examples

This program uses the WHILE...DO loop to display N, N^2, N^3 for N=1 to 10. The first cycle corresponds to N=0, so obviously the N<10 expression is TRUE. The last cycle corresponds to N=9 (at the beginning of the loop) so when N:=N+1, i.e. N=10 at line 5, the next test of N<10 gives FALSE so no further repetition occurs.

```
 0.VAR N:INTEGER;
 1.BEGIN
 2.N:=0;
 3.  WHILE N<10 DO
 4.   BEGIN
 5.     N:=N+1;
 6.     WRITELN(N:3,N*N:6,N*N*N:8);
 7.   END
 8.END.
```

```
G       <CR>
```

← "GO" COMMAND

```
 1     1       1
 2     4       8
 3     9      27
 4    16      64
 5    25     125
 6    36     216
 7    49     343
 8    64     512
 9    81     729
10   100    1000
```

← RESULTING DISPLAY

This program gives an example of a simple engine check routine. The pressure, temperature, fuel, oil are first read and if O.K. (i.e. the RUN yields a TRUE value), the compound statement following DO is executed. "ALL CONDITIONS O.K." is displayed, followed by a "re-read" and test of these parameters. While RUN=TRUE looping continues; as soon as RUN=FALSE execution jumps to line 14 and the particular fault is displayed.

```
 0.%ENGINE CHECK%
 1.VAR PRES,TEMP,FUEL,OIL:REAL;
 2.    RUN:BOOLEAN;
 3.BEGIN
 4.  READLN(PRES,TEMP,FUEL,OIL);
 5.  RUN:=(PRES)=10.0)AND(TEMP(=100.0)
 6.       AND(FUEL)=0.5)AND(OIL)=4.2);
 7.    WHILE RUN DO
 8.     BEGIN
 9.      WRITELN("ALL CONDITIONS O.K.");
10.      READLN(PRES,TEMP,FUEL,OIL);
11.      RUN:=(PRES)=10.0)AND(TEMP(=100.0)
12.           AND(FUEL)=0.5)AND(OIL)=4.2)
13.     END;
14.      IF PRES(10.0 THEN
15.        WRITELN("PRES LOW");
16.      IF TEMP)100.0 THEN
17.        WRITELN("TEMP HIGH");
18.      IF FUEL(0.5 THEN
19.        WRITELN("CHECK FUEL");
20.      IF OIL(4.2 THEN
21.        WRITELN("TOP UP OIL")
22.END.
```

```
G          <CR>
```

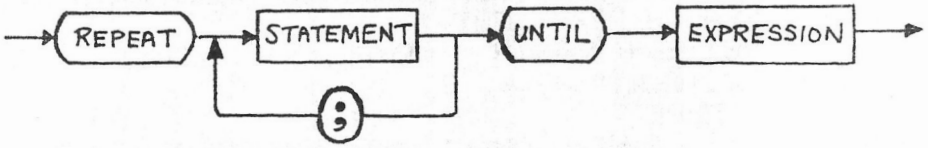
```
? 12.2 , 81.0 , 9.3 , 6.4 <CR>
  ↑      ↑      ↑      ↑
  ENTRY OF PRES, TEMP, FUEL OIL VALUES

ALL CONDITIONS O.K.
  ← DISPLAY OBTAINED

? ← AWAITING NEXT ENTRIES
```

4.5 REPETITION 2: THE REPEAT...UNTIL STATEMENT

The syntax diagram for the REPEAT...UNTIL form of statement is given below. This form is used, rather than WHILE...DO, when, for example, we do not know how many repetitions may be necessary and/or until some condition is satisfied.



The REPEAT statement causes the statement(s) grouped between REPEAT and UNTIL to be repeatedly executed until the expression immediately following UNTIL is TRUE. These statements are obeyed at least once. They must be sequenced correctly and contain at least one statement which has an effect on the terminating condition (the expression after UNTIL) and which eventually will cause looping to stop- otherwise the repetition will continue forever.

Program examples.

This program lists your "outstanding" loan each month until it is paid off. The REPEAT statement is used to work out and display the amount owing each month until LOAN<=0.0 is TRUE. Repetition is then discontinued and the program ends by displaying the total number of months required to pay off the loan. Try running the program by entering some values. Note the program will run forever if your repayments are insufficient. Can you correct the program so this condition cannot occur?

```
 0.% PAYOFF YOUR LOAN %
 1.VAR LOAN,INTERESTRATE,REPAYMENT:REAL;
 2.    MONTHNO:INTEGER;
 3.BEGIN
 4.  MONTHNO:=1;
 5.  WRITE("ENTER LOAN REQUIRED");
 6.  READLN(LOAN);
 7.  WRITE("INTEREST RATE %");
 8.  READLN(INTERESTRATE);
 9.  WRITE("AMOUNT OF REPAYMENT PER MONTH");
10.  READLN(REPAYMENT);
11.  WRITELN("MONTH   DEBT");
12.  WRITELN(" ");
13.    REPEAT
14.      LOAN:=LOAN*(1.0+INTERESTRATE/1200.0)
15.            -REPAYMENT;
16.      WRITELN(MONTHNO:4,LOAN:10:2);
17.      MONTHNO:=MONTHNO+1;
18.    UNTIL LOAN<=0.0;
19.  WRITELN(" ");
20.  WRITE("LOAN IS PAID OFF IN ");
21.  WRITELN(MONTHNO-1:3," MONTHS")
22.END.
```

The next example (shown at top of following page) uses the REPEAT statement to read letters typed in and count the total number and numbers of respective vowels "until" a full-stop is entered.
Note in running the program on the Sharp a carriage return must be entered after each letter.

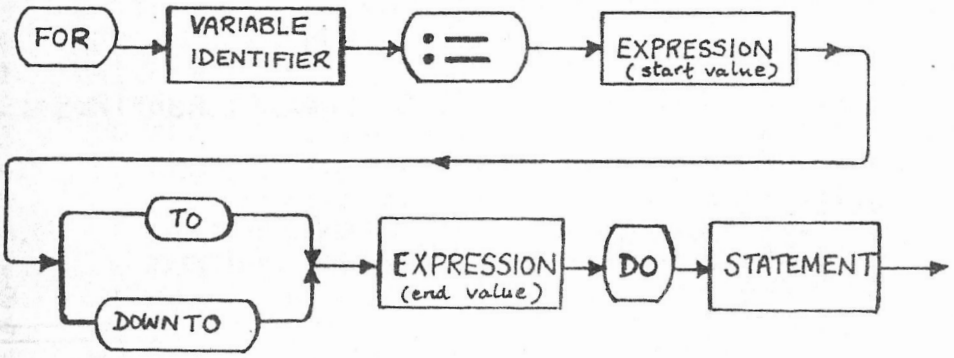
The final example (lower half of following page) simulates an acceptance test and count procedure. Lengths are entered in, tested to be within +1% of 100 and the numbers accepted and rejected counted. The process is terminated by entering LENGTH as 0.0

```
 0.VAR CH:CHAR;
 1.    N,A,E,I,O,U:INTEGER;
 2.BEGIN
 3.  N:=0;A:=0;E:=0;I:=0;O:=0;U:=0;
 4.    REPEAT
 5.      READ(CH);
 6.      N:=N+1;
 7.      IF CH='A'THEN A:=A+1;
 8.      IF CH='E'THEN E:=E+1;
 9.      IF CH='I'THEN I:=I+1;
10.      IF CH='O'THEN O:=O+1;
11.      IF CH='U'THEN U:=U+1
12.    UNTIL CH='.';
13.WRITELN("TOTAL NO. OF CHARACTERS=",N:5);
14.WRITELN("NO. OF As=",A:4);
15.WRITELN("NO. OF Es=",E:4);
16.WRITELN("NO. OF Is=",I:4);
17.WRITELN("NO. OF Os=",O:4);
18.WRITELN("NO. OF Us=",U:4);
19.END.
```

```
 0.% ACCEPTANCE COUNT PROCEDURE%
 1.VAR M,N:INTEGER;
 2.    LENGTH:REAL;
 3.    ACCEPT:BOOLEAN;
 4.BEGIN
 5.M:=0;N:=0;
 6.  REPEAT
 7.    WRITE("ENTER LENGTH ");
 8.    READLN(LENGTH);
 9.ACCEPT:=(LENGTH>=99.0)AND(LENGTH<=101.0);
10.    IF ACCEPT THEN
11.        BEGIN
12.         M:=M+1;
13.         WRITELN("WITHIN SPEC")
14.        END
15.      ELSE
16.        BEGIN
17.         N:=N+1;
18.         WRITELN("REJECT")
19.        END
20.  UNTIL LENGTH<=0.0;
21.WRITELN("NO. LENGTHS WITHIN SPEC.=",M:4);
22.WRITELN("NO. OUTSIDE SPEC.=",N-1:4)
23.END.
```

4.6 REPETITION 3:THE FOR...TO...DO STATEMENT

When we wish to execute a statement or compound statement for a given number of times (the number not depending on any statements within the loop) we use the FOR...TO...DO form of statement,the syntax diagram for which is given below.



The variable identifier after FOR is known as the control variable of the FOR statement. The control variable can be INTEGER or CHAR but must not be REAL.
The "start" and "end" values must be of the same type as the control variable.
When the start value is less than the end value TO applies, for the opposite situation DOWNTO applies.
For example:

```
0.VAR N:INTEGER;
1.BEGIN
2.   FOR N:=12 DOWNTO 1 DO
3.     WRITE(N:3)
4.END.
```

```
G    <CR>
```
← "GO" COMMAND

```
12 11 10  9  8  7  6  5  4  3  2  1
```
← DISPLAY

```
 0.VAR N:INTEGER;
 1.BEGIN
 2.   FOR N:=1 TO 12 DO
 3.     WRITE(N:3)
 4.END.
G     <CR>
 1  2  3  4  5  6  7  8  9 10 11 12
```

← DISPLAY

```
 0.VAR LETTER:CHAR;
 1.BEGIN
 2.   FOR LETTER:='A'TO 'M'DO
 3.     WRITE(LETTER:2)
 4.END.
G     <CR>
A B C D E F G H I J K L M A B
```

← DISPLAY

```
 0.VAR LETTER:CHAR;
 1.BEGIN
 2.   FOR LETTER:='Z'DOWNTO 'N'DO
 3.     WRITE(LETTER:2)
 4.END.
G     <CR>
Z Y X W V U T S R Q P O N
```

← DISPLAY

Program examples

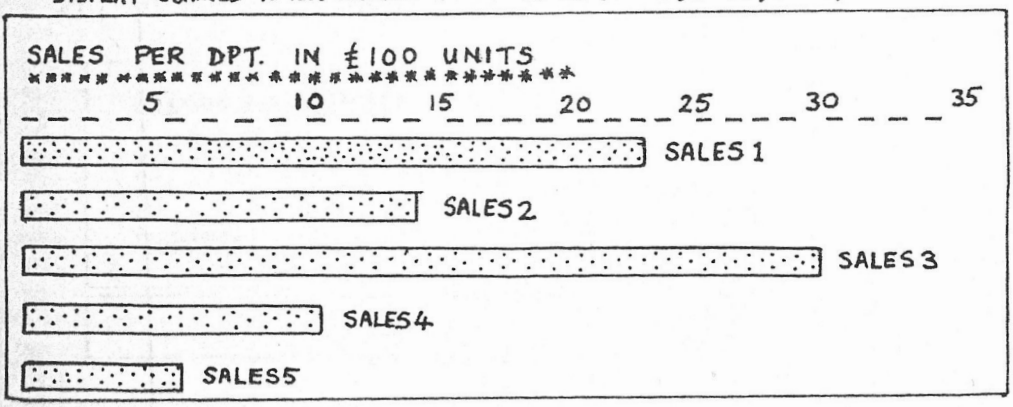
This example uses FOR...TO...DO... statements to draw rectangles. Enter the values of the sides required (A upto 22, B upto 21), the computer displays the rectangle.

```
 0.VAR HDASH1,VDASH,HDASH2:CHAR;
 1.     A,B,N:INTEGER;
 2.BEGIN
 3.   HDASH1:='_';VDASH:='|';HDASH2:='‾';
 4.   READ(A,B);
 5.   WRITE("▯");
 6.     FOR N:=1 TO A DO
 7.       WRITE(HDASH1:1);
 8.       WRITELN("▯");
 9.         FOR N:=1 TO B DO
10.         WRITELN(VDASH:1,VDASH:A);
11.           FOR N:=1 TO A DO
12.           WRITE(HDASH2:1)
13.END.
```

This program uses FOR loops to display a bar chart.

```
 0.%BAR CHART SHOWING SALES FIGURES%
 1.VAR SALES1,SALES2,SALES3,SALES4,SALES5,N:INTEGER;
 2.BEGIN
 3.WRITE("TYPE IN SALES1 TO NEAREST £100");
 4.READLN(SALES1);
 5.WRITE("SALES2=");READLN(SALES2);
 6.WRITE("SALES3=");READLN(SALES3);
 7.WRITE("SALES4=");READLN(SALES4);
 8.WRITE("SALES5=");READLN(SALES5);
 9.WRITELN("£");
10.WRITELN("SALES PER DPT. IN £100 UNITS");
11.WRITELN("*********************************");
12.WRITELN("    5    10   15   20   25   30   35");
13.WRITELN("---- ---- ---- ---- ---- ---- ---- ");
14.  FOR N:=1 TO SALES1 DO
15.  WRITE("▒");WRITELN(" SALES1");
16.  WRITELN(" ");
17.   FOR N:=1 TO SALES2 DO
18.   WRITE("▒");WRITELN(" SALES2");
19.   WRITELN(" ");
20.    FOR N:=1 TO SALES3 DO
21.    WRITE("▒");WRITELN(" SALES3");
22.    WRITELN(" ");
23.     FOR N:=1 TO SALES4 DO
24.     WRITE("▒");WRITELN(" SALES4");
25.     WRITELN(" ");
26.      FOR N:=1 TO SALES5 DO
27.      WRITE("▒");WRITELN(" SALES5")
28.END.
```

DISPLAY OBAINED AFTER ENTERING : 5A S1 23 ; 2:14 ; 3:30; 4:11; 5:6.



CHAPTER 5

PROCEDURES AND FUNCTIONS

5.1 INTRODUCTION

In this chapter we introduce the use of PROCEDURES and FUNCTIONS : how they are declared, their basic structure and how they are used in PASCAL programs.

A procedure is essentially a subroutine designed to accomplish a given task and once written may be "called" into action wherever and whenever it is required in the program.

A function is in many respects similar in both construction and use to a procedure. However,whereas a procedure is used to identify a particular set of actions a user defined function is used to perform a specific calcuation or similar task in an identical way as carried out by the standard functions previously considered in chapter 3, section 3.6

The use of PROCEDURES and FUNCTIONS have three important advantages:

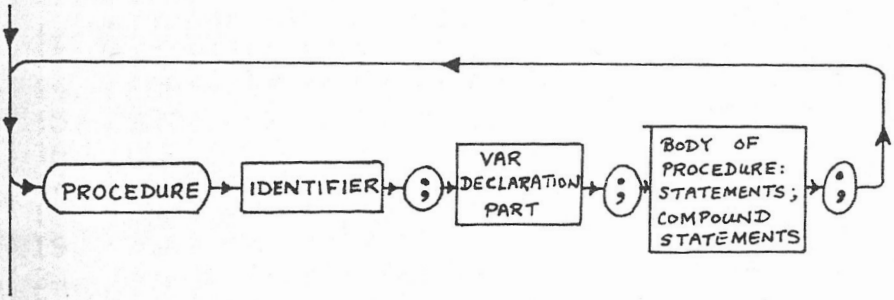
*** they avoid duplication in a program; each procedure and function is written only once but may be "called" (merely by writing their identifier) as many times as is required in the program;

*** larger and more complex programs are easier to develope (sections of the soltion can be developed and tested individually and then subsequently incorporated in the full program as procedures and functions);

*** complex programs are much easier for other user's to both read and understand.

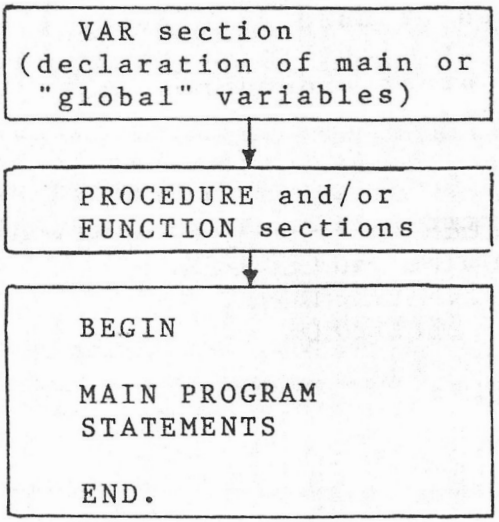
5.2 SIMPLE PROCEDURES: THEIR DECLARATION, STRUCTURE AND CALL

First let us consider the simple type of procedure in which no formal parameter list is used. The basic construction for this type is shown in the syntax diagram given below:



Every procedure must be allocated an identifier (i.e. a name, in the same way as for variables). This identifier follows immediately after PROCEDURE.
The PROCEDURE IDENTIFIER is followed by the VAR declaration for any variables to be used "locally" within the procedure. Then follows the body of the procedure: the simple and/or compound statements making up the execution part of the procedure.

Procedures (and functions) are declared immediately after the VAR declaration of the full progam and followed by the execution sections of the main program. Thus the order of a program containing one or more procedures or functions is:



A simple procedure is "invoked" or "called" into operation in the main program just by writing its identifier. For example, if we have a program containing a procedure whose identifier is PRINTALINE, any statement of the form:

```
PRINTALINE;
```

in the main program section will cause the task contained in this procedure to be carried out.

Program example.

The following program illustrates the basic ideas of how simple procedures are written, incorporated in the main program and how they may be called. Note the order of the program:

1. The variables of the main program are first declared (only one N in our example).
2. The procedures (MONEY and MILESTOKM in our example) are then declared.
3. The VAR and PROCEDURE sections are then followed by the main program statements, which in our example contain calls to procedures in lines 34 and 35.

The program can be used to convert £ to French francs or miles to kilometres. Any number of other procedures could, of course, be included.

```
 0.VAR N:INTEGER;
 1.%THE FOLLOWING PROCEDURES:%
 2.%1.MONEY. 2.MILESTOKM.%
 3.%ARE FIRST DECLARED%
 4.PROCEDURE MONEY;
 5.VAR F,P:REAL;
 6.   BEGIN
 7.     WRITE("ENTER SUM IN £ ");
 8.     READLN(P);
 9.     F:=10.63*P;
10.     WRITE("■■■■■£",P:8:2,"=");
11.     WRITELN(F:9:2," FRANCS");
12.     WRITE("AT THE CURRENT RATE");
13.     WRITE(" OF £1=FF10.63")
14.   END;
15.PROCEDURE MILESTOKM;
16.VAR M,KM:REAL;
17.   BEGIN
18.     WRITE("ENTER DISTANCE IN MILES ");
19.     READLN(M);
20.     KM:=1.6093*M;
21.     WRITE("■■■■■",M:8:2," MILES");
22.     KM:=1.6093*M;
23.     WRITE(KM:8:2," KILOMETRES");
24.END;
25.BEGIN
26.   WRITELN("■IF YOU WISH TO");
27.   WRITELN("1.CONVERT £ TO FRENCH FRANCS");
28.   WRITELN("*****ENTER 1*****");
29.   WRITELN(" ");
30.   WRITELN("2.CONVERT MILES TO KM.");
31.   WRITELN("****ENTER 2****");
32.     READLN(N);
33.     WRITELN("■");
34.       IF N=1 THEN MONEY
35.         ELSE MILESTOKM
36.END.
```

VARIBLE DECLARATION FOR MAIN PROGRAM

IDENTIFIER

VAR DECLARATION OF "LOCAL" VARIABLES

PROCEDURE MONEY

EXECUTABLE STATEMENTS (BODY OF PROCEDURE)

PROCEDURE MILESTOKM

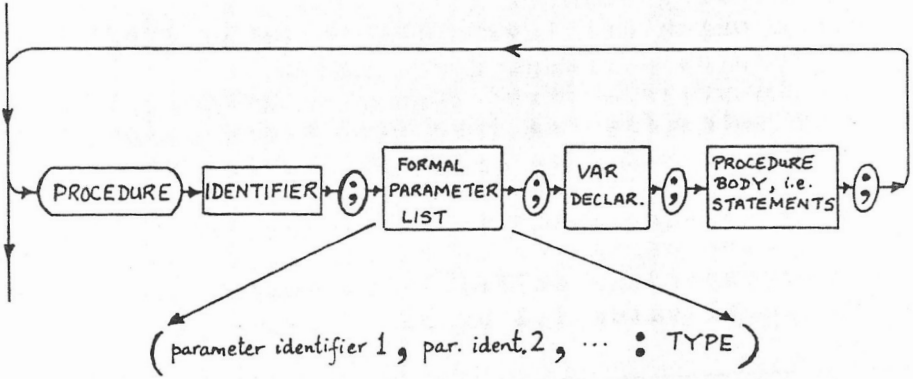
MAIN PROGRAM

CALL FOR PROCEDURE MONEY

CALL FOR PROCEDURE MILESTOKM

5.3 PROCEDURES WITH VALUE PARAMETERS

We can greatly increase the versatility of procedures by employing "value" parameter variables. Different values to these parameters may be given each time the procedure is called.
The value parameter variables are declaered by including a "formal" parameter list immediately after the procedure identifier in the manner shown in the syntax diagram below.



As an example of the use of value parameters let us consider the drawing of a bar chart using procedure calls to generate the bars of the specified length and also to provide a simple form of labelling.
We make use of two value parameter variables:
A to specify length; M to label the bar.
The procedure could then be written as follows:

```
PROCEDURE BAR(A,M:INTEGER);
VAR N : INTEGER;
BEGIN
FOR N:=1 TO A DO
WRITE(" ");
WRITE("QUANTITY",M:2);
WRITELN(" ");
END;
```

This procedure may then be called anywhere in the main program by assigning values to the value parameter variables in the following way:

```
BAR(18,5);
```

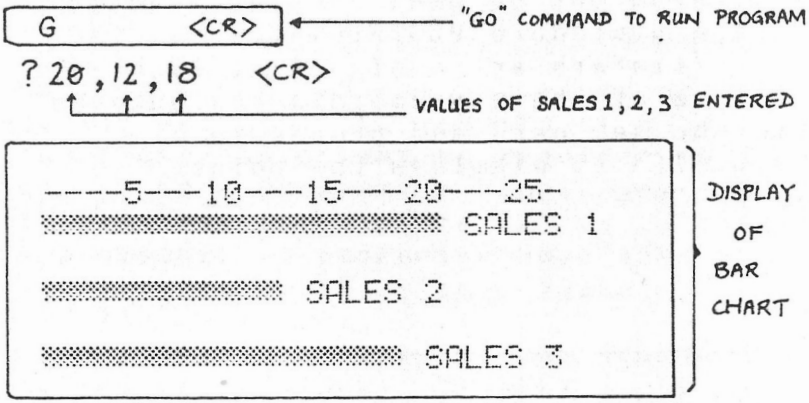
value parameter M is assigned the value 5

value parameter A is assigned the value 18

Such a call would cause a bar of length 18 units followed by it's label QUANTITY 5 to be displayed.

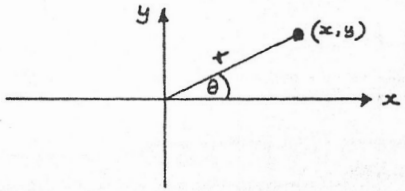
The following example illustates a complete program which displays a bar chart of sales figures plus corresponding labels.
The main program first requests us to enter in the sales figures (see READLN statement in line 10). It then calls procedure BAR (see lines 12,13 and 14) to effect the display of each SALES bar plus label; the A value parameter being assigned the the corresponding SALES1,2 or 3 value and M the label value 1,2 or 3.

```
 0.VAR SALES1,SALES2,SALES3:INTEGER;
 1.   PROCEDURE BAR(A,M:INTEGER);
 2.   VAR N:INTEGER;
 3.   BEGIN
 4.     FOR N:=1 TO A DO
 5.     WRITE("▒");
 6.     WRITELN(" SALES",M:2);
 7.     WRITELN(" ")
 8.   END;
 9.BEGIN
10.   READLN(SALES1,SALES2,SALES3);
11.   WRITELN("----5---10---15---20---25-");
12.   BAR(SALES1,1);
13.   BAR(SALES2,2);
14.   BAR(SALES3,3)
15.END.
```



Further program examples

This program converts (x,y) coordinates to polar coordinates (r,θ) using the procedure POLAR(X,Y).




```
 0.% TO CONVERT CATESIAN x,y TO POLAR %
 1.VAR X,Y:REAL;
 2.PROCEDURE POLAR(X,Y:REAL);
 3.VAR R,ANGLE:REAL;
 4.BEGIN
 5.   R:=SQRT(X*X+Y*Y);
 6.   ANGLE:=ARCTAN(Y/X)*180.0/3.1415927;
 7.    IF(X<0.0)THEN
 8.    ANGLE:=180.0+ANGLE;
 9.   WRITELN("R=",R:6:2,"  THETA=",ANGLE:6:2)
10.END;
11.   BEGIN
12.     WRITELN("ENTER VALUES OF x,y");
13.     WRITE("x= ");READLN(X);
14.     WRITE("y= ");READLN(Y);
15.     POLAR(X,Y)
16.   END.
```

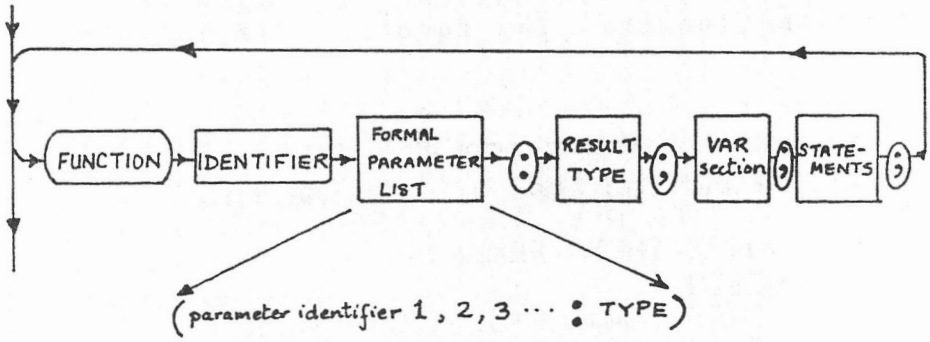
This program may be used to plot a graph using the procedure PLOTAPOINT(A,B). The value parameters are assigned at each call the values of the x,y coordinates entered in via the keyboard and procedure PLOTAPOINT(A,B) displays the point (x,y) by "*".

```
 0.% To plot a graph %
 1.VAR X,Y,P,M:INTEGER;
 2.PROCEDURE PLOTAPOINT(A,B:INTEGER);
 3.% LOCATING POSITION OF POINT%
 4.VAR N:INTEGER;
 5.BEGIN
 6.WRITE("■");
 7.  FOR N:=0 TO A DO
 8.  WRITE(" ");
 9.  FOR N:=0 TO B DO
10.  WRITE("■");
11.  WRITE("■*")
12.END;
13.% BEGINNING OF MAIN PROGAM%
14.BEGIN
15.WRITELN("■");
16.WRITELN("ENTER NO. OF POINTS TO BE PLOTTED");
17.READLN(M);
18.WRITELN("ENTER x,y CORDINATES OF EACH POINT");
19.WRITELN("■");
20.WRITE(" ----5----10---15---20---25---30---35");
21.WRITELN("■");
22.WRITE("■■■■■5■■■■■■10■■■■■■■15■■■■■■■20");
23.  FOR P:=1 TO M DO
24.  BEGIN
25.   WRITE("■");
26.   READ(X,Y);
27.   PLOTAPOINT(X,Y)
28.  END
29.END.
```

5.4 FUNCTIONS: DECLARATION AND USE

A function rather than a procedure is used when, for example, the results of a calculation or expression may be required more than once in a program. A simple way of distinguishing between the two is to think of a procedure as producing some effect whilst a function gives some value.

We have already met many of the standard functions (e.g. SIN(X), TRUNC(X), ORD(X)...) in chapter 3. We now consider how we can declare our own "user defined" functions, the syntax diagram for which is shown below. Note this is very similar to that of a procedure except that the result type must also be included.



Program examples

PASCAL provides no direct means of computing powers. This program provides such a function P(X,Y) which may be used to calculate X^Y The base X can take any REAL value greater than zero and the exponent or power Y is also REAL but may be positive or negative. The example given in the program calculates

$$C = 3.0^{2} + 2.0^{8}$$

```
 0.VAR A,B,C:REAL;
 1.% Power function %
 2.FUNCTION P(X,Y:REAL):REAL;
 3.BEGIN
 4.   P:=EXP(Y*LN(X))
 5.END;
 6.BEGIN
 7.   A:=2.0;B:=8.0;
 8.   C:=P(3.0,A)+P(2.0,B);
 9.   WRITELN(C:6:2)
10.END.
```

FUNCTION TO CALCULATE X^Y

This example applies the cosine rule:

$$Z = X + Y - 2XY\cos(\theta)$$

Enter in two lengths and the angle included by the lengths. The function A(X,Y,ANGLE) calculates the third side.

```
 0.VAR B,C,ANGLE:REAL;
 1.% Function applying cosine rule %
 2.FUNCTION A(X,Y,ANGLE:REAL):REAL;
 3.  VAR Z,THETA:REAL;
 4.  BEGIN
 5.    THETA:=ANGLE*3.14159/180.0;
 6.    Z:=X*X+Y*Y-2.0*X*Y*COS(THETA);
 7.    A:=SQRT(Z)
 8.  END;
 9.    BEGIN
10.     WRITE("ENTER LENGTH B ");
11.     READLN(B);
12.     WRITE("ENTER LENGTH C ");
13.     READLN(C);
14.     WRITE("ENTER ANGLE ");
15.     READLN(ANGLE);
16.     WRITE("LENGTH A=",A(B,C,ANGLE):8:2)
17.   END.
```

CHAPTER 6

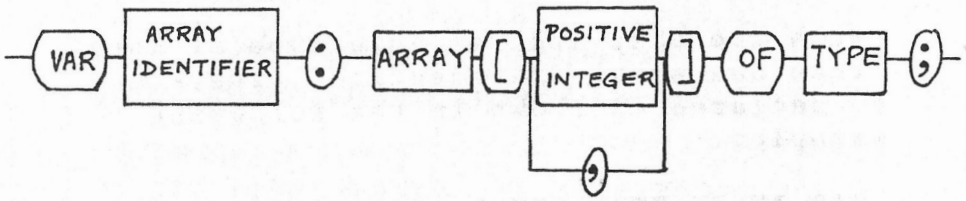
ARRAYS

6.1 INTRODUCTION

The use of arrays in programs makes it very much easier for us to handle and process larger volumes of related information.
We can then give, for example, an associated group of variables a collective name - the array identifier - rather than a series of individual identifiers. We can also easily refer to any of the individual variables by the array identifier plus subscript(s) to identify the particular element. This saves a great amount of "writing" space and is especially useful when reading in, processing and writing out large volumes of information.

6.2 ARRAY DECLARATION

The syntax diagram for declaring an array is given below:



Examples of array declaration

1. One-dimensional arrays

```
VAR XLIST:ARRAY[10] OF CHAR;
```

IDENTIFIER — SUBSCRIPT determining no. of elements — TYPE of variable elements

This delares a primary or one-dimensional array whose identifier is XLIST and which comprises 11 CHAR vaiables:
XLIST[0], XLIST[1], XLIST[3]....XLIST[10]

2. Two dimensional arrays

```
VAR XYTABLE:ARRAY[9,6] OF INTEGER;
```

This declares XYTABLE as a two-dimensional array of 10 x 7 = 70 INTEGER variables:

```
XYTABLE[0,0]  XYTABLE[0,1]....XYTABLE[0,6]
XYTABLE[1,0]  XYTABLE[1,1]....XYTABLE[1,6]
   ...           ...             ...
   ...           ...             ...

XYTABLE[9,0]      ...         XYTABLE[9,6]
```

3. Three-dimensional arrays

```
VAR XYZBLOCK:ARRAY[12,10,5] OF REAL;
```

declares XYZBLOCK as a three-dimensional array of 13 x 11 x 6 REAL variables

4. Declaration of similar size and type

When the size and variable type of more than one array are identical, they may be declared as shown in the following example:

```
VAR XTYPE,YTPE,ZTYPE:ARRAY[32] OF REAL;
```

6.3 EXAMPLES OF PROGRAMS USING ARRAYS

1. Two arrays NOITEMS and PRICE are declared in this program. When running the program we systematically enter in the number and price of each item, 10 pairs in all (see lines 4 and 5). The program then displays the data we have fed in.

```
0.VAR NOITEMS:ARRAY[9]OF INTEGER;
1.     PRICE:ARRAY[9]OF REAL;
2.     N:INTEGER;
3.BEGIN
4.   FOR N:=0 TO 9 DO
5.   READLN(NOITEMS[N],PRICE[N]);
6.    FOR N:=0 TO 9 DO
7.   WRITELN(NOITEMS[N]:4,PRICE[N]:8:2)
8.END.
```

2. A modification is made to the last program to not only list the number and price but also to display sub- and total prices.

```
 0.VAR NOITEMS,PRICE:ARRAY[9]OF REAL;
 1.     N:INTEGER;S:REAL;
 2.BEGIN
 3.S:=0.0;
 4.   FOR N:=0 TO 9 DO
 5.   READLN(NOITEMS[N],PRICE[N]);
 6.WRITELN("  NO.   PRICE      SUB-TOTAL");
 7.    FOR N:=0 TO 9 DO
 8.    BEGIN
 9.    WRITELN(NOITEMS[N]:4,PRICE[N]:8:2,
10.    NOITEMS[N]*PRICE[N]:12:2);
11.    S:=S+NOITEMS[N]*PRICE[N]
12.    END;
13.WRITELN(" ");
14.WRITELN("TOTAL=      ",S:13:2)
15.END.
```

```
G          <CR>
```
← "GO" COMMAND

```
 NO.   PRICE      SUB-TOTAL
 12    34.78       417.36
 21    56.00      1176.00
  9     5.13        46.17
 43    64.98      2794.14
 45    90.00      4050.00
 23    76.60      1761.80
 66      .87        57.42
 12    87.45      1049.40
 34     4.43       150.62
132      .05         6.60

TOTAL=             11509.51
```

EXAMPLE OF RESULTS DISPLAYED (after feeding in NO. ITEMS and PRICE information.

3. This program gives a simple example of the use of two-dimensional arrays. MARKS is the identifier for a 3x3 array. In lines 3,4,5 we read in data, e.g. student's marks. Lines 6-15 provide a tabular display of this data.

```
 0.VAR MARKS:ARRAY[2,2]OF REAL;
 1.    ROW,COLUMN:INTEGER;
 2.BEGIN
 3.  FOR ROW:=1 TO 2 DO
 4.    FOR COLUMN:=1 TO 2 DO
 5.      READLN(MARKS[ROW,COLUMN]);
 6.  FOR ROW:=1 TO 2 DO
 7.   BEGIN
 8.     WRITELN(" ");
 9.     WRITELN("--------------------");
10.     WRITE(ROW:3);
11.       FOR COLUMN:=1 TO 2 DO
12.         WRITE(MARKS[ROW,COLUMN]:6)
13.   END;
14.WRITELN(" ");
15.WRITELN("--------------------")
16.END.
```

4. The program below is essentially a modification of the last program using procedures INREAD and OUTWRITE to enter and display the marks of 6 students for 4 subjects followed by procedure AVSTUDMARK to display the average mark for each student in his/her subjects.

```
 0.VAR MARKS:ARRAY[6,4]OF REAL;
 1.    ROW,COLUMN:INTEGER;
 2.PROCEDURE INREAD;
 3.BEGIN
 4.  FOR ROW:=1 TO 6 DO
 5.    FOR COLUMN:=1 TO 4 DO
 6.      READLN(MARKS[ROW,COLUMN]);
 7.END;
 8.PROCEDURE OUTWRITE;
 9.BEGIN
10.  FOR ROW:=1 TO 6 DO
11.   BEGIN
12.     WRITELN(" ");
13.     WRITE("-------------------------");
14.    WRITELN("----");
15.     WRITE(ROW:3);
16.       FOR COLUMN:=1 TO 4 DO
17.         WRITE(MARKS[ROW,COLUMN]:6)
18.   END;
19.WRITELN(" ");
20.WRITELN("-----------------------------")
21.END;
22.PROCEDURE AVSTUDMARK;
23.VAR SUM:REAL;
24.BEGIN
25.  WRITELN("AVERAGE STUDENT'S MARKS:");
26.  FOR ROW:=1 TO 6 DO
27.   BEGIN
28.    SUM:=0.0;
29.    FOR COLUMN:=1 TO 4 DO
30.    SUM:=SUM+MARKS[ROW,COLUMN];
31.    WRITELN(ROW:3,(SUM/4.0):6:1)
32.   END
33.END;
34.BEGIN
35.INREAD;
36.OUTWRITE;
37.AVSTUDMARK
38.END.
```

On running the program we first enter in the subject marks for each student. These are then displayed in tabular form followed by a list of their average, e.g.

```
-----------------------------
  1    34    65    78    12
-----------------------------
  2    98    67    80    65
-----------------------------
  3     4    67    41    32
-----------------------------
  4    89    76    55    87
-----------------------------
  5     3     5    78    32
-----------------------------
  6     0    76    52    12
-----------------------------
AVERAGE STUDENT'S MARKS:
  1  47.2
  2  77.5
  3  36.0
  4  76.7
  5  29.5
  6  35.0
```

INDEX