PASCAL PROGRAMMING

The programming language PASCAL was developed in the late 1960's by Professor Niklaus Wirth at the Eidgeossische Technische Hochschule, Zurich, Switzerland. His aim was to produce a language containing a small number of fundamental programming concepts that would be suitable for teaching programming as a logical and systematic discipline, and also be capable of efficient implementation on most computers.

This is a simple Pascal program to display "Hello" on the screen.

```
PROGRAM first;
BEGIN
  Write('Hello')
END.
```

Exercises

1) Add another line to display "Good Morning" immediately after the previous line.
2) Modify the above program to display above two text in two different lines.

This is another simple program to read two numbers and display the summation of two numbers.

```
PROGRAM add;

VAR
  x,y,z : integer;

BEGIN
  read(x);
  read(y);
  z := x + y;
  write(z)
END.
```

Improved version of the above program

```
PROGRAM add;

VAR
  x,y,z : integer;

BEGIN
  write('Enter first number  : ');
  read(x);
  write('Enter second number : ');
  read(y);
  z := x + y;
  write('Sum = ', z)
END.
```

STANDARD DATA TYPES

INTEGER	-	Whole numbers
REAL	-	real numbers
CHAR	-	Characters
BOOLEAN	-	*true* or *false*

Exercises

1) Modify the above program to display the difference of two numbers.
2) Modify the above program to display the product of two numbers.
3) Modify the above program to display the summation of the squares of two numbers (සංඛ්‍යා දෙකේ වර්ගවල එකතුව).
4) Modify the above program to divide two numbers.
5) Write a program to calculate the hypotenuse (longest side) of a RIGTH TRIANGLE given other 2 sides. (සෘජුකෝණී ත්‍රිකෝණයක කුඩා පාද දෙකේ අගයන් දී ඇතිවිට කර්ණයේ (දිගම පාදය) දිග සෙවීම)

Consider our first program that display the word “Hello”. To display “Hello” in 3 times, we can use

```
writeln('Hello');
writeln('Hello');
writeln('Hello');
```

This is OK for small numbers, if you want to repeat it 100 times ?????

REPETITIVE STATEMENTS

An important class of action in computer programs is the *loop*, which enables the repetition of some statement, or group of statements, subject normally to some termination condition.

Pascal provides 3 repetitive constructs which reflect the needs of loop construction in most programming situations,

WHILE statement
REPEAT statement
FOR statement

FOR statement

The FOR statement may be used for operations which are to be carried out a pre-defined number of times.

For example, to display word “Hello” for 10 times, you can use following code.

```
PROGRAM rep1;
VAR
  i : integer;
BEGIN
  FOR i := 1 TO 10 DO
    writeln('Hello')
END;
```

Exercises:

1) Modify the above program to display as follows:

```
Hello  1
Hello  2
..
..
..
Hello 10
```

2) Modify the above program to display above line any number of times.

3) Modify the above program to display as follows:

```
Hello  1
Good Morning
Hello  2
Good Morning
..
..
..
Hello 10
Good Morning
```

4) Write a program to display following out put; 2 times table.

```
2
4
6
8
..
..
24
```

5) Modify the above program to display:

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
..
..
2 x 12 = 24
```

6) Modify the above program to display N times table (N වරක්).

This program will read set of numbers and calculate the total value.

```
PROGRAM sum;

VAR
  x, n, i, total : integer;

BEGIN
  write('How many numbers ? : ');
  read(n);
  writeln;
  total := 0;

  FOR i := 1 TO n DO
    BEGIN
      write('Enter number : ');
      read(x);
      total := total + x;
    END;

  Writeln;
  writeln('Total = ', total)
END.
```

Exercise

1) Modify the above program to calculate the AVERAGE also.

WHILE statement

WHILE statement may be used for operations which are to be carried out while a condition (expression) remains true.

We can re-write the previous program using WHILE statement as follows.

NOTE: We do not need to know the number of items at the beginning, instead read numbers one by one and at the end we enter an unusual number, say -999 to indicate that our number are over.

```
PROGRAM sum2;

VAR
  x, total, count : integer;
           average : real;

BEGIN
  total := 0;
  count := 0;

  write('Enter number : ');
  read(x);

  WHILE (x <> -999) DO
    BEGIN
      count := count + 1;
      total := total + x;
      write('Enter number : ');
      read(x)
    END;

  average := total / count;

  writeln;
  writeln('Total   = ', total);
  writeln('Average = ', average:5:2)
END.
```

REPEAT statement

The sequence of statements between the REPEAT and UNTIL symbols is executed and the condition (expression) is evaluated. If its value true then the REPEAT statement is terminated, otherwise execution of the statement sequence is repeated until the expression becomes true.

The essential difference between REPEAT and WHILE statements is the in the REPEAT statement the loop body is performed at least once, before the first evaluation of the termination condition, whereas in the WHILE statement the terminating condition is evaluated first and so the loop body may not be executed at all.

We can rewrite the previous program using REPEAT statement as follows.

```
PROGRAM sum3;

VAR
  x, total, count : integer;
           average : real;

BEGIN
  total := 0;
  count := 0;

  write('Enter number : ');
  read(x);

  REPEAT
    count := count + 1;
    total := total + x;
    write('Enter number : ');
    read(x)
  UNTIL (x = -999);

  average := total / count;

  writeln;
  writeln('Total   = ', total);
  writeln('Average = ', average:5:2);
END.
```

NESTED REPETITIVE statements

The body of the repetitive statement may itself contain another repetitive statement, in which case repetitive statements are said to be *nested.* This nesting of repetitive statements is illustrated in next programe, which reads an integer N and calculates the sum of the series

$1^1 + 2^2 + 3^3 + \ldots + n^n$.

```
PROGRAM sumofpowers;

VAR
  n, x, power, i, sum : integer;

BEGIN
  write('N = ');
  read(n);

  sum := 0;

  FOR x := 1 TO n DO
    BEGIN
      power := 1;
      FOR i := 1 TO x DO
        Power := power * x;
      sum := sum + power
    END;
  writeln('Sum = ', sum)
END.
```

CONDITIONAL STATEMENTS

It is often necessary to make the execution of a statement dependent upon some condition, or else at some point to choose to execute one of a number of possible statements depending upon some condition. Pascal provides 2 statement for this purpose

IF statement
CASE statement

IF statements

IF statement allows the conditional execution of one statement, or the choice between execution of 2 statements.

Following program reads AGE and display a warning message “Sorry! you are NOT allowed” if his/her AGE less than 18.

```
PROGRAM cond1;

VAR
  age : integer;

BEGIN
  write('Please enter your age in years : ');
  read(age);

  IF age < 18
    THEN
      writeln('Sorry! you are NOT allowed');
  writel('Thanks');
END.
```

Following program reads MARKS of a subject and display ‘PASS’ if marks greater than or equal to 40, otherwise display ‘Fail’.

```
PROGRAM cond2;

VAR
  marks : integer;

BEGIN
  write('Enter marks : ');
  read(marks);

  IF marks >= 40
    THEN
      writeln('Pass')
    ELSE
      writeln('Fail')
END.
```

Improved version of the above program using a BOOLEAN variable.

```
PROGRAM cond3;

VAR
  marks : integer;
  pass  : boolean;

BEGIN
  write('Enter marks : ');
  read(marks);

  pass := (marks >= 40);

  IF pass
    THEN
      Writeln('Pass')
    ELSE
      writeln('Fail')
END.
```

The following program read students marks and display the GRADE accordingly.

```
PROGRAM cond4;

VAR
  marks : integer;
  grade : char;

BEGIN
  write('Enter marks : ');
  read(marks);

  IF (marks >= 75)
    THEN
      grade := 'A'
    ELSE
      IF (marks >= 65)
        THEN
          grade := 'B'
        ELSE
          IF (marks >= 55)
            THEN
              grade := 'C'
            ELSE
              IF (marks >= 40)
                THEN
                  grade := 'S'
                ELSE
                   grade := 'W';

  writeln('Your grade is ', grade)
END.
```

Exercises

1) Write program to input the MONTH as a number (1,2,..12) and display the NUMBER OF DAYS in that month. Assume February has 28 days. (HINT: you have to combine several conditions with OR operator).
2) Write program to input the YEAR (assume the range 1901 to 2099) and display whether it is a LEAP year (අධික අවුරුද්දක්) or NOT.
3) Extend the above program for any year.

IN statement

The operator IN is used to test the presence of an individual member in a set.

For example, consider following conditional statement:

```
IF  (x = 1) OR (x = 5) OR (x = 7) OR (x = 12) OR (x = 15)
```

can be re-written more efficiently, clearly using IN statement as

```
IF x IN [1, 5, 7, 12, 15]
```

We can specify a RANGE in IN statement, like

x IN [0..100], letter IN ['a'..'z']

Exercises

1) Re-write the program in ex 1) above using IN statement.

Following program will read a sentence ending with period (.) and count the number of total characters.

```
PROGRAM cnt;

VAR
  ch : char;
  count : integer;

BEGIN
  count := 0;
  read(ch);
  WHILE ch <> '.' DO
    BEGIN
      count := count + 1;
      read(ch)
    END;
  Writeln('Number of characters = ', count);
END.
```

As above program count BLANK SPACES as characters, we can modify the program to count characters except BLANK CHARACTERS.

```
PROGRAM cnt;

VAR
  ch : char;
  count : integer;

BEGIN
  count := 0;
  read(ch);
  WHILE ch <> '.' DO
    BEGIN
      IF ch <> ' '
        THEN
          count := count + 1;
      read(ch)
    END;
  Writeln('Number of characters = ', count);
END.
```

Exercises

1) Modify the above program to count the number of SPACES (blank characters) in the statement.
2) Modify the above program to count number of VOWELS in the statement.
3) Modify the above program to count number of CONSONENTS (VOWELS නොවන ඉංග්‍රීසි අකුරු).
4) Modify the above program to count number of CAPITAL Letters.

The CASE statement

One complex selection pattern which occurs frequently in programming, and so deserves special consideration, is the SELECTION of one of a set of actions according to the value of some expression. This could be expressed ad a nested IF statement, e.g.,

```
IF ch = 'I' THEN n := 1 ELSE
IF ch = 'V' THEN n := 5 ELSE
IF ch = 'X' THEN n := 10 ELSE
IF ch = 'L' THEN n := 50
```

A more elegant way of expressing such in action in Pascal is to use the CASE statement.

```
CASE ch OF
  'I' : n := 1;
  'V' : n := 5;
  'X' : n := 10;
  'L' : n := 50
END;
```

We can re-write the previous exercise of finding the number of days in a month as follows.

```
PROGRAM ndays;

VAR
  Month, days : integer;

BEGIN
  write('Enter month (MM): ');
  read(month);

  CASE month OF
    1,3,5,7,8,10,12 : days := 31;
          4,6,9,11 : days := 30;
                 2 : days := 28
    ELSE             days :=  0
  END;

  write('Month', month:3, ' has', days:3, ' days');
END.
```

ARRAYS

Pascal provides a data structure called the ARRAY, which can store a fixed size sequential collection of elements of the same type.

Instead of declaring individual variables, such as *number1*, *number2*, .., *numberN*, you declare one array variable such as *numbers* and use *numbers[1], numbers[2]*, .., *numbers[n]* to represent individual variables.

This program reads set of numbers and display it in reverse order.

```
PROGRAM arr;

VAR
  i, n : integer;
  num  : ARRAY [1..50] of integer;

BEGIN
  write('How many numbers : ');
  read(n);

  FOR i := 1 TO n DO
    read(num[i]);

  FOR i := n DOWNTO 1 DO
    Write(num[i], ' ');
END.
```

This program reads characters as an array and displays 2nd and 3rd characters.

```
PROGRAM arr2;

VAR
  name : ARRAY [1..50] OF char;
  i    : integer;

BEGIN
  FOR i := 1 TO 10 DO
    read(name[i]);

  write(name[2]);
  writeln(name[3]);
END.
```

Character STRINGS

To read a sequence of character strings, we may use variable type STRING.

```
VAR
  name : string;
```

and name can be any sequence of characters, like

```
 name := 'Colombo' or name := 'Saman Perera' or name := 'Good Morning'.
```

Following program will read your name and display it on the screen.

```
PROGRAM str;

VAR
  name : string;

BEGIN
  write('Type your name here : ');
  read(name);
  writeln(name)
END.
```

SUB PROGRAMS

A sub-program is a program unit/module that performs a particular task. These sub-programs are combine to form a larger programs. A sub-program can be invoked by a program/sub-program which is called the calling program.

PROCEDURES

Procedures are sub-program that allow to obtain a group of results.

Following is the source code for the PROCEDURE findmax(). This will find the maximum number of two variables x and y. All variables x, y, max will be defined in the main program.

```
PROCEDURE findmax;
BEGIN
  IF x > y THEN max := x ELSE max := y;
END;
```

Calling the PROCEDURE

While creating a procedure, you give a definition of what the procedure has to do. To use the procedure, you will have to call that procedure to perform the defined task. When a program calls a procedure, program control is transferred to the called procedure. A called procedure performs the defined task, and when its last end statement is reached, it returns the control back to the calling program

```
PROGRAM maximum;

VAR
  x,y, max : integer;

  PROCEDURE findmax;
  BEGIN
    IF x > y THEN max := x ELSE max := y;
  END;

BEGIN
  write('Enter 2 numbers : ');
  read(x,y);
  findmax;
  writeln('Maximum is ', max);
END.
```

The above PROCEDURE find the maximum number of the variables x, and y. In practice, it is more convenient to declare a procedure which find maximum of any two numbers, not just given x, and y. Then, if at some point in the program we wish to find the maximum of x and y, we may write

findmax(x, y);

whereas if, at another point, we wish to find the maximum value of y and another integer variable z, we may write

findmax(y, z);

This is possible in Pascal by the declaration of a procedure which includes a formal-parameter-list in it heading as follows.

```
PROCEDURE findmax(p, q : integer);
BEGIN
  IF p > q THEN max := p ELSE max := q;
END;
```

Ex: Write a complete program to implement the above. Hint: Read 3 variables *x*, *y*, & *z* and find maximum of *x* and *y* and maximum of *y* and *z*.

Value Parameters

A value parameter is used when the parameter's only role is to carry a value into a procedure (input parameter).

The following procedure accepts two integer values representing a distance in feet and inches and writes it out in meters. In this case, *feet* and *inches* are value parameters.

```
PROCEDURE writeasmeters(feet, inches : integer);
VAR
  meters : real;
BEGIN
  inches := 12 * feet + inches;
  meters := inches / 39.39;
  write(meters:6:2);
END;
```

Ex: Write a complete program to implement the above.

Variable Parameters

A variable formal-parameter is used to denote an actual-parameter whose value may be altered by execution of the procedure.

In some cases, the procedure may use the existing value of an actual variable parameter before changing it. For example, the following procedure order examines and, if necessary, exchanges the values of its two parameters.

```
PROCEDURE order (VAR a,b : integer);

VAR i : integer;

BEGIN
  IF a > b
    THEN
      BEGIN
        t := a;
        a := b;
        b := t
      END
END;
```

Ex: Write a complete program to implement the above.

FUNCTIONS

Pascal not only provides standard functions (like sqr, sqrt, etc which do not require any declaration in the program) but also a means whereby the programmer can declare his / her own *functions* and have them evaluated, with appropriate parameters, as components of expressions. A function is a special form of procedure which describes a computation that produces a single value as its result.

The following is an example of a function-declaration defining a function *max* which yields as its result the larger of the values of its two real value parameters *x* and *y*.

```
FUNCTION max (x,y : integer) : integer;
BEGIN
  IF x > y THEN max := x ELSE max := y
END;
```

You may call this FUNCTION as

```
BEGIN
  write('Enter two numbers : ');
  readln(a,b);

  c := max(a,b);

  writeln('Maximum = ', c);
END.
```

Ex1: Write a function to convert the temperature from Centigrade to Fahrenheit and call the function in main program.

RECURSION

In Pascal, a procedure or function is permitted not only to call another procedure or function but also to call itself. Such a call is said to be *recursive*.

Consider for example the process which accepts a non-negative integer value and outputs the sequence of decimal digits representing the number in *reverse* oder.

BEGIN
output last digit of N;
IF *digits remain*
THEN *reverse remaining digits*
END

The above process may be written in Pascal, as a recursive procedure,

```
PROCEDURE reverse(n : longint);
BEGIN
  write(N MOD 10);
  IF N DIV 10 <> 0  THEN reverse(N DIV 10)
END;
```

Consider a variation of above – one which reads a sequence of characters of arbitrary length terminated by some special character as '.' say., and prints the sequence in reverse.

```
PROCEDURE reverseinput;
VAR c : char;
BEGIN
  read(c);
  IF c <> '.' THEN reverseinput;
  Write(c)
END;
```

Many mathematical functions are defined recursively and it thus appears natural to write recursive functions to compute their values. The factorial function is a familiar example which may be defined as

factorial(0) = 1
factorial(n) = n * factorial(n-1) for n > 0

and so a Pascal function may be declared to evaluate the factorial of a given value *n*.

```
FUNCTION fac(n : longint) : longint;
BEGIN
  IF n = 0 THEN fac := 1
           ELSE fac := n * fac(n-1)
END;
```