PASCAL - PROCEDURES

Procedures are subprograms that, instead of returning a single value, allow to obtain a group of results.

Defining a Procedure

In Pascal, a procedure is defined using the **procedure** keyword. The general form of a procedure definition is as follows −

```
procedure name(argument(s): type1, argument(s): type 2, ... );
   < local declarations >
begin
   < procedure body >
end;
```

A procedure **definition** in Pascal consists of a **header**, local **declarations** and a **body** of the procedure. The procedure header consists of the keyword **procedure** and a name given to the procedure. Here are all the parts of a procedure −

- **Arguments** − The argument*s* establish the linkage between the calling program and the procedure identifiers and also called the formal parameters. Rules for arguments in procedures are same as that for the functions.
- **Local declarations** − Local declarations refer to the declarations for labels, constants, variables, functions and procedures, which are applicable to the body of the procedure only.
- **Procedure Body** − The procedure body contains a collection of statements that define what the procedure does. It should always be enclosed between the reserved words begin and end. It is the part of a procedure where all computations are done.

Following is the source code for a procedure called *findMin*. This procedure takes 4 parameters x, y, z and m and stores the minimum among the first three variables in the variable named m. The variable m is passed by **reference** *wewilldiscusspassingargumentsbyreferencealittlelater* −

```
procedure findMin(x, y, z: integer; var m: integer);
(* Finds the minimum of the 3 values *)

begin
   if x < y then
      m := x
   else
      m := y;

   if z <m then
      m := z;
end; { end of procedure findMin }
```

Procedure Declarations

A procedure **declaration** tells the compiler about a procedure name and how to call the procedure. The actual body of the procedure can be defined separately.

A procedure declaration has the following syntax −

```
procedure name(argument(s): type1, argument(s): type 2, ... );
```

Please note that the **name of the procedure is not associated with any type**. For the above defined procedure *findMin*, following is the declaration −

```
procedure findMin(x, y, z: integer; var m: integer);
```

Calling a 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.

To call a procedure, you simply need to pass the required parameters along with the procedure name as shown below −

```
program exProcedure;
var
   a, b, c,  min: integer;
procedure findMin(x, y, z: integer; var m: integer);
(* Finds the minimum of the 3 values *)

begin
   if x < y then
      m:= x
   else
      m:= y;

   if z < m then
      m:= z;
end; { end of procedure findMin }

begin
   writeln(' Enter three numbers: ');
   readln( a, b, c);
   findMin(a, b, c, min); (* Procedure call *)

   writeln(' Minimum: ', min);
end.
```

When the above code is compiled and executed, it produces the following result −

```
Enter three numbers:
89 45 67
Minimum: 45
```

Recursive Subprograms

We have seen that a program or subprogram may call another subprogram. When a subprogram calls itself, it is referred to as a recursive call and the process is known as recursion.

To illustrate the concept, let us calculate the factorial of a number. Factorial of a number n is defined as −

```
n! = n*(n-1)!
   = n*(n-1)*(n-2)!
      ...
   = n*(n-1)*(n-2)*(n-3)... 1
```

The following program calculates the factorial of a given number by calling itself recursively.

```
program exRecursion;
var
   num, f: integer;
function fact(x: integer): integer; (* calculates factorial of x - x! *)

begin
   if x=0 then
      fact := 1
   else
      fact := x * fact(x-1); (* recursive call *)
```

```
end; { end of function fact}

begin
   writeln(' Enter a number: ');
   readln(num);
   f := fact(num);

   writeln(' Factorial ', num, ' is: ' , f);
end.
```

When the above code is compiled and executed, it produces the following result −

```
Enter a number:
5
Factorial 5 is: 120
```

Following is another example, which generates the **Fibonacci Series** for a given number using a **recursive** function −

```
program recursiveFibonacci;
var
   i: integer;
function fibonacci(n: integer): integer;

begin
   if n=1 then
      fibonacci := 0

   else if n=2 then
      fibonacci := 1

   else
      fibonacci := fibonacci(n-1) + fibonacci(n-2);
end;

begin
   for i:= 1 to 10 do

   write(fibonacci (i), '  ');
end.
```

When the above code is compiled and executed, it produces the following result −

```
0 1 1 2 3 5 8 13 21 34
```

Arguments of a Subprogram

If a subprogram (**function or procedure**) is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the subprogram.

The formal parameters behave like other local variables inside the subprogram and are created upon entry into the subprogram and destroyed upon exit.

While calling a subprogram, there are two ways that arguments can be passed to the subprogram −

Call Type	Description
Call by value	This method copies the actual value of an argument into the formal parameter of the subprogram. In this case, changes made to the parameter inside the subprogram have no effect on the argument. This method copies the address of an argument into the formal

Call by reference	parameter. Inside the subprogram, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, Pascal uses **call by value** to pass arguments. In general, this means that code within a subprogram cannot alter the arguments used to call the subprogram. The example program we used in the chapter 'Pascal - Functions' called the function named max using **call by value**.

Whereas, the example program provided here (*exProcedure*) calls the procedure findMin using **call by reference**.