PASCAL - FUNCTIONS

Subprograms

A subprogram is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design.' A subprogram can be invoked by a subprogram/program, which is called the calling program.

Pascal provides two kinds of subprograms −

- **Functions** − these subprograms return a single value.
- **Procedures** − these subprograms do not return a value directly.

Functions

A **function** is a group of statements that together perform a task. Every Pascal program has at least one function, which is the program itself, and all the most trivial programs can define additional functions.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

Pascal standard library provides numerous built-in functions that your program can call. For example, function **AppendStr** appends two strings, function **New** dynamically allocates memory to variables and many more functions.

Defining a Function

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

```
function name(argument(s): type1; argument(s): type2; ...): function_type;
local declarations;

begin
   ...
   < statements >
   ...
   name:= expression;
end;
```

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

- **Arguments** − The argument*s* establish the linkage between the calling program and the function identifiers and also called the formal parameters. A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of parameters of a function. Use of such formal parameters is optional. These parameters may have standard data type, user-defined data type or subrange data type.

  The formal parameters list appearing in the function statement could be simple or subscripted variables, arrays or structured variables, or subprograms.

- **Return Type** − All functions must return a value, so all functions must be assigned a type. The **function-type** is the data type of the value the function returns. It may be standard, user-defined scalar or subrange type but it cannot be structured type.

- **Local declarations** − Local declarations refer to the declarations for labels, constants, variables, functions and procedures, which are application to the body of function only.

- **Function Body** − The function body contains a collection of statements that define what the function does. It should always be enclosed between the reserved words begin and end. It is the part of a function where all computations are done. There must be an assignment statement of the type - **name := expression;** in the function body that assigns a value to the function name. This value is returned as and when the function is executed. The last statement in the body must be an end statement.

Following is an example showing how to define a function in pascal −

```
(* function returning the max between two numbers *)
function max(num1, num2: integer): integer;

var
   (* local variable declaration *)
   result: integer;

begin
   if (num1 > num2) then
      result := num1

   else
      result := num2;
   max := result;
end;
```

Function Declarations

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

A function declaration has the following parts −

```
function name(argument(s): type1; argument(s): type2; ...): function_type;
```

For the above-defined function max, following is the function declaration −

```
function max(num1, num2: integer): integer;
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a Function

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

To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value. Following is a simple example to show the usage −

```
program exFunction;
var
   a, b, ret : integer;

(*function definition *)
function max(num1, num2: integer): integer;
var
   (* local variable declaration *)
   result: integer;

begin
```

```
   if (num1 > num2) then
      result := num1

   else
      result := num2;
   max := result;
end;

begin
   a := 100;
   b := 200;
   (* calling a function to get max value *)
   ret := max(a, b);

   writeln( 'Max value is : ', ret );
end.
```

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

```
Max value is : 200
```