## 3.0 Introduction

Experienced programmer used to divide large (lengthy) programs in to parts, and then manage those parts to be solved one by one. This method of programming approach is to organize the typical work in a systematic manner. This aspect is practically achieved n C language thorough the concept known as 'Modular Programming".

The entire program is divided into a series of modules and each module is intended to perform a particular task. The detailed work to be solved by the module is described in the module (sub program) only and the main program only contains a series of modulus that are to be executed. Division of a main program in to set of modules and assigning various tasks to each module depends on the programmer's efficiency.

Whereas there is a need for us repeatedly execute one block of statements in one place of the program, loop statements can be used. But, a block of statements need to be repeatedly executed in many parts of the program, then repeated coding as well as wastage of the vital computer resource memory will wasted. . If we adopt modular programming technique, these disadvantages can be eliminated. The modules incorporated in C are called as the FUNCTIONS, and each function in the program is meant for doing specific task. C functions are easy to use and very efficient also.

## 3.1 Functions

**Definition**

A function can be defined as a subprogram which is meant for doing a specific task.

In a C program, a function definition will have name, parentheses pair contain zero or more parameters and a body. The parameters used in the parenthesis need to be declared with type and if not declared, they will be considered as of integer type.

The general form of the function is :

```
function type name <arg1,arg2,arg3, ————,argn>)
  data type  arg1, arg2,;
  data type argn;
    {
          body of function;
```

```
                    ______________________
                    ______________________
                    ______________________
                    return (<something>);
          }
```

From the above form the main components of function are

- Return type
- Function name
- Function body
- Return statement

**Return Type**

Refers to the type of value it would return to the calling portion of the program. It can have any of the basic data types such as int, float, char, etc. When a function is not supposed to return any value, it may be declared as type void

**Example**

```
void function name(- - - - - - - - - -);
int function name( - - - - - - - - - - );
char function name ( — - - - - - - );
```

**Function Name**

The function name can be any name conforming to the syntax rules of the variable.

A function name is relevant to the function operation.

**Example**

```
output( );
read data( );
```

**Formal arguments**

The arguments are called **formal arguments (or) formal parameters**, because they represent the names of data items that are transferred into the function from the calling portion of the program.

Any variable declared in the body of a function is said to be local to that function, other variable which were not declared either arguments or in the function body, are considered "**globol**" to the function and must be defined externally.

**Example**

```
int biggest (int a, int b)
{
___________________________
___________________________
___________________________
return( );
}
```

a, b are the formal arguments.

**Function Body**

Function body is a compound statement defines the action to be taken by the function. It should include one or more **"return"** statement in order to return a value to the calling portion of the program.

**Example**

```
int biggest(int a, int b)
{
if ( a > b)
return(a); body of function.
else
return(b);
}
```

Every C program consists of one or more functions. One of these functions must be called as **main.** Execution of the program will always begin by carrying out the instructions in main. Additional functions will be subordinate to main. If a program contains multiple functions, their definitions may appear in any order, though they must be independent of one another. That is, one function definition can't be embedded within another.

Generally a function will process information that is passed to it from the calling portion of the program and return a single value. Information is passed to the function via arguments (**parameters**) and returned via the **"return"** statement.

Some functions accept information but do not return anything (***ex:*** printf()) whereas other functions (***ex:*** scanf()) return multiple values.

## 3.1.1 The Return Statement

Every function subprogram in C will have return statement. This statement is used in function subprograms to return a value to the calling program/function. This statement can appear anywhere within a function body and we may have more than one return statement inside a function.

The general format of return statement is

```
return;
(or)
return (expression);
```

If no value is returned from function to the calling program, then there is no need of return statement to be present inside the function.

**Programs using function Call Techniques**

**Example 1**: Write a program to find factorial to the given positive integer ,using function technique.

```
# include <stdio.h>
main( )
{
    int n;
    printf ( " Enter any positive number\n");
    scanf( "%d", &n);
```

```
        printf( " The factorial of %d s %d \n",fact (n));
    }
    fact( i)
    int I;
    {
        int j; f = 1 ;
        for ( j = I; j>0; j - -)
        f  = f * I;
        return ( f ) ;
    }
```

In the above program function with name 'fact' is called by the main program. The function fact is called with n as parameter. The value is returned through variable f to the main program.

**Example 2**: Write a program to find the value of f(x) as $f(x) = x^2 + 4$, for the given of x. Make use of function technique.

```
     # include <stdio.h>
     main( )
    {
            f ( );
    }
    f ( )
     {  int  x,y ;
     printf( " Enter value of x \n");
     scanf( " %d", & x );
      y   = (x * x + 4);
     printf ( " The value of f (x) id %d \n", y ) ;
    }
```

## 3.2 Differences between Function and Procedures

| Procedure | Function |
|---|---|
| 1. Procedure is a sub program which is included with in main program. | 1. Functions is sub program which is intended for specific task. Eg. sqrt() |
| 2. Procedure donot return a value. | 2. Functions may or may not return a value. |
| 3. Procedure cannot be called again and again. | 3. Function once defined can be called any where n number of times. |
| 4. Global variables cannot be used in procedure. | 4. In functions both local and global variables can be used. |
| 5. Procedures can be writ-ten only in procedural pro-gramming such as Dbase, Foxpro. | 5. Functions can be written in modular programming such as C, C++ |

## 3.3 Advantages of Function

**The main advantages of using a function are:**

- Easy to write a correct small function
- Easy to read and debug a function.
- Easier to maintain or modify such a function
- Small functions tend to be self documenting and highly readable
- It can be called any number of times in any place with different parameters.

**Storage class**

A variable's storage class explains where the variable will be stored, its initial value and life of the variable.

**Iteration**

The block of statements is executed repeatedly using loops is called Iteration

**Categories of Functions**

A function, depending on, whether arguments are present or not and a value is returned or not.

A function may be belonging to one of the following types.

1. Function with no arguments and no return values.

2. Function with arguments and no return values.

3. Function with arguments and return values

## 3.4 Advanced Featured of Functions

a. Function Prototypes

b. Calling functions by value or by reference

c. Recursion.

**a. Function Prototypes**

The user defined functions may be classified as three ways based on the formal arguments passed and the usage of the **return** statement.

a. Functions with no arguments and no return value

b. Functions with arguments no return value

c. Functions with arguments and return value.

**a. Functions with no arguments and no return value**

A function is invoked without passing any formal arguments from the calling portion of a program and also the function does not return back any value to the called function. There is no communication between the calling portion of a program and a called function block.

**Example:**

```
#include <stdio.h>
main()
{
        void message( ); Function declaration
        message( ); Function calling
}
```

```
void message( )
{
        printf (“GOVT JUNIOR  COLLEGE \n”);
        printf (“\t HYDERABAD”);
}
```

**b. Function with arguments and no return value**

This type of functions passes some formal arguments to a function but the function does not return back any value to the caller. It is any one way data communication between a calling portion of the program and the function block.

***Example***

```
#include <stdio.h>
main()
{
        void square(int);
        printf (“Enter a value for n \n”);
        scanf (“%d”,&n);
        square(n);
}
void square (int n)
{
        int value;
        value = n * n;
        printf (“square of %d is %d “,n,value);
}
```

**c.  Function with arguments and return value**

The third type of function passes some formal arguments to a function from a calling portion of the program and the computer value is transferred back to the caller. Data are communicated between the calling portion and the function block.

**Example**

```
#include <stdio.h>
main()
{
        int square (int);
        int value;
        printf (“enter a value for n \n”);
        scanf(“%d”, &n);
        value = square(n);
        printf (“square of %d is %d “,n, value);
}
int square(int n)
{
        int p;
        p = n * n;
        return(p);
}
```

The keyword **VOID** can be used as a type specifier when defining a function that does not return anything or when the function definition does not include any arguments.

The presence of this keyword is not mandatory but it is good programming practice to make use of this feature.

**Actual and Formal Parameters (or) Arguments**

Function parameters are the means of communication between the calling and the called functions. The parameters may classify under two groups.

1. Formal Parameters
2. Actual Parameters

**1. Formal Parameters**

The formal parameters are the parameters given in function declaration and function definition. When the function is invoked, the formal parameters are replaced by the actual parameters.

**2. Actual Parameters**

The parameters appearing in the function call are referred to as actual parameters. The actual arguments may be expressed as constants, single variables or more complex expression. Each actual parameter must be of the same data type as its corresponding formal parameters.

**Example**

```
#include <stdio.h>
int sum (int a , int b )
{
        int c;
        c = a + b;
        return(c);
}
main( )
{
        intx,y,z;
        printf (“enter value for x,y \n”);
        scanf (“%d %d”,&x,&y);
        z = x + y;
        printf (“ sum is = %d”,z);
}
```

The variables **a and b** defined in function definition are known as **formal parameters**. The variables **x and y** are **actual parameters**.

## Local and Global Variable:

The variables may be classified as local or global variables.

### Local Variable

The variables defined can be accessed only within the block in which they are declared. These variables are called “Local” variables

### Example

```
funct (int ,int j)
{
        intk,m;
        ________;
        ________;
}
```

The integer variables **k** and **m** are defined within a function block of the “**funct()**”. All the variables to be used within a function block must be either defined at the beginning of the block or before using in the statement. Local variables one referred only the particular part of a block of a function.

### Global Variable

Global variables defined outside the main function block. Global variables are not contained to a single function. Global variables that are recognized in two or more functions. Their scope extends from the point of definition through the remainder of the program.

### b. Calling functions by value or by reference

The arguments are sent to the functions and their values are copied in the corresponding function. This is a sort of information inter change between the calling function and called function. This is known as Parameter passing. It is a mechanism through which arguments are passed to the called function for the required processing. There are two methods of parameter passing.

1. Call by Value
2. Call by reference.

**1. Call by value:** When the values of arguments are passed from calling function to a called function, these values are copied in to the called function. If