FUNCTION

A FUNCTION IS SELF-CONTAINED BLOCK OF CODE THAT PERFORMS A PARTICULAR TASK.

A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function −

- **Return Type** − A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **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 the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** − The function body contains a collection of statements that define what the function does.

- FUNCTION NAME
- FUNCTION TYPE
- LIST OF PARAMETERS
- LOCAL VARIABLE DECLARATIONS
- FUNCTION STATEMENT
- A RETURN STATEMENT

A Function which invokes other function is known as **calling function** and function which is invoked by other function is knows as C**alled function.**

Calling a Function

While creating a C 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, the program control is transferred to the called function. A called function performs a defined task and when

its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example –

```
#include <stdio.h>
#include <conio.h>

/* function declaration */
int max(int num1, int num2);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
```

```
    result = num1;
  else
    result = num2;

  return result;
}
```

OUTPUT : **Max value is : 200**

Difference between Function Definition and Declaration

Function Definition

- There is no semicolon at the end of the closing paranthesis of the parameter-list.
- The body of the function follows it.
- Mandatory for all functions.

Function declaration

- There is a semicolon at the end of the closing paranthesis of parameter list.
- The body of the function does not follow it.
- Optional for function returning **int** value.

Types of Functions

Based on the nature of the creation, the functions are divides as

1. **User-defined functions and**
2. **Built-in functions**

Built-in functions are predefined and supplied along with the compiler and these can be used in any C program. They are also known as **Library Functions.**

Function Definition

A function definition describes what a function does, how its actions are achieved and how it is used. It consists of a function header and function statements.

SYNTAX :

```
Function type   function Name (Parameters   List)
{
        Local variable declaration;
        Executable statement;
        Executable Statement;
            ……
        Return Statement
}
```

The advantages of using functions are:

- **Avoid repetition of codes.**
- **Increases program readability.**
- **Divide a complex problem into simpler ones.**
- **Reduces chances of error.**
- **Modifying a program becomes easier by using function.**

There are very few disadvantages to using functions in C.

- **Complexity of the program increases.**
- **Execution speed decreases.**
- **It requires a programmer must be expert in programming.**

Category of Functions :

- **Category 1 : Functions with no arguments and no return values.**
- **Category 2 : Functions with arguments and no return values.**
- **Category 3 : Functions with arguments and one return value.**
- **Category 4 : Functions with no arguments but return a value.**
- **Category 5 : Functions that return multiple values.**

CATOGORY 1 :

SAMPLE PROGRAM :

/* FACTORIAL NUMBER USING FUNCTON*/

```
#include<stdio.h>
#include<conio.h>
 void main ( )
 {
        int f, n, i=1;
        printf (“Enter the Number”);
        scanf (“%d”,&n);
        fun ( );
  }
void  fun ( )
{
        int  f=1,  i;
       for ( i=1;  i <= n;  i++)
       {
            f = f * i;
        }
        printf  (“ The factorial number is “, f);
}
```

CATOGORY 2 :

```
/* BIGGEST VALUE OF THREE NUMBERS USING FUNCTION */
 # include <stdio.h>
# include<conio.h>
void main ( )
{
        int a, b, c;
        void big(int, int, int);
        printf ("Enter three numbers");
        scanf ("%d  %d %d ", &a, &b, &c);
        big (a, b, c);
}
void   big (int  x, int  y, int  z)
{
      int  (x > y  && x  > z)
            printf (" A is Biggest ");
      else  if ( y  >  z   &&  y  > c )
              printf ("B is Biggest");
        else
          printf ("C is biggest");
   }
```

CATOGORY 3 :

```
/* AREA OF A TRIANGLE USING FUNCTION */

 # include <stdio.h>

# include<conio.h>

void main ( )

{

            float x, y, c, area( ) ;

             printf (“Enter Base and Height”);

             scanf (“%f  %f  “, &x, &y);

            c = area (x , y);

            printf ( “The area is  %f”, c);

}

float area ( float b, float h)

{

        return ( 0.5 * b * h);

}
```