Module – III

Functions: Defining and accessing: Passing arguments, Function prototypes, Function calls- Categories of functions- Nesting of functions- Recursion. Use of library functions, Scope , Visibility and Lifetime of variables.

C - Functions

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

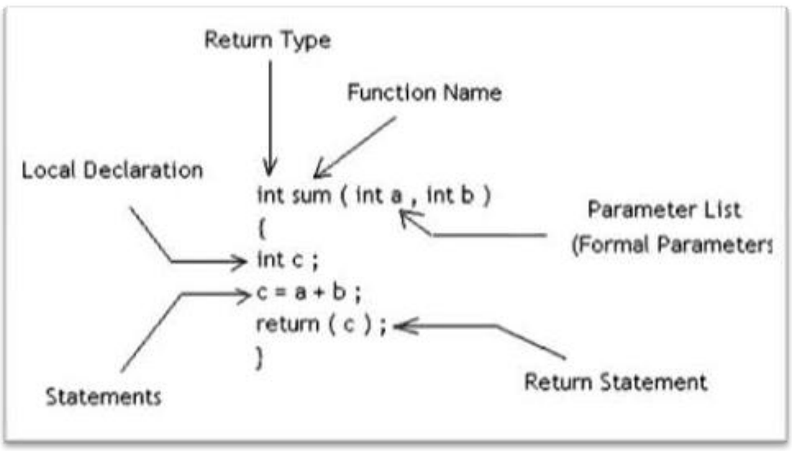
You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

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.

Defining a Function

The general form of a function definition in C programming language is as follows − return_type function_name(parameter list) {
  body of the function
}

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.

Example

Given below is the source code for a function called max(). This function takes two parameters num1 and num2 and returns the maximum value between the two −

```
int max(int num1, int num2)
{
  int result;
  if (num1 > num2)
    result = num1;
  else
    result = num2; return result;
}
```

Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways in which arguments can be passed to a function –

Sr.No.	**Call Type & Description**
1	**Call by value** This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
2	**Call by reference** This method copies the address of an argument into the formal parameter. Inside the function, 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, C uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

Function prototype

The function prototypes are used to tell the compiler about the number of arguments and about the required data types of a function parameter, it also tells about the return type of the function. By this information, the compiler cross-checks the function signatures before calling it. If the function prototypes are not mentioned, then the program may be compiled with some warnings, and sometimes generate some strange output.

Example Code

```
#include<stdio.h>
void function(int); //prototype
main() {
  function(50);
}
void function(int x) {
  printf("The value of x is: %d", x);
}
```

Output

```
The value of x is: 50
```

Categories of Functions

Depending on whether arguments are present or not and whether a value is returned or not, functions are categorized into −

- Functions without arguments and without return values
- Functions without arguments and with return values
- Functions with arguments and without return values
- Functions with arguments and with return values

C Recursion

In this tutorial, you will learn to write recursive functions in C programming with the help of an example.

A function that calls itself is known as a recursive function. And, this technique is known as recursion.
The recursion continues until some condition is met to prevent it.

To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call, and other doesn't.

Example: Factorial function using recursion

```
#include<stdio.h>

long factorial(int n)
{
  if (n == 0)
    return 1;
  else
    return(n * factorial(n-1));
}

void main()
{
  int number;
  long fact;
  printf("Enter a number: ");
  scanf("%d", &number);

  fact = factorial(number);
  printf("Factorial of %d is %ld\n", number, fact);
  return 0;
}
```

Initially, the `factorial` () is called from the main() function with number passed as an argument.

5! = 5*4*3*2*1 = 120

Library functions are built-in functions

Library functions are built-in functions that are grouped together and placed in a common location called library. Each function here performs a specific operation. We can use this library functions to get the pre-defined output.

All C standard library functions are declared by using many header files. These library functions are created at the time of designing the compilers. We include the header files in our C program by using #include<filename.h>. Whenever the program is run and executed, the related files are included in the C program.

Header File Functions

Some of the header file functions are as follows −

stdio.h − It is a standard i/o header file in which Input/output functions are declared
conio.h − This is a console input/output header file.
string.h − All string related functions are in this header file.
stdlib.h − This file contains common functions which are used in the C programs.
math.h − All functions related to mathematics are in this header file.
time.h − This file contains time and clock related functions.Built functions in stdio.h

C - Scope of variable

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language −

- Inside a function or a block which is called local variables.
- Outside of all functions which is called global variables.
- In the definition of function parameters which are called formalparameters.

Let us understand what are local and global variables, and formalparameters.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. The following example shows how local variables are used. Here all the variables a, b, and c are local to main() function.

Global Variables

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program. A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. The following program show how global variables are used in a program.

```
#include <stdio.h>
        /* global variable declaration */
int g;
void main () {
 /* local variable
 declaration */ int
 a, b;
 /* actual
 initializati
 on */ a =
 10;
 b = 20;
 g = a + b;
 printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
}
```