Unit-6
Functions
(Marks -8)

CTEVT Diploma in Computer Engineering
Subject : C Programming
Shree Sarwajanik Secondary Technical School
Prepared By: Er. Abinash Adhikari

C Functions

A function in C is a set of statements that when called perform some specific tasks. It is the basic building block of a C program that provides modularity and code reusability. The programming statements of a function are enclosed within { } braces, having certain meanings and performing certain operations. They are also called subroutines or procedures in other languages.

Syntax of Functions in C

The syntax of function can be divided into 3 aspects:

1. Function Declaration
2. Function Definition
3. Function Calls

Function Aspects

There are three aspects of a C function.

1. **Function declaration:**
   - ➢ A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.
2. **Function call:**
   - ➢ Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.
3. **Function definition:**
   - ➢ It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

1. Function declaration

In a function declaration, we must provide the function name, its return type, and the number and type of its parameters. A function declaration tells the compiler that there is a function with the given name defined somewhere else in the program.

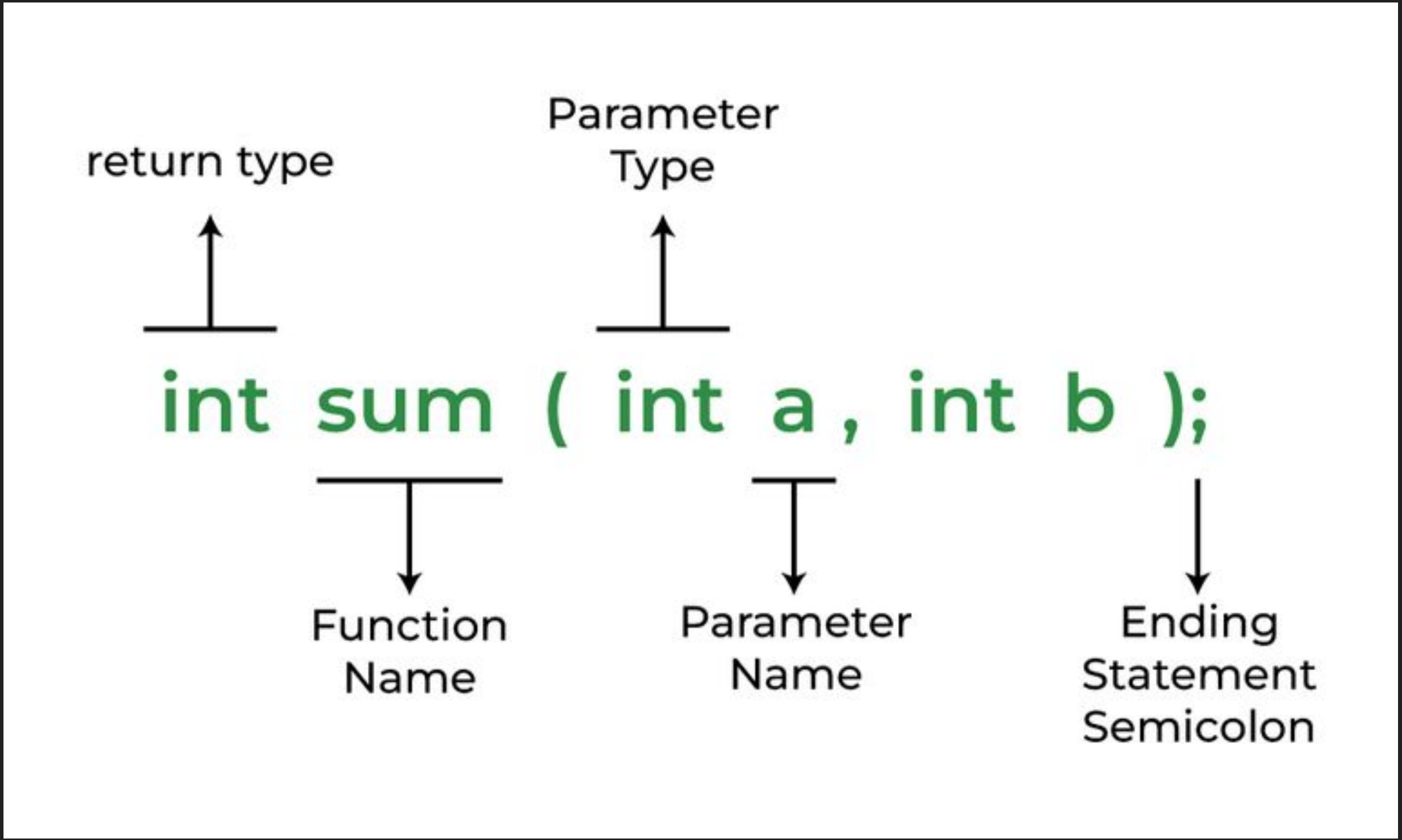
Syntax:

❖ return_type name_of_the_function (parameter_1, parameter_2);

The parameter name is not mandatory while declaring functions. We can also declare the function without using the name of the data variables.

Example:

❖ int sum(int a, int b); // Function declaration with parameter names
❖ int sum(int , int);   // Function declaration without parameter names



Note: A function in C must always be declared globally before calling it.

Function Definition

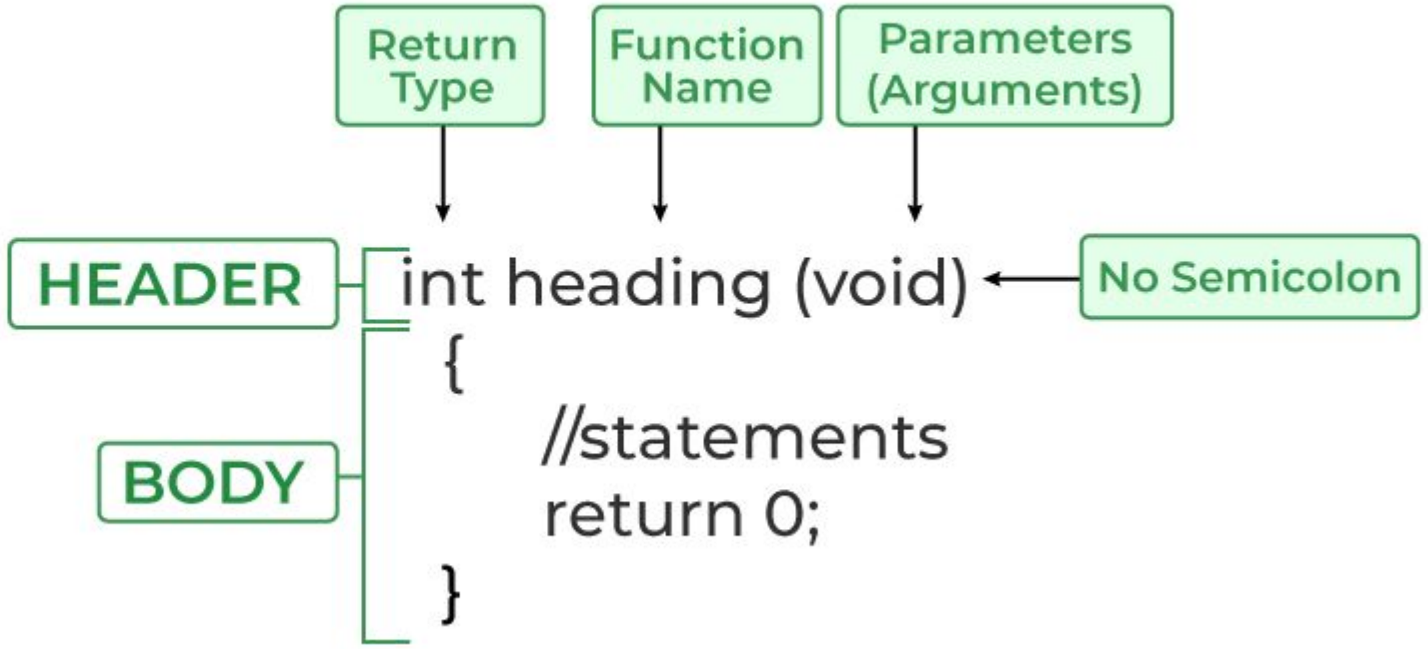
The function definition consists of actual statements which are executed when the function is called (i.e. when the program control comes to the function).

A C function is generally defined and declared in a single step because the function definition always starts with the function declaration so we do not need to declare it explicitly. The below example serves as both a function definition and a declaration.

Syntax:

```
return_type function_name (para1_type para1_name, para2_type
para2_name)
{
   // body of the function
}
```

Function Definition



```
int heading (void)
 {
      //statements
      return 0;
 }
```

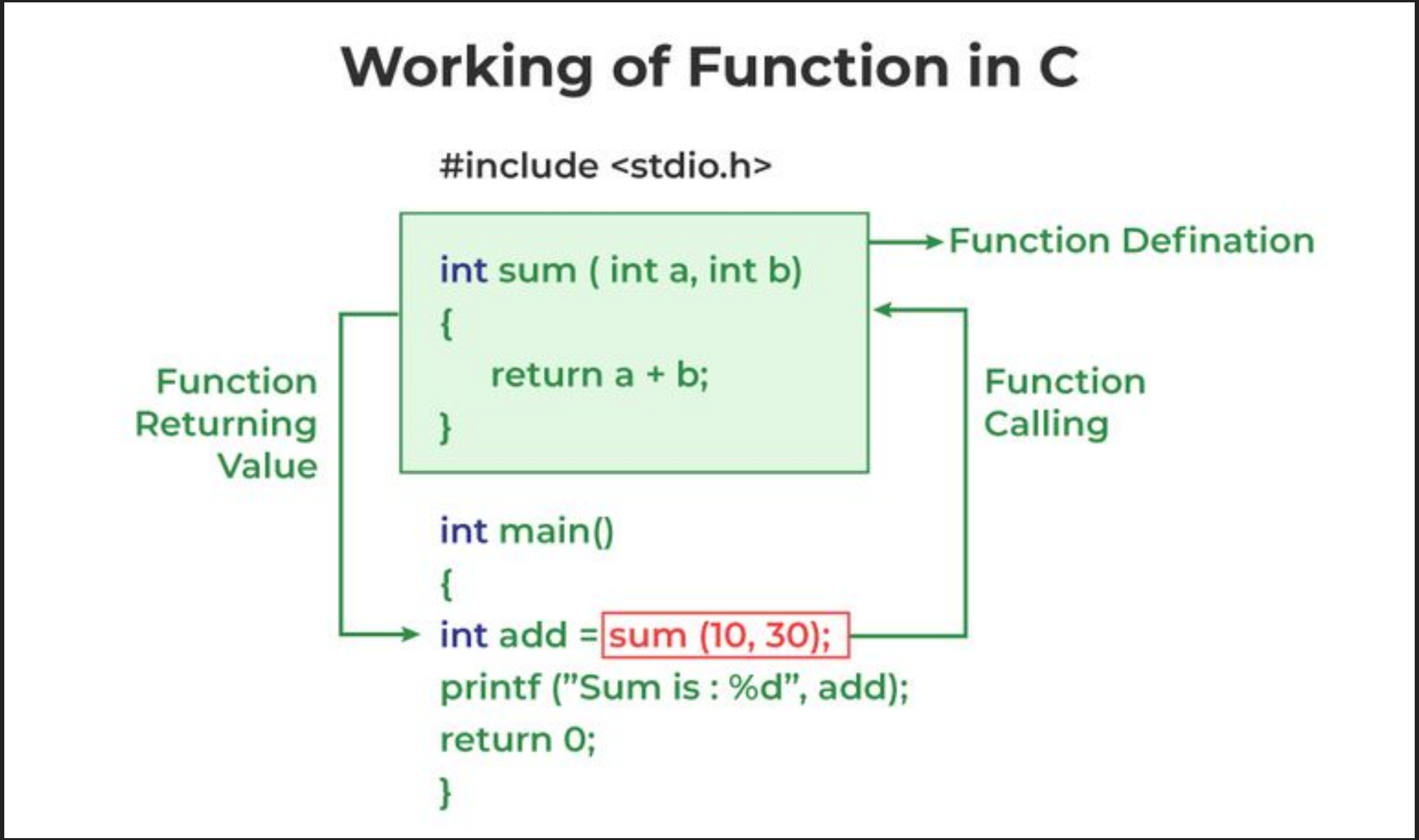
Function Call

A function call is a statement that instructs the compiler to execute the function. We use the function name and parameters in the function call.

In the below example, the first sum function is called and 10,30 are passed to the sum function. After the function call sum of a and b is returned and control is also returned back to the main function of the program.

Syntax:

```
int add = sum(a, b);
```

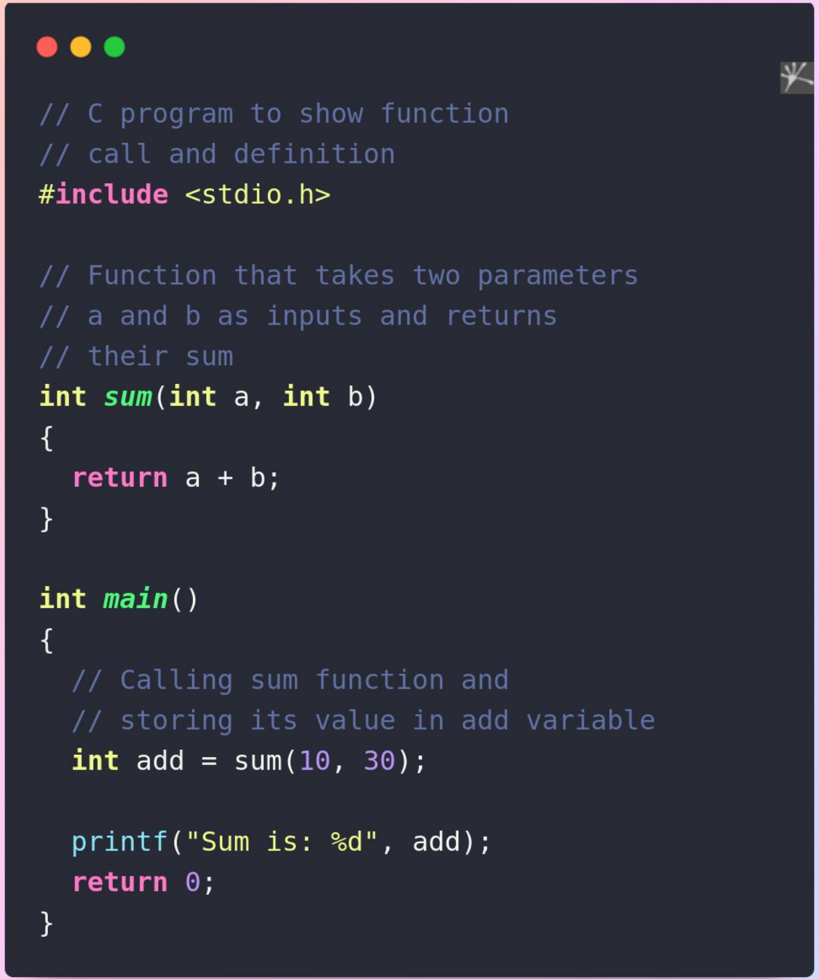


```
#include <stdio.h>

int sum ( int a, int b)
{
    return a + b;
}

int main()
{
int add = sum (10, 30);
printf ("Sum is : %d", add);
return 0;
}
```

Note: Function call is necessary to bring the program control to the function definition. If not called, the function statements will not be executed.



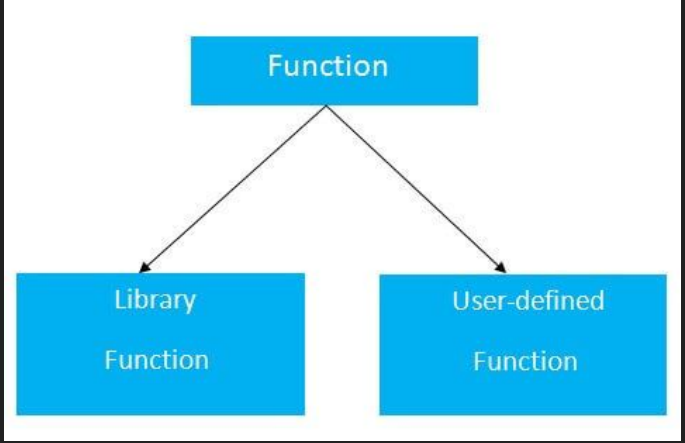
```c
// C program to show function
// call and definition
#include <stdio.h>

// Function that takes two parameters
// a and b as inputs and returns
// their sum
int sum(int a, int b)
{
  return a + b;
}

int main()
{
  // Calling sum function and
  // storing its value in add variable
  int add = sum(10, 30);

  printf("Sum is: %d", add);
  return 0;
}
```

Types of Functions



There are two types of functions in C programming:

1. **library/built-in Functions:** library/built-in Functions are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.
2. **User-defined functions:** User-defined functions are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.

1. Library Function

A library function is also referred to as a “built-in function”. A compiler package already exists that contains these functions, each of which has a specific meaning and is included in the package. Built-in functions have the advantage of being directly usable without being defined, whereas user-defined functions must be declared and defined before being used.

For Example:

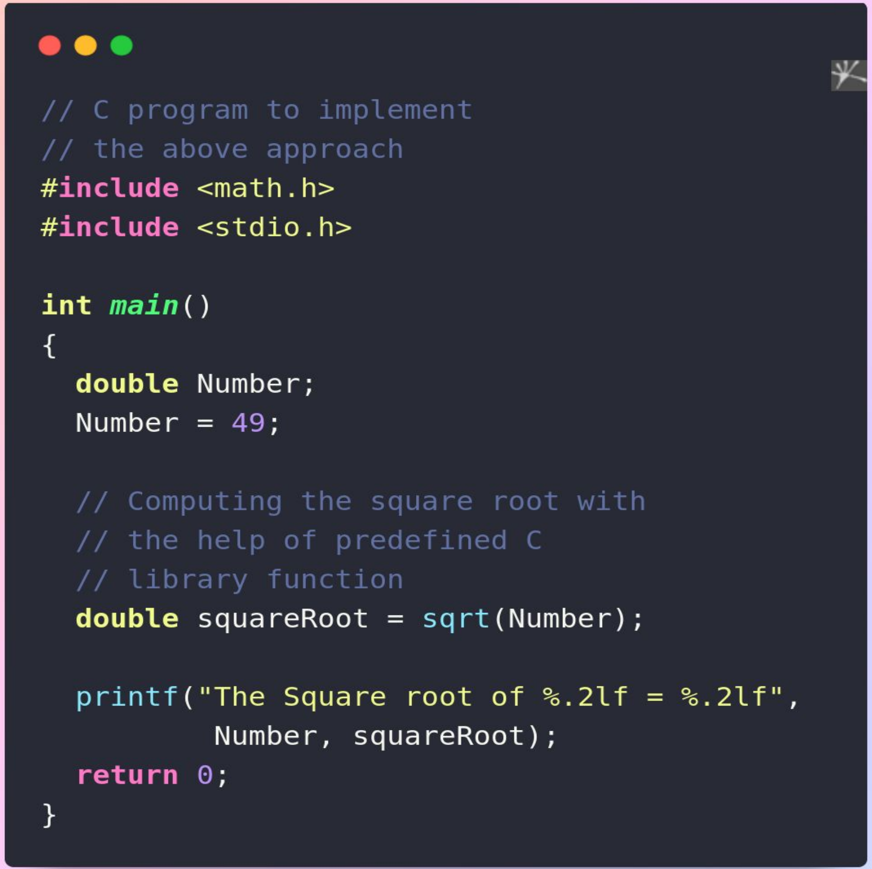
❖ pow(), sqrt(), strcmp(), strlen() etc.

Advantages of C library functions:

❖ C Library functions are easy to use and optimized for better performance.

❖ C library functions save a lot of time i.e, function development time.

❖ C library functions are convenient as they always work.



```c
// C program to implement
// the above approach
#include <math.h>
#include <stdio.h>

int main()
{
  double Number;
  Number = 49;

  // Computing the square root with
  // the help of predefined C
  // library function
  double squareRoot = sqrt(Number);

  printf("The Square root of %.2lf = %.2lf",
          Number, squareRoot);
  return 0;
}
```

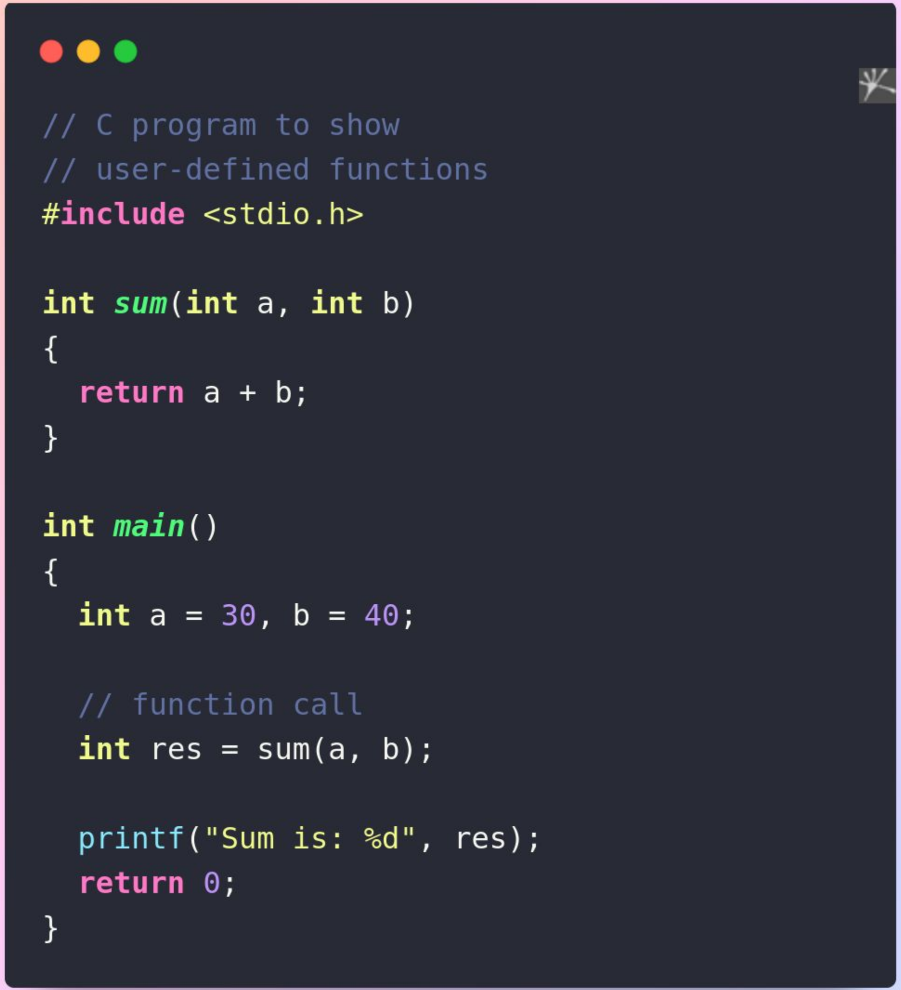
Output: The Square root of 49.00 = 7.00

2. User Defined Function

Functions that the programmer creates are known as User-Defined functions or “tailor-made functions”. User-defined functions can be improved and modified according to the need of the programmer. Whenever we write a function that is case-specific and is not defined in any header file, we need to declare and define our own functions according to the syntax.

Advantages of User-Defined Functions

1. Changeable functions can be modified as per need.
2. The Code of these functions is reusable in other programs.
3. These functions are easy to understand, debug and maintain.



```c
// C program to show
// user-defined functions
#include <stdio.h>

int sum(int a, int b)
{
  return a + b;
}

int main()
{
  int a = 30, b = 40;

  // function call
  int res = sum(a, b);

  printf("Sum is: %d", res);
  return 0;
}
```

Advantage of functions in C

There are the following advantages of C functions.

1. By using functions, we can avoid rewriting same logic/code again and again in a program.
2. We can call C functions any number of times in a program and from any place in a program.
3. We can track a large C program easily when it is divided into multiple functions.
4. Reusability is the main achievement of C functions.
5. However, Function calling is always a overhead in a C program.

Categories of Functions (Based on Return Value and Arguments)

- Functions in C can be categorized based on whether they return a value and whether they accept arguments:
- No return value and no arguments:
  - Example:

```
void display() {
   printf("This is a function without arguments.\n");
}
```

- No return value but with arguments:
  - Example:

```
void printNumber(int num) {
   printf("Number: %d\n", num);
}
```

- With return value but no arguments:
  - Example:

```
int getNumber() {
   return 42;
}
```

- With return value and with arguments:
  - Example:

```
int multiply(int a, int b) {
   return a * b;
}
```

Parameter Passing in C

Parameters allow functions to accept inputs and perform operations using those inputs. In C, there are two ways to pass parameters to functions:

1. **Call by Value:**
   a. A copy of the argument's value is passed to the function.
   b. Changes made to the parameter inside the function do not affect the original variable.
   c. Example:
```
void changeValue(int x) {
    x = 10; // Changes the copy, not the original variable
}
```

2. **Call by Reference:**
   a. The address of the argument is passed to the function, allowing it to modify the original variable.
   b. Achieved using pointers.
   c. Example:
```
void changeValue(int *x) {
    *x = 10; // Modifies the original variable
}
```

Recursion (Recursive Function)

Recursion occurs when a function calls itself. It is useful for problems that can be divided into smaller, similar subproblems (e.g., factorials, Fibonacci series).

Example: Factorial using Recursion

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

Key points about recursion:

- ❖ Every recursive function must have a base condition to terminate.
- ❖ Excessive recursion may cause stack overflow.

Passing Array to Function

An entire array can be passed to a function by passing its base address. Changes made to the array within the function affect the original array.

Example: Summing Array Elements

```
int sumArray(int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum;
}
```

Passing String to Function

Strings in C are arrays of characters terminated by a null character (\0). Functions can manipulate strings by accepting their base address as arguments.

Example: Counting String Length

```
int stringLength(char str[]) {

    int length = 0;

    while (str[length] != '\0') {

        length++;

    }

    return length;

}
```