# FUNCTION IN C

Dr. Sumit Srivastava
Dept. of CSE, BIT Mesra Ranchi
Email:- sumit@bitmesra.ac.in





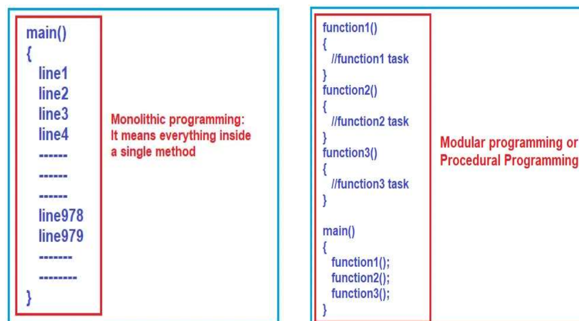
## Function

- A function is a block of code which only runs when it is called.
- Function is nothing but a group of codes put together and given a name. And these can be called anytime without writing the whole code again and again.
- Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times





## Function








## Example





## Function (Example)

- Program: Adding Two Numbers

```
#include <stdio.h>
int main ()
{
    int x, y;
    x = 10;
    y = 5;
    int z = x + y;
    printf ("sum is %d", z);
}
```

we have implemented the logic to add two numbers inside the main function only.

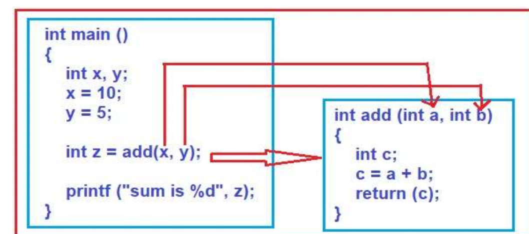




## Function (Example)

- Program: Adding Two Numbers

(Let us see how to write the same Program using Function)



## Function (Example)

- We created a function called add which takes two input parameters a and b of type integer. This add function adds the two integer numbers it received as input parameters and stores the result in variable c and returns that result.
- Now see the main function. From the main function, we are calling the add function and while calling the add function we are passing two parameters i.e. x and y (actually we are passing the values stored in x and y) and these parameters' values will go into a and b. The add function then adds these two values and returns the result to the calling function (the function is called the add method) i.e. the main method. The main method then store the result coming from the add method into the variable z and then print the result on the output window.

Sumit Srivastava @ BIT Mesra

7

## Function (Example)

- Program: Adding Two Numbers

```
#include <stdio.h>
int add (int a, int b)
{
    int c;
    c = a + b;
    return (c);
}
int main ()
{
    int x, y;
    x = 10;
    y = 5;
    int z = add (x, y);
    printf ("sum is %d", z);
}
```

Sumit Srivastava @ BIT Mesra

8

## Different Parts of a Function:

```
int add (int a, int b)        <= Prototype or Function Signature
{
  int c;
  c = a + b;                  <= Function Definition or Function Body
  return (c);
}
int main ()
{
  int x, y;
  x = 10;
  y = 5;
  int z = add (x, y);         <= Function Call
  printf ("sum is %d", z);
}
```

Sumit Srivastava @ BIT Mesra

9

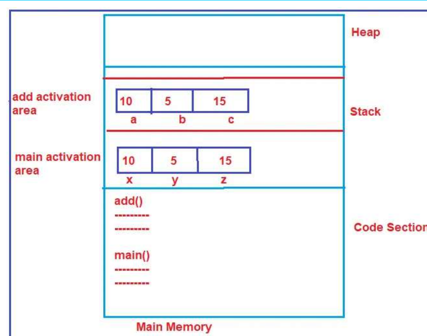
## Function (Example)

```
int add (int a, int b)        <- Formal Parameters
{
  int c;
  c = a + b;
  return (c);
}
int main ()
{
  int x, y;
  x = 10;
  y = 5;                      <- Actual Parameters
  int z = add (x, y);
  printf ("sum is %d", z);
}
```

Sumit Srivastava @ BIT Mesra

10

## How does it work inside the main memory?




Sumit Srivastava @ BIT Mesra

11

## What are Functions in C Language?

- A function in C is a self-contained program segment that carries out some specific, well-defined task.
- A-C Program is made of one or more functions, one of which must be named as the main. The execution of the program always starts and ends with the main, but it can call other functions to do special tasks.
- A function will carry out its intended action whenever it is called from some other portion (called calling function) of the program. Once the function (called function) has carried out its intended action, control will be returned to the point from which the function was called.

Sumit Srivastava @ BIT Mesra

12

# Function





## Types of Functions in C Language

- There are two types of functions in C Programming Language.
  1. **Library 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:** 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.





## Standard library functions

The standard library functions are built-in functions in C programming. These functions are defined in header files.

**For example,**

- The **printf()** is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the **stdio.h header file**.
  Hence, to use the printf()function, we need to include the stdio.h header file using #include <stdio.h>.
- The **sqrt()** function calculates the square root of a number. The function is defined in the **math.h** header file.





## User-defined function

- You can also create functions as per your need. Such functions created by the user are known as user-defined functions.
- **Example:**

```
#include <stdio.h>
void functionName()
{
   ... .. ...
   ... .. ...
}

int main()
{
   ... .. ...
   ... .. ...

   functionName();

   ... .. ...
   ... .. ...
}
```





## User-defined function

You can also create functions as per your need. Such functions created by the user are known as user-defined functions.
**Example:**

```
#include <stdio.h>
void functionName()
{
   ... .. ...
   ... .. ...
}

int main()
{
   ... .. ...
   ... .. ...

   functionName();

   ... .. ...
   ... .. ...
}
```





How function works in C programming?

```
#include <stdio.h>

void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

```c
#include <stdio.h>
int addNumbers(int a, int b);        // function prototype (Declaration)

int main()
{
   int n1,n2,sum;

   printf("Enters two numbers: ");
   scanf("%d %d",&n1,&n2);

   sum = addNumbers(n1, n2);       // function call
   printf("sum = %d",sum);

   return 0;
}

int addNumbers(int a, int b)       // function definition
{
   int result;
   result = a+b;
   return result;              // return statement
}
```





# Function Aspects

- There are three aspects of a C function.
  1. **Function Declaration**
  2. **Function Definition**
  3. **Function Calls**





# Function Aspects

- **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.
- **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.
- **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.





# Function Aspects

| SN | C function aspects | Syntax |
|---|---|---|
| 1 | Function declaration | return_type function_name (argument list); |
| 2 | Function call | function_name (argument_list) |
| 3 | Function definition | return_type function_name (argument list) {function body;} |





# Function Declaration (Prototype)

- A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.
- A function prototype gives information to the compiler that the function may later be used in the program.

- **Syntax of function prototype**

**returnType functionName(type1 argument1, type2 argument2, ...);**





# Function Declaration (Prototype)

- *In the above example, int addNumbers(int a, int b); is the function prototype which provides the following information to the compiler:*

- *name of the function is addNumbers()*
- *return type of the function is int*
- *two arguments of type int are passed to the function*

# Function Definition

- Function definition contains the block of code to perform a specific task. In our example, adding two numbers and returning it.

**Syntax of function definition**

```
returnType functionName(type1 argument1, type2 argument2, ...)
{
   //body of the function
}
```

Sumit Srivastava @ BIT Mesra

25

# Function Definition

- To create (Definition) your own function, specify the name of the function, followed by parentheses () and curly brackets {}.

**Example**

```
void myFunction()
{
     // code to be executed
}
```

Sumit Srivastava @ BIT Mesra

26

# Return Value

- A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

Example without return value

```
void hello()
{
    printf("hello c");
}
```

Sumit Srivastava @ BIT Mesra

27

# Return Value

- If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.

Example with return value

```
int get()
{
   return 10;
}
```

```
float get()
{
   return 10.2;
}
```

Sumit Srivastava @ BIT Mesra

28

# Call a Function

- Declared functions are not executed immediately. They are "saved for later use" and will be executed when they are called.
- To call a function, write the function's name followed by two parentheses () and a semicolon ;.

Sumit Srivastava @ BIT Mesra

29

# Function Call

- Control of the program is transferred to the user-defined function by calling it.

**Syntax of function definition**

```
functionName(argument1, argument2, ...);
```

*In the example, the function call is made using addNumbers(n1, n2); statement inside the main() function.*

Sumit Srivastava @ BIT Mesra

30

## Call a Function (Example)

- Inside main, call myFunction():

```
// Create a function
void myFunction()
{
  printf("I just got executed!");
}

int main()
{
  myFunction(); // call the function
  return 0;
}

// Outputs "I just got executed!"
```





## Call a Function

- A function can be called multiple times.
- **Example:**

```
void myFunction() {
  printf("I just got executed!");
}

int main() {
  myFunction();
  myFunction();
  myFunction();
  return 0;
}

// I just got executed!
// I just got executed!
// I just got executed!
```





## Passing Arguments to a Function

- In programming, argument refers to the variable passed to the function. In the example, two variables n1 and n2 are passed during the function call.
- The parameters a and b accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.
- A function can also be called without passing an argument.





## Passing Arguments to a Function

How to pass arguments to a function?

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    ... .. ...
}
```





## Return Statement

- The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement.
- In the above example, the value of the result variable is returned to the main function. The sum variable in the main() function is assigned this value.



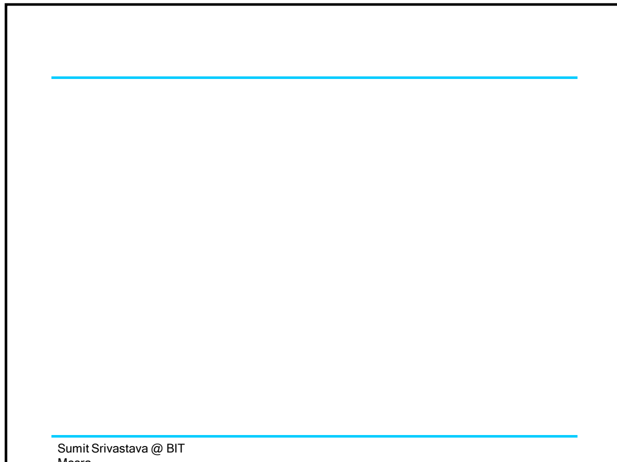


## Return Statement

Return statement of a Function

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    return result;
}
```

sum = result





37

# Function (Example)

```
#include <stdio.h>

// Function declaration
void myFunction();

// The main method
int main() {
  myFunction();  // call the function
  return 0;
}

// Function definition
void myFunction() {
  printf("I just got executed!");
}
```



38

# Function Arguments

- When a function is called the values of the arguments are passed to the function and stored in variables called formal parameters.
- These formal parameters must be defined in the function's code before they can be used to perform the desired task.
- The formal parameters of our given function operate just like any other local variables.
- When they enter a function, these arguments are formed. When it leaves after that, it is destroyed.



39

# Function Arguments (Example)

```
#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;
}
```

Formal Parameter: `int a, int b`

Actual Parameter: `10, 30`



40

# Function Arguments

- Actual parameter – This is the argument which is used in function call.
- Formal parameter – This is the argument which is used in function definition



41

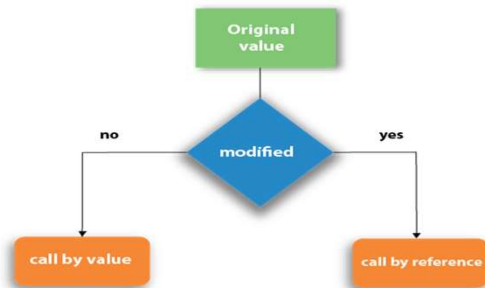
# Passing Parameters to Functions

- We can pass arguments to the C function in two ways:
  1. Pass by Value
  2. Pass by Reference



42

# Passing Parameters to Functions




Sumit Srivastava @ BIT Mesra

43

## HOW TO CALL FUNCTIONS IN A PROGRAM?

- There are two ways that a C function can be called from a program. They are,

1. Call by value
2. Call by reference

Sumit Srivastava @ BIT Mesra

44

# Pass (Call) by Value

- Parameter passing in this method copies values from actual parameters into formal function parameters.
- As a result, any changes made inside the functions do not reflect in the caller's parameters.

Sumit Srivastava @ BIT Mesra

45

# Pass (Call) by Value

```
// C program to show use
// of call by value
#include <stdio.h>

void swap(int var1, int var2)
{
  int temp = var1;
  var1 = var2;
  var2 = temp;
}

int main()
{
   int var1 = 3, var2 = 2;
   printf("Before swap Value of var1 and var2 is: %d,
           %d\n", var1, var2);
   swap(var1, var2);
   printf("After swap Value of var1 and var2 is: %d,
    %d",  var1, var2);
return 0;  }
```

Output

Before swap Value of var1 and var2 is: 3, 2

After swap Value of var1 and var2 is: 3, 2

Sumit Srivastava @ BIT Mesra

46

# Call by Value

- In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In call by value method, we can not modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Sumit Srivastava @ BIT Mesra

47

# Pass (Call) by Reference

- The caller's actual parameters and the function's actual parameters refer to the same locations, so any changes made inside the function are reflected in the caller's actual parameters.

Sumit Srivastava @ BIT Mesra

48

## Pass (Call) by Reference

```
// C program to show use
// of call by Reference
#include <stdio.h>

void swap(int *var1, int *var2)
{
 int temp = *var1;
 *var1 = *var2;
 *var2 = temp;
}

int main()
{
   int var1 = 3, var2 = 2;
   printf("Before swap Value of var1 and var2 is: %d,
           %d\n", var1, var2);
   swap(&var1, &var2);
   printf("After swap Value of var1 and var2 is: %d,
           %d", var1, var2);

  return 0;
}
```

Output

Before swap Value of var1 and var2 is: 3, 2

After swap Value of var1 and var2 is: 2, 3



## Call by Reference

- In call by reference, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.



## Types of User-Defined Function



## Types of User-Defined Function

- All C functions can be called either with arguments or without arguments in a C program. These functions may or may not return values to the calling function.
- There can be 4 different types of user-defined functions based of the return type & arguments passed.
  - Function with arguments (parameters) and with return value.
  - Function with arguments (parameters) and without return value.
  - Function without arguments (parameters) and without return value.
  - Function without arguments (parameters) and with return value.



## Types of User-Defined Function

- With arguments and with return values

function declaration:

```
int function ( int );
```

function call: function ( a );

function definition:

```
int function( int a )
{
statements;
return a;
}
```

Example

```
#include <stdio.h>
int add(int, int);
int add(int x, int y)
{
int sum = x+y;
return(sum);
}
int main()
{
int sum = add(23, 31);
printf("%d", sum);
return 0;
}
```



## Types of User-Defined Function

- With arguments and without return values

function declaration:

```
void function ( int );
```

function call: function( a );

function definition:

```
void function( int a )
{
statements;
}
```

Example:

```
#include <stdio.h>
void add(int, int);
void add(int x, int y)
{
int sum = x+y;
return(sum);
}
int main()
{
add(23, 31);
return 0;
}
```

# Types of User-Defined Function

- Without arguments and without return values

function declaration:

void function();

function call: function();

function definition:

void function()

{

statements;

}

```
#include <stdio.h>
void add();
void add()
{
int x = 20;
int y = 30;
int sum = x+y;
printf("sum %d", sum);
}
int main()
{
add();
return 0;
}
```

Sumit Srivastava @ BIT Mesra

55

# Types of User-Defined Function

- Without arguments and with return values

function declaration:

int function ( );

function call: function ( );

function definition:

int function( )

{

statements;

return a;

}

```
#include <stdio.h>
int add();
int add()
{
int x = 20;
int y = 30;
int sum = x+y;
return(sum);
}
int main()
{
int sum;
sum = add();
printf("sum %d", sum);
return 0;
}
```

Sumit Srivastava @ BIT Mesra

56

# Function with no arguments and no return value (Example)

```
#include<stdio.h>

void greatNum();  // function declaration

int main()
{
   greatNum();   // function call
   return 0;
}

void greatNum()      // function definition
{
   int i, j;
   printf("Enter 2 numbers that you want to
         compare...");
   scanf("%d%d", &i, &j);
   if(i > j) {
      printf("The greater number is: %d", i);
   }
   else {
      printf("The greater number is: %d", j);
   }
}
```

Sumit Srivastava @ BIT Mesra

57

# Function with no arguments and a return value (Example)

```
#include<stdio.h>

int greatNum();     // function declaration

int main()
{
  int result;
  result = greatNum();      // function call
  printf("The greater number is: %d",
          result);
  return 0;
}

int greatNum()      // function definition
{
   int i, j, greaterNum;
   printf("Enter 2 numbers that you want to
         compare...");
   scanf("%d%d", &i, &j);
   if(i > j) {
      greaterNum = i;
   }
   else {
      greaterNum = j;
   }
   // returning the result
   return greaterNum;
}
```

Sumit Srivastava @ BIT Mesra

58

# Function with arguments and no return value (Example)

```
#include<stdio.h>

void greatNum(int a, int b);  // function declaration

int main()
{
   int i, j;
   printf("Enter 2 numbers that you want to
compare...");
   scanf("%d%d", &i, &j);
   greatNum(i, j);       // function call
   return 0;
}

void greatNum(int x, int y)
\\function definition
{
   if(x > y) {
      printf("The greater number is:
             %d",  x);
   }
   else {
      printf("The greater number is:
             %d",  y);
   }
}
```

Sumit Srivastava @ BIT Mesra

59

# Function with arguments and a return value (Example)

```
#include<stdio.h>

int greatNum(int a, int b); // function declaration

int main()
{
   int i, j, result;
   printf("Enter 2 numbers that you want to
compare...");
   scanf("%d%d", &i, &j);
   result = greatNum(i, j); // function call
   printf("The greater number is: %d", result);
   return 0;
}}

int greatNum(int x, int y)
 // function definition
{
   if(x > y) {
      return x;
   }
   else {
      return y;
   }
}
```

Sumit Srivastava @ BIT Mesra

60

# Nesting of Functions

- C language also allows nesting of functions i.e to use/call one function inside another function's body.

```
function1()
{
   // function1 body here

   function2();

   // function1 body here
}
```



61

# Recursion

- A function that calls itself is known as a recursive function. And, this technique is known as recursion.

```
void recurse()
{
   ... .. ...
   recurse();
   ... .. ...
}

int main()
{
   ... .. ...
   recurse();
   ... .. ...
}
```



62

# Recursion

How does recursion work?

```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

recursive call



63

# 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.



64

# Recursion (Example)

Sum of Natural Numbers Using Recursion

```
#include <stdio.h>
int sum(int n);

int main() {
   int number, result;

   printf("Enter a positive integer: ");
   scanf("%d", &number);

   result = sum(number);

   printf("sum = %d", result);
   return 0;
}
```

```
int sum(int n) {
   if (n != 0)
      // sum() function calls itself
      return n + sum(n-1);
   else
      return n;
}
```

Output

```
Enter a positive integer:3
sum = 6
```



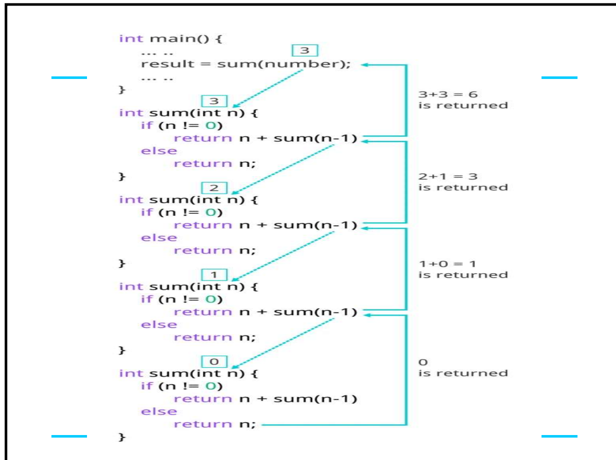
65

# Recursion (Example)

- Initially, the sum() is called from the main() function with number passed as an argument.
- Suppose, the value of n inside sum() is 3 initially. During the next function call, 2 is passed to the sum() function. This process continues until n is equal to 0.
- When n is equal to 0, the if condition fails and the else part is executed returning the sum of integers ultimately to the main() function.



66




67

# Recursion (Example)

Factorial of a number using Recursion

```
#include<stdio.h>

int factorial(int x); //declaring the function

void main()
{
   int a, b;

   printf("Enter a number...");
   scanf("%d", &a);
   b = factorial(a); //calling the function named factorial
   printf("%d", b);
}
```

```
int factorial(int x) //defining the function
{
   int r = 1;
   if(x == 1)
      return 1;
   else
      r = x*factorial(x-1);     //recursion, since the function calls itself

   return r;
}
```



68

# Advantages of functions

1. **Module Approach:** By using the function we can develop the application in module format i.e. procedure-oriented language concept.
2. **Reusability:** By using functions we can create re-usability blocks i.e. develop once and use multiple times.
3. **Code Maintenance:** When we are developing the application by using functions, then it is easy to maintain code for future enhancement.
4. **Code Sharing:** A function may be used by many other programs.
5. **Flexible Debugging:** It is easy to locate and isolate a faulty function for further investigations.
6. **Data Protection**: Functions can be used to protect data and local data. Local data is available only within a function when the function is being executed.
7. **Code Reduced:** Reduces the size of the code, duplicate statements are replaced by function calls.



69