

## #9 Functions

By Saurabh Shukla | 2017@mysirg.com

### What is Function?

Function is one of the most important topics in C language. It is more about the arrangement of the code which has many benefits.

- Function is a piece of code to perform a particular task
- Function is a block of code which performs usually a unit task
- It has some name for identification
- Action statements in C language can only reside in some function block.
- The format of function is as follows:

```
return_type function_name( arguments)
{
    statements;
}
```

- return\_type is data type (like int, char, float, etc) , which can be anything depending upon the requirement.
- function\_name is any name chosen by the programmer. Naming rules are same as applied in the case of variable names, that is, name is any combination of alphabet, digit or underscore and cannot start with digit.
- Arguments are 0 or more variable declarations.
- Function body or block is everything in the pair of curly braces ( { } ).
- You cannot write action statements without function block. The following code will not compile in C language

```
int a,b,c;
printf("Enter two numebrs");
scanf("%d%d",&a,&b);
c=a+b;
printf("Sum is %d",c);
```

- You must put action statements in a block. Above code is to add two numbers taken from user through keyboard. Say this is one task of adding two numbers. The correct way to write code is:

```
add()
{
    int a,b,c;
    printf("Enter two numebrs");
    scanf("%d%d",&a,&b);
    c=a+b;
    printf("Sum is %d",c);
}
```

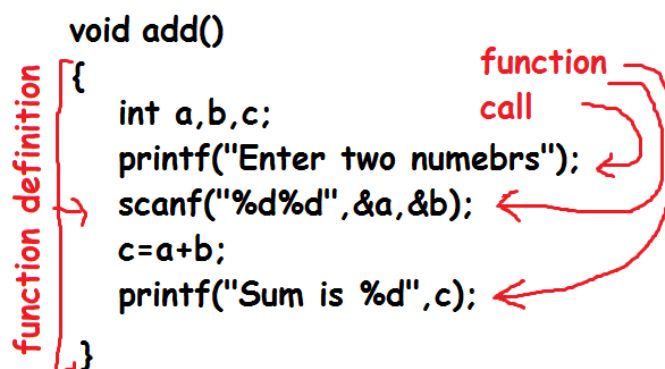
- add is function name.
- Where is return type? In the absence of return type it is considered as int.
- Function has no arguments thus the parenthesis left empty.
- Return type of add() function shouldn't be int, as function is not actually returning a value, because of which a warning occurs during compilation of the program, stated- "Function should return a value". As the default return type is int, compiler expected from function to return a value of type int.
- How a function can actually return a value. It can be achieved by using return keyword, which we are going to cover in this chapter only.
- The more important point is what should be the return type of add(). As it is not going to return any value, return type should be void. So the correct version of the function is:

```
void add()
{
    int a,b,c;
    printf("Enter two numebrs");
    scanf("%d%d",&a,&b);
    c=a+b;
    printf("Sum is %d",c);
}
```

- The above code is called function definition of add()
- You can have any number of function in a C program
- Function names must be unique
- You can define functions in any order
- main() is also a function
- Execution of program always begins from main() function, irrespective of its position in the program.
- You cannot define a function inside a function

- Function definition is different from function declaration and function call.
- There are two types of functions
  - Predefined Functions
  - User defined Functions
- add() is a user defined function.
- main() is also a user defined function.
- There must be some function with the name main() in the program so that program can begin its execution, otherwise it will not run.
- printf, scanf are prime examples of predefined functions. Notice that in the above function add(), we used a statement printf and statement scanf. Please do not get confused, you are not defining printf or scanf inside the function add(), but you are calling functions printf() and scanf().
- Function call is a representative of function code.

```
void add()  
{  
    int a,b,c;  
    printf("Enter two numebrs");  
    scanf("%d%d",&a,&b);  
    c=a+b;  
    printf("Sum is %d",c);  
}
```



- You can call a function any number of times. You are calling printf two times in add function
- Where is printf defined? All predefined functions reside in C library, which is a set of library files. After compilation of your code, linker associates your function calls to the appropriate definition of function from the library.
- Function declaration is another jargon which is different from function definition and function call.
- Function declaration is also known as function prototype.
- It is a single line statement written for the compiler to process.
- You declare variables before using them, why? It is because; variable names are new words to the compiler. You need to inform compiler about the purpose of the new word in advance. In the same way, function names are also new to the compiler, you should first declare them.
- ANSI stated about C function declarations- "Function declaration is recommended but not compulsory". You can use function without actually declaring them, but recommendation is to always declare explicitly.

- In the absence of function declaration, compiler automatically declares it in a preset manner which is sometimes do not fit for the definition. This may lead to abnormal behavior of the code. So it is strongly recommended to write function declaration by your own.
- The general format of function prototype is
  - `return_type function_name(argument);`
- `return_type` is any data type, `function_name` is name of the function and arguments are 0 or more variable types.
- Return type, function name and arguments in function declaration should match with the return type function name and arguments in the function definition.
- Function declaration can be done outside the function body, called global declaration. This is useful when more than one function in your program wants to call declared function.
- Function can be declared inside the function prior to any action statement in the block.

```

1
2  void add(); ← function declaration
3  main()      (Global declaration)
4  {
5
6      add(); ← function call
7
8  }
9  void add()
10 {
11     int a,b,c;
12     printf("Enter two numbers:");
13     scanf("%d%d",&a,&b);
14     c=a+b;
15     printf("Sum is %d",c);
16 }

```

The image shows a C program snippet with line numbers 1 to 16. Annotations with red arrows and text identify key parts: 'function declaration (Global declaration)' points to line 2, 'function call' points to line 6, and 'function definition' points to the block from line 9 to 16.

- C Program must have at least one function.
- Function is a block of code which can be used many times, by simply calling it any number of times.
- Function is a way to achieve modularization. Modularization is splitting up of a bigger task into several smaller subtasks, which reduces the complexity of the problem.
- Suppose you have to make a program of calculator. It is a big task. You can logically divide it into several independent subtasks. Like addition, subtraction, multiplication, division, etc. Now you can define each subtask with the help of function. This reduces the complexity of your program.

### Benefits of function

You can make a big complex program by simply writing whole code in main function, which should be discouraged, because of many reasons:

- Messy look, hard to read and understand
- Wastage of memory
- Not easy to debug or modify
- Hard to maintain

Logical breakdown of overall task of the program make it easy to create, maintain and understand. Following are the benefits of functions.

- It avoids re-writing of same code over and over
- Achieve modularization
- Easy to read
- Easy to debug
- Easy to modify
- Better memory utilization
  - Function consumes memory on its call
  - Function releases memory after its execution

### Ways to define a function

So far you have understood how a function can be defined, but there will be more to study. How a function can take arguments? How a function can return a value? Such questions are answered in this section. On the basis of arguments and return value, functions can be defined in the following four ways.

- Takes Nothing, Returns Nothing
- Takes Something, Returns Nothing
- Takes Nothing, Return Something
- Takes Something, Returns Something

### Takes nothing, returns nothing

```
1
2 void add(); ← function declaration
3 main()      (Global declaration)
4 {
5
6     add(); ← function call
7
8 }
9 void add() ← void means returns nothing
10           Empty parenthesis means takes nothing
11 {
12     int a,b,c;
13     printf("Enter two numbers:");
14     scanf("%d%d", &a, &b);
15     c=a+b;
16     printf("Sum is %d",c);
17 }
```

function definition

In the above program, function add is defined as takes nothing and return something. Empty parenthesis of function add() indicates that the function is not taking any value. Data type void before the function name is used to tell compiler that this function is not going to return any value.

Execution of the program begins with main function. Main function calls add function. Now add function is executed. After completion of add function, control goes back to main function, exactly from where the add function was invoked.

- Function can use only those variables which are declared in its body. Function cannot access variables of other function.
- To return a value, you have to use return keyword. In this example, there is no need of return keyword, as it is of nature, returns nothing.

### Takes Something, Returns Nothing

Suppose main function has two variables, x and y, there are some values in the variables. Now you want to add them (or any other operation using values of x and y). You know that there is a add() function defined to add two numbers, but the way it was designed in previous example, can add only values stored in variables, a and b. To accomplish this task

- You can replace a and b with x and y respectively, but you are wrong. Function add() can't access variables, x and y, as they belong to main function. You know that function can't access variables of another function.
- Another option is to copy data of x and y into a and b correspondingly. Yes this is the way.

In the following example, observe how we copy data of x and y of main() function into a and b of add() function. This is called function call by passing values or simply function call by value.

```

1 void add(int, int);
2 main()
3 {
4     int x,y;
5     printf("Enter two numbers");
6     scanf("%d%d", &x, &y);
7     add(x,y);
8 }
9 void add(int a,int b)
10 {
11     int c;
12     c=a+b;
13     printf("Sum is %d",c);
14 }
15

```

**x and y are Actual Arguments (parameters)**

**a and b are Formal Arguments**

**Takes Something (non empty parenthesis)**

**Returns Nothing ( void )**

- First line is the declaration of function add(). You can see two times int is written in the parenthesis, informing compiler about two arguments of type int must be passed on call to the function add().
- In main() function, there are two variables x and y. call to scanf function allows input of two integers and collected in variables x and y.

- In `add(x,y)`, values of `x` and `y` are copied to variables `a` and `b` of `add()` function. The value of `x` goes into `a` and value of `y` goes into `b`. Here `x` and `y` are called actual arguments. `a` and `b` are called formal arguments.
- In the definition of `add()` function, parenthesis is not blank, therefore it is called takes something. You can see the declaration of two variables in the parenthesis. They are `a` and `b`, known as formal arguments. Do not declare them again in the function's body; otherwise it will be a re-declaration error. You cannot declare formal arguments like this `void add(int,a,b)`; this is wrong  
You have to use data type for each variable in the formal arguments.
- The number of formal arguments decides exactly how many values need to be supplied during call of function. Here, since we have declared two variables in the parenthesis, we can pass exactly two `int` type values during call of `add()` function.
- It would be a blunder if you think about taking values from user using `scanf()` for variables `a` and `b`, because you have already received values in `a` and `b`.

### Takes Nothing, Returns Something

Another way to define a function is takes nothing, returns something. You already know about takes nothing, so our more focus is towards returns something.

```

1  int add();
2  main()
3  {
4      int s;
5      s=add();
6      printf("Sum is %d",s);
7  }
8  int add()
9  {
10     int a,b,c;
11     printf("Enter two numbers");
12     scanf("%d%d",&a,&b);
13     c=a+b;
14     return(c);
15 }

```

Returned value returns exactly from where the function was called

Takes Nothing

Returns Something

- First line is function declaration. Return type is `int`, which means the function `add` is going to return value of type `int`. Empty parenthesis shows its takes nothing nature.
- In the `main` function, variable `s` is declared to hold result which you will be getting on call of function `add()`
- In the line `s=add()`, function call will be going to replace by the value getting returned from the function `add()`.
- In the `add` function, as it is of nature takes nothing, you have to take values from user. So `printf` and `scanf` are present in the code.
- Once result is calculated and stored in `c`, it is returned to the calling function.

- In the main function, returned value is collected in variable s.
- Next is the printf statement to print value of s.

### Takes Something, Return Something

Another way to define a function is takes something and returns something. Since you have understood takes something in the second way and return something in the third way of defining functions, now you can apply your brain to understand the following example.

```
1  int add(int,int);
2  main()
3  {
4      int s,x,y;
5      printf("Enter two numbers");
6      scanf("%d%d",&x,&y);
7      s=add(x,y);
8      printf("Sum is %d",s);
9  }
10 int add(int a,int b)
11 {
12     int c;
13     c=a+b;
14     return(c);
15 }
```

**Takes Something** (points to the parameters `int a, int b` in the function definition)

**Return something** (points to the `return(c);` statement in the function definition)

### References:

#### YouTube video links

- Lecture 9 Functions in C language part 1
  - <https://youtu.be/5ED2NGXvv6s?list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW>
- Lecture 9 Functions in C language part 2
  - [https://youtu.be/1egX1-DoS\\_s?list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW](https://youtu.be/1egX1-DoS_s?list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW)
- Lecture 9 Functions in C language part 3
  - <https://youtu.be/t7ApbGmiGB4?list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW>
- Lecture 9 Functions in C language part 4
  - <https://youtu.be/AMXno1XVvol?list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW>
- Lecture 9 Functions in C language part 5
  - <https://youtu.be/ty43LEprTPo?list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW>
- Why I am not using header files.
  - <https://youtu.be/j51aSRF5c10?list=PL7ersPsTyYt2sohwBWF03kSpW1Ymt0Q9R>

### Exercise



1. Write a function to calculate area of circle. (Takes something, returns something)
2. Write a function to calculate **area of triangle.** (Takes something, returns something)
3. Write a function to calculate sum of first N natural number. ((Takes something, returns something)
4. Write a function to calculate factorial of a number. (Takes something, returns something)
5. Write a function to calculate  $x^y$ . (Takes something, returns something)
6. Write a function count digits in a given number. (Takes something, returns something)
7. Write a function to calculate number of combination to select r items from n items. (Takes something, returns something)
8. Write a function to reverse a number. (Takes something, returns something)
9. Write a function to calculate LCM of two numbers. (Takes something, returns something)
10. Write a function to print N terms of Fibonacci series. (Takes something, returns Nothing)
11. Write a program to express a given number as a sum of two prime numbers. Print all possible solutions. Define as many functions as required in the program.
12. Write a function to check whether a given number is even or odd. Function prototype must be
  - a. `int isEven(int);`
13. Write a function to check whether a given number is Prime or not. Function prototype must be
  - a. `int isPrime(int);`
14. Write a function to print all prime numbers between two given numbers. Function prototype must be (you can use `isPrime()` function)
  - a. `void printAllPrime(int, int);`
15. Write a function to find the immediately next prime number of a given number. Function prototype must be (you can use `isPrime()` function)
  - a. `int nextPrime(int);`
16. Write a function to print N lines of Pascal triangle. Function prototype must be (You can use functions to calculate factorial and combination)
  - a. `void pascalTriangle(int);`
17. Explore functions of `math.h`, `conio.h` and `stdlib.h`
18. Write a program to print System Date.
19. Write a program to generate a random number using predefined functions.
20. Write a function to print first N prime numbers.

