### Functions in C++

Functions are used to provide modularity to a program. Creating an application using function makes it easier to understand, edit, check errors etc.

#### Syntax of Function

*return-type* **function-name** (*parameters*)

{

// function-body

}

* **return-type :** suggests what the function will return. It can be int, char, some pointer or even a class object. There can be functions which does not return anything, they are mentioned with **void**.
* **Function Name :** is the name of the function, using the function name it is called.
* **Parameters :** are variables to hold values of arguments passed while function is called. A function may or may not contain parameter list.
* void sum(int x, int y)
* {
* int z;
* z = x + y;
* cout << z;
* }
* int main()
* {
* int a = 10;
* int b = 20;
* **sum (a, b)**;
* }

Here, **a** and **b** are sent as arguments, and **x** and **y** are parameters which will hold values of a and b to perform required operation inside function.

* **Function body :** is he part where the code statements are written.

#### Declaring, Defining and Calling Function

Function declaration, is done to tell the compiler about the existence of the function. Function's return type, its name & parameter list is mentioned. Function body is written in its definition. Lets understand this with help of an example.

#include < iostream>

using namespace std;

int sum (int x, int y); //declaring function

int main()

{

int a = 10;

int b = 20;

int c = sum (a, b); //calling function

cout << c;

}

int sum (int x, int y) //defining function

{

return (x + y);

}

Here, initially the function is **declared**, without body. Then inside main() function it is **called**, as the function returns sumation of two values, hence z is their to store the value of sum. Then, at last, function is **defined**, where the body of function is mentioned. We can also, declare & define the function together, but then it should be done before it is called.

#### Calling a Function

Functions are called by their names. If the function is without argument, it can be called directly using its name. But for functions with arguments, we have two ways to call them,

1. Call by Value
2. Call by Reference

#### Call by Value

In this calling technique we pass the values of arguments which are stored or copied into the formal parameters of functions. Hence, the original values are unchanged only the parameters inside function changes.

void calc(int x);

int main()

{

int x = 10;

calc(x);

printf("%d", x);

}

void calc(int x)

{

x = x + 10 ;

}

Output : 10

In this case the actual variable x is not changed, because we pass argument by value, hence a copy of x is passed, which is changed, and that copied value is destroyed as the function ends(goes out of scope). So the variable **x** inside main() still has a value 10.

But we can change this program to modify the original **x**, by making the function **calc()** return a value, and storing that value in x.

int calc(int x);

int main()

{

int x = 10;

x = calc(x);

printf("%d", x);

}

int calc(int x)

{

x = x + 10 ;

return x;

}

Output : 20

#### Call by Reference

In this we pass the address of the variable as arguments. In this case the formal parameter can be taken as a reference or a pointer, in both the case they will change the values of the original variable.

void calc(int \*p);

int main()

{

int x = 10;

calc(&x); // passing address of x as argument

printf("%d", x);

}

void calc(int \*p)

{

\*p = \*p + 10;

}

Output : 20

**NOTE :** If you do not have a prior knowledge of pointers, do study Pointers first.

# **C++ Recursion**

A [function](https://www.programiz.com/cpp-programming/function) that calls itself is known as recursive function. And, this technique is known as recursion.

## How recursion works in C++?

void recurse()

{

... .. ...

recurse();

... .. ...

}

int main()

{

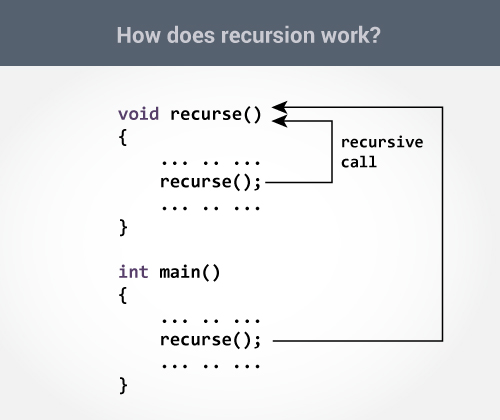
... .. ...

recurse();

... .. ...

}

The figure below shows how recursion works by calling itself over and over again.



The recursion continues until some condition is met.

To prevent infinite recursion, [if...else statement](https://www.programiz.com/cpp-programming/if-else) (or similar approach) can be used where one branch makes the recursive call and other doesn't.

## Example 1: Factorial of a Number Using Recursion

// Factorial of n = 1\*2\*3\*...\*n

#include <iostream>

using namespace std;

int factorial(int);

int main()

{

int n;

cout<<"Enter a number to find factorial: ";

cin >> n;

cout << "Factorial of " << n <<" = " << factorial(n);

return 0;

}

int factorial(int n)

{

if (n > 1)

{

return n\*factorial(n-1);

}

else

{

return 1;

}

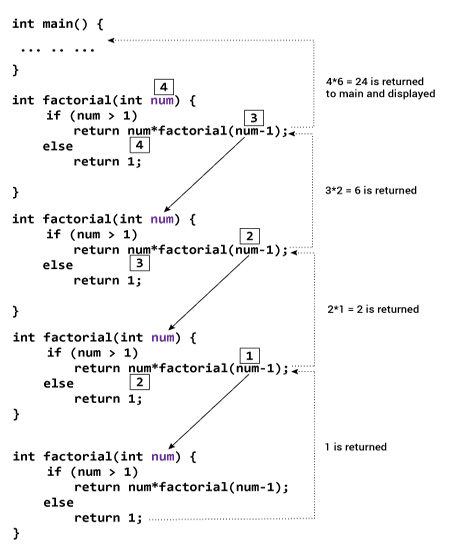
}

**Output**

Enter a number to find factorial: 4

Factorial of 4 = 24

### Explanation: How this example works?



Suppose the user entered 4, which is passed to the factorial() function.

1. In the first factorial() function, test expression inside [if statement](https://www.programiz.com/cpp-programming/if-else) is true. The return num\*factorial(num-1); statement is executed, which calls the second factorial() function and argument passed is num-1which is 3.
2. In the second factorial() function, test expression inside if statement is true. The return num\*factorial(num-1); statement is executed, which calls the third factorial() function and argument passed is num-1 which is 2.
3. In the third factorial() function, test expression inside if statement is true. The return num\*factorial(num-1); statement is executed, which calls the fourth factorial() function and argument passed is num-1 which is 1.
4. In the fourth factorial() function, test expression inside if statement is false. The return 1; statement is executed, which returns 1 to third factorial() function.
5. The third factorial() function returns 2 to the second factorial() function.
6. The second factorial() function returns 6 to the first factorial() function.
7. Finally, the first factorial() function returns 24 to the main() function, which is displayed on the screen.