C++ Functions

The function in C++ language is also known as procedure or subroutine in other programming languages.

To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

## Types of Functions

There are two types of functions in C programming:

**1. Library Functions:** are the functions which are declared in the C++ header files such as ceil(x), cos(x), exp(x), 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 complexity of a big program and optimizes the code.

The syntax of creating function :

return\_type function\_name(data\_type parameter...)

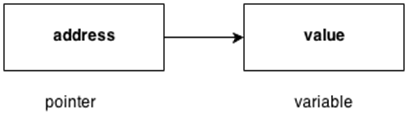
{

//code to be executed

}

# C++ Pointers

The pointer in C++ language is a variable, it is also known as locator or indicator that points to an address of a value.



|  |  |  |
| --- | --- | --- |
| & (ampersand sign) | Address operator | Determine the address  of a variable. |
| ∗ (asterisk sign) | Indirection operator | Access the value of an  address. |

The pointer in C++ language can be declared using ∗ (asterisk symbol).

**int** ∗   a; //pointer to int

**char** ∗  c; //pointer to char

#include <iostream>

**using** **namespace** std;

**int** main()

{

**int** number=30;

**int** ∗   p;

p=&number;//stores the address of number variable

cout<<"Address of number variable is:"<<&number<<endl;

cout<<"Address of p variable is:"<<p<<endl;

cout<<"Value of p variable is:"<<\*p<<endl;

**return** 0;

}

## Call by value in C++

In call by value, **original value is not modified.**

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

## Call by reference in C++

In call by reference, original value is modified because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

**Note:** To understand the call by reference, you must have the basic knowledge of pointers.

Let's try to understand the concept of call by reference in C++ language by the example given below:

#include<iostream>

**using** **namespace** std;

**void** swap(**int** \*x, **int** \*y)

{

**int** swap;

 swap=\*x;

 \*x=\*y;

 \*y=swap;

}

**int** main()

{

**int** x=500, y=100;

 swap(&x, &y);  // passing value to function

 cout<<"Value of x is: "<<x<<endl;

 cout<<"Value of y is: "<<y<<endl;

**return** 0;

}

## Difference between call by value and call by reference in C++

|  |  |  |
| --- | --- | --- |
| **No.** | **Call by value** | **Call by reference** |
| 1 | A copy of value is passed to the function | An address of value is passed to the function |
| 2 | Changes made inside the function  is not reflected on other functions | Changes made inside the function is reflected  outside the function also |
| 3 | Actual and formal arguments will  be created in different memory location | Actual and formal arguments will be created  in same memory location |