

# Functions in C++ with example

A function is block of code which is used to perform a particular task, for example let's say you are writing a large C++ program and in that program you want to do a particular task several number of times, like displaying value from 1 to 10, in order to do that you have to write few lines of code and you need to repeat these lines every time you display values. Another way of doing this is that you write these lines inside a function and call that function every time you want to display values. This would make you code simple, readable and reusable.

## Syntax of Function

```
return_type function_name (parameter_list)
{
    //C++ Statements
}
```

Let's take a simple example to understand this concept.

## A simple function example

```
#include <iostream>
using namespace std;
/* This function adds two integer values
 * and returns the result
 */int
sum(int num1, int num2){
    int num3 = num1+num2; return num3;
}
```

```
int main(){
    //Calling the function
    cout<<sum(1,99);
    return 0;
}
```

### Output:

100

**The same program can be written like this:** Well, I am writing this program to let you understand an important term regarding functions, which is function declaration. Lets see the program first and then at the end of it we will discuss function declaration, definition and calling of function.

```
#include <iostream>
using namespace std;
//Function declaration
int sum(int,int);

//Main function
int main(){
    //Calling the function
    cout<<sum(1,99);
    return 0;
}
/* Function is defined after the main method
 */
int sum(int num1, int num2){
```

```
int num3 = num1+num2;  
return num3;  
}
```

**Function Declaration:** You have seen that I have written the same program in two ways, in the first program I didn't have any function declaration and in the second program I have function declaration at the beginning of the program. The thing is that when you define the function before the main() function in your program then you don't need to do function declaration but if you are writing your function after the main() function like we did in the second program then you need to declare the function first, else you will get compilation error.

**syntax of function declaration:**

```
return_type function_name(parameter_list);
```

**Note:** While providing parameter\_list you can avoid the parameter names, just like I did in the above example. I have given `int sum(int,int);` instead of `int sum(int num1,int num2);`.

**Function definition:** Writing the full body of function is known as defining a function.

**syntax of function definition:**

```
return_type function_name(parameter_list) {  
    //Statements inside function  
}
```

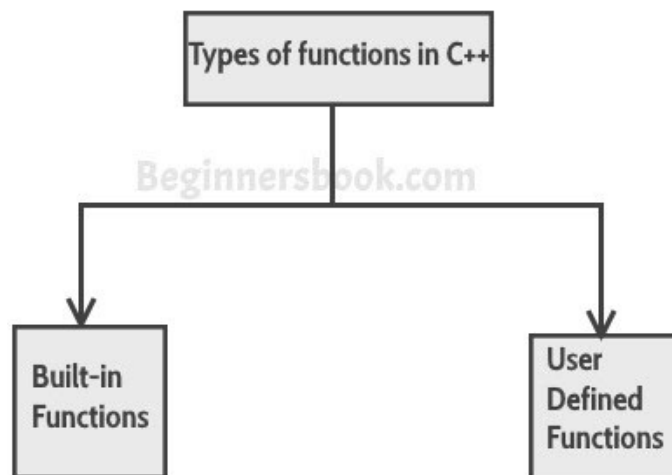
**Calling function:** We can call the function like this:

```
function_name(parameters);
```

Now that we understood the **working of function**, let's see the types of function in C++

## Types of function

We have two types of function in C++:



- 1) Built-in functions
- 2) User-defined functions

### 1) Build-it functions

Built-in functions are also known as library functions. We need not to declare and define these functions as they are already written in the C++ libraries such as `iostream`, `cmath` etc. We can directly call them when we need.

## Example: C++ built-in function example

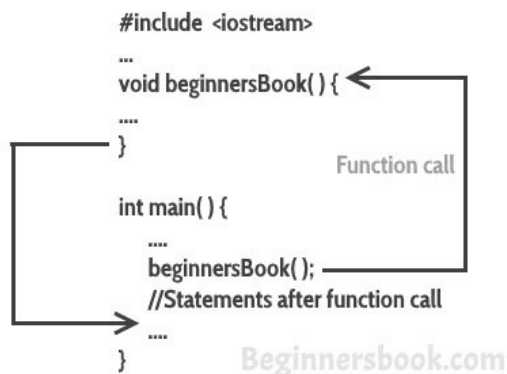
Here we are using built-in function `pow(x,y)` which is  $x$  to the power  $y$ . This function is declared in `cmath` header file so we have included the file in our program using `#include` directive.

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    /* Calling the built-in function
     * pow(x, y) which is x to the power y
     * We are directly calling this function
     */
    cout<<pow(2,5);
    return 0;
}
```

**Output:**

32

## 2) User-defined functions



We have already seen user-defined functions, the example we have given at the beginning of this tutorial is an example of user-defined function. The functions that we declare and write in our programs are user-defined functions. Lets see another example of user-defined functions.

### User-defined functions

```
#include <iostream>
#include <cmath>
using namespace std;
//Declaring the function sum
int sum(int,int);

int main() {
    int x, y;
    cout<<"enter first number: ";
    cin>> x;

    cout<<"enter second number: ";
    cin>>y;

    cout<<"Sum of these two : "<<sum(x,y);
    return 0;
}
```

```
//Defining the function sum
int sum(int a, int b) {
    int c = a+b;
    return c;
}
```

**Output:**

```
enter first number: 22
enter second number: 19
Sum of these two :41
```

## Default Arguments in C++ Functions

The default arguments are used when you provide no arguments or only few arguments while calling a function. The default arguments are used during compilation of program. For example, let's say you have a **user-defined function** `sum` declared like this: `int sum(int a=10, int b=20)`, now while calling this function you do not provide any arguments, simply called `sum()`; then in this case the result would be 30, compiler used the default values 10 and 20 declared in function signature. If you pass only one argument like this: `sum(80)` then the result would be 100, using the passed argument 80 as first value and 20 taken from the default argument.

### Example: Default arguments in C++

```
#include <iostream>
using namespace std;
int sum(int a, int b=10, int c=20);
```

```
int main() {
    /* In this case a value is passed as
     * 1 and b and c values are taken from
     * default arguments.
     */
    cout<<sum(1)<<endl;
```

```
    /* In this case a value is passed as
     * 1 and b value as 2, value of c values is
     * taken from default arguments.
     */
    cout<<sum(1, 2)<<endl;
```

```
    /* In this case all the three values are
     * passed during function call, hence no
     * default arguments have been used.
     */
    cout<<sum(1, 2, 3)<<endl;
    return 0;
}
int sum(int a, int b, int c) {
    int z;
    z = a+b+c;
    return z;
}
```

**Output:**

31  
23  
6

## Rules of default arguments

As you have seen in the above example that I have assigned the default values for only two arguments b and c during function declaration. It is up to you to assign default values to all arguments or only selected arguments but remember the following rule while assigning default values to only some of the arguments:

**If you assign default value to an argument, the subsequent arguments must have default values assigned to them, else you will get compilation error.**

**For example:** Lets see some valid and invalid cases.

**Valid:** Following function declarations are valid –

```
int sum(int a=10, int b=20, int c=30);
int sum(int a, int b=20, int c=30);
int sum(int a, int b, int c=30);
```

**Invalid:** Following function declarations are invalid –

```
/* Since a has default value assigned, all the
 * arguments after a (in this case b and c) must have
 * default values assigned
 */
int sum(int a=10, int b, int c=30);
```

```
/* Since b has default value assigned, all the
 * arguments after b (in this case c) must have
 * default values assigned
 */
int sum(int a, int b=20, int c);
```

```
/* Since a has default value assigned, all the
 * arguments after a (in this case b and c) must have
 * default values assigned, b has default value but
 * c doesn't have, thats why this is also invalid
 */
int sum(int a=10, int b=20, int c);
```

## C++ ‘this’ Pointer

The **this** pointer holds the address of current object, in simple words you can say that this pointer points to the current object of the class. Let’s take an example to understand this concept.

## C++ Example: this pointer

Here you can see that we have two data members num and ch. In member function setMyValues() we have two local variables having same name as data members name. In such case if you want to assign the local variable value to the data members then you won’t

be able to do until unless you use this pointer, because the compiler won't know that you are referring to object's data members unless you use this pointer. This is one of the example where you must use **this** pointer.

```
#include <iostream>
using namespace std;
class Demo {
private:
    int num;
    char ch;
public:
    void setMyValues(int num, char ch){
        this->num =num;
        this->ch=ch;
    }
    void displayMyValues(){
        cout<<num<<endl;
        cout<<ch;
    }
};
int main(){
    Demo obj;
    obj.setMyValues(100, 'A');
    obj.displayMyValues();
    return 0;
}
```

**Output:**

```
100
A
```

## Example 2: function chaining calls using this pointer

Another example of using this pointer is to return the reference of current object so that you can chain function calls, this way you can call all the functions for the current object in one go. Another important point to note in this program is that I have incremented the value of object's num in the second function and you can see in the output that it incremented the value that we have set in the first function call. This shows that the chaining is sequential, and the changes made to the object's data members retains for further chaining calls.

```
#include <iostream>
using namespace std;
class Demo {
private:
    int num;
    char ch;
public:
    Demo &setNum(int num){
        this->num =num;
        return *this;
    }
    Demo &setCh(char ch){
        this->num++;
        this->ch =ch;
        return *this;
    }
    void displayMyValues(){
        cout<<num<<endl;
        cout<<ch;
    }
};
int main(){
    Demo obj;
    //Chaining calls
    obj.setNum(100).setCh('A');
    obj.displayMyValues();
    return 0;
}
```

**Output:**

```
101
A
```