Functions and Modular Programming in C++

Murtaza Khan

May 2025

Learning Objectives

By the end of this lesson, students will be able to:

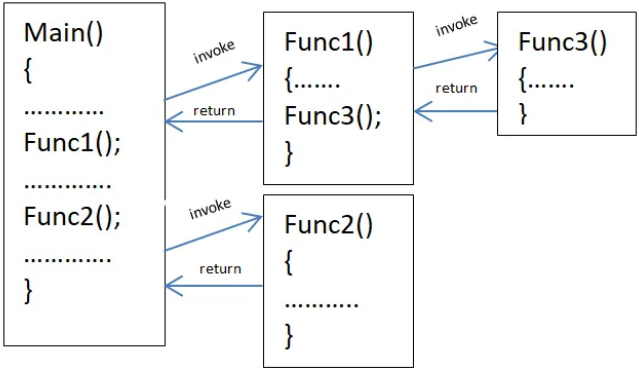
- Explain what a function is in C++.
- Understand why modular programming is useful.
- Define and call functions with and without parameters.
- Write a simple C++ program using function.

Cake Recipe

Imagine you are baking a cake. Do you repeat the whole recipe every time, or do you follow a set of steps?

- Just like recipes, functions let us group steps into reusable blocks. Instead of writing the same code over and over, we define it once.

- Modularity is a key concept in programming. Functions promote modularity by breaking problems into smaller, manageable units.



- A function is a block of code that performs a specific task.
- A function is written once and can be called one or more times to perform the same task.

Syntax of a Function

```
returnType functionName(parameter list)
{
    // body of the function
}
```

Example of a Function

```
int add(int a, int b)
{
    return a + b;
}
```

- A complete C++ program with a main function that calls the add function

```
#include <iostream>
using namespace std;

int add(int a, int b) {
    return a + b;
}

int main() {
    int num1 = 5;
    int num2 = 10;
    int sum = add(num1, num2); // Function call
    cout << sum << endl;
    return 0;
}
```

Output 15

Types of Parameters and Return Value

No parameters, no return value

```
void greetUser()
{
    cout << "Welcome" << endl;
}
```

Parameters and return value

```
int add(int a, int b)
{
    return a + b;
}
```

No parameters, returns a value

```
int getLuckyNumber()
{
    return 7;
}
```

Parameters, no return value

```
void printMessage(string message)
{
    cout << message << endl;
}
```

- What happens if we call the following?

```
string printMessage(string name){
    return "Welcome " + name;
}
```

```
int sum = add(3, -5);
```

```
int sum = add(2, 3) + add(4, 1);
```

Swap Function

- Let's define a function that **swaps (exchanges)** the values of two variables, a and b.
  Example:
  - Before swap: `a = 10, b = 20`
  - After swap: `a = 20, b = 10`

Pass by Value:

- The function parameters store local copies of variables passed to the function from the main method.
- Changing local variables inside the function does not affect the variables in main.
- To get updated values in main, the function must return the new values.

Swap Function (Incorrect: Pass by Value)

```cpp
using namespace std;

// Swap function (Incorrect: Pass by Value)
// It swaps copies of a and b, not the originals.
void mySwap(int a, int b) {
    int c = a;
    a = b;
    b = c;
}
int main(){
    int a = 10, b = 20;
    cout << "Before swap: a = " << a << ", b = " << b << endl;
    mySwap(a, b);
    cout << "After swap:  a = " << a << ", b = " << b << endl;
    return 0;
}
```

```
Before swap: a = 10, b = 20
After swap:  a = 10, b = 20
```

Pass by Reference:

- The function parameters share the same memory location as the variables passed from the main method.
- Changing variables inside the function also changes the variables in main.
- The function does not need to return updated values. The main variables are automatically updated via shared memory.

Swap Function (Correct: Pass by Reference)

```
using namespace std;

// Swap function (Correct: Pass by Reference)
// It swaps the actual variables.
void mySwap(int &a, int &b) {
    int c = a;
    a = b;
    b = c;
}
int main(){
    int a = 10, b = 20;
    cout << "Before swap: a = " << a << ", b = " << b << endl;
    mySwap(a, b);
    cout << "After swap:  a = " << a << ", b = " << b << endl;
    return 0;
}
```

```
Before swap: a = 10, b = 20
After swap:  a = 20, b = 10
```

Incorrect Swap function (Pass by Value)

```
void mySwap(int a, int b) {
    int c = a;
    a = b;
    b = c;
}
```

```
Before swap: a = 10, b = 20
After swap:  a = 10, b = 20
```

Correct Swap function (Pass by Reference)

```
void mySwap(int &a, int &b) {
    int c = a;
    a = b;
    b = c;
}
```

```
Before swap: a = 10, b = 20
After swap:  a = 20, b = 10
```

- Functions promote **modularity** by dividing code into logical blocks.
- Functions enable **reuse of code**, reducing duplication and errors.
- **Pass by Value** sends a copy of the variable to the function - changes do not affect the original.
- **Pass by Reference** allows the function to modify the original variable by sharing memory.
- Using functions improves code clarity, maintenance, and efficiency.

- Exploring Cpp Adventure Begins Basics by Jason James
- http://www.craie-programming.org/
- https://medium.com/@scitechexplorer/function-in-c-2c5f2dea75e
- https://chatgpt.com/
- https://www.overleaf.com/
- https://www.pexels.com/ (high-quality, royalty-free images and videos)