University of Anbar
College of Computer Science and Information Technology
Department of Computer Networks

C++ Functions I

Lecture 4

Dr. Ali Al-Kubaisi

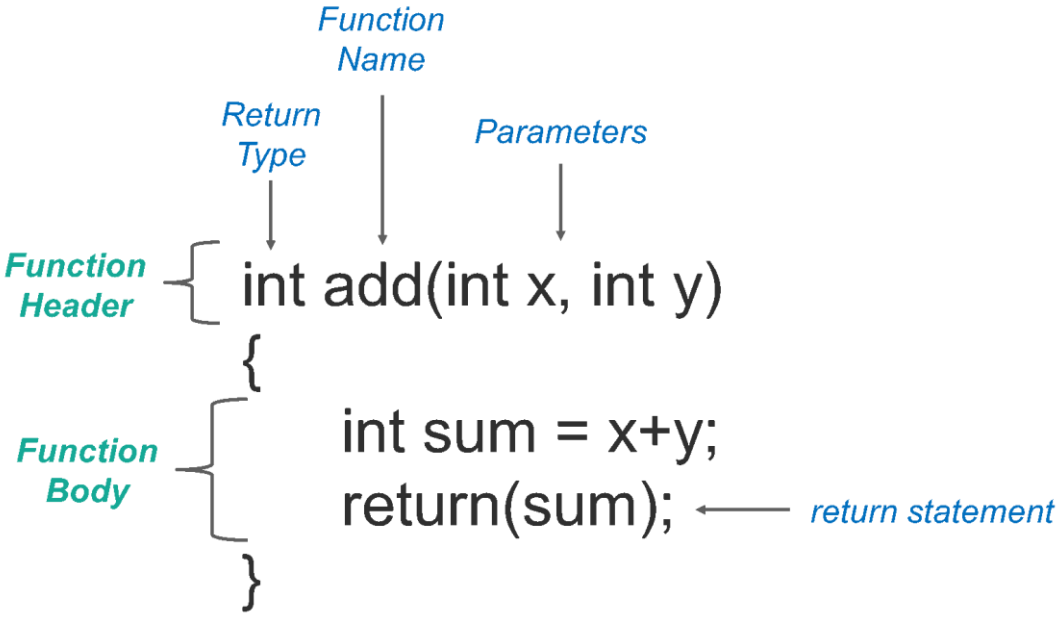
Asst.Lect. Abdulrahman Abbas Mukhlif

Outlines

- Create a Function
- Call a Function
- Function Declaration and Definition
- C++ Function Parameters
  - Parameters and Arguments
  - Default Parameters
  - Multiple Parameters
  - Return Values
  - Pass By Reference

C++ Functions

- A function is a **block of code which only runs when it is called**.
- You can pass data, known as **parameters**, into a function.
- Functions are used to perform certain actions, and they are important for **reusing code**: Define the code once, and use it many times.



Create a Function

- C++ provides some pre-defined functions, such as main(), which is used to execute code. But you can also create your own functions to perform certain actions.
- To create (often referred to as ***declare***) a function, specify the name of the function, followed by parentheses **()**:
- **Syntax**:

```
void myFunction()
{
  // code to be executed
}
```

Call a Function

- Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are called.
- To call a function, write the function's name followed by two parentheses () and a semicolon ;
- In the following example, myFunction() is used to print a text (the action), when it is called:

```
// Create a function
void myFunction() {
  cout << “Hello There!";
}

int main() {
  myFunction(); // call the function
  return 0;
}
// Outputs “Hello There!"
```

Function Declaration and Definition

- A C++ function consist of two parts:
  - **Declaration:** the return type, the name of the function, and parameters (if any)
  - **Definition:** the body of the function (code to be executed)

```
void myFunction() { // declaration
  // the body of the function (definition)
}
```

  - If a user-defined function, such as myFunction() is declared after the main() function, **an error will occur**:

```
int main() {
  myFunction();
  return 0;
}
void myFunction() {
  cout << “Hello There!";
}
// Error
```

Function Declaration and Definition (Cont.)

- It is possible to separate the declaration and the definition of the function - for code optimization.
- You will often see C++ programs that have function declaration above main(), and function definition below main(). This will make the code better organized and easier to read:

```
// Function declaration
void myFunction();

// The main method
int main() {
  myFunction();  // call the function
  return 0;
}

// Function definition
void myFunction() {
  cout << "Hello There!";
}
```

Parameters and Arguments

- Information can be passed to functions as a parameter.
- Parameters act as variables inside the function.
- Parameters are specified after the function name, inside the parentheses.
- You can add as many parameters as you want, just separate them with a comma.
- **Syntax:**

```
void functionName(parameter1, parameter2, parameter3)
{
  // code to be executed
}
```

Parameters and Arguments (Cont.)

```
void myFunction(string fname)
{
  cout <<“Hello ”<< fname <<“\n”;
}
int main()
{
  myFunction(“Ali");
  myFunction(“Omar");
  myFunction(“Taha");
  return 0;
}
// Hello Ali
// Hello Omar
// Hello Taha
```

When a **parameter** is passed to the function, it is called an **argument**. So, from the example above: `fname` is a **parameter**, while `Ali` ،`Omar` and `Taha` are **arguments**.

C++ Default Parameters

```
void myFunction(string country = “Iraq")
{
  cout << country << "\n";
}
int main()
{
  myFunction(“Qatar");
  myFunction("India");
  myFunction();
  myFunction("USA");
  return 0;
}
// Qatar
// India
// Iraq
// USA
```

A parameter with a default value, is often known as an "**optional parameter**". From the example above, **country** is an optional parameter and “**Iraq**" is the default value.

Multiple Parameters

- Inside the function, you can add as many parameters as you want:

```
void myFunction(string fname, int age)
{
  cout << " My name is " << fname <<“ and my age is” << age << " years old. \n";
}
int main() {
  myFunction(“Amna", 3);
  myFunction(“Omar", 14);
  myFunction(“Hajer", 30);
  return 0;
}
// My name is Amna and my age is 3 years old.
// My name is Omar and my age is 14 years old.
// My name is Hajer and my age is 30 years old.
```

Note that when you are working with multiple parameters, the function call must have the **same number of arguments** as there are parameters, and the arguments must be **passed in the same order**.

Return Values

- The **void** keyword, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as **int**, **string**, etc.) instead of **void**, and use the **return** keyword inside the function:

```
int myFunction(int x)
{
  return 5 + x;
}

int main()
{
  cout << myFunction(3);
  return 0;
}

// Outputs 8 (5 + 3)
```

One parameter

```
int myFunction(int x, int y)
{
  return x + y;
}

int main()
{
  cout << myFunction(5, 3);
  return 0;
}

// Outputs 8 (5 + 3)
```

Two parameters

```
int myFunction(int x, int y)
{
  return x + y;
}

int main()
{
  int z = myFunction(5, 3);
  cout << z;
  return 0;
}
// Outputs 8 (5 + 3)
```

store the result in a variable

Pass By Reference

- In C++, **pass by reference** means passing the **address** of a variable to a function, allowing the function to directly **modify** the original variable.
- We use pass by reference to:
  - ✓To **modify the original data** inside the function.
  - ✓To **avoid copying large objects**, improving performance.

Pass by Reference vs. Pass by Value		
Feature	**Pass by Value**	**Pass by Reference**
Copies the value?	Yes	No
Original modified?	No	Yes
Memory efficiency	Less efficient (copy)	More efficient

Pass By Reference (Cont.)

```
void swapNums(int &x, int &y)
{
  int z = x;
  x = y;
  y = z;
}
int main() {
  int firstNum = 10;
  int secondNum = 20;
  cout << "Before swap: " << "\n";
  cout << firstNum << secondNum << "\n";
  // Call the function, which will change the values of firstNum and secondNum
  swapNums(firstNum, secondNum);
  cout << "After swap: " << "\n";
  cout << firstNum << secondNum << "\n";
  return 0;
}
```