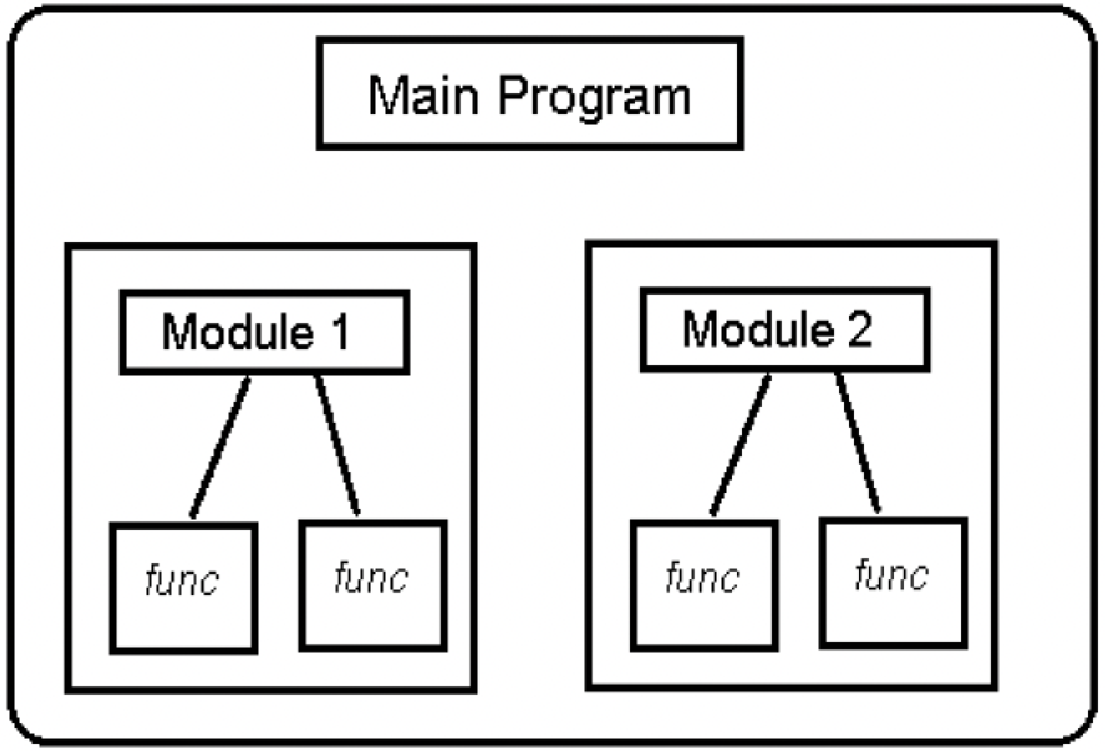
Fundamentals of Computer Programming

CS-110

Course Instructor: Dr. Momina Moetesum

Introduction to Modular Programming

Week 6



Learnning Objectives

01

To understand the need for Modular Programming

02

To introduce the concept of functions

03

To understand the basic syntax of functions

Modular Programming

Modular programming originated in the 1960s when developers began breaking up larger software programs into smaller parts.

Modular programming is a general programming concept where developers separate program functions into independent pieces.

These pieces then act like building blocks, with each block containing all the necessary parts to execute one aspect of functionality.

Each block is separate, strong, and testable, so you can stack them together at the end to create your application.

Levels of Modularity

MODULARITY WORKS ON MULTIPLE LEVELS:

FUNCTIONS WITHIN FILES

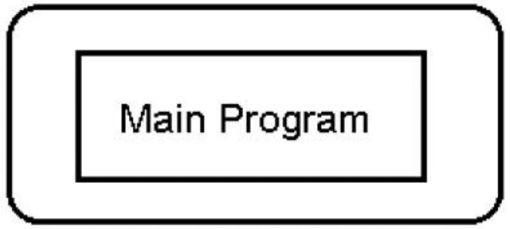
FILES WITHIN REPOSITORIES/LIBRARIES

LIBRARIES/REPOSITORIES WITHIN PROJECTS

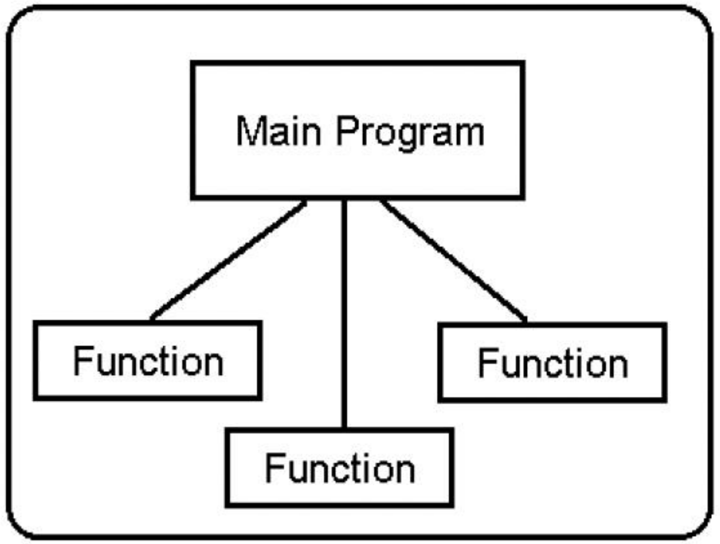
Advantages of Modular Programming

- CODE IS EASIER TO READ
- CODE IS EASIER TO DEBUG
- CODE REUSABILITY
- EASIER TO COLLABORATE
- EASIER REFACTORING (IN FRONT-END PROGRAMMING CASES)
- IMPROVES MAINTAINABILITY

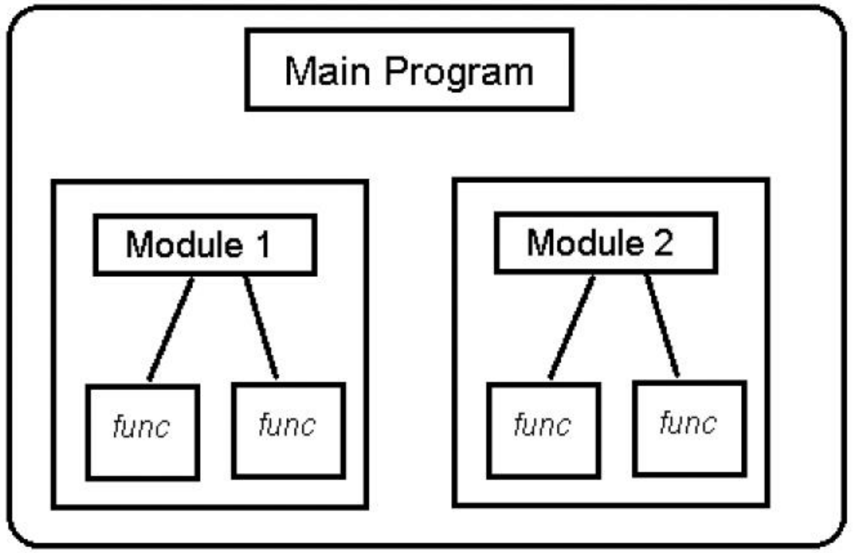
Unstructured vs Procedural vs Modular Programming



Unstructured



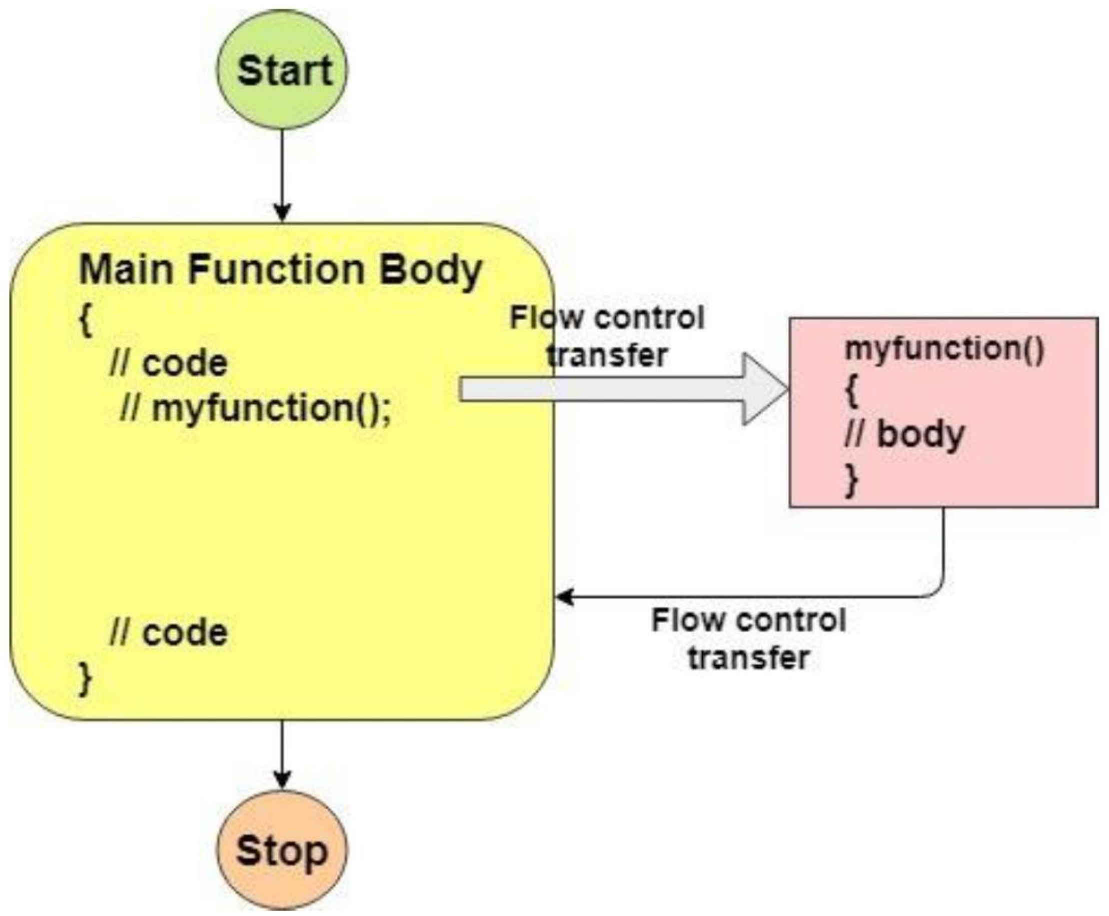
Procedural



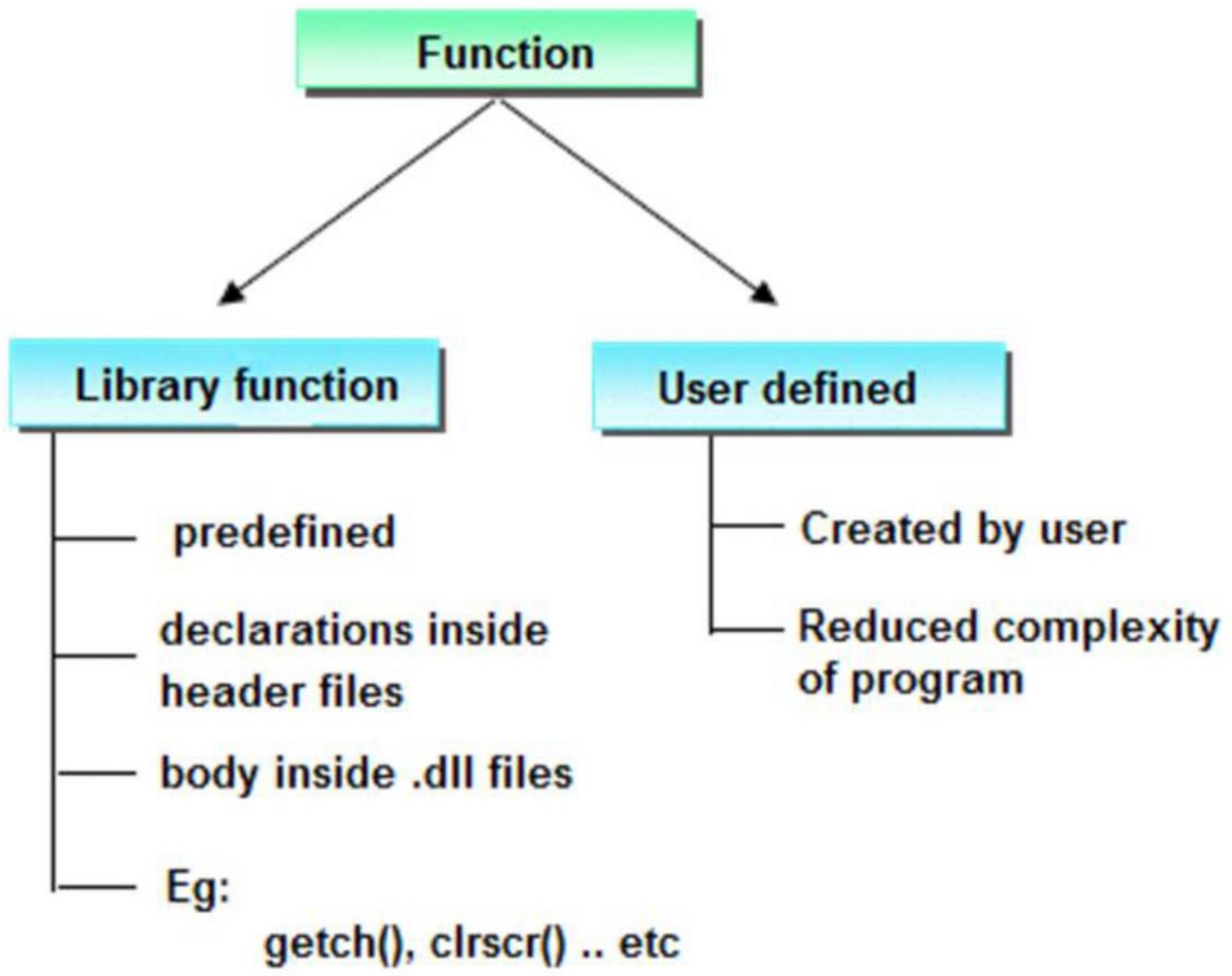
Modular

Functions

- **Functions** are a group of statements in a single logical unit to perform some specific task.
- Along with the main function, a program can have multiple functions that are designed for different operation.
- The results of functions can be used throughout the program without concern about the process and the mechanism of the function.



Types of Functions


Function
Library function
User defined
predefined
declarations inside header files
body inside .dll files
Eg: getch(), clrscr() .. etc
Created by user
Reduced complexity of program

Library Functions

Library functions are those which are predefined in C/C++ compiler.

The implementation part of pre-defined functions is available in library files that are .lib/.obj files. **.lib or .obj** files are contained pre-compiled code.

printf(), scanf(), clrscr(), pow() etc. are pre-defined functions.

As a programmer we do not having any controls on predefined function implementation part is there in machine readable format.

In implementation whenever a predefined function is not supporting user requirement then go for user defined function.

User Defined Functions

These functions are created by programmer according to their requirement.

For example, suppose you want to create a function for adding two numbers then you create a function with name sum(num, num). This type of function is called user defined function.

How Functions Work

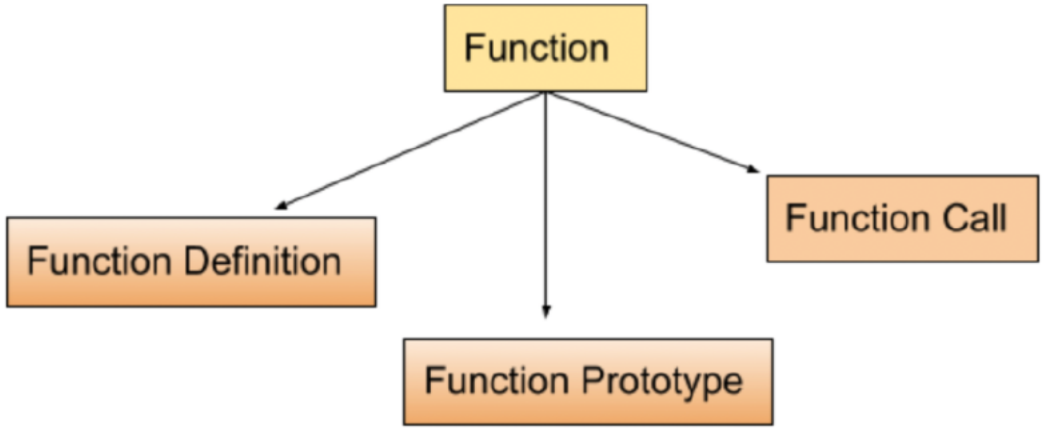
- Working of the C/C++ functions can be broken into the following steps as mentioned below:
- **Declaring a function:**Declaring a function is a step where we declare a function. Here we define the return types and parameters of the function.
- **Defining a function:**Defining the function is a step where we can code all the statements inside the function to get the final result.
- **Calling the function:**Calling the function is a step where we call the function by passing the arguments in the function.
- **Returning a value:**Returning a value is the step where the calculated value after the execution of the function is returned. Exiting the function is the final step where all the allocated memory to the variables, functions, etc is destroyed before giving full control to the main function.

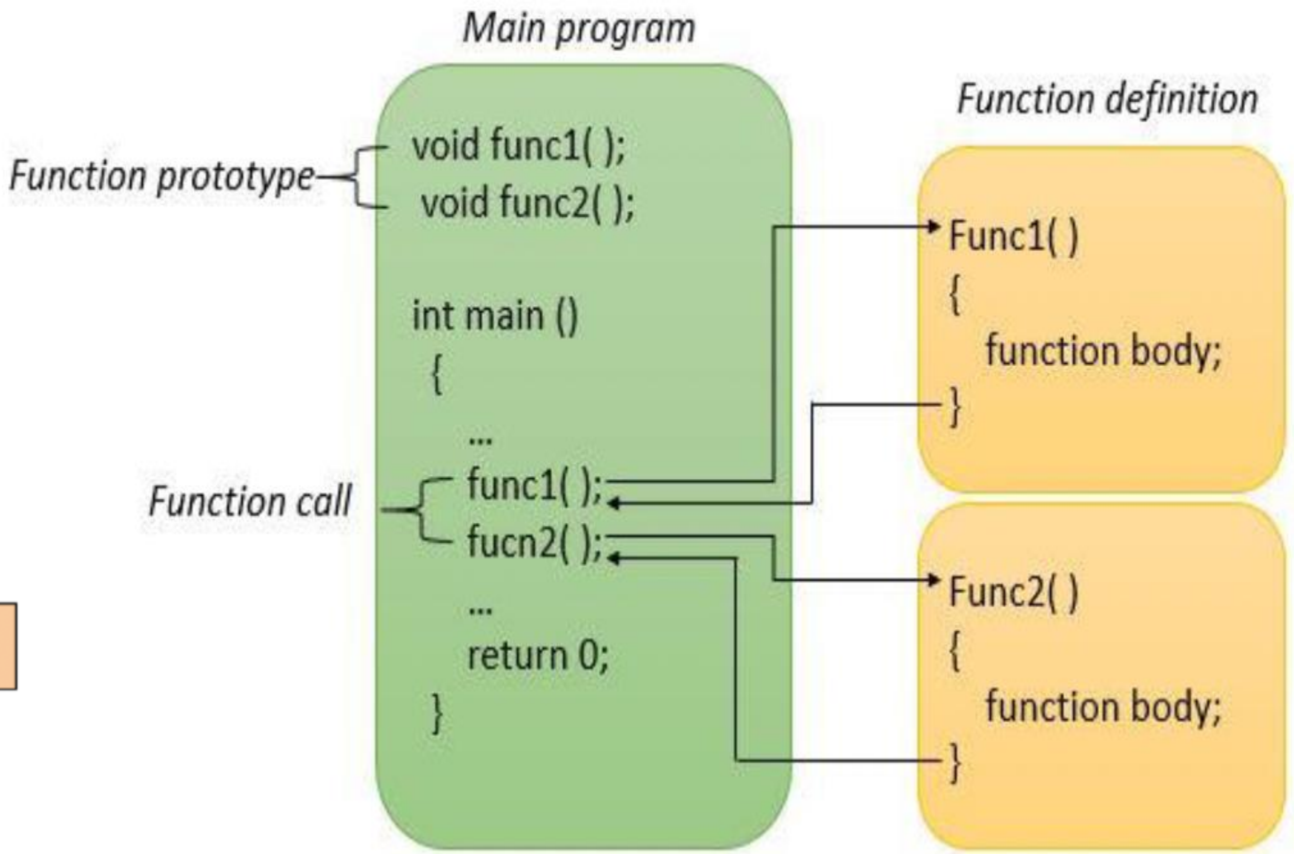
Advantages of Functions

- Functions reduce the repetition of the same statements in the program.
- Functions make code readable by providing modularity to our program.
- There is no fixed number of calling functions it can be called as many times as you want.
- Functions reduce the size of the program.
- Once the function is declared you can just use it without thinking about the internal working of the function.

Syntax of Functions in C/C++

- Generally, a C/C++ function has three parts:
  - **Function Prototype**
  - **Function Definition**
  - **Function Call**





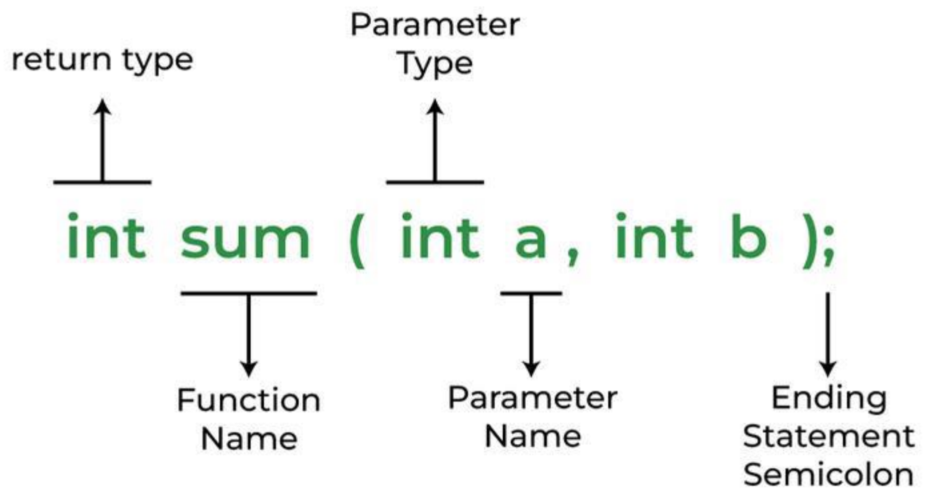
```
void func1( );
 void func2( );

int main ()
 {
   ...
   func1( );
   fucn2( );
   ...
   return 0;
 }
```

```
Func1( )
{
  function body;
}
```

```
Func2( )
{
  function body;
}
```

Function Prototype (Declaration)

- In a function declaration, we must provide the function name,
- its return type, and
- the number and type of its parameters.
- A function declaration tells the compiler that there is a function with the given name defined somewhere else in the program.
- Parameter name is **optional** in most cases
- Function prototype must end with a semicolon



Function Prototype (Declaration)

return_type: It is the data type of the value that the function returns. It can be any data type int, float, void, etc. If the function does not return anything, void is used as the return type.

function_name: It is the identifier of the function. Use appropriate names for the functions that specify the purpose of the function.

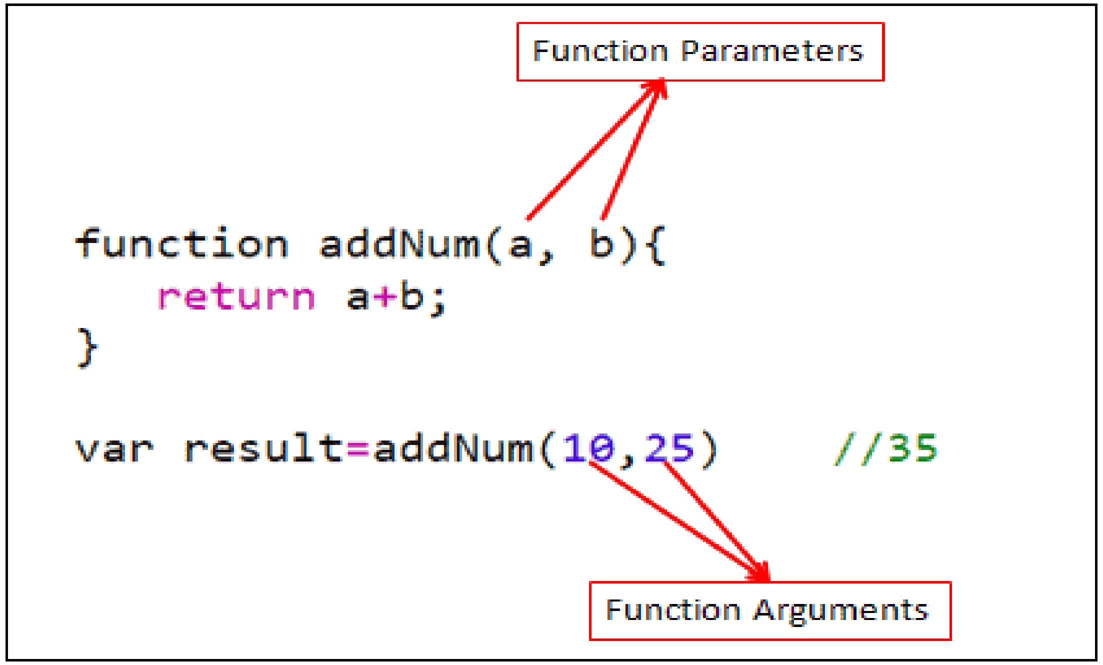
parameter_list: It is the list of parameters that a function expects in parentheses. A parameter consists of its data type and name. If we don’t want to pass any parameter, we can leave the parentheses empty.

Function-Return Type

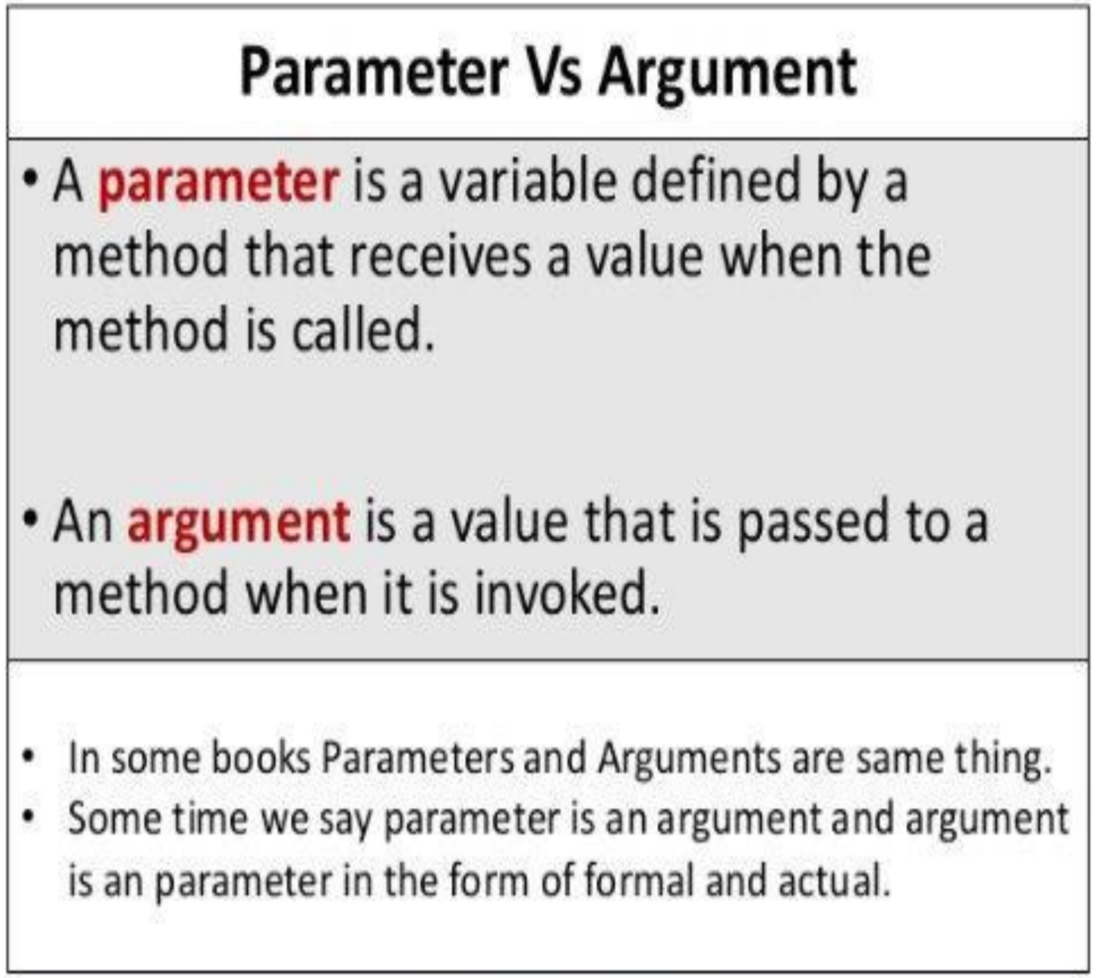
- Function return type tells what type of value is returned after all function is executed. When we don’t want to return a value, we can use the void data type.
    - Example: int func(parameter_1,parameter_2);
- The above function will return an integer value after running statements inside the function.
- ***Note:*** *Only one value can be returned from a C/C++ function. To return multiple values, we have to use pointers or structures.*

Function Parameters or Arguments

- Function Arguments (also known as Function Parameters in some cases) are the data that is passed to a function.



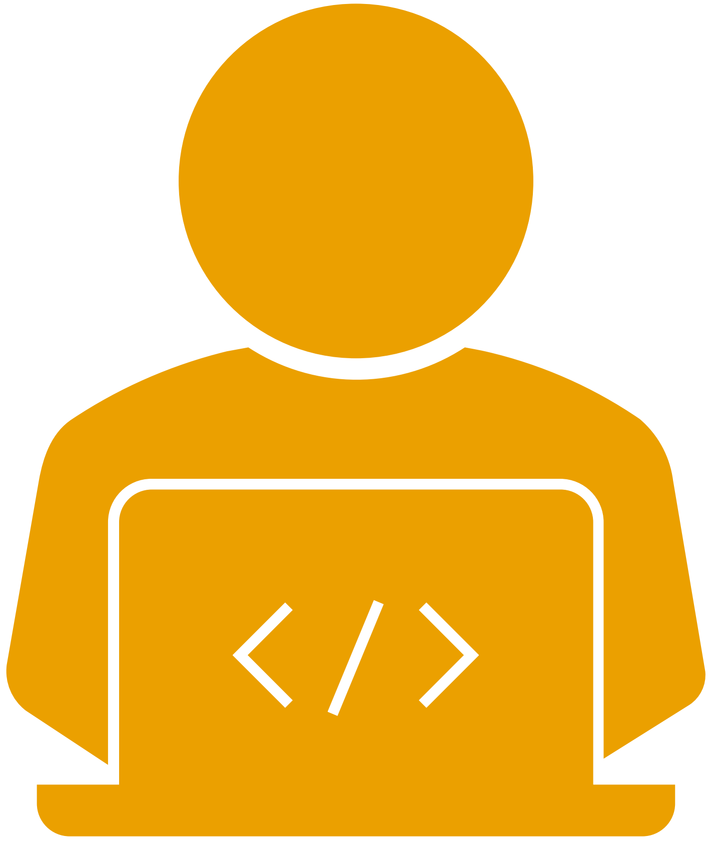
```
function addNum(a, b){
   return a+b;
}

var result=addNum(10,25)    //35
```



Parameter Vs Argument

- A **parameter** is a variable defined by a method that receives a value when the method is called.
- An **argument** is a value that is passed to a method when it is invoked.

- In some books Parameters and Arguments are same thing.
- Some time we say parameter is an argument and argument is an parameter in the form of formal and actual.

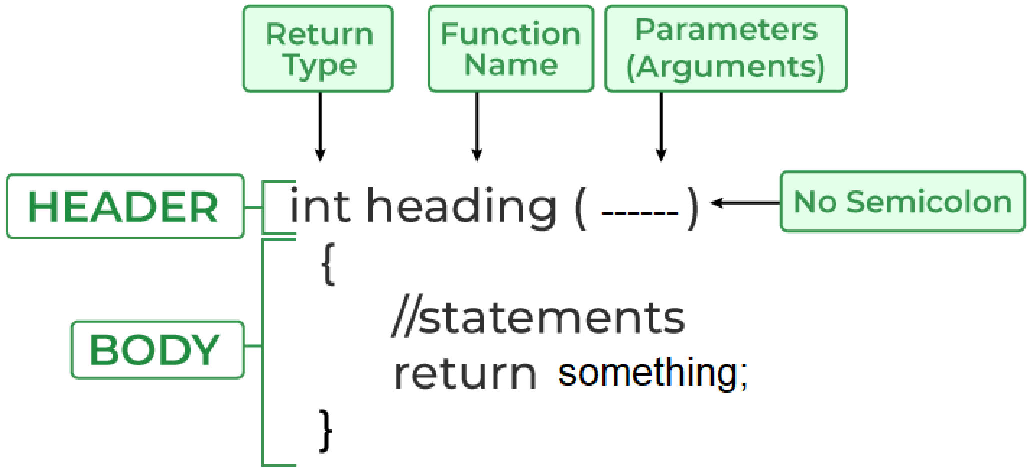


Function Parameters or Arguments

- In C/C++ programming language, functions can be called either with or without arguments and may or might not return values.
- Function with no arguments and no return value
- Function with no arguments and with return value
- Function with argument and with no return value
- Function with arguments and with return value

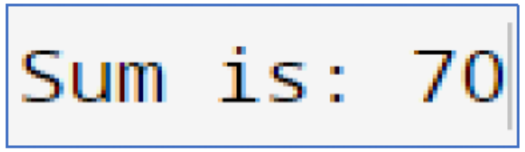
Function Call

- A function call is a statement that instructs the compiler to execute the function. We use the function name and parameters in the function call.
- In the below example, the first sum function is called and 10,30 are passed to the sum function.
- After the function call sum of a and b is r
returned back to the main function of the
- Types:
  - Call by Value
  - Call by Reference

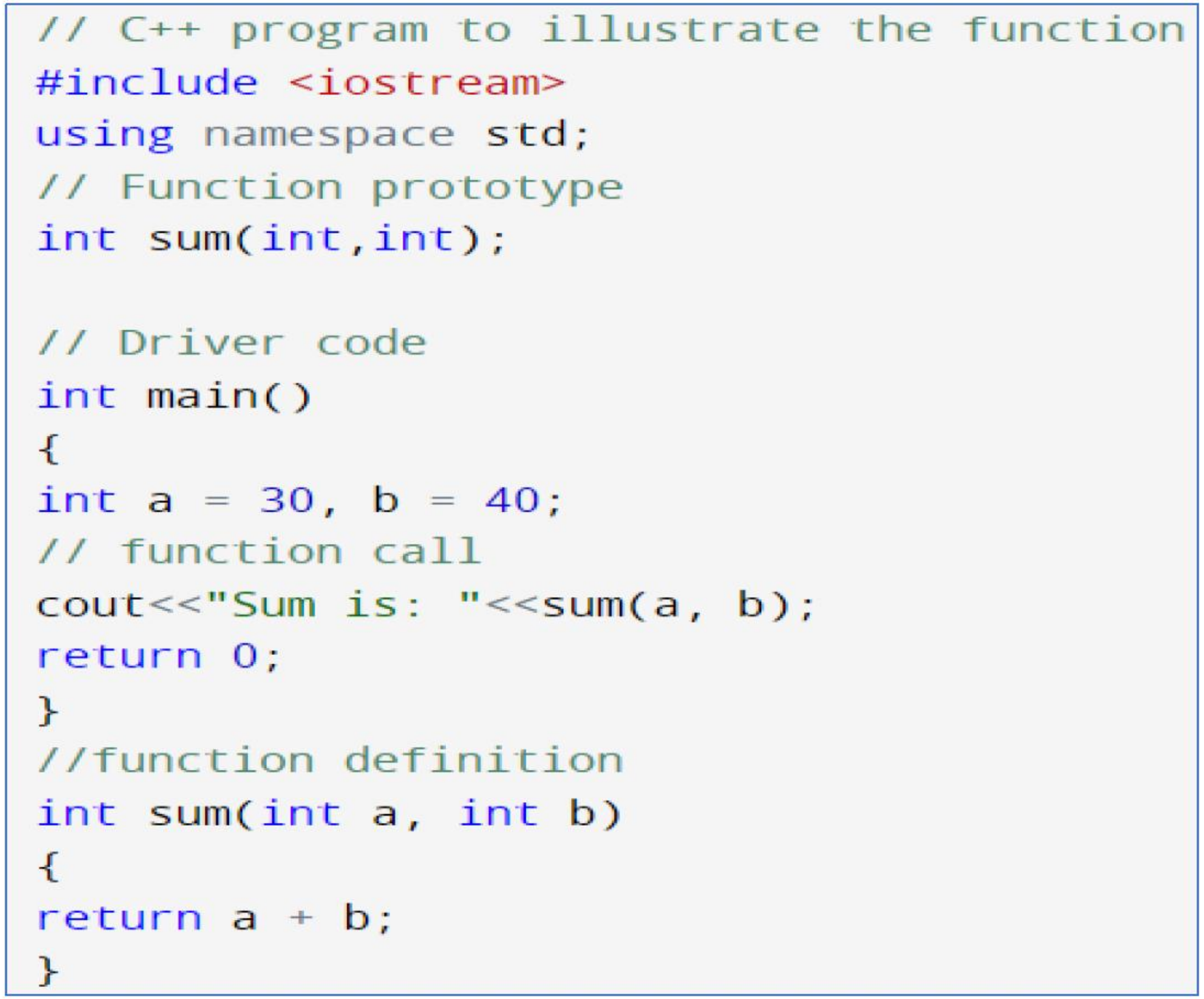


```
int heading ( ------ )
{
      //statements
      return something;
}
```

Example

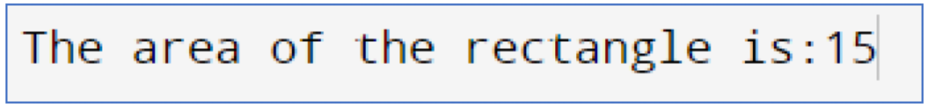


```
Sum is: 70
```

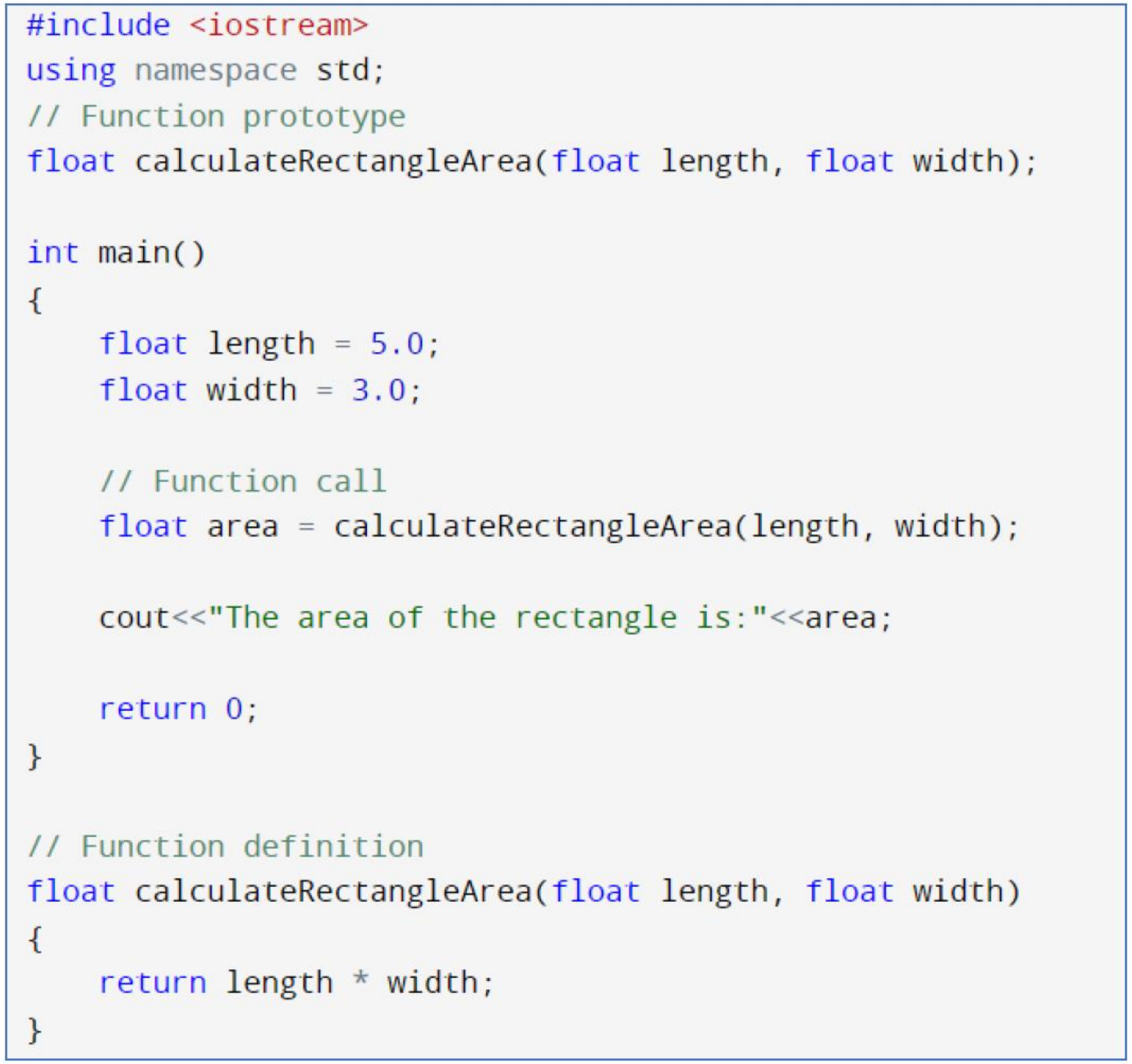


```
// C++ program to illustrate the function
#include <iostream>
using namespace std;
// Function prototype
int sum(int,int);

// Driver code
int main()
{
int a = 30, b = 40;
// function call
cout<<"Sum is: "<<sum(a, b);
return 0;
}
//function definition
int sum(int a, int b)
{
return a + b;
}
```

Example

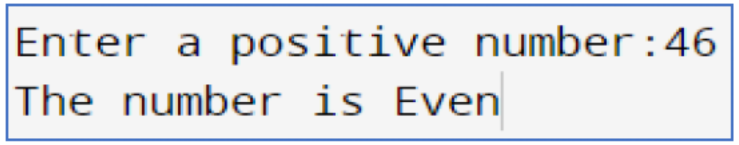


```
The area of the rectangle is:15
```

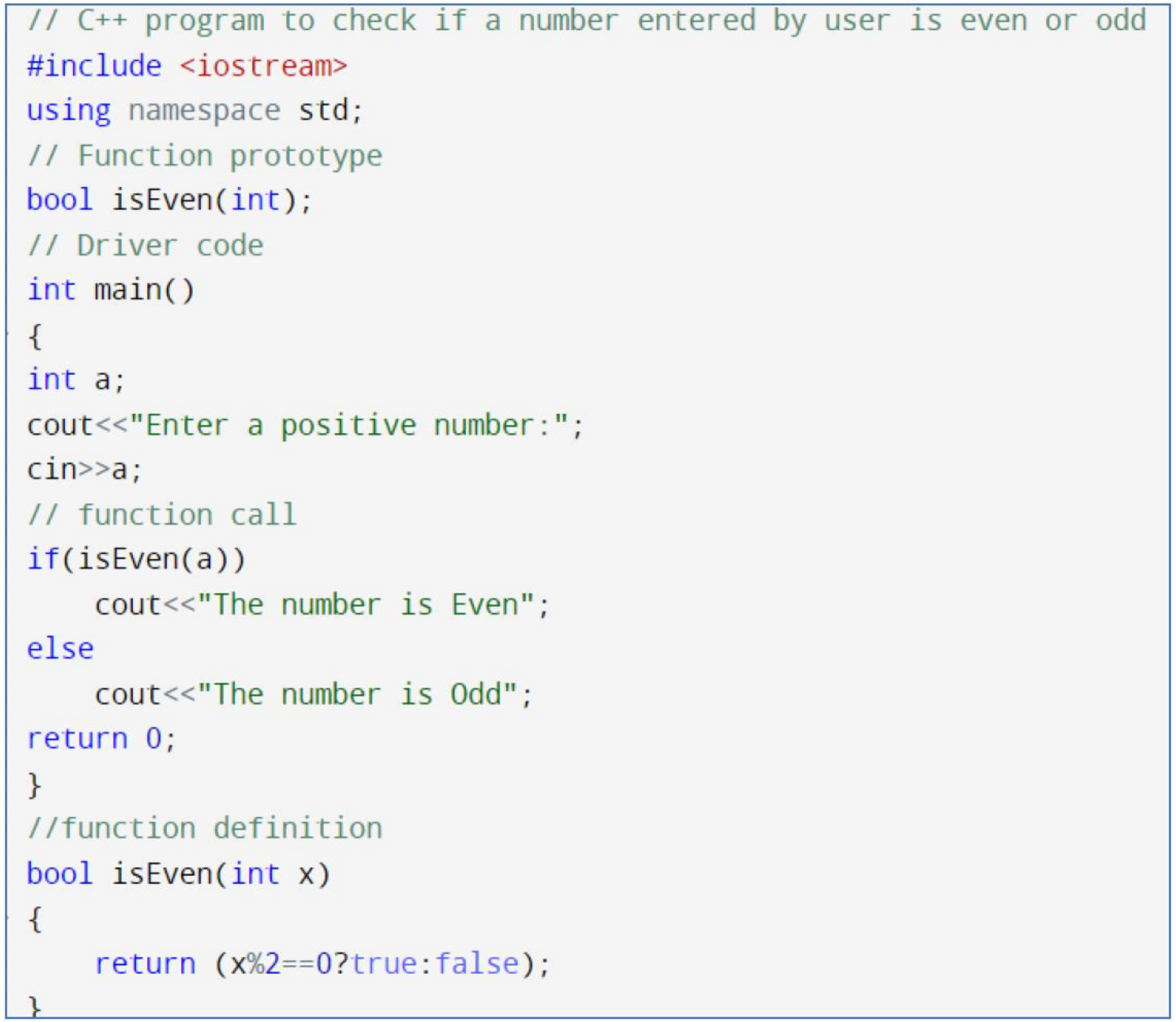


```cpp
#include <iostream>
using namespace std;
// Function prototype
float calculateRectangleArea(float length, float width);

int main()
{
    float length = 5.0;
    float width = 3.0;

    // Function call
    float area = calculateRectangleArea(length, width);

    cout<<"The area of the rectangle is:"<<area;

    return 0;
}

// Function definition
float calculateRectangleArea(float length, float width)
{
    return length * width;
}
```

Example



```
Enter a positive number:46
The number is Even
```



```cpp
// C++ program to check if a number entered by user is even or odd
#include <iostream>
using namespace std;
// Function prototype
bool isEven(int);
// Driver code
int main()
{
int a;
cout<<"Enter a positive number:";
cin>>a;
// function call
if(isEven(a))
    cout<<"The number is Even";
else
    cout<<"The number is Odd";
return 0;
}
//function definition
bool isEven(int x)
{
    return (x%2==0?true:false);
}
```

Acknowledgment

- Content of these slides are taken from:
  - https://www.geeksforgeeks.org/
  - https://www.tutorialspoint.com/
  - https://www.programiz.com/
  - https://www.trytoprogram.com/cplusplus-programming/functions/