



Programming Fundamentals with C++

Lecture 14 – Functions

Dr. Muhammad Sajjad

RA: Asad Ullah



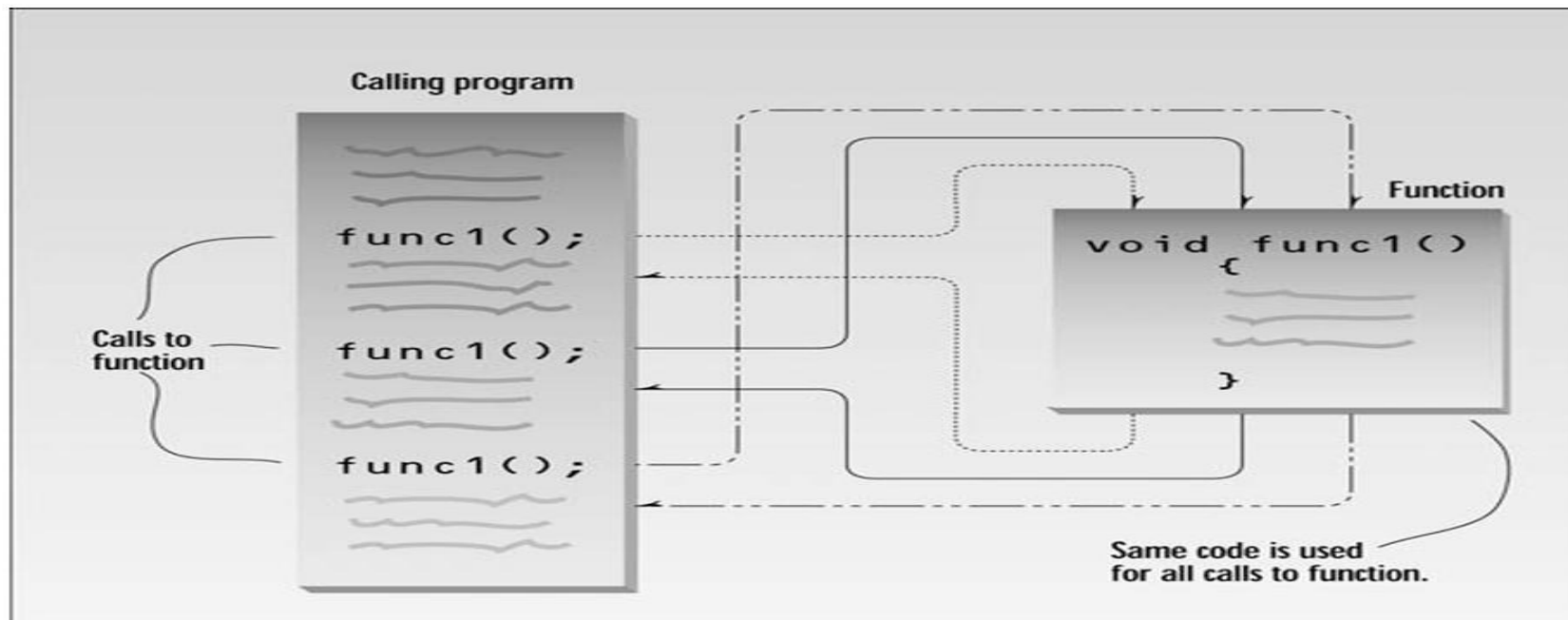
Overview

- What are Functions
- Advantages of Using Functions
- A Simple Function
- Steps to use a Function
- Code Example



What are Functions

- **Definition:** A function is a named unit of code that groups program statements and can be invoked multiple times.
- **Purpose:**
 - **Conceptual Organization:** Simplifies complex programs by dividing them into logical units.
 - **Code Reusability:** Avoids repetition by storing code in one place, reducing program size.
- **Origin:** Functions were invented to streamline programming and improve efficiency.
- **Usage:** Invoked from different parts of a program, allowing modular and structured programming.





Advantages of Using Functions

- **Code Reusability:** Write once, use multiple times.
- **Modularity:** Divide a large program into smaller, manageable parts.
- **Readability:** Makes the program easier to read and understand.
- **Debugging:** Errors are easier to locate and fix.

Advantages of Functions

- Easier to Code
- Easier to Modify
- Easier to Maintain
- Reusability
- Less Programming Time
- Easier to Understand



A Simple Function

- **Program Objective:** To create a simple table with lines of asterisks for better readability using a function.

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

void starline(); // Function declaration (prototype)

int main() {
    starline(); // Call to function
    cout << "Data type  Range" << endl;
    starline(); // Call to function
    cout << "char      -128 to 127" << endl
         << "short     -32,768 to 32,767" << endl
         << "int       System dependent" << endl
         << "long      -2,147,483,648 to 2,147,483,647" << endl;
    starline(); // Call to function
    return 0;
}
```

```
// starline()
// Function definition
void starline() {
    for (int j = 0; j < 45; j++) //
        Function body
        cout << '*';
    cout << endl;
}
```

```
*****
Data type  Range
*****
char      -128 to 127
short     -32,768 to 32,767
int       System dependent
long      -2,147,483,648 to
2,147,483,647
*****
```

A Simple Function

- **Key Components of the Program:**
 - **Function Declaration:** `void starline();` informs the compiler about the function's existence and its signature.
 - **Function Calls:** `starline();` is used in `main()` to invoke the function multiple times.
 - **Function Definition:** Provides the implementation of `starline()`, printing 45 asterisks followed by a new line.
- **Advantages:**
 - **Modular Code:** Simplifies readability and maintenance.
 - **Reusability:** Eliminates repetition by reusing `starline()` wherever required.
 - **Structure:** Demonstrates a clear flow of control between the main program and the function.

Steps to use a Function

1. The Function Declaration.
2. The Function Definition (write the code).
3. Calling the Function.

1. The Function Declaration

- Before using a function, the compiler needs to know its existence and structure. This is done using a function declaration, also known as a prototype.
- In the program TABLE, the function `starline()` is declared as:

```
#include<iostream>
// other libraries if exist
void starline(); // Declaration
.
.
.
.
```

Steps to use a Function

1. The Function Declaration

Purpose of Declaration:

Informs the compiler about the function's name (starline), return type (void), and any arguments it may take (none in this case).

Acts as a blueprint for the function, allowing the compiler to process calls to the function before its definition appears later in the code.

Components:

Return Type:

- Specifies the type of value the function returns.
- void means the function does not return any value.

Function Name:

- The name used to call the function in the program (e.g., starline).

Parameter List:

- The parentheses () indicate whether the function takes arguments.
- Empty parentheses mean no arguments are passed.

Steps to use a Function

1. The Function Declaration

Syntax of Function Declaration:

```
return_type function_name(argument_list);
```

Key Notes:

- The declaration ends with a semicolon (;).
- Declaring the function does not include its implementation; the definition (body) comes later.
- This step allows you to reference the function before defining it.

Example Prototype:

```
int add(int a, int b); // Declares a function 'add' that takes two integers and returns an integer  
void display();        // Declares a function 'display' that takes no arguments and returns nothing
```

Steps to use a Function

2. The Function Definition

Definition:

The function definition is where the actual code of the function is written. It specifies what the function does when it is called.

Components:

Declarator:

The first line of the definition that specifies the **function's name**, **return type**, and **parameters**.

Must match the function's declaration (prototype) exactly in **name**, **argument types**, **order**, and **return type**.

No semicolon at the end of the declarator.

Function Body:

Enclosed in curly braces {}.

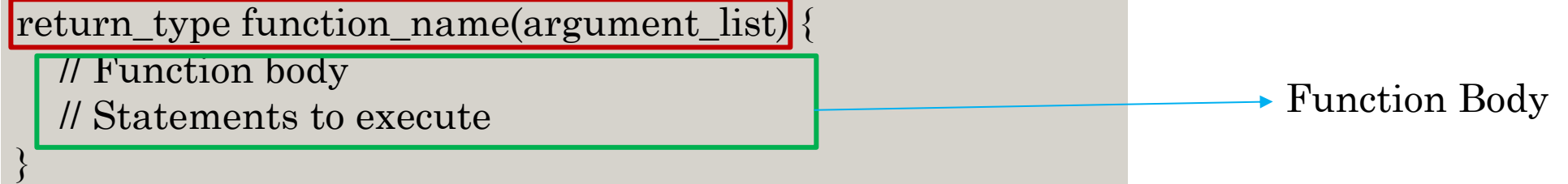
Contains the statements that are executed when the function is called.

Steps to use a Function

2. The Function Definition

Syntax:

```
return_type function_name(argument_list) {  
    // Function body  
    // Statements to execute  
}
```



The diagram illustrates the syntax of a function definition. A red box highlights the declarator part, `return_type function_name(argument_list)`, with a blue arrow pointing to the label "Declarator". A green box highlights the function body, which contains `// Function body` and `// Statements to execute`, with a blue arrow pointing to the label "Function Body".

Example Definition:

```
void starline() {           // Declarator  
    for (int j = 0; j < 45; j++) { // Function body  
        cout << '*';  
    }  
    cout << endl;  
}
```

Steps to use a Function

2. The Function Definition

Key Points:

- The declarator must exactly match the earlier function declaration.
- The function body contains all the logic or operations that the function will perform.
- No semicolon at the end of the declarator.

Workflow:

- Function is declared to inform the compiler.
- Function is defined to provide its implementation.
- Function is called from the program to execute the body.

Why the Function Definition is Critical:

- This is where the functionality of the function is implemented.
- Ensures reusability by allowing the same code to be called multiple times.

Steps to use a Function

3. Calling The Function

What is Function Calling:

- A function is "called" or "invoked" to execute the code defined in its body.

Syntax for Function Call:

```
function_name();
```

- Use the function name followed by parentheses.
- The statement ends with a semicolon.
- Unlike the declaration, the return type is not included when calling the function.

How it Works:

- Control transfers to the function being called.
- The statements in the function's body are executed.
- After execution, control returns to the point where the function was called, resuming the program from there.

Steps to use a Function

3. Calling The Function

Example:

- From the earlier `starline()` example:

```
starline(); // Call to the function
```

- This statement causes the `starline()` function to execute, printing a line of 45 asterisks.

Visualizing the Process:

- If the function is called multiple times in a program.

```
starline(); // First call: Executes the function  
cout << "Some data here" << endl;  
starline(); // Second call: Executes the function again
```

Output:

```
*****  
Some data here  
*****
```

Steps to use a Function

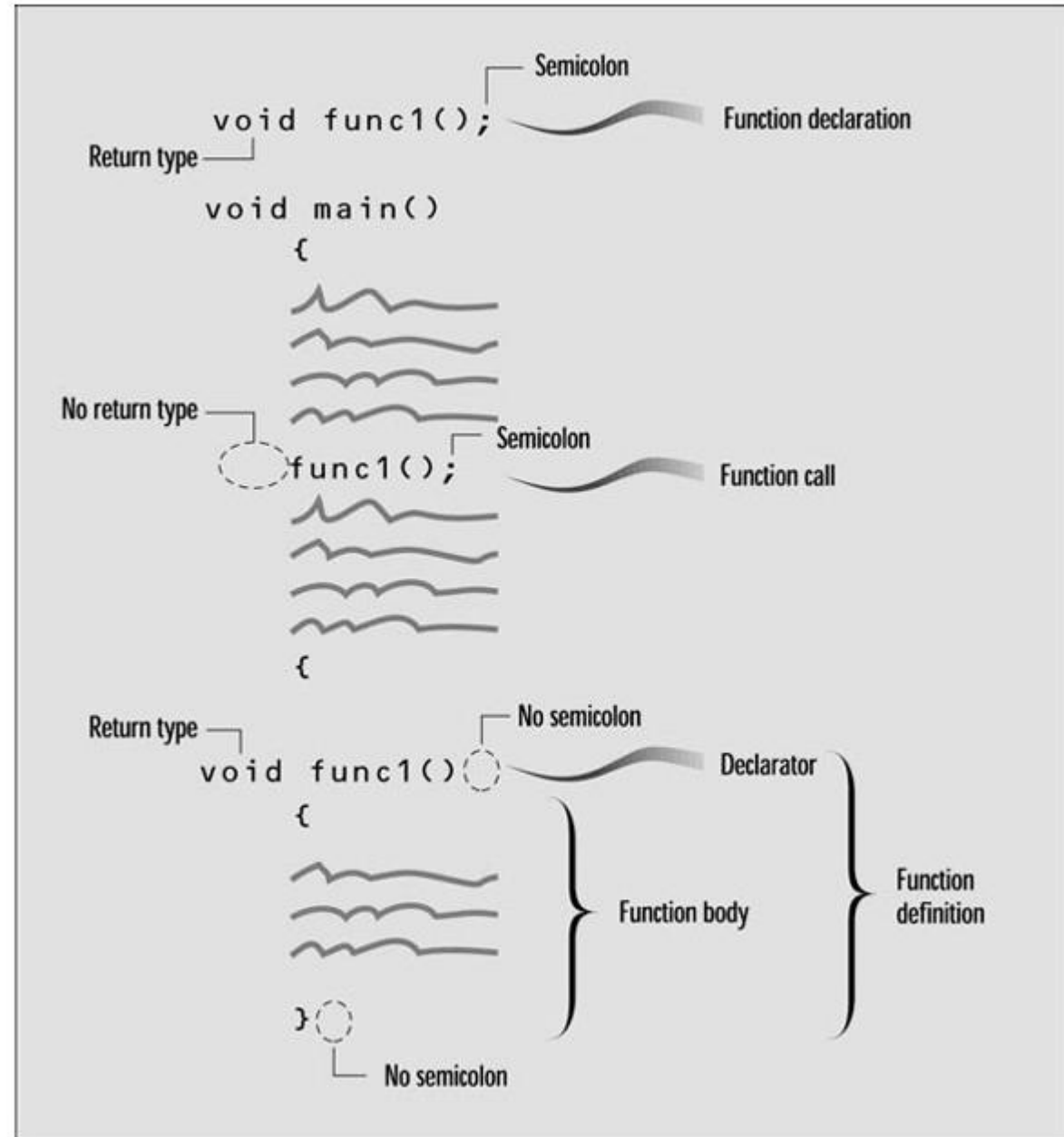
3. Calling The Function

Key Points:

- Function calls can be made multiple times from different parts of the program.
- Each call causes the function's body to execute from start to finish.
- Control automatically returns to the point following the function call after execution.

Note:

- In figure you can see that first the function is declared i.e. `void func1();`
- Then it is defined outside main i.e. at the last.
- And hence called in the main i.e. `func1();`



Code Example

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

// Function declaration (prototype)
void greet();

// Function definition
void greet() {
    cout << "Hello! Welcome to learning functions in C++." << endl;
}

// Main function
int main() {
    greet(); // Function call
    return 0;
}
```

Note: You can also define the function before the main. In that case the function will be declare and define at the same time (no need for explicit declaration). But it is recommended to declare the function always.

Thank You