Programming Fundamentals with C++

Lecture 15 – Functions

Dr. Muhammad Sajjad

RA: Asad Ullah

Overview

- **Declaring Functions that takes Arguments**
- **Defining Functions that takes Arguments**
- **Calling Functions that takes Arguments**
- **Passing Arguments to Functions**
- **Passing Arrays as Arguments**

Declaring Functions that takes Arguments

- **Function Declaration (Prototype)** tells the compiler about the function before it is defined.
- It includes the function name, return type, and parameters (arguments).

- **Syntax:**

```
returnType functionName(dataType param1, dataType param2, ...);
```

- **Example:**

```
void greet (string name, int age);  // Function declaration
```

Defining Functions that takes Arguments

- The function definition contains the actual code.
- The parameters receive values when the function is called.
- **Syntax:**

```
returnType functionName (dataType param1,
dataType param2, ...) {
    // Function body
}
```

- **Example:**

```
#include <iostream>
using namespace std;
void greet (char name[], int age);  // Function declaration
void greet (char name[], int age) {
    cout << "Hello, " << name << "! You are " << age << " years old."
<< endl;
}
```

Calling Functions that takes Arguments

- The function is executed when it is called in main(), with actual values (arguments) passed.
- **Example:**

```
int main() {
    greet("Ali", 20);  // Function call with arguments
    return 0;
}
```

- **Output:**

```
Hello, Ali! You are 20 years old.
```

Passing Arguments to Functions

- **Pass-by-Value**
  - A copy of the argument is passed to the function.
  - Changes inside the function **do not affect** the original variable.
  - **Example:**

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

void increase (int num) {
    num += 5;  // Only changes the local copy
    cout << "Inside function: " << num << endl;
}

int main() {
    int value = 10;
    increase (value);
    cout << "Outside function: " << value << endl; // Original remains
unchanged
    return 0;
}
```

Output:

```
Inside function: 15
Outside function: 10
```

Passing Arguments to Functions

- **Pass-by-Reference**
  - The actual variable is passed to the function using &.
  - Changes inside the function **affect** the original variable.
  - **Example:**

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

void increaseByReference (int &num) {  // Reference parameter
    num += 5;  // Modifies the actual variable
}

int main() {
    int value = 10;
    increaseByReference (value);
    cout << "After function call: " << value << endl;  // Value is modified
    return 0;
}
```

Output:

```
After function call: 15
```

Passing Arrays as Arguments

- Arrays are always passed by reference (without &).
- Modifications inside the function affect the original array.
- The function does **not** know the array size, so we pass it separately.
- **Example: Printing an Array**

```
#include <iostream>
using namespace std;
void printArray (int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int numbers[] = {1, 2, 3, 4, 5};
    int size = sizeof(numbers) / sizeof(numbers[0]);
    printArray (numbers, size);
    return 0;
}
```

Output

```
1 2 3 4 5
```

Passing Arrays as Arguments

- **Example: Modifying an Array**

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

void doubleArray (int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i] *= 2;  // Modifies original array
    }
}

int main() {
    int numbers[] = {1, 2, 3, 4, 5};
    int size = sizeof(numbers) / sizeof(numbers[0]);

    doubleArray (numbers, size);  // Function call
    cout << "Modified array: ";
    for (int i = 0; i < size; i++) {
        cout << numbers[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Output

```
Modified array: 2 4 6 8 10
```

Summary

- **Function Declaration** specifies the function name, return type, and parameters.
- **Function Definition** contains the actual function body.
- **Function Call** executes the function by passing arguments.
- **Pass-by-Value** keeps the original variable unchanged.
- **Pass-by-Reference** modifies the actual variable.
- **Passing Arrays** always passes a reference to the array.

Practice Questions for Students

- Write a function to find the **sum of array elements**.
- Write a function that **finds the maximum element** in an array.
- Write a function that **swaps two numbers** using pass-by-reference.
- Write a function that **reverses an array**.

Thank You