HIMALAYA COLLEGE OF ENGINEERING

Advanced C++ Programming Lab report

Lab 2: Use of array, Inline Function, Default Argument, Structure, Pass by reference, Return by reference, DMA in C++

Prepared By: Bikram Panthi

Subject: Object Oriented Programming (OOP)

Program: Bachelors in Electronics Engineering

Institution: Himalaya College of Engineering

Date: June 15, 2025

Objectives:

- To understand and implement arrays for storing and processing collection of data.
- To learn the use of inline functions for optimizing performance in function calls.
- To apply default arguments in functions for greater flexibility in function calls.
- To define and use structures for grouping related data types under a single unit
- To explore the concepts of passing and returning values by reference in functions
- To perform dynamic memory allocation and deallocation using pointers in C++
- To strengthen understanding of memory management and modular programming in C++

Tools and Libraries Used:

- Programming language: C++
- IDE: G++
- Libraries: #include <iostream>, include <string>

Theory:

Use of Array

An array is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow us to store multiple values under a single variable name, making it easier to manage and process large sets of data efficiently using loops.

```
// Declaration data_type
array_name[array_size]; //
```

```
Initialization int numbers[5] =
{10, 20, 30, 40, 50};

// Accessing elements cout <<
numbers[0]; // Outputs 10

// Using loop to access all elements
for (int i = 0; i < 5; i++)
{     cout << numbers[i] << " ";
}
```

Inline Function

An **inline function** is a function where the compiler attempts to insert the complete function code at each point the function is called, instead of performing a regular function call. This helps reduce the overhead of function calls, especially for small and frequently used functions.

```
Syntax: inline return_type
function_name(parameters) {
   // function body
}
```

Default argument

Default argument in C++ allow a function to have a default values for one or more parameters. If the caller does not provide values for those parameters, the default values are used automatically. This makes function calls more flexible and reduces the need for overloading. Default arguments are defined in the function declaration or definition, and they must be assigned from right to left.

Syntax:

```
return_type function_name(parameter1, parameter2 = default_value) {
   // function body }
```

Structure

A structure in C++ is a user-defined data type that groups related variables of different types under a single name. It is useful when you want to organize and manage complex data more efficiently, such as representing a student, employee, or product. Structures help improve code readability and modularity by keeping related information together.

Syntax:

```
struct structure_name {
data_type member1;     data_type
member2;
   // more members
};
```

Pass by reference

Pass by reference allows a function to access and modify the original variable directly by passing its memory address. Unlike pass by value, it doesn't create a copy, which makes it more efficient for large data and enables changes made inside the function to reflect outside. It is commonly used when multiple functions need to share and update the same data.

Syntax:

```
void function_name(data_type &parameter);
```

Return by reference

Return by reference allows a function to return a reference to a variable instead of a copy. This is useful when you want the caller to access or modify the original variable directly. It improves performance by avoiding unnecessary copying and is often used when working with large data or when a function needs to return a modifiable value.

Syntax:

```
 data_type& function_name(parameters);
```

DMA in C++

DMA in C++ usually refers to **Dynamic Memory Allocation**. It means allocating memory at runtime (during program execution) rather than at compile time. C++ provides operators new and delete to allocate and deallocate memory dynamically.

Basic Syntax:

- Allocate memory for a single object:

```
Type* ptr = new Type;
```

- Allocate memory for an array of objects:

```
Type* arr = new Type[size];
```

- Deallocate memory for a single object:

```
delete ptr;
```

- Deallocate memory for an array:

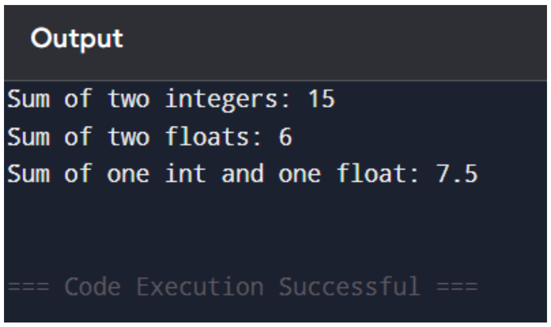
```
delete[] arr;
```

Q1: Write a C++ program to overload a function add() to handle:

Two integers, Two floats ,One integer and one float

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

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

float add(float x, float y) {
return x + y;
}
// Function to add one integer and one float
float add(int a, float b) {
return a + b;
}

int main() {
int a = 5, b = 10;
float x = 3.5f, y = 2.5f;

cout << "Sum of two integers: " << add(a, b) << endl;
cout << "Sum of two floats: " << add(x, y) << endl;
cout << "Sum of one int and one float: " << add(a, y) << endl;

return 0;
}
```

Output:

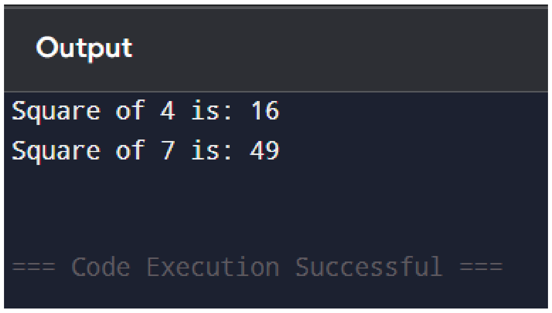


```
Sum of two integers: 15
Sum of two floats: 6
Sum of one int and one float: 7.5


=== Code Execution Successful ===
```

Q2: Write an inline function in C++ to calculate the square of a number and demonstrate it with at least two function calls.

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

inline int square(int x) {
    return x * x;
}

int main() {
    int a = 4;
    int b = 7;

    cout << "Square of " << a << " is: " << square(a) << endl;
    cout << "Square of " << b << " is: " << square(b) << endl;

    return 0;
}
```

Output:

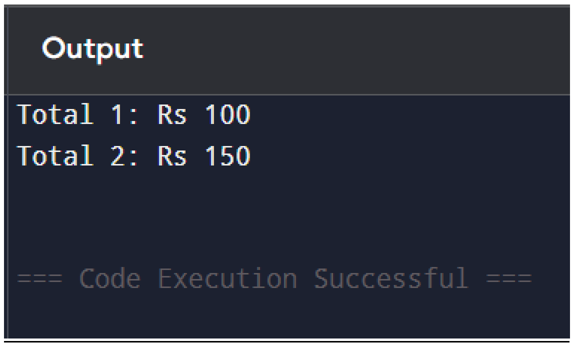


```
Square of 4 is: 16
Square of 7 is: 49


=== Code Execution Successful ===
```

Q3: Write a program using a function with default arguments for calculating total price. The function should take the item price and quantity, with quantity defaulting to 1.

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

float totalPrice(float price, int quantity = 1) {
    return price * quantity;
}

int main() {
    cout << "Total 1: Rs " << totalPrice(100) << endl;
    cout << "Total 2: Rs " << totalPrice(50, 3) << endl;

    return 0;
}
```

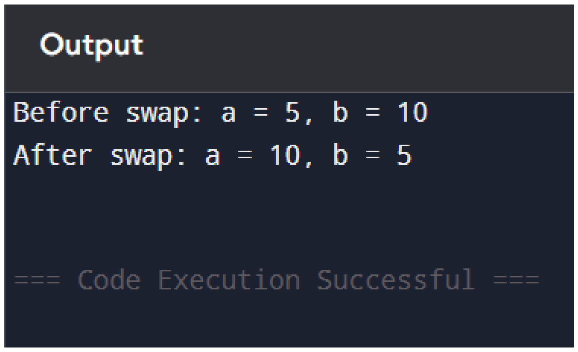
Output:



```
Total 1: Rs 100
Total 2: Rs 150


=== Code Execution Successful ===
```

Q4: Write a C++ program to swap two numbers using pass-by-reference.

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

void swap(int &x, int &y) {
    int temp = x;
    x = y;
    y = temp;
}

int main() {
    int a = 5, b = 10;
    cout << "Before swap: a = " << a << ", b = " << b << endl;

    swap(a, b);

    cout << "After swap: a = " << a << ", b = " << b << endl;

    return 0;
}
```

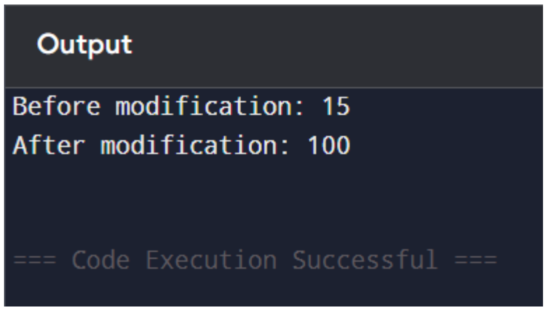


Output

```
Before swap: a = 5, b = 10
After swap: a = 10, b = 5


=== Code Execution Successful ===
```

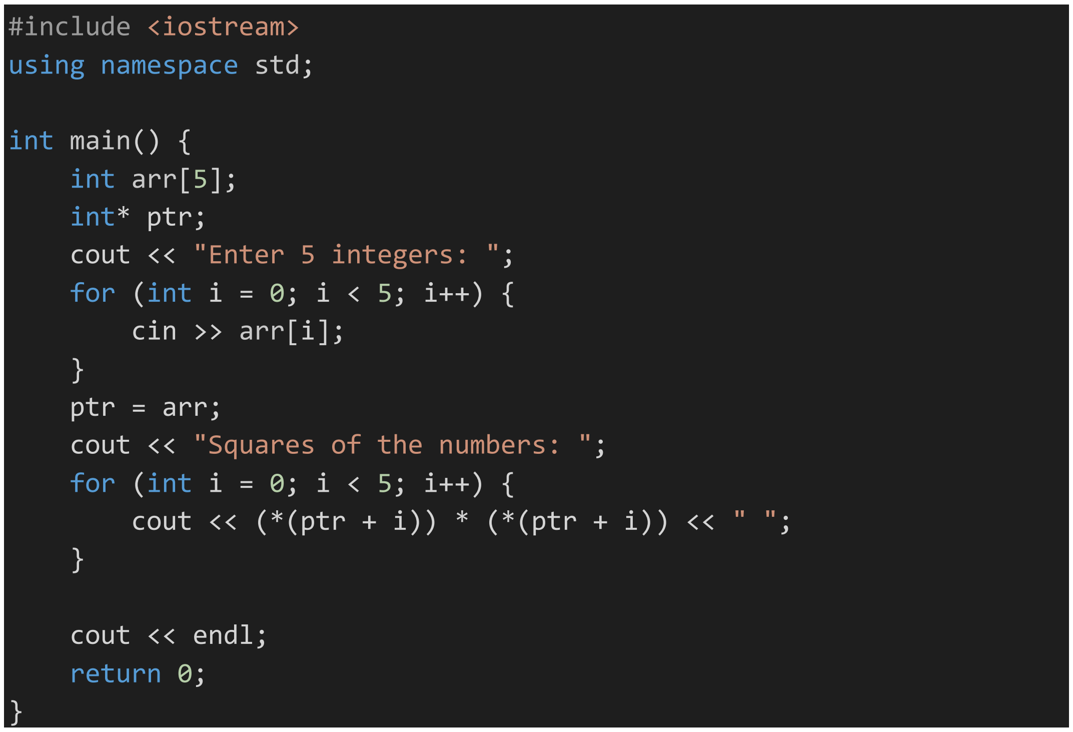
Q5: Create a function that returns a reference to an element in an array and modifies it.

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

int& getArray(int arr[], int index) {
    return arr[index];
}

int main() {
    int numbers[5] = {3, 6, 9, 12, 15};

    cout << "Before modification: " << numbers[4] << endl;
    getArray(numbers, 4) = 100;
    cout << "After modification: " << numbers[4] << endl;

    return 0;
}
```

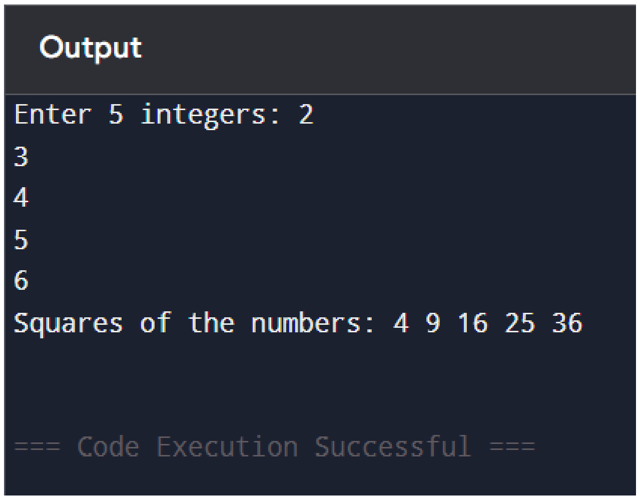


```
Before modification: 15
After modification: 100


=== Code Execution Successful ===
```

Q6: Write a program to input 5 integers in an arrayand print their squares using a pointer.



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

int main() {
    int arr[5];
    int* ptr;
    cout << "Enter 5 integers: ";
    for (int i = 0; i < 5; i++) {
        cin >> arr[i];
    }
    ptr = arr;
    cout << "Squares of the numbers: ";
    for (int i = 0; i < 5; i++) {
        cout << (*(ptr + i)) * (*(ptr + i)) << " ";
    }

    cout << endl;
    return 0;
}
```



```
Enter 5 integers: 2
3
4
5
6
Squares of the numbers: 4 9 16 25 36


=== Code Execution Successful ===
```

Discussion:

In this lab, we worked with arrays, inline functions, default arguments, structures, and pass/return by reference. These features helped us write more organized, flexible, and efficient C++ programs. We saw how arrays store multiple values, structures group related data, and functions become more powerful with default arguments and references. Each part improved our understanding of writing better code.

Conclusion:

This lab helped us understand important features of C++ like arrays, structures, default arguments, and references. By using them, we learned how to make our programs more readable, efficient, and easier to manage. It also gave us confidence in writing modular and well-structured code.