TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: - 02

Title: - Exploring Advanced C++ Programming Concepts

Submitted by: -

Name: - Diwas Pokhrel

Roll NO: -HCE081BEI014

Submitted To: -

Department of Electronics and Computer Engineering

Checked by: -

Date of submission: - 2025/06/08

Objectives:

- To demonstrate function overloading & implement inline functions.
- To manipulate arrays using pointers&compare structure and union.
- To explore defalut argument and understand pass-by-reference.
- To use eunumerations and manage dynamic memory allocation.

Tools and Libraries Used

• Programming Language: C++

• IDE: Clyang

• Libraries: #include <iostream>, #include <cmath>, #include <string>

THEORY

INTRODUCTION

C++ is a high-performance programming language widely used in software development, game programming, and system applications. One of its key strengths is its support for multiple programming paradigms, including procedural, object-oriented, and generic programming.It focuses on fundamental C++ features that enhance code efficiency and flexibility.

1. Function Overloading

Function overloading allows multiple functions to share the same name but have different parameters (type or number). The compiler selects the correct version based on the arguments provided.

Why use it?

- Simplifies code by using one function name for similar operations.
- Handles different data types without needing separate function names.

Example:

```
int add(int a, int b);     // Adds two integer

                    float add(float x, float y); // Adds two floats

              float add(int a, float b); // Adds an integer and a float
```

2. Inline Functions

Inline functions are expanded at compile time, reducing the overhead of function calls. They are best for small, simple functions called frequently.

Key Points:

- Improves performance by avoiding function call overhead.
- Defined using the inline keyword.

Example:

```
inline int square(int n) {return n*n;}
```

2. Default Arguments

Functions can have parameters with default values, which are used if no argument is provided.

Why use it?

- Makes functions more flexible.
- Reduces the need for multiple similar functions.

Example:

```
float calculateTotal(float price, int quantity = 1);

// If quantity is not provided, it defaults to 1
```

4. Pass-by-Reference

Pass-by-reference allows functions to modify the original variable directly, avoiding the need for pointers.

Example

```
Void swapnum(int &a,int &b);
```

5.Pointers and Arrays

A pointer stores the memory address of a variable. It can be used to traverse and manipulate arrays efficiently.

Why use pointers?

- Provides direct memory access.
- Enables dynamic memory allocation.

Example:

```
int arr[5]:

int *ptr = arr; // Pointer to the first element

cout << *(ptr + 2); // Accesses the third element
```

6. Structures & Unions

Both store multiple data types, but they differ in memory usage:

Example:

```
struct Student (int roll; float marks; ); // Allocates 8 bytes (4 + 4)

union Data f int i; float f; / /Allocates 4 bytes (largest member)
```

7. Enumerations (Enums)

Enums assign names to integer constants, making code more readable.

Why use enums?

- Improves clarity (e.g., Monday instead of 0).
- Reduces errors from using arbitrary numbers.

Example:

```
enum Day ( Sunday, Monday, Tuesday ):

Day today = Monday; // today holds value
```

8. Dynamic Memory Allocation

new : Allocates memory at runtime (e.g., for arrays).

delete :Frees memory to prevent leaks.

Why use it?

- Memory is allocated only when needed.
- Prevents wasted memory for unused variables.

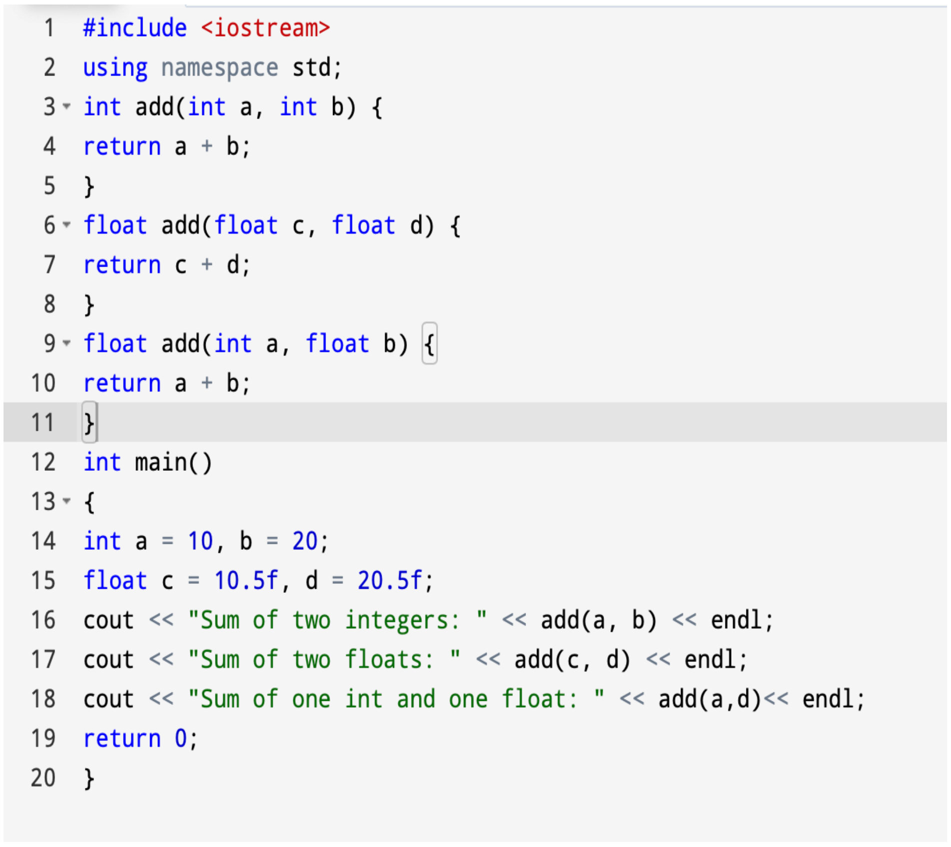
Example:

```
int *arr = new int[10]:  // Allocates memory for 10 integers

delete[] arr; // Frees the memory
```

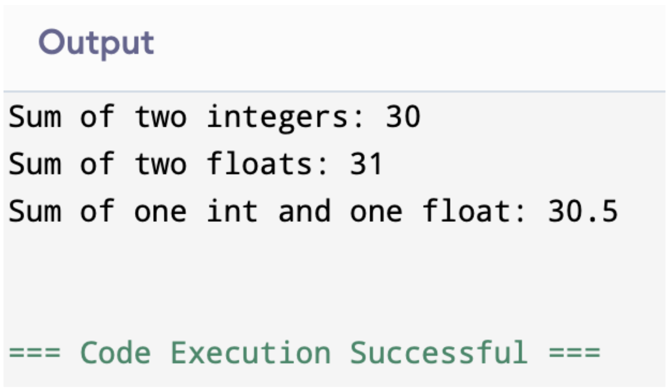
Certain Programs are:

Q.1) 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 c, float d) {
return c + d;
}
float add(int a, float b) {
return a + b;
}
int main()
{
int a = 10, b = 20;
float c = 10.5f, d = 20.5f;
cout << "Sum of two integers: " << add(a, b) << endl;
cout << "Sum of two floats: " << add(c, d) << endl;
cout << "Sum of one int and one float: " << add(a,d)<< endl;
return 0;
}
```

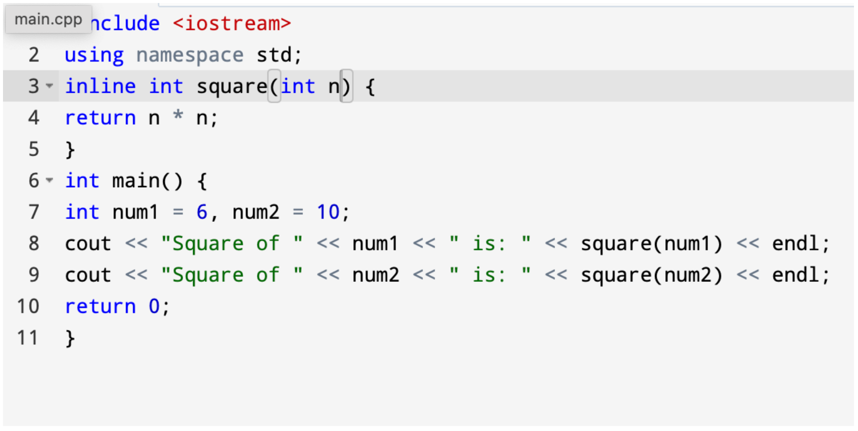


Output

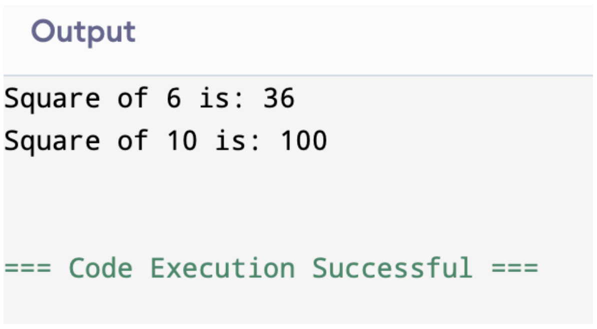
```
Sum of two integers: 30
Sum of two floats: 31
Sum of one int and one float: 30.5


=== 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.

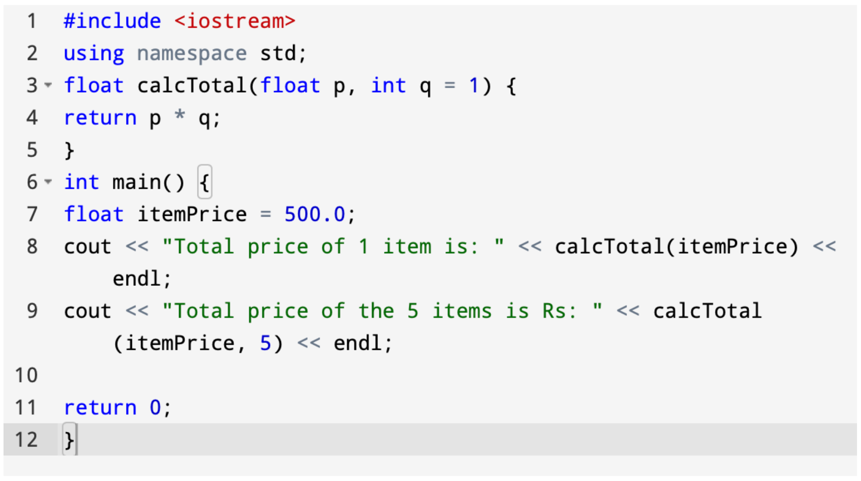


```cpp
nclude <iostream>
using namespace std;
inline int square(int n) {
return n * n;
}
int main() {
int num1 = 6, num2 = 10;
cout << "Square of " << num1 << " is: " << square(num1) << endl;
cout << "Square of " << num2 << " is: " << square(num2) << endl;
return 0;
}
```

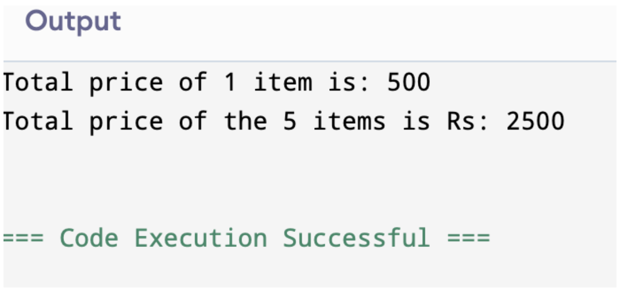


```
Square of 6 is: 36
Square of 10 is: 100


=== 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



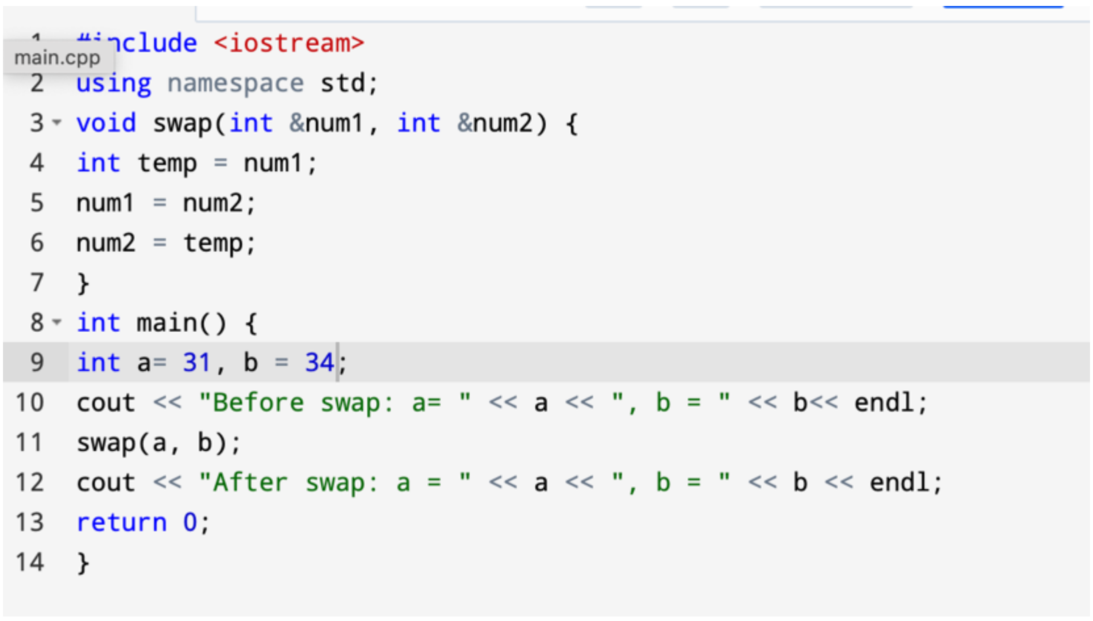
```cpp
#include <iostream>
using namespace std;
float calcTotal(float p, int q = 1) {
return p * q;
}
int main() {
float itemPrice = 500.0;
cout << "Total price of 1 item is: " << calcTotal(itemPrice) <<
    endl;
cout << "Total price of the 5 items is Rs: " << calcTotal
    (itemPrice, 5) << endl;

return 0;
}
```



Output

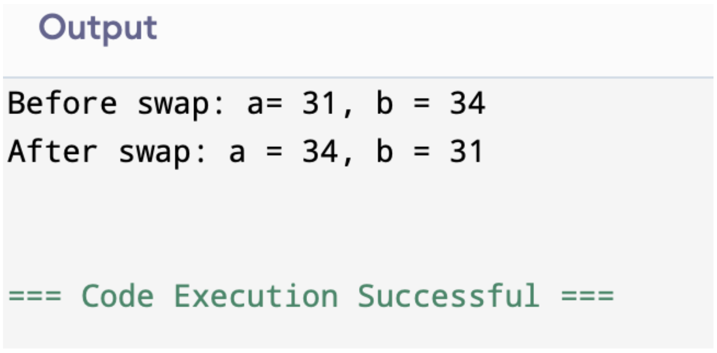
```
Total price of 1 item is: 500
Total price of the 5 items is Rs: 2500


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

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



main.cpp

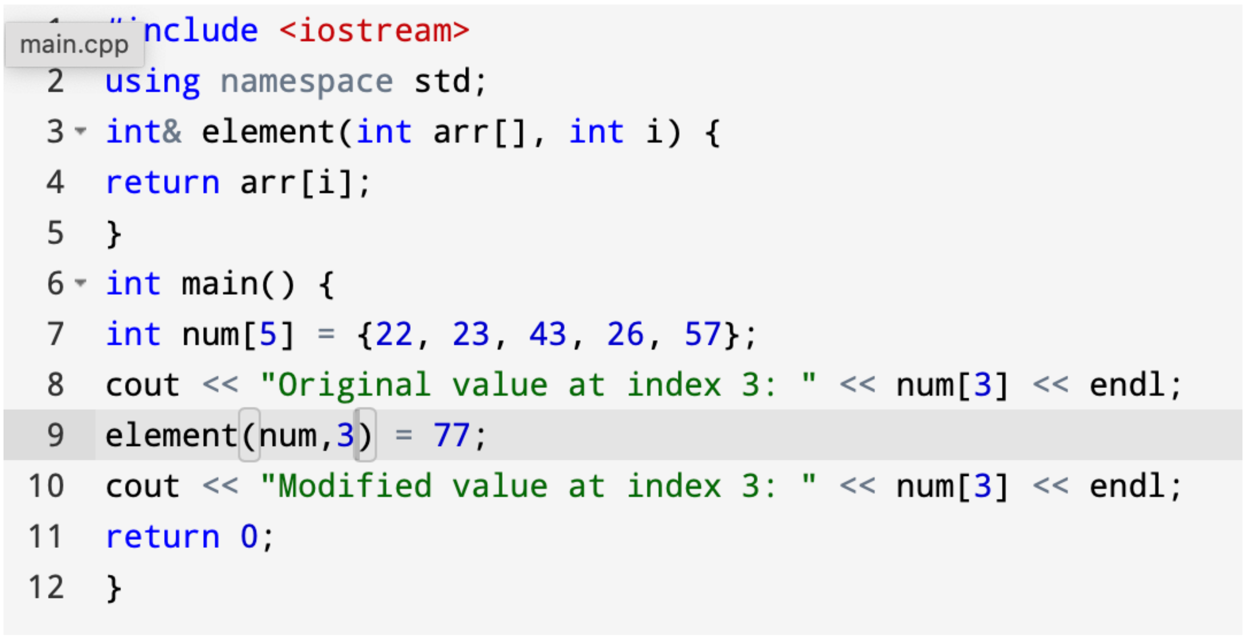
```cpp
#include <iostream>
using namespace std;
void swap(int &num1, int &num2) {
int temp = num1;
num1 = num2;
num2 = temp;
}
int main() {
int a= 31, b = 34;
cout << "Before swap: a= " << a << ", b = " << b<< endl;
swap(a, b);
cout << "After swap: a = " << a << ", b = " << b << endl;
return 0;
}
```



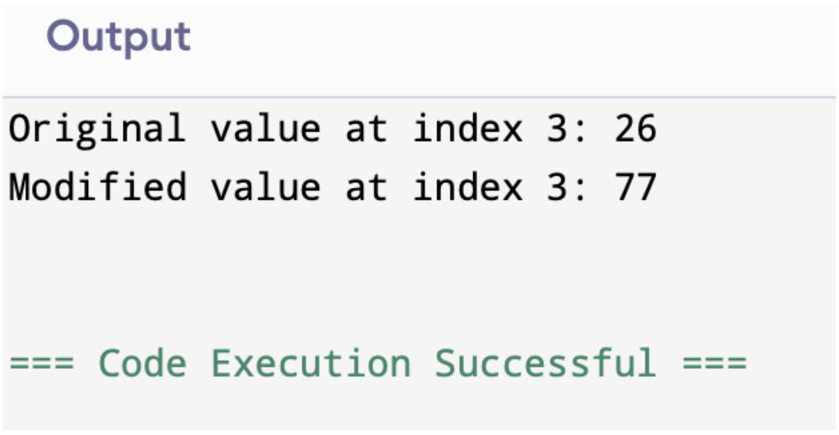
Output

```
Before swap: a= 31, b = 34
After swap: a = 34, b = 31


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

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

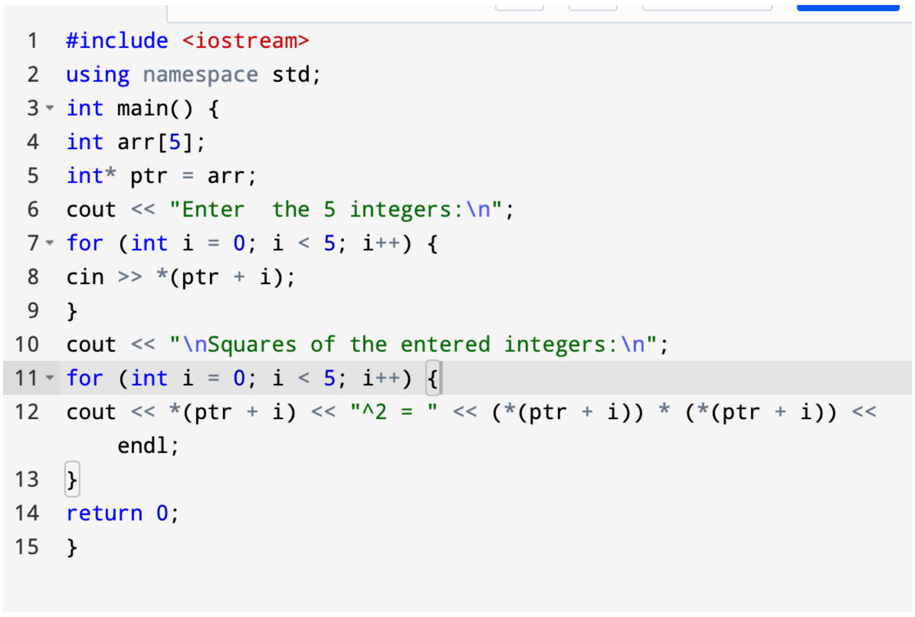


```cpp
#include <iostream>
using namespace std;
int& element(int arr[], int i) {
return arr[i];
}
int main() {
int num[5] = {22, 23, 43, 26, 57};
cout << "Original value at index 3: " << num[3] << endl;
element(num,3) = 77;
cout << "Modified value at index 3: " << num[3] << endl;
return 0;
}
```

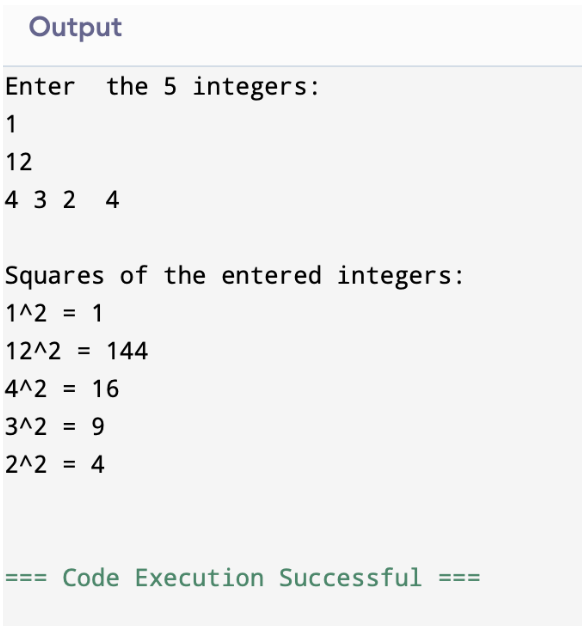


```
Original value at index 3: 26
Modified value at index 3: 77


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

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



```
#include <iostream>
using namespace std;
int main() {
int arr[5];
int* ptr = arr;
cout << "Enter  the 5 integers:\n";
for (int i = 0; i < 5; i++) {
cin >> *(ptr + i);
}
cout << "\nSquares of the entered integers:\n";
for (int i = 0; i < 5; i++) {
cout << *(ptr + i) << "^2 = " << (*(ptr + i)) * (*(ptr + i)) <<
    endl;
}
return 0;
}
```



Output

```
Enter  the 5 integers:
1
12
4 3 2  4

Squares of the entered integers:
1^2 = 1
12^2 = 144
4^2 = 16
3^2 = 9
2^2 = 4


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

Discussion:

This lab covered key C++ concepts such as function overloading, inline functions, default arguments, pass-by-reference, pointer manipulation, structures vs unions, enums, and dynamic memory allocation. Each program demonstrated how these features work in real scenarios, improving code flexibility, efficiency, and clarity. Practical tasks helped reinforce how memory and functions can be handled efficiently in C++.

Conclusion

The lab successfully enhanced understanding of C++ programming fundamentals. Through hands-on practice, we gained valuable experience in memory management, function design, and data structures. These skills are essential for writing optimized and maintainable code in modern software development.