TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: - 02

Title: - Exploring Advanced C++ Programming Concepts

Submitted by: -

Name: -Gaurab Pandey

Roll NO: - 15

Submitted To: -

Department Of C and Electronics

Checked by: -

Date of submission: -

Objectives:

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

Tools and Libraries Used

• Programming Language: C++

• IDE: Clang

• 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
```

The C++ programs are:

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
```

2.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 function to calculate square
inline int square(int n)
{
return n * n;
}
int main()
{
int num1 = 5, num2 = 9;
cout << "Square of " << num1 << " is: " << square(num1) << endl;
cout << "Square of " << num2 << " is: " << square(num2) << endl;
return 0;
}
```

```
Square of 5 is: 25
Square of 9 is: 81
```

3: 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 calculateTotal(float price, int quantity = 1)
{
return price * quantity;
}
int main() {
float itemPrice = 200.0;
cout << (const char [28])"Total price (7 items): Rs. " emPrice) << endl;
cout << "Total price (7 items): Rs. " << calculateTotal(itemPrice, 7) << endl;
return 0;
}
```

```
Total price (1 item): Rs. 200
Total price (7 items): Rs. 1400
```

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

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

void swapNumbers(int &a, int &b)
{
int temp = a;
a = b;
b = temp;
}
int main()
{
int x = 11, y = 33;
cout << "Before swap: x = " << x << ", y = " << y << endl;
swapNumbers(x, y);
cout << "After swap: x = " << x << ", y = " << y << endl;
return 0;
}
```

```
Before swap: x = 11, y = 33
After swap: x = 33, y = 11
```

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

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

int& getElement(int arr[], int index)
{
return arr[index];
}
int main()
{
int numbers[5] = {10, 20, 30, 40, 50};
cout << "Original value at index 2: " << numbers[2] << endl;

getElement(numbers, 2) = 76;
cout << "Modified value at index 2: " << numbers[2] << endl;

return 0;
}
```

```
Original value at index 2: 30
Modified value at index 2: 76
```

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

```cpp
#include <iostream>
using namespace std;
int main()
{
int arr[5];
int* ptr = arr;
cout << "Enter 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;
}
```

```
Enter 5 integers:
2
3
4
5
6

Squares of the entered integers:
2^2 = 4
3^2 = 9
4^2 = 16
5^2 = 25
6^2 = 36
```

Discussion

This lab provided comprehensive exposure to advanced C++ programming concepts that form the foundation of efficient system-level programming. Function overloading demonstrated compile-time polymorphism, allowing multiple functions with the same name but different parameter signatures, which enhanced code readability and reduced duplication while maintaining type safety. Inline functions revealed the balance between performance optimization and code maintainability by eliminating function call overhead for small, frequently used functions, though we learned that the compiler makes the final in-lining decision based on optimization criteria. Pass by reference emerged as a crucial mechanism for efficient parameter passing, enabling functions to work directly with original variables without copying overhead, particularly beneficial for large data structures, while also allowing functions to modify caller variables safely.

Conclusion

This advanced C++ programming lab successfully demonstrated the sophisticated features that distinguish C++ as a systems programming language capable of both high-level abstraction and low-level control. The exploration of function overloading, inline functions, pass by reference, and pointer programming provided practical experience with concepts that form the foundation of efficient C++ development.