HIMALAYA COLLEGE OF ENGINEERING

Advanced C++ Programming Lab Report

Lab 2: Functions, Structures, and Memory in C++

Prepared By: Nawnit Paudel

Subject: Object Oriented Programming (OOP)

Program: Bachelors of Electronics and Computer Engineering

Institution: Himalaya College of Engineering

Date: June 15, 2025

Objectives:

- To understand and implement function overloading in C++.
- To utilize inline functions for efficiency.
- To work with default arguments in functions.
- To implement call-by-reference.
- To manipulate arrays and pointers.
- To differentiate between structures and unions.
- To understand enumeration types in C++.
- To dynamically allocate and deallocate memory.

Tools and Libraries Used:

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

Theory:

Function Overloading

Function overloading allows multiple functions to have the same name with different parameters. The compiler determines which function to invoke based on the function signature.

Example:

```cpp
int add(int a, int b);
float add(float x, float y);
float add(int a, float b);
```

Inline Functions

Inline functions are used to reduce the overhead of function calls. When a function is marked as inline, the compiler attempts to expand it at the point of call.

Example:

```
inline int square(int n);
```

Default Arguments

Default arguments are specified in function declarations and allow functions to be called with fewer arguments than declared.

Example:

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

Call-by-Reference

Using reference variables in function parameters enables the function to modify the original values passed to it.

Example: i) Call by refrence

```
void swapNumbers(int &a, int &b);
```

i) Return refrence

```
int& getElement(int arr[], int index);
```

Pointers and Arrays

Pointers can access and manipulate array elements directly using pointer arithmetic.

Example:

```
int* ptr = arr;
```

Structures vs Unions

Structure: Allocates separate memory for each member.

Example:

```
struct StdStructure {
  int roll;
  string name;
  float marks;
};
```

Union: Allocates shared memory, and only one member can be used at a time.

Example:

```
union StdUnion {
  int roll;
  string name;
  float marks;
};
```

Enumerations

Enums allow defining a set of named integral constants to make code more readable and maintainable.

Example:

```
enum Day { Sunday, Monday, ... };
```

Dynamic Memory Allocation

Using new and delete operators in C++, memory can be allocated and deallocated at runtime.

Example:

```
int* arr = new int[n];
delete[] arr;
```

Lab Task :

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

1. **Two integers**
2. **Two floats**
3. **One integer and one float.**

```
#include <iostream>

using namespace std;
void add(int x, int y){
int sum = x+y;
cout<<sum;
}
void add(float x, float y){
float sum = x+y;
cout<<sum;
}
void add(int x, float y){
float sum = x+y;
cout<<sum;}
int main()
{
   float a,b;
   cout << "Enter 1st numbers (a) : " << endl;
   cin>>a;
   cout<< "Enter 2nd numbers (b) : "<<endl;
   cin>>b;
   add(a,b);
   return 0;
}
```

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

```
#include <iostream>

using namespace std;
inline void square(float n){
cout<<"Square of the number "<<n <<" is : ";
float sq = n*n ;
cout<<sq<<endl;
}
int main()
{
   float n1,n2;
   cout << "Enter integer number to find its square : " << endl;
   cin>>n1;
   square(n1);
   cout<< "Enter float number to find its square : "<<;
   cin>>n2;
   square(n2);
   return 0;
}
```

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.

```
#include <iostream>

using namespace std;
void totPri(float price, int quantity=1){
float Price = quantity*price ;
cout<<"Total price is : "<<Price;
}
int main()
{
   float p;
   cout << "Enter the price of the product : " << endl;
   cin>>p;
   totPri(p);
   cout<<endl<<"For 7 rpoducts : ";
   totPri (p,7);
   return 0;
}
```

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

```
#include <iostream>

using namespace std;
void swap(float &a,float &b){
float c =  a;
a=b;
b=c;
}
int main()
{
   float x,y;
   cout << "Enter 2 numbers : " << endl;
   cin>>x>>y;
   swap(x,y);
   cout<<"New value of x = "<<x<<"    New value of y = "<<y<<endl;
   return 0;
}
```

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

```
#include <iostream>

using namespace std;
int &func(int arr[], int index){
return arr[index];
}
int main()
{
   int wow[] = {2,32,44,52,62};
    for(int i=0;i<5;i++){
      cout<<wow[i]<<" ";
   }
   cout<<endl<<"Initial value at index 2 : "<<wow[2]<<endl;
   func(wow,2)=42;
   cout<<"Modified value at index 2 : "<<wow[2]<<endl;
   for(int i=0;i<5;i++){
      cout<<wow[i]<<" ";
   }
   return 0;
}
```

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 5 integers:\n";
   for (int i = 0; i < 5; i++) {
      cin >> *(ptr + i);
   }

   cout << "Squares of the numbers:\n";
   for (int i = 0; i < 5; i++) {
      cout << (*(ptr + i)) * (*(ptr + i)) << endl;
   }

   return 0;
}
```

DISCUSSION

In this lab, we explored some essential C++ concepts that help make programs more efficient and organized. We worked with arrays, functions, inline functions, and default arguments, and got hands-on experience with structures and dynamic memory allocation (DMA). A big focus was on how to pass and return values by reference, which helps boost performance by avoiding unnecessary copying. Using structures and dynamic memory allowed us to handle more complex data and memory management in a flexible way. Overall, these tools and techniques helped us write cleaner, more efficient, and modular code.

CONCLUSION

This lab helped deepen our understanding of fundamental C++ programming techniques that are essential for writing efficient and effective code. We explored key concepts like passing and returning values by reference, as well as dynamic memory allocation (DMA), both of which play a vital role in managing memory and building scalable programs. These skills lay a strong foundation for developing more advanced applications and applying object-oriented programming principles in future projects.