HIMALAYA COLLEGE OF ENGINEERING

Advanced C++ Programming Lab Report

Lab 2: Basics concepts of C++ programming

Prepared By: Inisha Mali

Subject: Object Oriented Programming (OOP)

Program: Bachelors of Electronics and Computer Engineering

Institution: Himalaya College of Engineering

Date: June 8, 2025

Objectives:

- To apply default arguments in functions.
- To utilize inline functions for efficiency.
- To perform dynamic memory allocation and deallocation.
- To get familiar with call-by-reference.
- To manipulate arrays and pointers.
- To understand the differentiate between structures and unions.
- To understand enumeration types in C++.

Tools and Libraries Used:

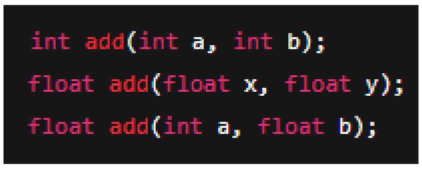
- Programming Language: C++
- IDE: Code::Blocks
- Libraries: include <iostream>, include <string>

Theory:

Function Overloading

Function overloading in C++ allows multiple functions to have the same name, provided they differ in their parameter lists (type or number of arguments). The compiler determines the correct function to invoke based on the arguments used during the function call.

Example:

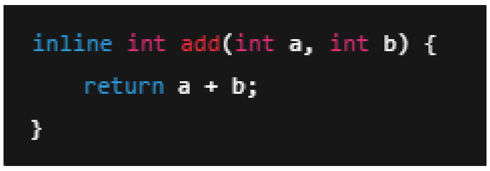


```cpp
int add(int a, int b);
float add(float x, float y);
float add(int a, float b);
```

Inline Functions

Inline functions are expanded in-line at the point of function call, reducing overhead by avoiding stack operations and function jumps. They are ideal for small, frequently used functions.

Example:

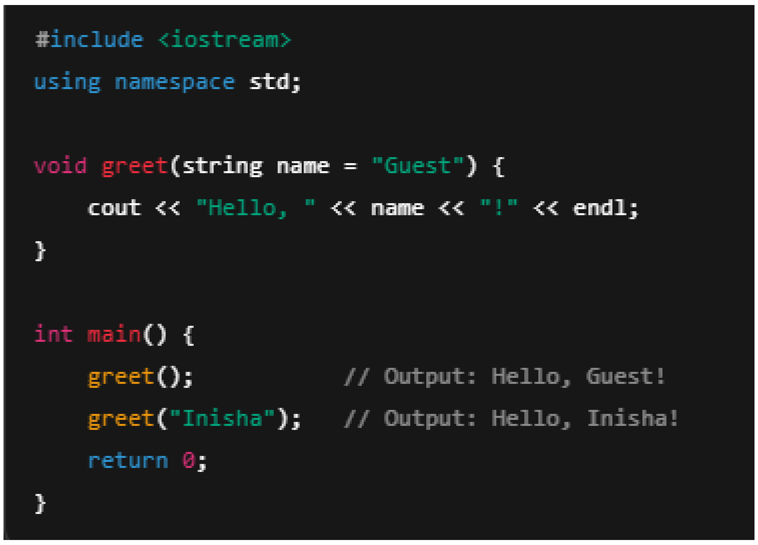


```cpp
inline int add(int a, int b) {
    return a + b;
}
```

Default Arguments

Default arguments allow a function to be called with fewer arguments than declared. The compiler uses default values for any parameters that are not explicitly passed.

Example:

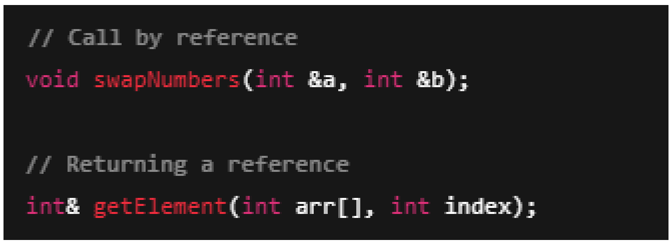


```cpp
#include <iostream>
using namespace std;

void greet(string name = "Guest") {
    cout << "Hello, " << name << "!" << endl;
}

int main() {
    greet();            // Output: Hello, Guest!
    greet("Inisha");    // Output: Hello, Inisha!
    return 0;
}
```

Call-by-Reference

Using reference parameters (&), functions can directly access and modify the actual arguments passed to them. This avoids copying and is useful for efficient memory usage and real-time updates.

Example:

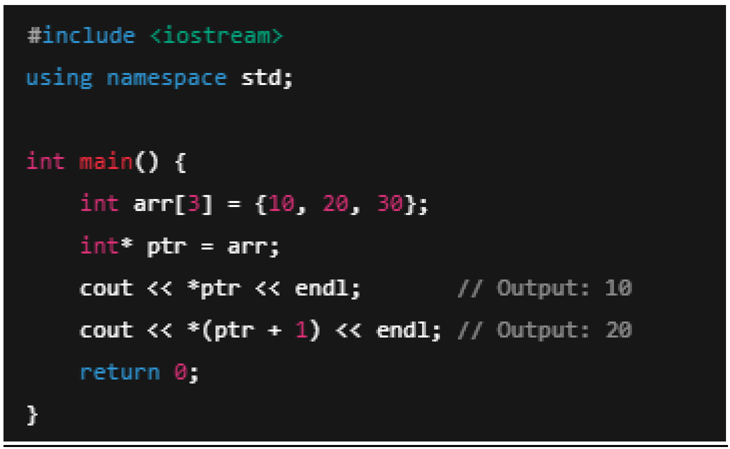


```cpp
// Call by reference
void swapNumbers(int &a, int &b);

// Returning a reference
int& getElement(int arr[], int index);
```

Pointers and Arrays

Pointers provide direct access to memory and can be used to manipulate array elements using pointer arithmetic.

Example:



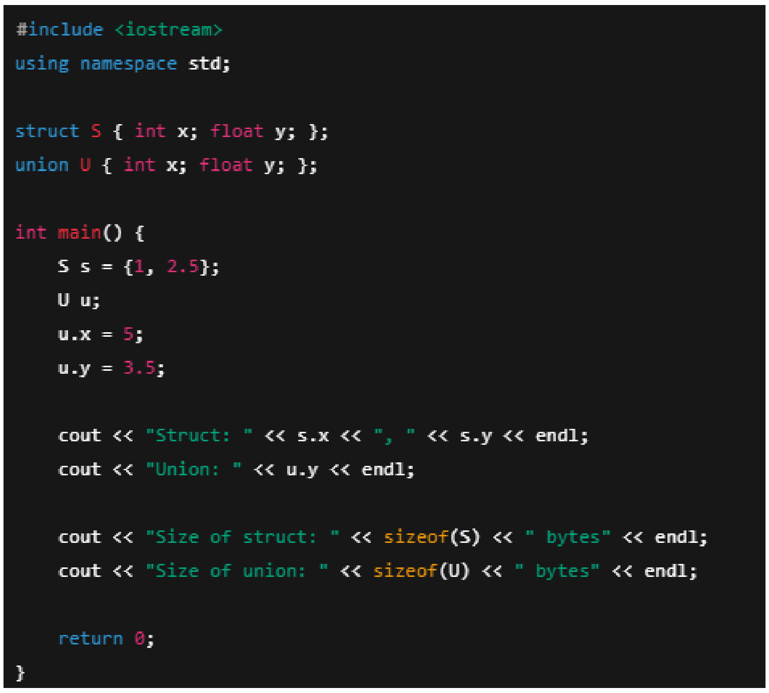
```cpp
#include <iostream>
using namespace std;

int main() {
    int arr[3] = {10, 20, 30};
    int* ptr = arr;
    cout << *ptr << endl;       // Output: 10
    cout << *(ptr + 1) << endl; // Output: 20
    return 0;
}
```

Structures vs Unions

In structure each member has its own memory location.

In union all members share the same memory space. Only one member can hold a value at a time.union

Example:

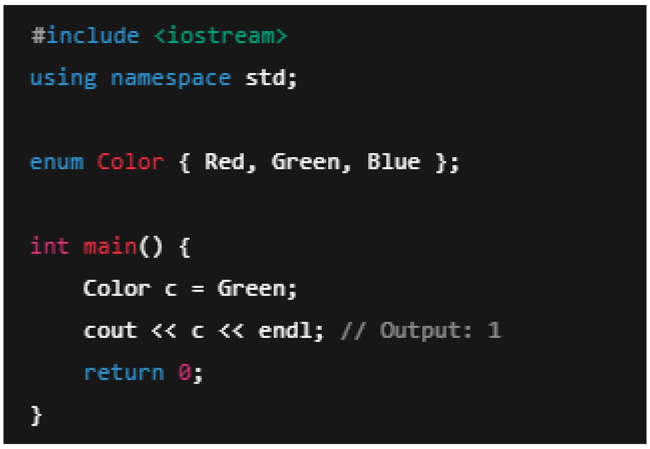


```cpp
#include <iostream>
using namespace std;

struct S { int x; float y; };
union U { int x; float y; };

int main() {
    S s = {1, 2.5};
    U u;
    u.x = 5;
    u.y = 3.5;

    cout << "Struct: " << s.x << ", " << s.y << endl;
    cout << "Union: " << u.y << endl;

    cout << "Size of struct: " << sizeof(S) << " bytes" << endl;
    cout << "Size of union: " << sizeof(U) << " bytes" << endl;

    return 0;
}
```

Enumerations

Enums define a set of named integer constants, making code more readable and organized..

Example:

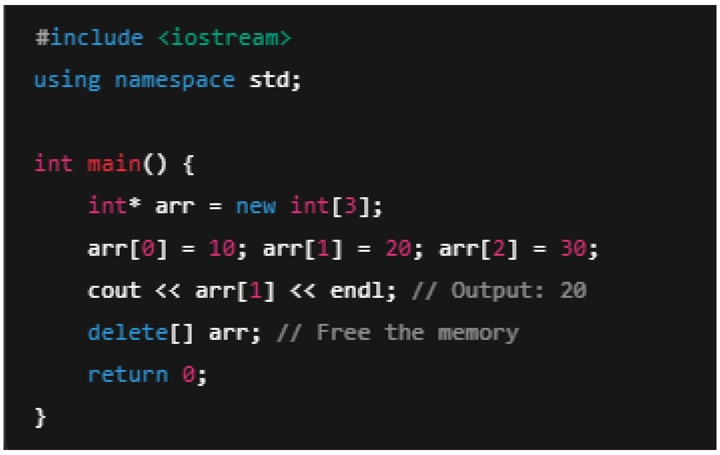


```
#include <iostream>
using namespace std;

enum Color { Red, Green, Blue };

int main() {
    Color c = Green;
    cout << c << endl; // Output: 1
    return 0;
}
```

Dynamic Memory Allocation

C++ supports dynamic memory management using the new and delete operators, allowing memory allocation at runtime.

Example:

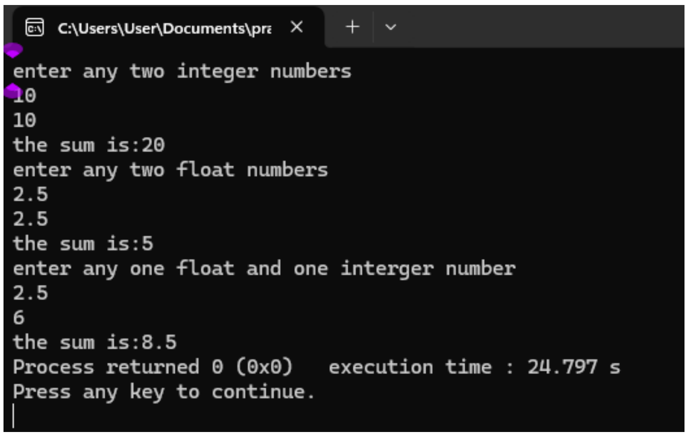


```
#include <iostream>
using namespace std;

int main() {
    int* arr = new int[3];
    arr[0] = 10; arr[1] = 20; arr[2] = 30;
    cout << arr[1] << endl; // Output: 20
    delete[] arr; // Free the memory
    return 0;
}
```

Q1: Write a C++ program to overload a function add() to handle: two integers, two float, one integer and one float.

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

int add(int a,int b)
{
 return a+b;
}
float add(float c, float d)
{
   return c+d;
}
float add(float e, int f)
{
   return e+f;
}

int main()
{
   float num1, num2,result1, result2, result3;

   cout<<"enter any two integer numbers"<<endl;
   cin>>num1>>num2;
   result1=add(num1,num2);
   cout<<"the sum is:"<<result1<<endl;

   cout<<"enter any two float numbers"<<endl;
   cin>>num1>>num2;
   result2=add(num1,num2);
   cout<<"the sum is:"<<result2;

   cout<<"enter any one float and one interger number"<<endl;
   cin>>num1>>num2;
   result3=add(num1,num2);
   cout<<"the sum is:"<<result3;

   return 0;
}
```

Output:

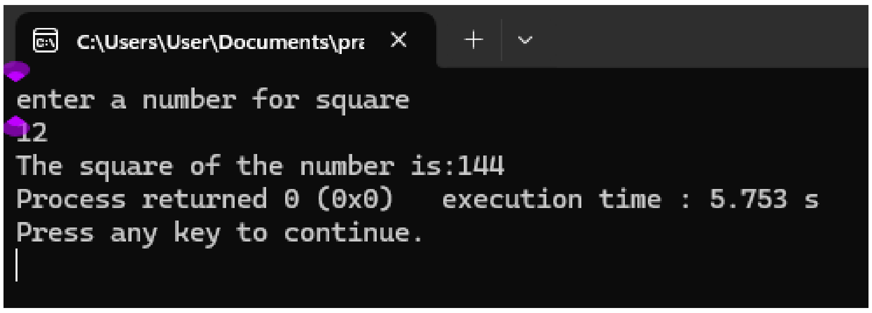


```
enter any two integer numbers
10
10
the sum is:20
enter any two float numbers
2.5
2.5
the sum is:5
enter any one float and one interger number
2.5
6
the sum is:8.5
Process returned 0 (0x0)   execution time : 24.797 s
Press any key to continue.
```

2. 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 float square(float n)
{
 return n*n;
}
int main()
{
   float num1,result;
   cout<<"enter a number for square"<<endl;
   cin>>num1;
   result=square(num1);
   cout<<"The square of the number is:"<<result;
   return 0;
}
```

Output



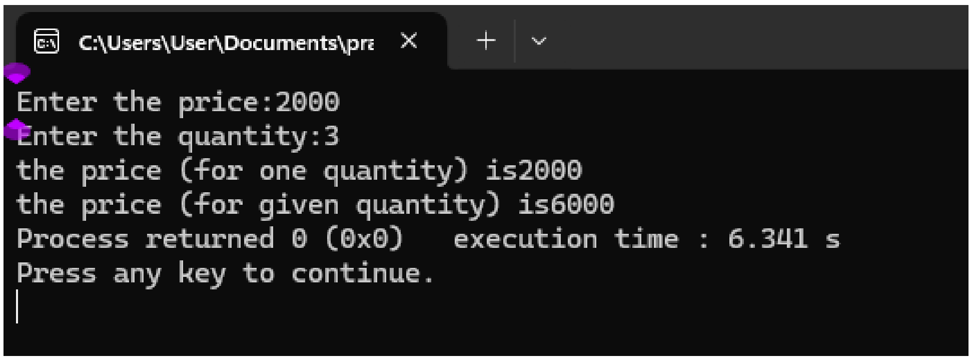
```
enter a number for square
12
The square of the number is:144
Process returned 0 (0x0)   execution time : 5.753 s
Press any key to continue.
```

3.

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.

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

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

int main()
{
   float price, result,result1;
   int quantity;
   cout<<"Enter the price ";
   cin>>price;

   cout<<"Enter the quantity";
   cin>>quantity;

   result1=calculate(price);
   cout<<"the price is(for one quantity)"<<result1<<endl;
   result=calculate(price,quantity);
   cout<<"the price is(for given quantity)"<<result;
   return 0;
}
```

Output

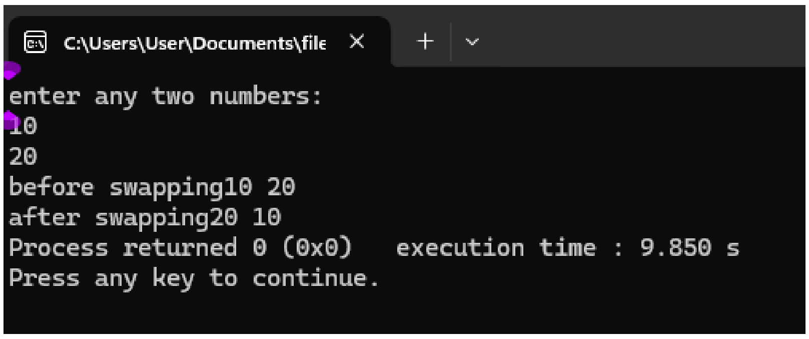


```
Enter the price:2000
Enter the quantity:3
the price (for one quantity) is2000
the price (for given quantity) is6000
Process returned 0 (0x0)   execution time : 6.341 s
Press any key to continue.
```

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

```
#include<iostream>
using namespace std;
void swap (int &a, int &b)
{
   int temp;
   temp=a;
   a=b;
   b=temp;
}
int main()
{
   int num1,num2;
   cout<<"enter any two numbers:"<<endl;
   cin>>num1>>num2;
   cout<<"before swapping"<<num1<<" "<<num2<<endl;
   swap(num1,num2);
   cout<<"after swapping"<<num1<<" "<<num2;
   return 0;
}
```

Output

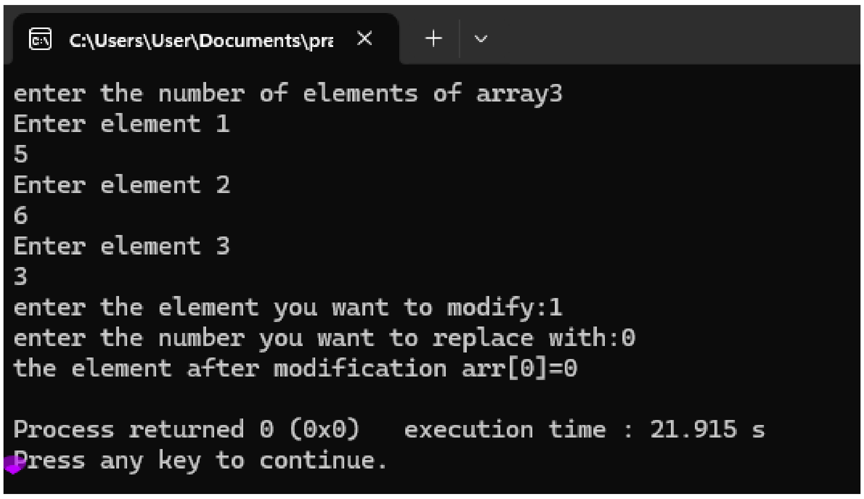


```
enter any two numbers:
10
20
before swapping10 20
after swapping20 10
Process returned 0 (0x0)   execution time : 9.850 s
Press any key to continue.
```

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

```
#include<iostream>
using namespace std;
int& modifyElement(int arr[],int index)
{
   return arr[index];
}
int main()
{
   int arr[100],n,num,result;
   cout<<"enter the number of elements of array";
   cin>>n;
   for(int i=0;i<n;i++)
   {
     cout<<"Enter element "<<i+1<<endl;
     cin>>arr[i];

   }
   int index1;
   cout<<"enter the element you want to modify";
   cin>>index1;
   cout<<"enter the number you want to replace with";
   cin>>num;
   result=modifyElement(arr,index1)=num;
   cout<<"the element after modification"<<" "<<"arr["<<index1<<"]"<<result<<endl;
   return 0;
   }
```

Output

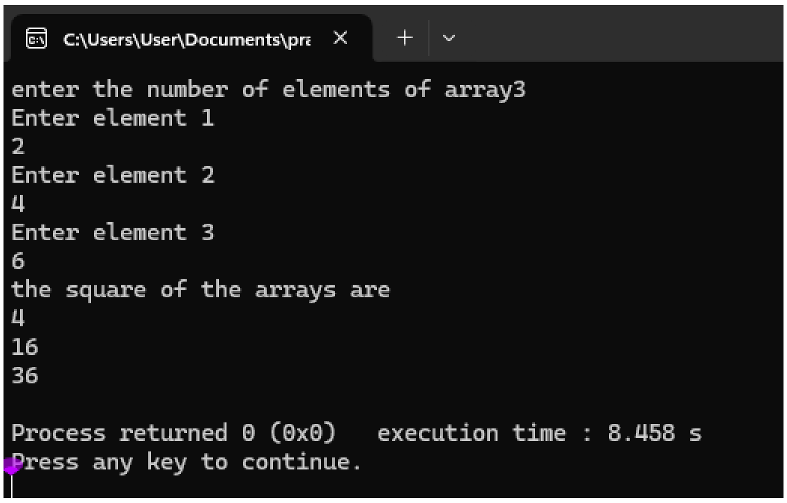


```
enter the number of elements of array3
Enter element 1
5
Enter element 2
6
Enter element 3
3
enter the element you want to modify:1
enter the number you want to replace with:0
the element after modification arr[0]=0

Process returned 0 (0x0)   execution time : 21.915 s
Press any key to continue.
```

6. 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[40],n;
   int *ptr=arr;
   cout<<"enter the number of elements of array";
   cin>>n;
   for(int i=0;i<n;i++)
   {
       cout<<"Enter element "<<i+1<<endl;
       cin>>*(ptr+i);

   }
   //for square
   cout<<"the square of the arrays are"<<endl;
   for(int i=0;i<n;i++)
   {
       cout<<(*(ptr+i))*(*(ptr+i))<<endl;
   }
   return 0;
}
```

Output



```
enter the number of elements of array3
Enter element 1
2
Enter element 2
4
Enter element 3
6
the square of the arrays are
4
16
36

Process returned 0 (0x0)   execution time : 8.458 s
Press any key to continue.
```

Discussion:
This lab introduced essential C++ features that improve code efficiency and flexibility. Function overloading allowed using the same function name for different argument types. Inline functions reduced overhead by expanding code at the call site. Default arguments simplified function calls by using preset values. Call-by-reference enabled direct modification of variables, improving performance.Pointers and arrays demonstrated memory-level data access.

Conclusion:
The lab successfully deepened our understanding of advanced C++ features by exploring how they enhance code efficiency, readability, and flexibility. Concepts such as function overloading, inline functions, and default arguments showed how to write more maintainable and cleaner code. Working with references, pointers, and dynamic memory management highlighted the importance of memory control and performance optimization in C++. The practical comparison between structures and unions demonstrated different ways data can be stored and accessed.