

LAB ASSIGNMENTS

1. Write a function template swapValues() that swaps two variables of any data type. Demonstrate its use with int, float, and char.

Source Code:

```
#include <iostream>

using namespace std;

template <typename T>

void swapValues(T& a, T& b) {

    T temp = a;

    a = b;

    b = temp;

}
```

```
int main() {

    int x = 5, y = 10;

    float f1 = 1.5f, f2 = 2.5f;

    char c1 = 'A', c2 = 'B';

    swapValues(x, y);

    cout << "Swapped int: x = " << x << ", y = " << y << endl;

    swapValues(f1, f2);

    cout << "Swapped float: f1 = " << f1 << ", f2 = " << f2 << endl;

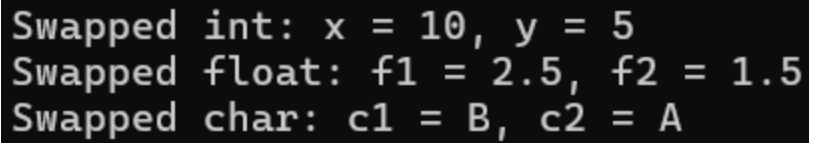
    swapValues(c1, c2);

    cout << "Swapped char: c1 = " << c1 << ", c2 = " << c2 << endl;

    return 0;

}
```

Output:



```
Swapped int: x = 10, y = 5
Swapped float: f1 = 2.5, f2 = 1.5
Swapped char: c1 = B, c2 = A
```

2. Write a program to overload a function template `maxValue()` to find the maximum of two values (for same type) and three values (for same type). Call it using `int`, `double`, and `char`.

Source Code

```
#include <iostream>

using namespace std;

template <typename T>
T maxValue(T a, T b) {
    return (a > b) ? a : b;
}

template <typename T>
T maxValue(T a, T b, T c) {
    return maxValue(maxValue(a, b), c);
}

int main() {
    cout<<"Enter 2 integers: ";
    int a,b;
    cin>>a>>b;
    cout<<"Enter the third integer: ";
    int c;
    cin>>c;
    cout<<"Enter 2 characters: ";
    char ch1,ch2;
    cin>>ch1>>ch2;
```

```
cout<<"Enter the third character: ";
```

```
char ch3;
```

```
cin>>ch3;
```

```
cout<<"Enter 2 doubles: ";
```

```
double d1,d2;
```

```
cin>>d1>>d2;
```

```
cout<<"Enter the third double: ";
```

```
double d3;
```

```
cin>>d3;
```

```
cout<<"Enter 2 float numbers: ";
```

```
float f1,f2;
```

```
cin>>f1>>f2;
```

```
cout<<"Enter the third float: ";
```

```
float f3;
```

```
cin>>f3;
```

```
cout << "Max of 2 integers: " << maxValue(a, b) << endl;
```

```
cout << "Max of 3 integers: " << maxValue(a, b, c) << endl;
```

```
cout << "Max of 2 doubles: " << maxValue(d1, d2) << endl;
```

```
cout << "Max of 3 doubles: " << maxValue(d1, d2, d3) << endl;
```

```
cout << "Max of 2 chars: " << maxValue(ch1, ch2) << endl;
```

```
cout << "Max of 3 chars: " << maxValue(ch1, ch2, ch3) << endl;
```

```
return 0;
```

```
}
```

Output:

```
Enter 2 integers: 4 5
Enter the third integer: 7
Enter 2 characters: s n
Enter the third character: p
Enter 2 doubles: 1.0 0.2
Enter the third double: 0.05
Enter 2 float numbers: 1.0 0.5
Enter the third float: 0.03
Max of 2 integers: 5
Max of 3 integers: 7
Max of 2 doubles: 1
Max of 3 doubles: 1
Max of 2 chars: s
Max of 3 chars: s
```

3. Create a class template `Calculator<T>` that performs addition, subtraction, multiplication, and division of two data members of type `T`. Instantiate it with `int` and `float`.

Source Code:

```
#include <iostream>

using namespace std;

template <typename T>

class Calculator {

    T a, b;

public:

    Calculator(T x, T y) : a(x), b(y) {}

    T add() {

        return a + b;

    }

    T subtract() {

        return a - b;

    }

    T multiply() {

        return a * b;

    }

    T divide() {

        return a / b;

    }

};

int main() {
```

```

cout<<"Enter 2 integers: ";
int a,b;
cin>>a>>b;
Calculator<int> calcInt(a, b);
cout<<"Addition: "<<calcInt.add()<<endl;
cout<<"Subtraction: "<<calcInt.subtract()<<endl;
cout<<"Multiplication: "<<calcInt.multiply()<<endl;
cout<<"Division: "<<calcInt.divide()<<endl;
cout<<"Enter 2 float numbers: ";
float c,d;
cin>>c>>d;
Calculator<float> calcFloat(c, d);
cout<<"Addition: "<<calcFloat.add()<<endl;
cout<<"Subtraction: "<<calcFloat.subtract()<<endl;
cout<<"Multiplication: "<<calcFloat.multiply()<<endl;
cout<<"Division: "<<calcFloat.divide()<<endl;
return 0;
}

```

Output:

```

Enter 2 integers: 2 3
Addition: 5
Subtraction: -1
Multiplication: 6
Division: 0
Enter 2 float numbers: 2.3 4.2
Addition: 6.5
Subtraction: -1.9
Multiplication: 9.66
Division: 0.547619

```

4. Define a class template Base<T> with a protected data member and a member function to display it. Derive a class Derived<T> from it, add another data member, and display both data members. Use string and int types to test.

Source Code:

```
#include <iostream>

#include <string>

using namespace std;

template <typename T>

class Base {

protected:

    T data;

public:

    Base(T d) : data(d) {}

    void display() {

        cout << "Base data: " << data << endl;

    }

};

template <typename T>

class Derived : public Base<T> {

    T newdata;

public:

    Derived(T d, T e) : Base<T>(d), newdata(e) {}

    void display() {

        cout << "Base data: " << this->data << ", New Data: " << newdata << endl;

    }

}
```

```
};
```

```
int main() {
```

```
    Derived<int> d1(10, 20);
```

```
    d1.display();
```

```
    Derived<string> d2("Hello", "World");
```

```
    d2.display();
```

```
    return 0;
```

```
}
```

Output:

```
Base data: 10, New Data: 20
Base data: Hello, New Data: World
```

5. Write a program to demonstrate the use of Container, Iterator, and Algorithm components in a single program using a vector<int> and performing sorting using sort().

Source Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

vector<int> numbers = {23, 10, 45, 15, 5};

cout << "Original Vector Elements:\n";

vector<int>::iterator it;

for (it = numbers.begin(); it != numbers.end(); ++it) {

cout << *it << " ";

}

sort(numbers.begin(), numbers.end());

cout << "\nSorted Vector Elements (Ascending):\n";

for (it = numbers.begin(); it != numbers.end(); ++it) {

cout << *it << " ";

}

return 0;

}
```

Output:

```
Original Vector Elements:
23 10 45 15 5
Sorted Vector Elements (Ascending):
5 10 15 23 45
```


6. Write a program to use the STL algorithm functions: sort(), reverse(), find(), and count() on a vector<int>.

Source Code:

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {
    vector<int> numbers = {23, 10, 45, 15, 5};
    cout << "Original Vector Elements:\n";
    vector<int>::iterator it;
    for (it = numbers.begin(); it != numbers.end(); ++it) {
        cout << *it << " ";
    }

    sort(numbers.begin(), numbers.end());
    cout << "\n\nSorted Vector Elements (Ascending):\n";
    for (it = numbers.begin(); it != numbers.end(); ++it) {
        cout << *it << " ";
    }

    return 0;
}
```

Output:

```
Original Vector Elements:
23 10 45 15 5
Sorted Vector (Ascending):
5 10 15 23 45
Vector after Reversing:
45 23 15 10 5
Element 8 not found.
Number of times 2 appears: 0
```

DISCUSSION

In this lab, we were able to understand the core concept of templates, which are a powerful feature in C++ that enable generic programming. We learnt how function templates and class templates allow us to write flexible and reusable code for different data types without duplication. Additionally, we explored the concept of template specialization and overloading templates to handle specific cases. Although understanding template syntax and its implementation was challenging at first, with the help of examples and online resources, I was able to understand and implement it.

CONCLUSION

From this lab, we can conclude that the use of templates enhances code maintainability, reduces the need for rewriting code, and facilitates polymorphism which is a core concept in OOP.