# LAB QUESTIONS

Q1. Write a function template swapValues() that swaps two variables of any data type. Demonstrate its use with int, float, and char.

#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, y;

cout << "Enter two integers: ";

cin >> x >> y;

swapValues(x, y);

cout << "After swapping (int): x = " << x << ", y = " << y << endl << endl;

float f1, f2;

cout << "Enter two float values: ";

cin >> f1 >> f2;

swapValues(f1, f2);

cout << "After swapping (float): f1 = " << f1 << ", f2 = " << f2 << endl << endl;

char c1, c2;

cout << "Enter two characters: ";

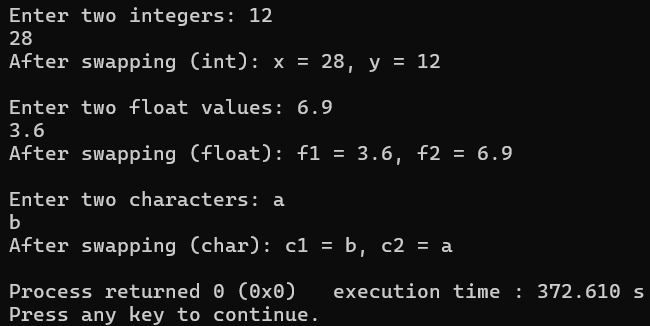
cin >> c1 >> c2;

swapValues(c1, c2);

cout << "After swapping (char): c1 = " << c1 << ", c2 = " << c2 << endl;

return 0;

}



QN2. 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.

#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() {

int i1 = 10, i2 = 25, i3 = 15;

cout << "Max of 10 and 25 (int): " << maxValue(i1, i2) << endl;

cout << "Max of 10, 25 and 15 (int): " << maxValue(i1, i2, i3) << endl << endl;

double d1 = 5.5, d2 = 3.3, d3 = 7.7;

cout << "Max of 5.5 and 3.3 (double): " << maxValue(d1, d2) << endl;

cout << "Max of 5.5, 3.3 and 7.7 (double): " << maxValue(d1, d2, d3) << endl << endl;

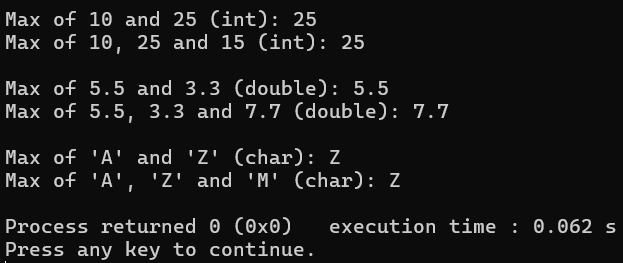
char c1 = 'A', c2 = 'Z', c3 = 'M';

cout << "Max of 'A' and 'Z' (char): " << maxValue(c1, c2) << endl;

cout << "Max of 'A', 'Z' and 'M' (char): " << maxValue(c1, c2, c3) << endl;

return 0;

}



Q3. Create a class template Calculator that performs addition, subtraction, multiplication, and division of two data members of type T. Instantiate it with int and float

#include <iostream>

using namespace std;

template <typename T>

class Calculator {

private:

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() {

if (b != 0)

return a / b;

else {

cout << "Error: Division by zero!" << endl;

return 0;

}}};

int main() {

Calculator<int> intCalc(20, 5);

cout << "Integer operations:" << endl;

cout << "Addition: " << intCalc.add() << endl;

cout << "Subtraction: " << intCalc.subtract() << endl;

cout << "Multiplication: " << intCalc.multiply() << endl;

cout << "Division: " << intCalc.divide() << endl << endl;

Calculator<float> floatCalc(5.5f, 2.2f);

cout << "Float operations:" << endl;

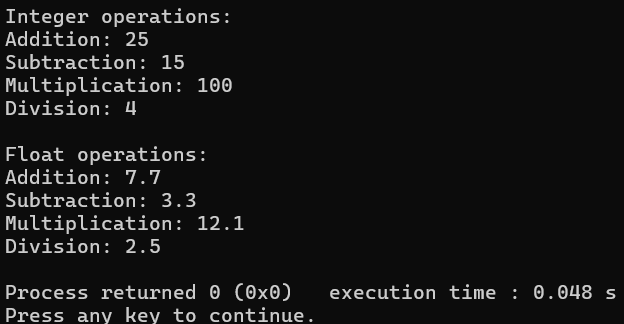
cout << "Addition: " << floatCalc.add() << endl;

cout << "Subtraction: " << floatCalc.subtract() << endl;

cout << "Multiplication: " << floatCalc.multiply() << endl;

cout << "Division: " << floatCalc.divide() << endl;

return 0;}



Q4. Define a class template Base with a protected data member and a member function to display it. Derive a class Derived from it, add another data member, and display both data members. Use string and int types to test.

#include <iostream>

#include <string>

using namespace std;

template <typename T>

class Base {

protected:

T baseData;

public:

Base(T val) {

baseData = val;

}

void displayBase() {

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

}};

template <typename T>

class Derived : public Base<T> {

private:

T derivedData;

public:

Derived(T baseVal, T derivedVal) : Base<T>(baseVal) {

derivedData = derivedVal;

}

void displayAll() {

cout << "Base data: " << this->baseData << endl;

cout << "Derived data: " << derivedData << endl;

}};

int main() {

cout << "--- Integer Example ---" << endl;

Derived<int> intObj(100, 200);

intObj.displayAll();

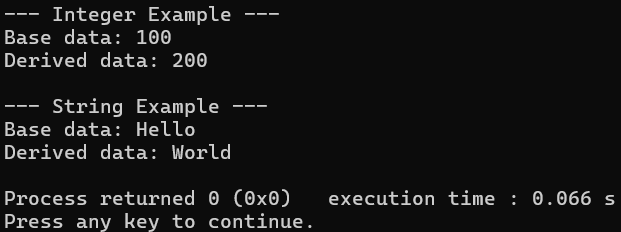
cout << "\n--- String Example ---" << endl;

Derived<string> strObj("Hello", "World");

strObj.displayAll();

return 0;

}



Q5. Write a program to demonstrate the use of Container, Iterator, and Algorithm components in a single program using a vector and performing sorting using sort().

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

vector<int> numbers = {33, 25, 45, 15, 10};

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";

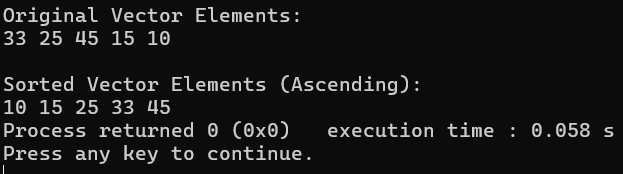
for (it = numbers.begin(); it != numbers.end(); ++it) {

cout << \*it << " ";

}

return 0;

}



Q6. Write a program to use the STL algorithm functions: sort(), reverse(), find(), and count() on a vector.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

vector<int> nums = {5, 3, 8, 3, 9, 1, 3, 7};

cout << "Original vector: ";

for (int n : nums) cout << n << " ";

cout << endl;

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

cout << "Sorted vector: ";

for (int n : nums) cout << n << " ";

cout << endl;

reverse(nums.begin(), nums.end());

cout << "Reversed vector: ";

for (int n : nums) cout << n << " ";

cout << endl;

int target = 3;

auto it = find(nums.begin(), nums.end(), target);

if (it != nums.end())

cout << "Element " << target << " found at index: " << distance(nums.begin(), it) << endl;

else

cout << "Element " << target << " not found." << endl;

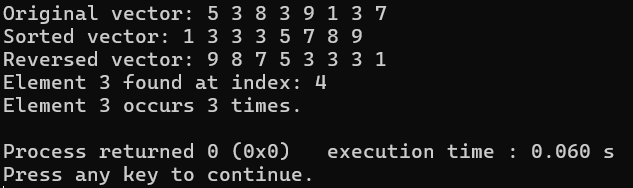
int countVal = 3;

int countResult = count(nums.begin(), nums.end(), countVal);

cout << "Element " << countVal << " occurs " << countResult << " times." << endl;

return 0;

}



# DISCUSSIONS

# In this lab session, we discussed about templates in C++. Templates allow us to write generic and reusable code that works with any data type. We learned how to define both function templates and class templates, enabling us to perform operations such as swapping values, building calculators, and creating data structures without duplicating code for each data type. We also explored how templates support compile-time polymorphism by allowing the same function or class definition to work with different types like int, float, and string. This promotes code efficiency, type safety, and flexibility.

# CONCLUSIONS

In conclusion, this lab session on templates helped us understand how templates enhance code reusability, flexibility, and maintainability by allowing generic programming. By using function and class templates, we were able to create type-independent code, reducing duplication and improving efficiency. This feature supports the principles of object-oriented programming such as abstraction and reusability, by enabling the creation of clean and scalable solutions available to different data types.