OBJECTIVE

- To understand the concept and syntax of class templates in C++.
- To learn how to create generic classes that work with different data types.
- To implement a Calculator<T> class that performs basic arithmetic operations: addition, subtraction, multiplication, and division.

BACKGROUND THEORY

Templates are a powerful feature in C++ that allow the creation of generic and reusable code components. They enable programmers to write functions and classes that can operate with any data type without rewriting the same code for each type. Templates promote code reusability, maintainability, and type safety.

Types of Templates

1. **Function Templates**:
   Function templates allow writing a single function to work with different data types. The compiler generates the appropriate function based on the type of arguments used in the call.

Example: A generic function to find the maximum of two values.

2. **Class Templates**:
   Class templates define a blueprint for a class that can handle any data type. The compiler generates the specific class definition based on the data type provided.

Example: A generic Pair class to store two values of any data type.

Syntax:

```
template <typename T>

T functionName(T param1, T param2) {

   // function body

}
```

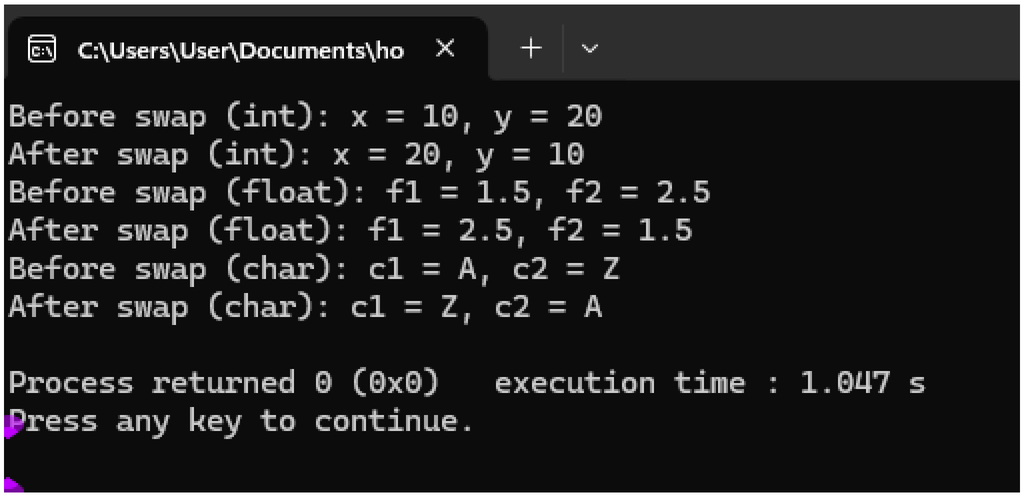
For classes:

```
template <typename T>

class ClassName {

   T memberVariable;

   // class body

};
```

1. **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 = 10, y = 20;
   cout << "Before swap (int): x = " << x << ", y = " << y << endl;
   SwapValues(x, y);
   cout << "After swap (int): x = " << x << ", y = " << y << endl;

   float f1 = 1.5f, f2 = 2.5f;
   cout << "Before swap (float): f1 = " << f1 << ", f2 = " << f2 << endl;
   SwapValues(f1, f2);
   cout << "After swap (float): f1 = " << f1 << ", f2 = " << f2 << endl;

   char c1 = 'A', c2 = 'Z';
   cout << "Before swap (char): c1 = " << c1 << ", c2 = " << c2 << endl;
   SwapValues(c1, c2);
   cout << "After swap (char): c1 = " << c1 << ", c2 = " << c2 << endl;
   return 0;}
```

Output:



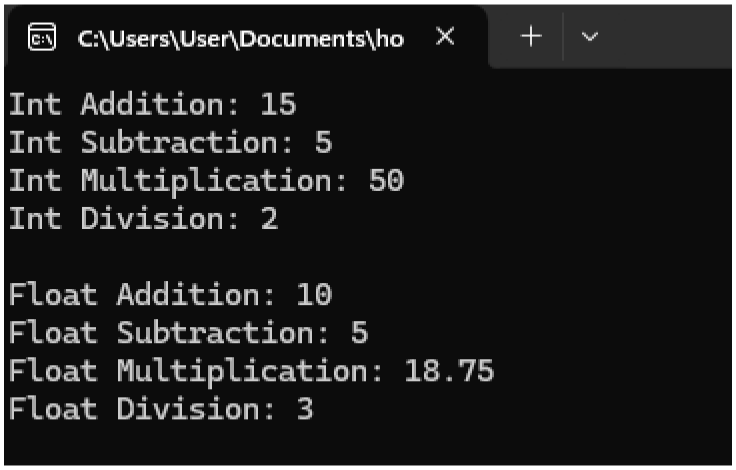
```
Before swap (int): x = 10, y = 20
After swap (int): x = 20, y = 10
Before swap (float): f1 = 1.5, f2 = 2.5
After swap (float): f1 = 2.5, f2 = 1.5
Before swap (char): c1 = A, c2 = Z
After swap (char): c1 = Z, c2 = A

Process returned 0 (0x0)   execution time : 1.047 s
Press any key to continue.
```

2. **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.**

```
#include <iostream>

using namespace std;

template <typename T>

class Calculator {

   T num1, num2;

public:

   Calculator(T a, T b) : num1(a), num2(b) {}

   T add() {

      return num1 + num2;

   }

   T subtract() {

      return num1 - num2;

   }

   T multiply() {

      return num1 * num2;

   }
```

```
    T divide() {
        if (num2 != 0)
            return num1 / num2;
        else {
            cout << "Error: Division by zero!" << endl;
            return T{}; // Return default value of T
        }
    }
};
int main() {
    // Instantiate Calculator with int
    Calculator<int> calcInt(10, 5);
    cout << "Int Addition: " << calcInt.add() << endl;
    cout << "Int Subtraction: " << calcInt.subtract() << endl;
    cout << "Int Multiplication: " << calcInt.multiply() << endl;
    cout << "Int Division: " << calcInt.divide() << endl;
    cout << endl;
    Calculator<float> calcFloat(7.5f, 2.5f);
    cout << "Float Addition: " << calcFloat.add() << endl;
    cout << "Float Subtraction: " << calcFloat.subtract() << endl;
    cout << "Float Multiplication: " << calcFloat.multiply() << endl;
    cout << "Float Division: " << calcFloat.divide() << endl;
    return 0;
}
```

Output



```
Int Addition: 15
Int Subtraction: 5
Int Multiplication: 50
Int Division: 2

Float Addition: 10
Float Subtraction: 5
Float Multiplication: 18.75
Float Division: 3
```

Discussion

The implementation of the class template Calculator<T> in this lab demonstrates the versatility and power of templates in C++. By designing a generic class that performs arithmetic operations on two data members, we were able to apply the same logic to multiple data types such as int and float without duplicating code.

Conclusion

This lab successfully illustrated how templates in C++ can be used to create flexible, reusable, and type-safe code. The Calculator<T> class template worked effectively with different data types, proving its generic functionality. Using templates not only simplifies development but also reduces redundancy and enhances maintainability.