Experiment 1

Student Name: Harsh Kumar **UID: 22BCS15754**
Branch:BE-CSE **Section/Group: FL_IOT_603'B**
Semester: 5th **Date of Performance:18/07/24**
Subject Name:Design and Analysis Algorithms
Subject Code:22CSH-311

1. **Aim:** Analyse if the stack is empty or full, and if elements are present, return the top element in the stack using templates. Also, perform push and pop operations on the stack.
2. **Objective:** Demonstrate the use of the predefined stack container from the C++ Standard Library for basic stack operations. • Perform and verify stack operations such as pushing, popping, checking if the stack is empty, and accessing the top element. • Provide a clear example of stack manipulation, including checking the stack's status before and after operations
3. **Implementation/Code:**

```
#include <iostream>

#define MAX_SIZE 100 // Maximum size of the stack

using namespace std;

template <typename T>

class CustomStack {

private:

   T elements[MAX_SIZE];

   int topIndex;
```

```
public:
    CustomStack() : topIndex(-1) {}
    void push(T value) {
        if (topIndex >= MAX_SIZE - 1) {
            cout << "Stack overflow" << endl;
        } else {
            elements[++topIndex] = value;
            cout << value << " pushed into stack" << endl;
        }
    }
    void pop() {
        if (topIndex < 0) {
            cout << "Stack underflow" << endl;
        } else {
            T value = elements[topIndex--];
            cout << value << " popped from stack" << endl;
        }
    }

    bool isEmpty() {
        return topIndex == -1;
    }
```

```
    bool isFull() {
        return topIndex >= MAX_SIZE - 1;
    }
    T peek() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return T();
        } else {
            return elements[topIndex];  }  }
    void display() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
        } else {
            cout << "Stack elements are: ";
            for (int i = 0; i <= topIndex; ++i) {
                cout << elements[i] << " ";
            }
            cout << endl;
        }}};
int main() {
    CustomStack<int> intStack;
```

```
    CustomStack<double> doubleStack;
    intStack.push(5);
    intStack.push(15);
    intStack.push(25);
    intStack.display();
    cout << "Top element of int stack is: " << intStack.peek() << endl;
    intStack.pop();
    intStack.pop();
    intStack.display();
    cout << "Top element of int stack is: " << intStack.peek() << endl;
    doubleStack.push(5.5);
    doubleStack.push(15.5);
    doubleStack.push(25.5);
    doubleStack.display();
    cout << "Top element of double stack is: " << doubleStack.peek() << endl;
    doubleStack.pop();
    doubleStack.pop();
    doubleStack.display();
    cout << "Top element of double stack is: " << doubleStack.peek() << endl;

    return 0;
}
```

4. Output

```
5 pushed into stack
15 pushed into stack
25 pushed into stack
Stack elements are: 5 15 25
Top element of int stack is: 25
25 popped from stack
15 popped from stack
Stack elements are: 5
Top element of int stack is: 5
5.5 pushed into stack
15.5 pushed into stack
25.5 pushed into stack
Stack elements are: 5.5 15.5 25.5
Top element of double stack is: 25.5
25.5 popped from stack
15.5 popped from stack
Stack elements are: 5.5
Top element of double stack is: 5.5


=== Code Execution Successful ===
```

Time Complexity

Push operation: O(1) - Constant time to push an element onto the stack.
Pop operation: O(1) - Constant time to pop an element from the stack.
isEmpty(): O(1) - Constant time to check if the stack is empty.
isFull(): O(1) - Constant time to check if the stack is full.
peek(): O(1) - Constant time to get the top element of the stack.
display(): O(n) - Linear time to display all elements of the stack, where n is the number of elements in the stack.