United International University

Department of Computer Science and Engineering

Course Code: CSI 217 | Course name: Data Structure and Algorithms - I Laboratory

Total Marks: 75 (will be converted to 25)

1: Implementing Two Stacks Using a Single Array (Mark 20)

You are tasked with **implementing two stacks using a single array.** Design a class TwoStacks that allows you to perform operations on these stacks efficiently. The class should have the following methods:

```
TwoStacks(int maxSize);   // Constructor to initialize the array and size
void pushStack1(int x);   // Push element x onto the first stack (Mark 4)
void pushStack2(int x);   // Push element x onto the second stack (Mark 4)
int popStack1();          // Pop and return an element from the first stack (Mark 4)
int popStack2();          // Pop and return an element from the second stack (Mark 4)
bool isEmptyStack1();     // Check if the first stack is empty (Mark 2)
bool isEmptyStack2();     // Check if the second stack is empty (Mark 2)
```

2. Testing the Stack Implementation: (Mark 5)

```
    TwoStacks myStacks(6);

    // Pushing elements onto Stack 1
    myStacks.pushStack1(5);
    myStacks.pushStack1(10);

    // Pushing elements onto Stack 2
    myStacks.pushStack2(20);
    myStacks.pushStack2(25);

    // Popping elements from both stacks
    int popped1 = myStacks.popStack1(); // Should be 10
    int popped2 = myStacks.popStack2(); // Should be 25

    // Pushing more elements onto Stack 2
    myStacks.pushStack2(30);

    // Popping more elements from both stacks
    int popped3 = myStacks.popStack1(); // Should be 5
    int popped4 = myStacks.popStack2(); // Should be 30

    // Checking if stacks are empty
    bool isEmpty1 = myStacks.isEmptyStack1(); // Should be true
    bool isEmpty2 = myStacks.isEmptyStack2(); // Should be false
```

Helper Code :

```
class TwoStacks {
private:
    int* arr;
    int size;
    int top1;
    int top2;

public:
    TwoStacks(int maxSize) {
        size = maxSize;
        arr = new int[size];
        top1 = ;
        top2 = ;
    }

    void pushStack1(int x) {
    }

    void pushStack2(int x) {
    }

    int popStack1() {
        return -1; // Stack 1 is empty
    }

    int popStack2() {
        return -1; // Stack 2 is empty
    }

    bool isEmptyStack1() {   return;}

    bool isEmptyStack2() { return ; }};
```

3 .You are given **a string** representing a mathematical expression. The expression contains **digits(0-9), parentheses ('(', ')', '[', ']', '{', '}')**, and valid **mathematical operators (+, -, *, or /).**

You need to determine whether the **expression is valid** according to the following conditions:

- The expression should contain valid parentheses, meaning that all opening parentheses ('(', '[', '{') should be properly closed in the correct order (')', ']', '}'). The parentheses should also match in quantity, meaning for every opening parenthesis, there should be a corresponding closing parenthesis. **(mark 9)**

- The expression should not contain empty sets of parentheses, meaning there should be valid expressions inside at least one set of parentheses. **(mark 4)**

- The digits in the expression should be from palindrome, meaning that it reads the same backward as forward. **(mark 8)**

- The expression should contain at least one valid mathematical operator (+, -, *, or /) between the operands. **(mark 4)**

- You write a function isValidExpression that takes a string as input and returns true if the expression is valid based on the conditions mentioned above; otherwise, return false.

Examples:

- Valid : ((4++1)*4)
- Valid : ([(2*2)])
- Valid : ({(9)-{3+9}})
- Invalid : (((2+3)*4)) cause digits not palindrome
- Invalid : ([2++]*2) cause missing operand
- Invalid : [] cause no expression inside
- Invalid : (2++) cause no end operand
- Invalid : (+) cause no digit

4.**Basic Queue Implementation: (Mark 8)**

Implement a basic queue data structure with the following functions:

- void enqueue(int x): Add element x to the back of the queue.
- int dequeue(): Remove and retrieve the element from the front of the queue.
- bool isEmpty(): Determine if the queue is empty, returning true if it is, and false otherwise.

5 .Stack Implementation Using Queues: (Mark 12)

Using the basic queue implementation from the previous step, implement a stack data structure with the following functions:

- void push(int x): Add element x to the top of the stack.
- int pop(): Remove and retrieve the element from the top of the stack.
- bool isEmpty(): Determine if the stack is empty, returning true if it is, and false otherwise.

You can use as many queue you needed

6. Testing the Stack Implementation: (Mark 5)

For above Stack demonstrate the following operations:

Push(4)
Push(2)
Push(5)
Pop()
isEmpty()
Pop()
Pop()
isEmpty()

Helper Code :

```
#include <iostream>

class Queue {
private:
    static const int MAX_SIZE = 100; // Maximum size of the queue
    int arr[MAX_SIZE];              // Array to store queue elements
    int frontIndex;                 // Index of the front element
    int rearIndex;                  // Index of the rear element

public:
    Queue() {
        frontIndex = -1; // Initialize front index
        rearIndex = -1;  // Initialize rear index
    }

    void enqueue(int x) {

    }

    int dequeue() {

        Return ;
    }

    bool isEmpty() { return frontIndex == -1 && rearIndex == -1; }
```

By completing above structure you can declare like : Queue Q1,Q2 and used them in implementing stack