DSA STACKS ASSIGNMENT

Q1:

#include <iostream>

using namespace std;

#define MAX 100

class Stack {

private:

    int arr[MAX];

    int top;

    int capacity;

public:

    Stack(int size) {

        capacity = size;

        top = -1;

    }

    bool isEmpty() {

        return top == -1;

    }

    bool isFull() {

        return top == capacity - 1;

    }

    void push(int item) {

        if (isFull()) {

            cout << "Stack is FULL. Cannot push.\n";

            return;

        }

        arr[++top] = item;

        cout << "Pushed " << item << " to stack.\n";

    }

    void pop() {

        if (isEmpty()) {

            cout << "Stack is EMPTY. Cannot pop.\n";

            return;

        }

        cout << "Popped item: " << arr[top--] << endl;

    }

    void peek() {

        if (isEmpty()) {

            cout << "Stack is EMPTY.\n";

            return;

        }

        cout << "Top element is: " << arr[top] << endl;

    }

    void display() {

        if (isEmpty()) {

            cout << "Stack is EMPTY.\n";

            return;

        }

        cout << "Stack elements (top to bottom):\n";

        for (int i = top; i >= 0; i--) {

            cout << arr[i] << endl;

        }

    }

};

int main() {

    int size;

    cout << "Enter the capacity of the stack: ";

    cin >> size;

    if (size > MAX) {

        cout << "Error: Max allowed size is " << MAX << ".\n";

        return 1;

    }

    Stack s(size);

    int choice, item;

    while (true) {

        cout << "\n=== Stack Menu ===\n";

        cout << "1. Push\n";

        cout << "2. Pop\n";

        cout << "3. Peek\n";

        cout << "4. isEmpty\n";

        cout << "5. isFull\n";

        cout << "6. Display\n";

        cout << "7. Exit\n";

        cout << "Enter your choice (1-7): ";

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter item to push: ";

                cin >> item;

                s.push(item);

                break;

            case 2:

                s.pop();

                break;

            case 3:

                s.peek();

                break;

            case 4:

                cout << (s.isEmpty() ? "Stack is EMPTY.\n" : "Stack is NOT empty.\n");

                break;

            case 5:

                cout << (s.isFull() ? "Stack is FULL.\n" : "Stack is NOT full.\n");

                break;

            case 6:

                s.display();

                break;

            case 7:

                cout << "Exiting...\n";

                return 0;

            default:

                cout << "Invalid choice. Please select from 1 to 7.\n";

        }

    }

    return 0;

}

Q2:

#include <iostream>

#include <stack>

#include <string>

using namespace std;

string reverseUsingStack(const string& input) {

    stack<char> s;

    for (char ch : input) {

        s.push(ch);

    }

    string reversed = "";

    while (!s.empty()) {

        reversed += s.top();

        s.pop();

    }

    return reversed;

}

int main() {

    string input;

    cout << "Enter a string to reverse: ";

    getline(cin, input);

    string reversed = reverseUsingStack(input);

    cout << "Reversed string: " << reversed << endl;

    return 0;

}

Q3:

#include <iostream>

#include <stack>

#include <string>

using namespace std;

bool isMatchingPair(char opening, char closing) {

    return (opening == '(' && closing == ')') ||

           (opening == '{' && closing == '}') ||

           (opening == '[' && closing == ']');

}

bool isBalanced(const string& expr) {

    stack<char> s;

    for (char ch : expr) {

        if (ch == '(' || ch == '{' || ch == '[') {

            s.push(ch);

        } else if (ch == ')' || ch == '}' || ch == ']') {

            if (s.empty() || !isMatchingPair(s.top(), ch)) {

                return false;

            }

            s.pop();

        }

    }

    return s.empty();

}

int main() {

    string expression;

    cout << "Enter an expression: ";

    getline(cin, expression);

    if (isBalanced(expression)) {

        cout << "The expression has balanced parentheses.\n";

    } else {

        cout << "The expression does NOT have balanced parentheses.\n";

    }

    return 0;

}

Q5:

#include <iostream>

#include <stack>

#include <sstream>

#include <string>

#include <cctype>

using namespace std;

int applyOperator(int operand1, int operand2, char op) {

    switch (op) {

        case '+': return operand1 + operand2;

        case '-': return operand1 - operand2;

        case '\*': return operand1 \* operand2;

        case '/':

            if (operand2 == 0) {

                throw runtime\_error("Division by zero");

            }

            return operand1 / operand2;

        default: throw runtime\_error("Invalid operator");

    }

}

int evaluatePostfix(const string& expr) {

    stack<int> s;

    stringstream ss(expr);

    string token;

    while (ss >> token) {

        if (isdigit(token[0]) || (token.length() > 1 && token[0] == '-')) {

            s.push(stoi(token));

        }

        else if (token.length() == 1 && string("+-\*/").find(token) != string::npos) {

            if (s.size() < 2)

                throw runtime\_error("Invalid expression");

            int operand2 = s.top(); s.pop();

            int operand1 = s.top(); s.pop();

            int result = applyOperator(operand1, operand2, token[0]);

            s.push(result);

        }

        else {

            throw runtime\_error("Invalid token: " + token);

        }

    }

    if (s.size() != 1)

        throw runtime\_error("Invalid postfix expression");

    return s.top();

}

int main() {

    string expression;

    cout << "Enter a postfix expression (tokens separated by spaces): ";

    getline(cin, expression);

    try {

        int result = evaluatePostfix(expression);

        cout << "Result: " << result << endl;

    } catch (const exception& e) {

        cerr << "Error: " << e.what() << endl;

    }

    return 0;

}