

Data Structures & Algorithms

Lab Assignment 3

Stack

Shefali Sharma
1024030284
2C35

1. Develop a menu driven program demonstrating the following operations on a Stack using array:
(i) push(), (ii) pop(), (iii) isEmpty(), (iv) isFull(), (v) display(), and (vi) peek().

A1)

```
#include <iostream>
using namespace std;

#define MAX 100

class Stack {
    int arr[MAX];
    int top;

public:
    Stack() { top = -1; }

    bool isFull() { return top == MAX - 1; }
    bool isEmpty() { return top == -1; }

    void push(int x) {
        if (isFull()) {
            cout << "Stack Overflow\n";
            return;
        }
        arr[++top] = x;
        cout << x << " pushed to stack\n";
    }

    void pop() {
        if (isEmpty()) {
            cout << "Stack Underflow\n";
            return;
        }
        cout << arr[top--] << " popped from stack\n";
    }

    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty\n";
            return -1;
        }
        return arr[top];
    }

    void display() {
        if (isEmpty()) {
            cout << "Stack is empty\n";
            return;
        }
        cout << "Stack elements: ";
        for (int i = top; i >= 0; i--)
            cout << arr[i] << " ";
        cout << "\n";
    }
}
```

```

};

int main() {
    Stack s;
    int choice, val;

    do {
        cout << "\n1. Push\n2. Pop\n3. isEmpty\n4. isFull\n5. Display\n6. Peek\n7. Exit\nEnter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to push: ";
                cin >> val;
                s.push(val);
                break;
            case 2:
                s.pop();
                break;
            case 3:
                cout << (s.isEmpty() ? "Stack is empty\n" : "Stack is not empty\n");
                break;
            case 4:
                cout << (s.isFull() ? "Stack is full\n" : "Stack is not full\n");
                break;
            case 5:
                s.display();
                break;
            case 6:
                val = s.peek();
                if (val != -1)
                    cout << "Top element: " << val << "\n";
                break;
            case 7:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice\n";
        }
    } while (choice != 7);

    return 0;
}

```

Output

Cle

1. Push

2. Pop

3. isEmpty

4. isFull

5. Display

6. Peek

7. Exit

Enter your choice: 5

Stack is empty

1. Push

2. Pop

3. isEmpty

4. isFull

5. Display

6. Peek

7. Exit

Enter your choice: 1

Enter value to push: 4

4 pushed to stack

1. Push

2. Pop

3. isEmpty

4. isFull

Q2. Given a string, reverse it using STACK. For example “DataStructure” should be output as “erutcurtSataD.”

A2)

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
    string str;
    cout << "Enter string: ";
    getline(cin, str);

    stack<char> s;
    for (char c : str) {
        s.push(c);
    }
}
```

```

string rev = "";
while (!s.empty()) {
    rev += s.top();
    s.pop();
}

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

return 0;
}

```

main.cpp	Output
<pre> 1 #include <iostream> 2 #include <stack> 3 using namespace std; 4 5 int main() { 6 string str; 7 cout << "Enter string: "; 8 getline(cin, str); 9 10 stack<char> s; 11 for (char c : str) { 12 s.push(c); 13 } 14 15 string rev = ""; 16 while (!s.empty()) { 17 rev += s.top(); 18 s.pop(); 19 } 20 21 cout << "Reversed string: " << rev << endl; 22 23 return 0; 24 } 25 </pre>	<pre> Enter string: HELLO WORLD Reversed string: DLROW OLLEH === Code Execution Successful === </pre>

Q3. Write a program that checks if an expression has balanced parentheses.

A3)

```

#include <iostream>
#include <stack>
using namespace std;

bool isMatchingPair(char left, char right) {
    return (left == '(' && right == ')') ||
           (left == '{' && right == '}') ||
           (left == '[' && right == ']');
}

bool isBalanced(string expr) {
    stack<char> s;

    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') {
            s.push(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (s.empty() || !isMatchingPair(s.top(), c))
                return false;
            else
                s.pop();
        }
    }
    return s.empty();
}

```

```

int main() {
    string expr;
    cout << "Enter expression: ";
    getline(cin, expr);

    if (isBalanced(expr))
        cout << "Balanced\n";
    else
        cout << "Not balanced\n";

    return 0;
}

```

main.cpp	Output
<pre> 1 #include <iostream> 2 #include <stack> 3 using namespace std; 4 5 bool isMatchingPair(char left, char right) { 6 return (left == '(' && right == ')') 7 (left == '{' && right == '}') 8 (left == '[' && right == ']'); 9 } 10 11 bool isBalanced(string expr) { 12 stack<char> s; 13 14 for (char c : expr) { 15 if (c == '(' c == '{' c == '[') { 16 s.push(c); 17 } else if (c == ')' c == '}' c == ']') { 18 if (s.empty() !isMatchingPair(s.top(), c)) 19 return false; 20 else 21 s.pop(); 22 } 23 } 24 return s.empty(); 25 } 26 </pre>	<pre> Enter expression: a+c Balanced === Code Execution Successful === </pre>

Q4. Write a program to convert an Infix expression into a Postfix expression.

```

A4)
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 3;
    return 0;
}

bool isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/' || c == '^';
}

string infixToPostfix(string infix) {
    stack<char> s;
    string postfix = "";

    for (char c : infix) {

```

```

    if (isalnum(c)) {
        postfix += c;
    } else if (c == '(') {
        s.push(c);
    } else if (c == ')') {
        while (!s.empty() && s.top() != '(') {
            postfix += s.top();
            s.pop();
        }
        if (!s.empty()) s.pop(); // pop '('
    } else if (isOperator(c)) {
        while (!s.empty() && precedence(s.top()) >= precedence(c)) {
            if (c == '^' && s.top() == '^') // right associative
                break;
            postfix += s.top();
            s.pop();
        }
        s.push(c);
    }
}
while (!s.empty()) {
    postfix += s.top();
    s.pop();
}
return postfix;
}

int main() {
    string infix;
    cout << "Enter infix expression: ";
    getline(cin, infix);

    cout << "Postfix expression: " << infixToPostfix(infix) << endl;

    return 0;
}

```

Output

```

Enter infix expression: a * (b + c)
Postfix expression: abc+*

```

```

=== Code Execution Successful ===

```

Q5. Write a program for the evaluation of a Postfix expression.

A5)

```
#include <iostream>
```

```
#include <stack>
```

```

#include <string>

using namespace std;

int evaluatePostfix(string postfix) {
    stack<int> s;

    for (char c : postfix) {
        if (isdigit(c)) {
            s.push(c - '0');
        } else {
            int val2 = s.top(); s.pop();
            int val1 = s.top(); s.pop();

            switch (c) {
                case '+': s.push(val1 + val2); break;
                case '-': s.push(val1 - val2); break;
                case '*': s.push(val1 * val2); break;
                case '/': s.push(val1 / val2); break;
                case '^': {
                    int res = 1;
                    for (int i = 0; i < val2; i++) res *= val1;
                    s.push(res);
                    break;
                }
            }
        }
    }

    return s.top();
}

int main() {
    string postfix;
    cout << "Enter postfix expression (single digit operands): ";
    getline(cin, postfix);

```

```
cout << "Evaluated result: " << evaluatePostfix(postfix) << endl;

return 0;
}
```

Output

```
Enter postfix expression (single digit operands): 23+
Evaluated result: 5
```

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