Data Structures & Algorithms :

Assignment - 3
(Stacks)

Aadita Garg
1024030461
2C32

(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().

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

#define MAX 5  // size of stack

class Stack {
   int arr[MAX];
   int top;
public:
   Stack() { top = -1; }

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

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

   void push(int val) {
      if (isFull()) {
         cout << "Stack Overflow, Cannot push " << val << endl;
      } else {
         arr[++top] = val;
         cout << val << " pushed into stack." << endl;
      }
   }

   void pop() {
      if (isEmpty()) {
         cout << "Stack Underflow, Cannot pop." << endl;
      } else {
         cout << arr[top--] << " popped from stack." << endl;
      }
   }

   void peek() {
      if (isEmpty()) {
         cout << "Stack is empty. Nothing to peek." << endl;
      } else {
         cout << "Top element is: " << arr[top] << endl;
      }
   }
```

```
void display() {
        if (isEmpty()) {
            cout << "Stack is empty." << endl;
        } else {
            cout << "Stack elements (top to bottom): ";
            for (int i = top; i >= 0; i--) {
                cout << arr[i] << " ";
            }
            cout << endl;
        }
    }
};

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

    do {
        cout << "Stack Menu :" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter value to push: ";
            cin >> value;
            s.push(value);
            break;
        case 2:
            s.pop();
            break;
        case 3:
            cout << (s.isEmpty() ? "Stack is Empty" : "Stack is NOT Empty") << endl;
            break;
        case 4:
            cout << (s.isFull() ? "Stack is Full" : "Stack is NOT Full") << endl;
            break;
        case 5:
            s.display();
            break;
        case 6:
            s.peek();
            break;
        case 7:
            cout << "Exiting program..." << endl;
            break;
```

```
default:
        cout << "Invalid choice! Try again." << endl;
      }
   } while (choice != 7);

   return 0;
}
```

```
TERMINAL

Stack Menu :
Enter your choice: 1
Enter value to push: 3
3 pushed into stack.
Stack Menu :
Enter your choice: 1
Enter value to push: 4
4 pushed into stack.
Stack Menu :
Enter your choice: 1
Enter value to push: 5
5 pushed into stack.
Stack Menu :
Enter your choice: 2
5 popped from stack.
Stack Menu :
Enter your choice: 3
Stack is NOT Empty
Stack Menu :
Enter your choice: 4
Stack is NOT Full
Stack Menu :
Enter your choice: 5
Stack elements (top to bottom): 4 3
Stack Menu :
Enter your choice: 6
Top element is: 4
Stack Menu :
Enter your choice: 7
Exiting program...
```

(2) Given a string , reverse it using stack operations.
String = DataStructure

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

string reverseString(string s) {
    stack<char> st;

    // Push all characters
    for (char c : s) {
        st.push(c);
    }

    // Pop and build reversed string
    string reversed = "";
    while (!st.empty()) {
        reversed += st.top();
        st.pop();
    }
    return reversed;
}

int main() {
    string str = "DataStructure";
    cout << "Original: " << str << endl;

    string rev = reverseString(str);
    cout << "Reversed: " << rev << endl;

    return 0;
}
```

```
TERMINAL

Original: DataStructure
Reversed: erutcurtSataD


** Process exited - Return Code: 0 **
```

(3) Check if the following string has valid parenthesis

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

bool isValid(string s) {
   stack<char> st;

   for (char c : s) {
      if (c == '(' || c == '{' || c == '[') {
         st.push(c); // push opening
      } else {
         if (st.empty()) return false; // no matching opening

         char top = st.top();
         st.pop();

         if ((c == ')' && top != '(') ||
            (c == '}' && top != '{') ||
            (c == ']' && top != '[')) {
            return false; // mismatch
         }
      }
   }
   return st.empty(); // valid only if stack is empty
}
int main() {
   string s;
   cout << "Enter parentheses string: ";
   cin >> s;

   if (isValid(s))
      cout << "Valid Parentheses" << endl;
   else
      cout << "Invalid Parentheses" << endl;

   return 0;
}
```

```
TERMINAL
Enter parentheses string: ()[{()}]
Valid Parentheses


** Process exited - Return Code: 0 **
```

(4) Write a program to convert an Infix expression into a Postfix expression.

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

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

bool isRightAssociative(char op) {
   return (op == '^');
}

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

   for (char c : infix) {
      if (isalnum(c)) {
         postfix += c; // operand directly to output
      }
      else if (c == '(') {
         st.push(c);
      }
      else if (c == ')') {
         while (!st.empty() && st.top() != '(') {
            postfix += st.top();
            st.pop();
         }
         if (!st.empty()) st.pop(); // remove '('
      }
      else { // operator
         while (!st.empty() &&
               (precedence(st.top()) > precedence(c) ||
               (precedence(st.top()) == precedence(c) && !isRightAssociative(c))) &&
               st.top() != '(') {
            postfix += st.top();
            st.pop();
         }
         st.push(c);
      }
   }
```

```
    // Pop remaining operators
    while (!st.empty()) {
        postfix += st.top();
        st.pop();
    }

    return postfix;
}

int main() {
    string infix;
    cout << "Enter an infix expression: ";
    cin >> infix;

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

    return 0;
}
```

TERMINAL

```
Enter an infix expression: A^B^C
Postfix expression: ABC^^


** Process exited - Return Code: 0 **
```

(5) Write a program for the evaluation of a Postfix expression.

```
#include <iostream>
#include <stack>
#include <cmath>
using namespace std;
int evaluatePostfix(string postfix) {
   stack<int> st;

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

         switch (c) {
            case '+': st.push(val1 + val2); break;
            case '-': st.push(val1 - val2); break;
            case '*': st.push(val1 * val2); break;
            case '/': st.push(val1 / val2); break;
            case '^': st.push(pow(val1, val2)); break;
         }
      }
   }
   return st.top();
}


int main() {
   string postfix;
   cout << "Enter postfix expression (digits only): ";
   cin >> postfix;

   int result = evaluatePostfix(postfix);
   cout << "Result = " << result << endl;

   return 0;
}
```

TERMINAL

```
Enter postfix expression (digits only): 231*+9-
Result = -4
```