```
//Vansh Arora
//1024030514

//Q1
#include<iostream>
using namespace std;
#include <iostream>
using namespace std;
#define MAX 5
class Stack {
   int arr[MAX];
   int top;
public:
   Stack() {
      top = -1;
   }
 void push(int x) {
      if (isFull()) {
         cout << "Stack Overflow! Cannot push " << x << endl;
      } else {
         arr[++top] = x;
         cout << x << " pushed into stack." << endl;
      }
   }
void pop() {
      if (isEmpty()) {
         cout << "Stack Underflow! Cannot pop." << endl;
      } else {
         cout << arr[top--] << " popped from stack." << endl;
      }
   }
```

```
    bool isEmpty() {
        return top == -1;
    }
  bool isFull() {
        return top == MAX - 1;
    }
  void display() {
        if (isEmpty()) {
            cout << "Stack is empty." << endl;
        } else {
            cout << "Stack elements: ";
            for (int i = top; i >= 0; i--) {
                cout << arr[i] << " ";
            }
            cout << endl;
        }
    }
  void peek() {
        if (isEmpty()) {
            cout << "Stack is empty. No top element." << endl;
        } else {
            cout << "Top element: " << arr[top] << endl;
        }
    }
};
int main() {
    Stack s;
    int choice, value;
do {
        cout << "\n--- Stack Menu ---\n";
        cout << "1. Push\n";
```

```
cout << "2. Pop\n";
cout << "3. isEmpty\n";
cout << "4. isFull\n";
cout << "5. Display\n";
cout << "6. Peek\n";
cout << "7. Exit\n";
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:
      if (s.isEmpty())
         cout << "Stack is empty." << endl;
      else
         cout << "Stack is not empty." << endl;
      break;
   case 4:
      if (s.isFull())
         cout << "Stack is full." << endl;
      else
         cout << "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;
}
//Q2)
#include <iostream>
#include <stack>
using namespace std;
string reverseString(string str) {
   stack<char> s;
    for (char c : str) {
       s.push(c);
   }
   string reversed = "";
   while (!s.empty()) {
       reversed += s.top();
       s.pop();
   }
    return reversed;
}
int main() {
```

```
    string str = "DataStructure";
    cout << "Original String: " << str << endl;
    cout << "Reversed String: " << reverseString(str) << endl;
    return 0;
}
//Q3)
#include <iostream>
#include <stack>
using namespace std;

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

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;
            }
            s.pop();
        }
    }
```

```
    return s.empty();
}

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

    if (isBalanced(expr))
        cout << "Balanced" << endl;
    else
        cout << "Not Balanced" << endl;

    return 0;
}
//Q4)
#include <iostream>
#include <stack>
using namespace std;

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

// Function to check if operator is right associative
bool isRightAssociative(char op) {
    return (op == '^');
```

```
}

// Function to check if character is operator
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

// Function to convert Infix to Postfix
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(); // remove '('
        }
        else if (isOperator(c)) {
            while (!s.empty() && isOperator(s.top())) {
                if ((precedence(c) < precedence(s.top())) ||
                    (precedence(c) == precedence(s.top()) && !isRightAssociative(c))) {
                    postfix += s.top();
                    s.pop();
```

```
            } else break;
        }
        s.push(c);
    }
  }
  while (!s.empty()) {
    postfix += s.top();
    s.pop();
  }
 return postfix;
}
int main() {
  string infix;
  cout << "Enter Infix Expression: ";
  cin >> infix;
cout << "Postfix Expression: " << infixToPostfix(infix) << endl;
return 0;
}
//Q5)
#include <iostream>
#include <stack>
#include <cmath>
using namespace std;
int evaluatePostfix(string postfix) {
  stack<int> s;
  for (char c : postfix) {
    if (isdigit(c)) {
      s.push(c - '0');
    }
    else {
      int op1, op2;
```

```
            if (!s.empty()) { op1 = s.top(); s.pop(); }
            else { cout << "Invalid Expression!"; return -1; }
            if (!s.empty()) { op2 = s.top(); s.pop(); }
            else { cout << "Invalid Expression!"; return -1; }
     switch (c) {
                case '+': s.push(op2 + op1); break;
                case '-': s.push(op2 - op1); break;
                case '*': s.push(op2 * op1); break;
                case '/': s.push(op2 / op1); break;
                case '^': s.push(pow(op2, op1)); break;
                default:
                    cout << "Invalid operator: " << c << endl;
                    return -1;
            }
        }
    }
    return s.top();
}
int main() {
    string postfix;
    cout << "Enter Postfix Expression (operands must be single-digit): ";
    cin >> postfix;
    int result = evaluatePostfix(postfix);
    cout << "Result = " << result << endl;
    return 0;
}

//EQ1)
#include <iostream>
#include <vector>
#include <stack>
```

```
using namespace std;

vector<int> nearestSmallerToLeft(vector<int>& arr) {
    stack<int> s;
    vector<int> result;

    for (int i = 0; i < arr.size(); i++) {

        while (!s.empty() && s.top() >= arr[i]) {
            s.pop();
        }


        if (s.empty())
            result.push_back(-1);
        else
            result.push_back(s.top());


        s.push(arr[i]);
    }

    return result;
}

int main() {
    vector<int> arr = {4, 5, 2, 10, 8};
    vector<int> ans = nearestSmallerToLeft(arr);

    cout << "Nearest Smaller to Left: ";
    for (int x : ans) cout << x << " ";
```

```
    cout << endl;


    return 0;
}
```