**Q.1) Array implementation of stack.**

import java.util.Scanner;

class stack {

    public int arr[];

    public int size;

    public int top = -1;

    stack(int size) {

        this.size = size;

        arr = new int[size];

    }

    public boolean isEmpty() {

        return (top == -1);

    }

    public boolean isFull() {

        return (top == size - 1);

    }

    public int peek() {

        if (isEmpty()) {

            return -1;

        } else {

            return arr[top];

        }

    public void push(int x) {

        if (isFull()) {

            System.out.println("Stack is full");

        } else {

            arr[++top] = x;

        }

    }

    public int pop() {

        int ans;

        if (isEmpty()) {

            return -1;

        } else {

            ans = arr[top];

            top--;

        }

        return ans;

    }

    public void display() {

        if (isEmpty()) {

            System.out.println("Stack is empty");

            return;

        }

        for (int i = 0; i <= top; i++) {

            System.out.print(arr[i] + " ");

        }

        System.out.println();

    }

}

public class StackArr {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the size of the stack : ");

        int n = sc.nextInt();

        stack st = new stack(n);

        int option;

        char ch = 'Y';

        while (ch == 'Y' || ch == 'y') {

            System.out.println("1.To push\n2.To pop\n3.To display\n4.Top element");

            option = sc.nextInt();

            switch (option) {

                case 1:

                    System.out.print("Enter the element : ");

                    int ele = sc.nextInt();

                    st.push(ele);

                    break;

                case 2:

                    int popped = st.pop();

                    System.out.println("Popped element is : " + popped);

                    break;

                case 3:

                    st.display();

                    break;

                case 4:

                    int top = st.peek();

                    System.out.println("Top element is : " + top);

                    break;

                default:

                    System.out.println("Invalid input");

            }

            System.out.println("Do you want to continue ?");

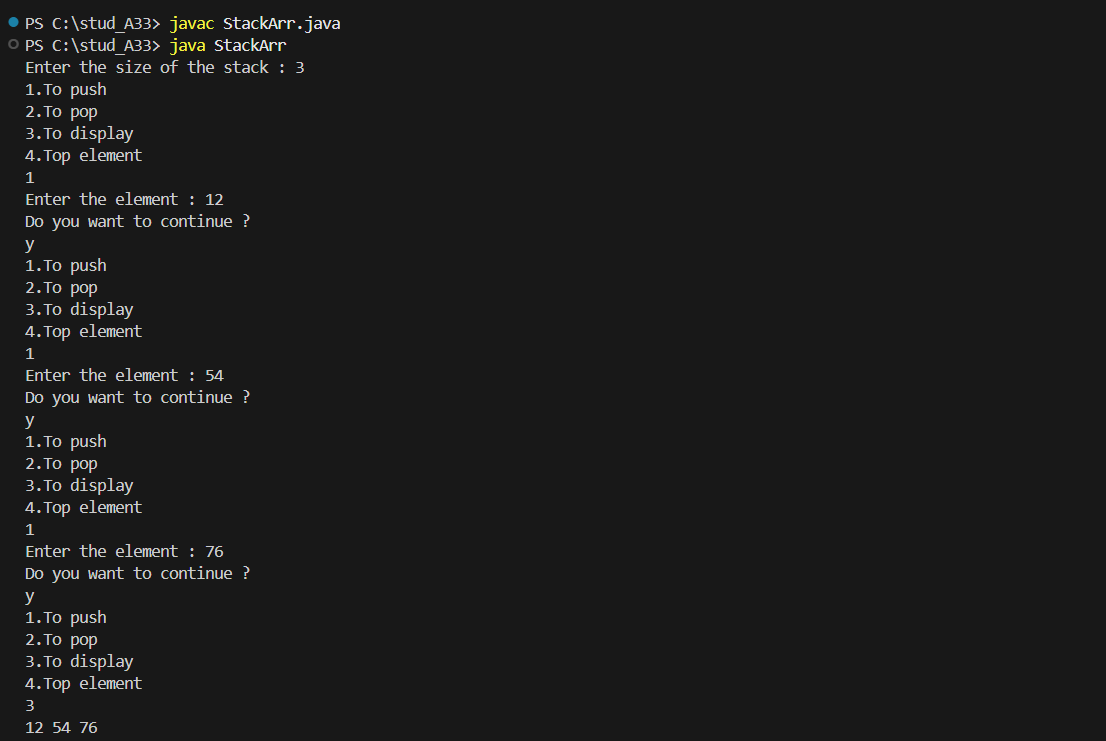
            ch = sc.next().charAt(0);

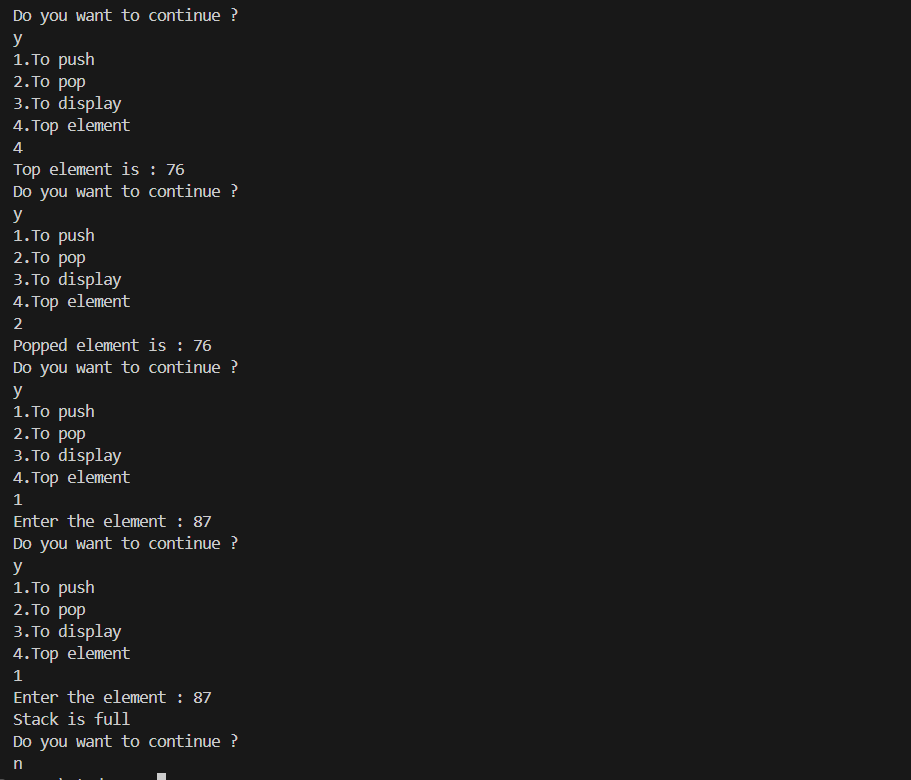
        }

    }

}

Output :





**Q.2) Array implementation of queue.**

import java.util.Scanner;

class queue {

    int arr[];

    int front = -1;

    int rear = -1;

    int size

    queue(int size) {

        this.size = size;

        arr = new int[size];

    }

    public boolean isEmpty() {

        return (front == -1);

    }

    public boolean isFull() {

        return (rear == size - 1);

    }

    public void enqueue(int x) {

        if (isFull()) {

            System.out.println("Queue full");

            return;

        }

        if (isEmpty()) {

            front = rear = 0;

        } else {

            rear++;

        }

        arr[rear] = x;

    }

    public int dequeue() {

        if (isEmpty()) {

            System.out.println("Queue empty");

            return -1;

        }

        int ans = arr[front];

        if (front == rear) {

            front = rear = -1;

        } else {

            front++;

        }

        return ans;

    }

    public void display() {

        if (isEmpty()) {

            System.out.println("Queue is empty");

            return;

        }

        System.out.print("Queue elements: ");

        for (int i = front; i <= rear; i++) {

            System.out.print(arr[i] + " ");

        }

        System.out.println();

    }

}

public class QueueArr {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the size of the queue : ");

        int n = sc.nextInt();

        queue q = new queue(n);

        int option;

        char ch = 'Y';

        while (ch == 'Y' || ch == 'y') {

            System.out.println("1.To enqueue\n2.To dequeue\n3.To display");

            option = sc.nextInt();

            switch (option) {

                case 1:

                    System.out.print("Enter the element : ");

                    int ele = sc.nextInt();

                    q.enqueue(ele);

                    break;

                case 2:

                    int popped = q.dequeue();

                    System.out.println("Popped front element is : " + popped);

                    break;

                case 3:

                    q.display();

                    break;

                default:

                    System.out.println("Invalid input");

            }

            System.out.println("Do you want to continue ?");

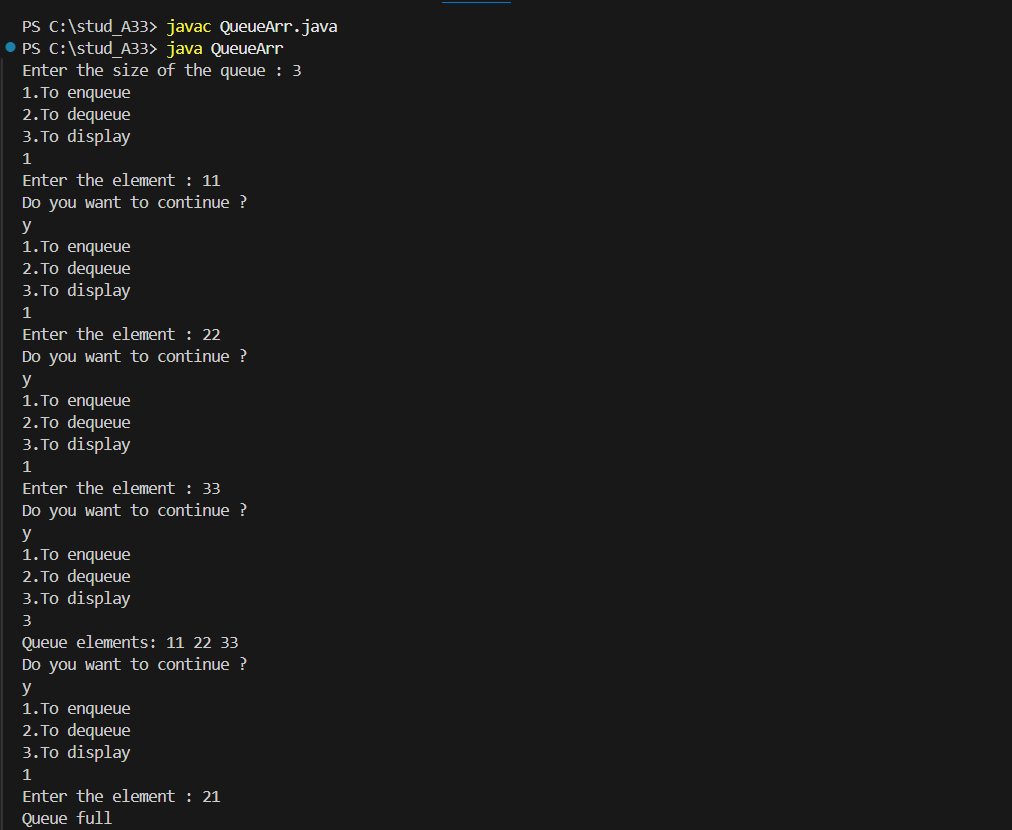
            ch = sc.next().charAt(0);

        }

    }

}

Output :





**Q.2) Array implementation of circular queue.**

import java.util.Scanner;

class queue

    int arr[];

    int front = -1;

    int rear = -1;

    int size;

    queue(int size) {

        this.size = size;

        arr = new int[size];

    }

    public boolean isEmpty() {

        return (front == -1);

    }

    public boolean isFull() {

        return ((rear + 1) % size == front);

    }

    public void enqueue(int x) {

        if (isFull()) {

            System.out.println("Queue is full");

            return;

        }

        if (isEmpty()) {

            front = rear = 0;

        } else {

            rear = (rear + 1) % size;

        }

        arr[rear] = x;

    }

    public int dequeue() {

        if (isEmpty()) {

            System.out.println("Queue is empty");

            return -1;

        }

        int ans = arr[front];

        if (front == rear) {

            front = rear = -1;

        } else {

            front = (front + 1) % size;

        }

        return ans;

    }

    public void display() {

        if (isEmpty()) {

            System.out.println("Queue is empty");

            return;

        }

        System.out.print("Queue elements: ");

        int i = front;

        while (i != rear) {

            System.out.print(arr[i] + " ");

            i = (i + 1) % size;

        }

        System.out.println(arr[rear]);

    }

}

public class QueueCirc {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the size of the queue: ");

        int n = sc.nextInt();

        queue q = new queue(n);

        int option;

        char ch = 'Y';

        while (ch == 'Y' || ch == 'y') {

            System.out.println("1. To enqueue\n2. To dequeue\n3. To display");

            option = sc.nextInt();

            switch (option) {

                case 1:

                    System.out.print("Enter the element: ");

                    int ele = sc.nextInt();

                    q.enqueue(ele);

                    break;

                case 2:

                    int popped = q.dequeue();

                    System.out.println("Dequeued front element is: " + popped);

                    break;

                case 3:

                    q.display();

                    break;

                default:

                    System.out.println("Invalid input");

            }

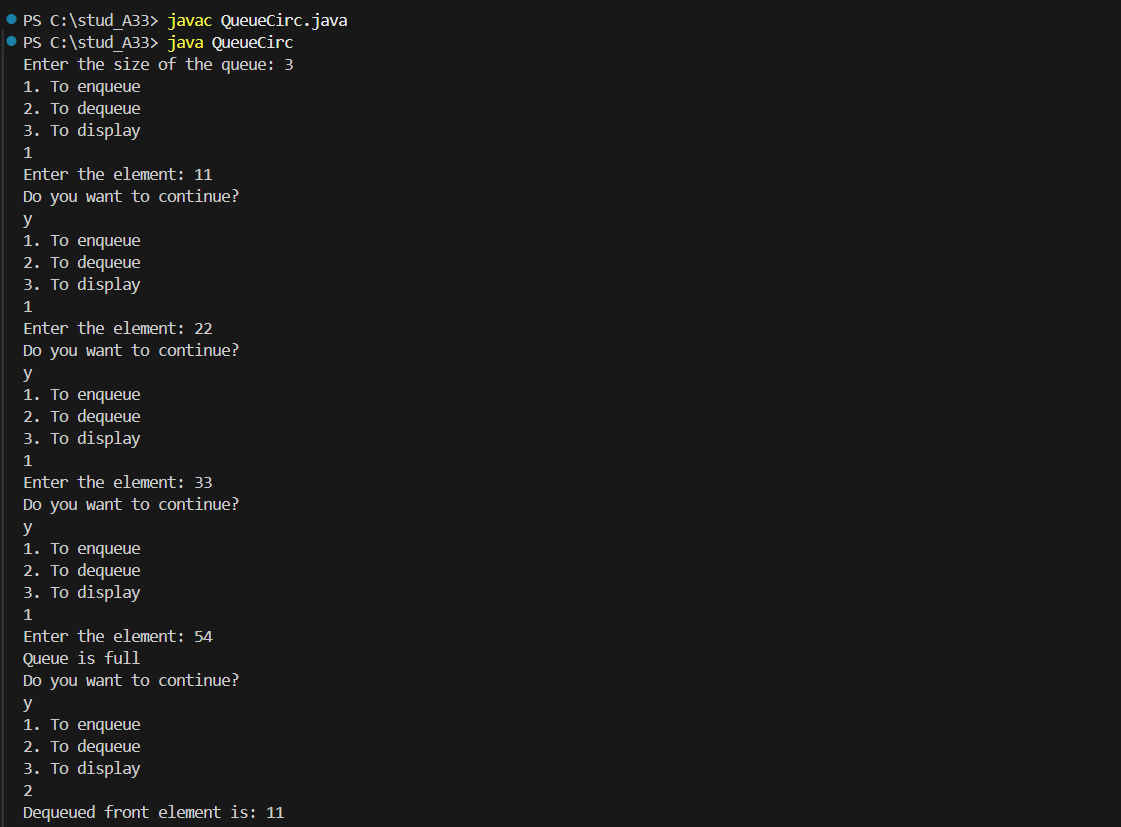
            System.out.println("Do you want to continue?");

            ch = sc.next().charAt(0);

        }

    } }

Output :





**Q.3) Conversion of infix to postfix notation.**

import java.util.Stack;

class Solution {

    boolean isOperator(char ch) {

        return ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^';

    }

    int precedence(char ch) {

        if (ch == '\*' || ch == '/') {

            return 2;

        }

        if (ch == '+' || ch == '-') {

            return 1;

        }

        if (ch == '^') {

            return 3;

        }

        return 0;

    }

    String infixToPostfix(String str) {

        String ans = "";

        Stack<Character> st = new Stack<>();

        int size = str.length();

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

            if (Character.isLetterOrDigit(str.charAt(i))) {

                ans += str.charAt(i);

            } else if (isOperator(str.charAt(i))) {

                while (!st.isEmpty() && precedence(st.peek()) >= precedence(str.charAt(i))) {

                    ans += st.pop();

                }

                st.push(str.charAt(i));

            } else if (str.charAt(i) == '(') {

                st.push(str.charAt(i));

            } else if (str.charAt(i) == ')') {

                while (!st.isEmpty() && st.peek() != '(') {

                    ans += st.pop();

                }

                st.pop();

            }

        }

        while (!st.isEmpty()) {

            ans += st.pop();

        }

        return ans;

    }

}

public class inToPost {

    public static void main(String[] args) {

        Solution s = new Solution();

        System.out.println(s.infixToPostfix("a+b\*(c^d-e)^(f+g\*h)-i"));

    }

}

Output :



**Q.4) Evaluation of postfix expression and balancing of parentheses.**

import java.util.Stack;

class Solution {

    public boolean isBalanced(String expression) {

        Stack<Character> stack = new Stack<>();

        for (char ch : expression.toCharArray()) {

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

                stack.push(ch);

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

                if (stack.isEmpty()) {

                    return false;

                }

                char top = stack.pop();

                if ((ch == ')' && top != '(')

                        || (ch == '}' && top != '{')

                        || (ch == ']' && top != '[')) {

                    return false;

                }

            }

        }

        return stack.isEmpty();

    }

    public int evaluatePostfix(String expression) {

        Stack<Integer> stack = new Stack<>();

        for (char ch : expression.toCharArray()) {

            if (Character.isDigit(ch)) {

                stack.push(ch - '0');

            } else {

                int operand2 = stack.pop();

                int operand1 = stack.pop();

                switch (ch) {

                    case '+':

                        stack.push(operand1 + operand2);

                        break;

                    case '-':

                        stack.push(operand1 - operand2);

                        break;

                    case '\*':

                        stack.push(operand1 \* operand2);

                        break;

                    case '/':

                        stack.push(operand1 / operand2);

                        break;

                    case '^':

                        stack.push((int) Math.pow(operand1, operand2));

                        break;

                }

            }

        }

        return stack.pop();

    }

}

public class Test {

    public static void main(String[] args) {

        Solution s = new Solution();

        System.out.println("Result of Postfix Evaluation: " + s.evaluatePostfix("231\*+9-"));

        System.out.println("Is Parentheses Balanced? :  " + s.isBalanced("{[()]}"));

    }

}

Output :

