

**Assignment Problem 1. Infix to Postfix Conversion**

**Concept Used:** Stack for operator precedence and order

**Problem:**

Convert an infix expression (e.g., A + B \* C) into postfix form.

**Hint:** Use a stack to store operators and apply precedence rules (\* > +, etc.). Pop when precedence drops.

PROGRAM

import java.util.\*;

public class InfixToPostfix {

public static int precedence(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

public static String infixToPostfix(String exp) {

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

StringBuilder result = new StringBuilder();

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

if (Character.isLetterOrDigit(ch))

result.append(ch);

else if (ch == '(')

stack.push(ch);

else if (ch == ')') {

while (!stack.isEmpty() && stack.peek() != '(')

result.append(stack.pop());

stack.pop();

} else {

while (!stack.isEmpty() && precedence(ch) <= precedence(stack.peek()))

result.append(stack.pop());

stack.push(ch);

}

}

while (!stack.isEmpty())

result.append(stack.pop());

return result.toString();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter infix expression: ");

String exp = sc.nextLine();

System.out.println("Postfix: " + infixToPostfix(exp));

}

}

OUTPUT

Sample Input:

A+B\*C

Output:

Postfix: ABC\*+

**Assignment Problem 2. Implement Queue Using Array**

**Concept Used:** Linear Queue implementation using list (array indexing) **Problem:**

Implement a Queue with basic operations — enqueue, dequeue, peek, isFull, and isEmpty.

**Hint:** Maintain front and rear pointers to track insertion/removal positions.

PROGRAM

import java.util.\*;

public class QueueUsingArray {

int size, front, rear;

int[] queue;

QueueUsingArray(int capacity) {

size = capacity;

queue = new int[size];

front = rear = -1;

}

void enqueue(int data) {

if (rear == size - 1) {

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

return;

}

if (front == -1) front = 0;

queue[++rear] = data;

System.out.println(data + " enqueued.");

}

void dequeue() {

if (front == -1 || front > rear) {

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

return;

}

System.out.println(queue[front++] + " dequeued.");

}

void display() {

if (front == -1 || front > rear) {

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

return;

}

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

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

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

System.out.println();

}

public static void main(String[] args) {

QueueUsingArray q = new QueueUsingArray(5);

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.display();

q.dequeue();

q.display();

}

}

OUTPUT

10 enqueued.

20 enqueued.

30 enqueued.

Queue: 10 20 30

10 dequeued.

Queue: 20 30

**Assignment Problem 3. Circular Queue Implementation**

**Concept Used:** Circular Queue logic (modular arithmetic)

**Problem:**

Implement a Circular Queue with operations to insert, delete, and display elements. **Hint:** Use (rear + 1) % size to make the queue circular.

PROGRAM

import java.util.\*;

public class CircularQueue {

int front, rear, size;

int[] queue;

CircularQueue(int capacity) {

size = capacity;

queue = new int[size];

front = rear = -1;

}

void enqueue(int data) {

if ((rear + 1) % size == front) {

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

return;

}

if (front == -1) front = 0;

rear = (rear + 1) % size;

queue[rear] = data;

System.out.println(data + " inserted.");

}

void dequeue() {

if (front == -1) {

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

return;

}

System.out.println(queue[front] + " deleted.");

if (front == rear)

front = rear = -1;

else

front = (front + 1) % size;

}

void display() {

if (front == -1) {

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

return;

}

System.out.print("Elements: ");

int i = front;

while (true) {

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

if (i == rear)

break;

i = (i + 1) % size;

}

System.out.println();

}

public static void main(String[] args) {

CircularQueue cq = new CircularQueue(5);

cq.enqueue(10);

cq.enqueue(20);

cq.enqueue(30);

cq.display();

cq.dequeue();

cq.display();

}

}

OUTPUT

10 inserted.

20 inserted.

30 inserted.

Elements: 10 20 30

10 deleted.

Elements: 20 30

**Assignment Problem 4. Stack-Based Undo Feature**

**Concept Used:** Stack for undo operations (reversal of last action)

**Problem:**

Simulate a simple text editor that supports undo. Each action is stored in a stack. **Hint:** Push every action to stack; on undo, pop the last action and display remaining state.

PROGRAM

import java.util.\*;

public class UndoFeature {

public static void main(String[] args) {

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

Scanner sc = new Scanner(System.in);

while (true) {

System.out.print("Command (TYPE <word>/UNDO/PRINT/EXIT): ");

String cmd = sc.next();

if (cmd.equalsIgnoreCase("TYPE")) {

String word = sc.next();

stack.push(word);

} else if (cmd.equalsIgnoreCase("UNDO")) {

if (!stack.isEmpty()) stack.pop();

else System.out.println("Nothing to undo!");

} else if (cmd.equalsIgnoreCase("PRINT")) {

System.out.println(String.join(" ", stack));

} else if (cmd.equalsIgnoreCase("EXIT")) {

break;

} else {

System.out.println("Invalid command!");

}

}

}

}

OUTPUT

Input:

TYPE Namaste

TYPE Bharat

UNDO

TYPE India

PRINT

Output:

Namaste India

**Assignment Problem 5. Next Greater Element**

1



**Concept Used:** Stack for array traversal and element comparison

**Problem:**

For each element in an array, find the next greater element to its right.

**Example:**

Input: [4, 5, 2, 25] → Output: [5, 25, 25, -1]

**Hint:** Traverse array, use stack to keep track of elements waiting for a greater number.

PROGRAM

import java.util.\*;

public class NextGreaterElement {

public static void main(String[] args) {

int[] arr = {4, 5, 2, 25};

int n = arr.length;

int[] nge = new int[n];

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

for (int i = n - 1; i >= 0; i--) {

while (!stack.isEmpty() && stack.peek() <= arr[i])

stack.pop();

nge[i] = stack.isEmpty() ? -1 : stack.peek();

stack.push(arr[i]);

}

System.out.println("Next Greater Elements:");

for (int i = 0; i < n; i++)

System.out.println(arr[i] + " → " + nge[i]);

}

}

OUTPUT

Next Greater Elements:

4 → 5

5 → 25

2 → 25

25 → -1

**Assignment Problem 6. Queue Simulation for Customer Service**

**Concept Used:** Queue for task scheduling / FIFO operations

**Problem:**

Simulate a customer service system where customers are enqueued on arrival and dequeued for service.

**Hint:** Use enqueue() to add customers and dequeue() to serve. Display queue after every operation.

PROGRAM

import java.util.\*;

public class CustomerServiceQueue {

public static void main(String[] args) {

Queue<String> queue = new LinkedList<>();

Scanner sc = new Scanner(System.in);

while (true) {

System.out.print("Command (ARRIVE <name>/SERVE/STATUS/EXIT): ");

String cmd = sc.next();

if (cmd.equalsIgnoreCase("ARRIVE")) {

String name = sc.next();

queue.add(name);

System.out.println(name + " joined the queue.");

} else if (cmd.equalsIgnoreCase("SERVE")) {

if (!queue.isEmpty())

System.out.println("Serving " + queue.poll());

else

System.out.println("No customers to serve!");

} else if (cmd.equalsIgnoreCase("STATUS")) {

System.out.println("Waiting: " + queue);

} else if (cmd.equalsIgnoreCase("EXIT")) {

break;

} else {

System.out.println("Invalid command!");

}

}

}

}

OUTPUT

Input:

ARRIVE Rohan

ARRIVE Priya

ARRIVE Aarav

SERVE

STATUS

Output:

Rohan joined the queue.

Priya joined the queue.

Aarav joined the queue.

Serving Rohan

Waiting: [Priya, Aarav]