STACK & QUEUE

HARD Q: :-1 Evaluate Reverse Polish Notation.

You are given an array of strings tokens that represent an arithmetic expression in Reverse Polish Notation.

Evaluate the expression. Return *an integer that represents the value of the expression*.

Note that:

- The valid operators are '+', '-', '*', and '/'.
- Each operand may be an integer or another expression.
- The division between two integers always truncates toward zero.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in reverse polish notation.
- The answer and all the intermediate calculations can be represented in a 32-bit integer.

Example 1:
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
Example 2:
Input: tokens = ["4", "13", "5","/","+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
Example 3:
Input: tokens = ["10", "6", "9", "3","+","-11","*","/","*", "17","+", "5","+"]
Output: 22
Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22

Constraints:

- 1 <= tokens.length<= 10^4
- tokens[i] is either an operator: "+", "-", "*", or "/", or an integer in the range [-200, 200].

Solution:-

```
class Solution {
  public int evalRPN(String[] tokens) {
      Stack<Integer>st = new Stack<>();
      for (String token: tokens) {
         if (token.equals("+") || token.equals("-") || token.equals("*") || token.equals("/")) {
```

```
            int b = st.pop(); // right operand
            int a = st.pop(); // left operand
            int res = 0;
            switch (token) {
               case "+": res = a + b; break;
               case "-": res = a - b; break;
               case "*": res = a * b; break;
               case "/": res = a / b; break;
            }
st.push(res);
         } else {
st.push(Integer.parseInt(token));
         }
      }
      return st.pop();
   }
}
```

MEDIUM Q:-2 Valid Parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' , and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:
Input: s = "()"
Output: true
Example 2:
Input: s = "()[]{}"
Output: true
Example 3:
Input: s = "(]"
Output: false
Example 4:
Input: s = "([])"
Output: true

Constraints:

- 1 <= s.length<= 10^4
- s consists of parentheses only '()[]{}'.

Solution:-

```
class Solution {
   public boolean isValid(String s) {
      Stack<Character>st=new Stack<Character>();
for(int i=0;i<s.length();i++)
      {
         char ch=s.charAt(i);
         if(ch=='(')
         {
st.push('(');
         }
        else if(ch=='{')
         {
st.push('{');
         }
        else if(ch=='[')
         {
st.push('[');
         }
```

```
        else if(ch==')')
        {
          if(st.size()==0)
          {
            return false;
          }
          else if(st.peek()!='(')
          {
            return false;
          }
          else
          {
st.pop();
          }
        }
        else if(ch=='}')
        {
          if(st.size()==0)
          {
            return false;
          }
          else if(st.peek()!='{')
          {
            return false;
          }
          else
          {
st.pop();
          }
        }
        else if(ch==']')
        {
          if(st.size()==0)
          {
            return false;
          }
          else if(st.peek()!='[')
          {
            return false;
          }
          else
          {
st.pop();
```

```
            }
          }
        }
        if(st.size()>0)
        {
          return false;
        }
        else
        {
          return true;
        }

    }
}
```

MEDIUM Q:-3 Implement

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

Implement the MyStack class:

- void push(int x) Pushes element x to the top of the stack.
- int pop() Removes the element on the top of the stack and returns it.
- int top() Returns the element on the top of the stack.
- boolean empty() Returns true if the stack is empty, false otherwise.

Notes:

- You must use only standard operations of a queue, which means that only push to back, peek/pop from front, size , and is empty operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Example 1:

Input

["MyStack", "push", "push", "top", "pop", "empty"]

[[], [1], [2], [], [], []]

Output

[null, null, null, 2, 2, false]

Explanation

MyStackmyStack = new MyStack();

myStack.push(1);

myStack.push(2);

myStack.top(); // return 2

myStack.pop(); // return 2

myStack.empty(); // return False

Constraints:

- 1 <= x <= 9
- At most 100 calls will be made to push, pop, top, and empty.
- All the calls to pop and top are valid.

Solution:-

```
class MyStack {

Queue<Integer>q1;
Queue<Integer>q2;
   public MyStack() {
      q1=new LinkedList();
      q2=new LinkedList();
```

```
    }

    public void push(int x) {
        while(!q2.isEmpty())
        {
            q1.add(q2.poll());

        }
        q2.add(x);
         while(!q1.isEmpty())
        {
            q2.add(q1.poll());

        }

    }

    public int pop() {
        return q2.poll();

    }

    public int top() {
        return q2.peek();
    }
    public boolean empty() {
      return q2.isEmpty();

    }
}

/**
 * Your MyStack object will be instantiated and called as such:
 * MyStackobj = new MyStack();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.top();
 * boolean param_4 = obj.empty();
 */
```

Q:-4 HARD Online Stock Span

Design an algorithm that collects daily price quotes for some stock and returns the span of that stock's price for the current day.

The span of the stock's price in one day is the maximum number of consecutive days (starting from that day and going backward) for which the stock price was less than or equal to the cost of that day.

- For example, if the prices of the stock in the last four days are [7,2,1,2] and the cost of the stock today is 2, then the span of today is 4 because starting from today, the price of the stock was less than or equal to 2 for 4 consecutive days.
- Also, if the prices of the stock in the last four days are [7,34,1,2] and the cost of the stock today is 8, then the span of today is 3 because starting from today, the price of the stock was less than or equal to 8 for 3 consecutive days.

Implement the StockSpanner class:

- StockSpanner() Initializes the object of the class.
- int next(int price) Returns the span of the stock's price given that today's price is price.

Example 1:
Input
["StockSpanner", "next", "next", "next", "next", "next", "next", "next"]
[[], [100], [80], [60], [70], [60], [75], [85]]
Output
[null, 1, 1, 1, 2, 1, 4, 6]

Explanation
StockSpannerstockSpanner = new StockSpanner();
stockSpanner.next(100); // return 1
stockSpanner.next(80); // return 1
stockSpanner.next(60); // return 1
stockSpanner.next(70); // return 2
stockSpanner.next(60); // return 1
stockSpanner.next(75); // return 4, because the last 4 prices (including today's price of 75) were less than or equal to today's price.
stockSpanner.next(85); // return 6

Constraints:

- $1 <= price <= 10^5$
- At most 10^4 calls will be made to the next.

Solution:-

```
class StockSpanner {
   Stack<Node>st;

   public StockSpanner() {
```

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

    }

    public int next(int price) {
        int count=0;
        while(!st.isEmpty() &&st.peek().preprice<=price)
        {
            count=count+st.peek().precount;
st.pop();
        }
        count++;
st.push(new Node(count,price));
        return count;
    }
    static class Node
    {
        int precount;
        int preprice;
Node(int precount, int preprice)
        {
this.precount=precount;
this.preprice=preprice;
        }

    }

}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpannerobj = new StockSpanner();
 * int param_1 = obj.next(price);
 */
```

Q:- 5 HARD Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

Example 1:
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null, null, null,null,-3,null,0,-2]

Explanation
MinStackminStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2

Constraints:

- $-2^{31} <= val <= 2^{31} - 1$
- Methods pop, top and getMin operations will always be called on non-empty stacks.
- At most $3 * 10^4$ calls will be made to push, pop, top, and getMin.

Solution:-

```
class MinStack {
  Stack<Integer> stack;
    Stack<Integer>minstack;
    public MinStack() {
        stack=new Stack();
minstack=new Stack();
    }
    public void push(int val) {
```

```
stack.push(val);
        if(minstack.isEmpty() || minstack.peek()>val)
        {
minstack.push(val);
        }
        else
        {
minstack.push(minstack.peek());
        }

    }

    public void pop() {
stack.pop();
minstack.pop();

    }

    public int top() {
      return stack.peek();
    }

    public int getMin() {
      return minstack.peek();

    }
}

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStackobj = new MinStack();
 * obj.push(val);
 * obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.getMin();
 */
```

Q:-6 MEDIUM Implement Queue using Stacks.

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).
Implement the MyQueue class:

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Notes:

- You must use only standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Example 1:
Input
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 1, 1, false]

Explanation
MyQueuemyQueue = new MyQueue();
myQueue.push(1); // queue is: [1]
myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)
myQueue.peek(); // return 1
myQueue.pop(); // return 1, queue is [2]
myQueue.empty(); // return false

Constraints:

- 1 <= x <= 9
- At most 100 calls will be made to push, pop, peek, and empty.
- All the calls to pop and peek are valid.

Solution:-

```
class MyQueue {
    Stack<Integer> st1;
    Stack<Integer>st2;

    public MyQueue() {
        st1=new Stack();
        st2=new Stack();
```

```
    }

    public void push(int x) {
while(!st2.isEmpty())
        {
            st1.push(st2.pop());
        }
        st2.push(x);

while(!st1.isEmpty())
        {
            st2.push(st1.pop());
        }
    }

    public int pop() {
        return st2.pop();

    }

    public int peek() {
        return st2.peek();
    }

    public boolean empty() {
        return st2.isEmpty();

    }
}

/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueueobj = new MyQueue();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.peek();
 * boolean param_4 = obj.empty();
 */
```

Q:-7 EASY Implementation of Queue using arrays:-

```
package com.ds;
public class NormalQueueDemo
{
int front,rear,size,queue[];
public NormalQueueDemo()
{

front = -1;
rear = -1;
size = 5;
queue = new int[size];
}
void insert(int value)
{
if(rear==size)
{
System.out.println("NQ is full");
return;
}
if(front==rear)
{
front=rear=0;
}
queue[rear++]=value;
}
void delete(){
if(front==rear){
System.out.println("NQ is empty");
return;
}
System.out.println("Deleted object is: "+queue[front]);
front++;
if(front==rear)
{
front=rear=-1;
}
}
void display(){
if(front==rear)
{
System.out.println("Normal Queue is empty");
return;
}
for(int i=front;i<rear;i++)
{
System.out.print(queue[i]+" ");
}
System.out.println();
}
```

```
public static void main(String[] args)
{
NormalQueueDemonqd = new NormalQueueDemo();
nqd.insert(111);
nqd.insert(222);
nqd.insert(333);
nqd.insert(444);
nqd.insert(555);
nqd.display();
nqd.delete();
nqd.display();
}
}
```

Result:-

```
111 222 333 444 555
Deleted object is: 111
222 333 444 555
```

Q:-8 EASY implementation of Queue using linked list:-

```
package com.ds;
public class NormalQueueList
{
Node front,rear;
int size;
class Node
{
int data;
Node next;
Node(int data,Node next)
{
this.data = data;
this.next = next;
}
}
public NormalQueueList()
{
front = null;
rear = null;
size = 0;
}
void display()
{
if(size==0)
{
System.out.println("q is empty");
return;
}
Node temp = front;
while(temp!=null)
{
System.out.print(temp.data+" ");
temp = temp.next;
}
System.out.println();
}
void insert(int value)
{
Node newNode = new Node(value,null);
if(front==null && rear==null)
{
front = newNode;
rear = newNode;
}
else
{
rear.next = newNode;
rear = newNode;
```

```
}
size++;
}
void delete()
{
if(size==0)
{
System.out.println("Queue is empty");
return;
}
System.out.println("Deleted item is: "+front.data);
front = front.next;
size--;
}
public static void main(String[] args)
{
NormalQueueListnql = new NormalQueueList();
nql.display();//q is empty
nql.insert(111);
nql.insert(222);
nql.insert(333);
nql.display();//111, 222, 333
nql.delete();//111
nql.display();//222, 333
nql.delete();//222
nql.display();//333
nql.delete();
nql.display();//q is empty
}
}
```

Result:-

```
Queue is empty
111 222 333
Deleted item is: 111
222 333
Deleted item is: 222
333
Deleted item is: 333
Queue is empty
```

Q:-9 EASY implementation of Circular Queue using arrays:-

```
package com.ds;
public class CircularQueueArray
{int front,rear,size,count,circularQueue[];
CircularQueueArray()
{
front = -1;
rear = -1;
count = 0;
size = 5;
circularQueue = new int[size];
}
void insert(int value)
{
if(count==size)
{
System.out.println("Queue is full");
return;
}
if(front==-1)
front=rear=0;
else
rear = (rear+1)%size;
circularQueue[rear] = value;
count++;
}
void delete(){
if(count==0){
System.out.println("queue is empty");
return;
}
System.out.println("Deleted item is: "+circularQueue[front]);
if(front==rear)
front=rear=-1;
else
front = (front+1)%size;
count--;
}
void display(){
if(count==0){
System.out.println("queue is empty");
return;
}
int i=front;
if(front<=rear){
while(i<=rear)
System.out.print(circularQueue[i++]+" ");
}
else{
while(i!=rear){
```

```
System.out.print(circularQueue[i]+" ");
i=(i+1)%size;
}
System.out.print(circularQueue[i]);
}
System.out.println();
}
public static void main(String[] args)
{
CircularQueueArraycqa = new CircularQueueArray();
cqa.insert(111);
cqa.insert(222);
cqa.insert(333);
cqa.insert(444);
cqa.insert(555);
cqa.display();
cqa.insert(666);
cqa.delete();//111
cqa.display();
cqa.insert(666);
cqa.display();
}
}
```

Result:-

```
111 222 333 444 555
Queue is full
Deleted item is: 111
222 333 444 555
222 333 444 555 666
```

Q:-10 EASY Implementation of Circular Queue using linked list:-

```
package com.ds;
public class CircularQueueList
{
Node front,rear;
class Node
{
int data;
Node next;
Node(int data,Node next)
{
this.data = data;
this.next = next;
}
}
void insert(int data)
{
Node newNode = new Node(data,null);
if(front==null)
{
front = newNode;
}
else
{
rear.next = newNode;
}
rear=newNode;
rear.next=front;
}
void delete(){
if(front==null){
System.out.println("queue is empty");
return;
}
System.out.println("Deleted item is: "+front.data);
if(front==rear){
front=null;
rear=null;
}
else{
front = front.next;
rear.next = front;
}
}
void display(){
Node temp = front;
if(temp==null){
System.out.println("queue is empty");
return;
```

```
}
while(temp.next!=front){
System.out.print(temp.data+" ");
temp=temp.next;
}
System.out.println(temp.data);
}
public static void main(String[] args)
{
CircularQueueListcql = new CircularQueueList();
cql.display();
cql.insert(111);
cql.insert(222);
cql.insert(333);
cql.insert(444);
cql.insert(555);
cql.display();
cql.delete();
cql.delete();
cql.display();
}
}
```

Result:-

```
queue is empty
111 222 333 444 555
Deleted item is: 111
Deleted item is: 222
333 444 555
```

Q:-11 MEDIUM Implementation of deque using arrays.

```
package com.ds;
public class DequeArray
{
int deque[],front,rear,size;
public DequeArray()
{
front = -1;
rear = -1;
size = 5;
deque = new int[size];
}
void insertAtFront(int value){
if((front==0 && rear==size-1)||(front==rear+1)){
System.out.println("Q is full");
return;
}
if(front==-1)
{
front=rear=0;
}
else if(front==0)
{
front=size-1;
}
else
{
front=front-1;
}
deque[front]=value;
}
void insertAtRear(int value){
if((front==0 && rear==size-1)||(front==rear+1)){
System.out.println("Q is full");
return;
}
if(front==-1)
{
front=rear=0;
}
else if(rear==size-1)
{
rear=0;
}
else
{
rear=rear+1;
}
deque[rear]=value;
}
```

```
void deleteFromFront(){
if(front==-1)
{
System.out.println("DQ is empty");
return;
}
System.out.println("deleted item is: "+deque[front]);
if(front==rear)
{
front=rear=-1;
}
else
{
if(front==size-1)
{
front=0;
}
else
{
front=front+1;
}
}
}
void deleteFromRear(){
if(front==-1){
System.out.println("dq is empty");
return;
}
System.out.println("Deleted object : "+deque[rear]);
if(front==rear)
{
front=rear=-1;
}
else
{
if(rear==0)
{
rear=size-1;
}
else
{
rear=rear-1;
}
}
}
void display(){
if(front==-1){
System.out.println("dq is empty");
return;
}
```

```
int left=front,right=rear;
if(left<=right){
while(left<=right)
{
System.out.print(deque[left++]+" ");
}
}
else{
while(left<=size-1)
System.out.print(deque[left++]+" ");
left=0;
while(left<=right)
System.out.print(deque[left++]+" ");
}
System.out.println();
}
public static void main(String[] args)
{
DequeArraydq = new DequeArray();
dq.insertAtRear(111);
dq.insertAtRear(222);
dq.insertAtRear(333);
dq.display();//111 222 333
dq.insertAtFront(777);
dq.insertAtFront(888);
dq.insertAtFront(999);//queue is full
dq.display();//888 777 111 222 333
dq.deleteFromFront();//888
dq.display();//777 111 222 333
dq.deleteFromRear();//333
dq.display();//777 111 222
}
}
```

Result:-

```
888 777 111 222 333
deleted item is: 888
777 111 222 333
Deleted object : 333
777 111 222
```

Q:-12 MEDIUM Implementation of deque using linked list:-

```
package com.ds;
public class DequeList
{
Node front,rear;
int size;//number of element in deque
DequeList(){
front = null;
rear = null;
size = 0;
}
class Node{
int data;
Node next;
Node(int data,Nodenext){
this.data = data;
this.next = next;
size++;
}
}
void insertAtFront(int value){
Node newNode = new Node(value,null);
if(front==null){
front = newNode;
rear = newNode;
return;
}
newNode.next = front;
front = newNode;
}
void insertAtRear(int value){
Node newNode = new Node(value,null);
if(front==null){
front = newNode;
rear = newNode;
return;
}
rear.next = newNode;
rear = newNode;
}
void deleteAtFront(){
if(front==null){
System.out.println("dq is empty");
return;
}
System.out.println("deleted obj is: "+front.data);
front = front.next;
size--;
}
```

```
void deleteAtRear(){
if(front==null){
System.out.println("dq is empty");
return;
}
System.out.println("deleted obj is: "+rear.data);
size--;
if(front==rear){
front=null;
rear=null;
return;
}
Node temp = front;
while(temp.next!=rear)
temp = temp.next;
rear = temp;
rear.next=null;
}
void display(){
if(front==null){
System.out.println("dq is empty");
return;
}
Node temp = front;
while(temp!=null){
System.out.print(temp.data+" ");
temp=temp.next;
}
System.out.println();
}
public static void main(String[] args)
{
DequeListdq = new DequeList();
dq.insertAtFront(333);
dq.insertAtFront(222);
dq.insertAtFront(111);
dq.display();//111 222 333
dq.insertAtRear(444);
dq.insertAtRear(555);
dq.insertAtFront(999);
dq.display();//999 111 222 333 444 555
dq.deleteAtFront();
dq.display();//111 222 333 444 555
dq.deleteAtRear();
dq.display();//111 222 333 444
}
}
```

Result:-

999 111 222 333 444 555

```
deleted obj is: 999
111 222 333 444 555
deleted obj is: 555
111 222 333 444
```