**Assignment 1)**

## 1) Sum of the series

**Recursive program to find the Sum of the series 1 – 1/2 + 1/3 – 1/4 … 1/N Given a positive integer N, the task is to find the sum of the series 1 – (1/2) + (1/3) – (1/4) +…. (1/N) using recursion.**

import java.io.\*;

import java.util.\*;

public class Solution {

public static float sum\_of\_series(int i, int n, float s)

{

if(i>n)

return s;

else

{

if(i%2==0)

s-=(float)1/i;

else

s+=(float)1/i;

return sum\_of\_series(i+1,n,s);

}

}

public static void main(String[] args) {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

Scanner scan=new Scanner(System.in);

int n=scan.nextInt();

float res=sum\_of\_series(1,n,0);

System.out.println(String.format("%.16f",res));

}

}

## 3)Tower-of-Hanoi Problem

## import java.io.\*;

## import java.util.\*;

## public class Solution {

## static void towerOfHanoi(int n, char from\_rod,

## char to\_rod, char aux\_rod)

## {

## if (n == 1)

## {

## System.out.println("Disk 1 moved from "+

## from\_rod+" to "+to\_rod);

## return;

## }

## towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

## System.out.println("Disk "+ n + " moved from " +

## from\_rod +" to " + to\_rod );

## towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

## }

## public static void main(String[] args) {

## /\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

## Scanner sc=new Scanner(System.in);

## int n =sc.nextInt(); // Number of disks

## towerOfHanoi(n, 'A', 'C', 'B');

## }

## }

## 2)Negative Number

**Sample Input 0**

-12, 11, -13, -5, 6, -7, 5, -3, -6

**Sample Output 0**

-12 -13 -5 -7 -3 -6 11 6 5

## import java.io.\*;

## import java.util.\*;

## public class Solution {

## public static void main(String[] args) {

## /\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

## int arr[] = { -12, 11, -13, -5, 6, -7, 5, -3, -6};

## 

## for (int i=0; i<arr.length; i++)

## {

## 

## if(arr[i]>0)

## {

## for (int j=i; j<arr.length; j++)

## {

## if(arr[i]>0 && arr[j]<0)

## {

## int a=arr[i];

## arr[i]=arr[j];

## arr[j]=a;

## }

## }

## }

## }

## 

## for (int i=0; i<arr.length; i++)

## {

## for (int j=0; j<arr.length; j++)

## {

## if (arr[i] >0 && arr[j]>0)

## {

## if(arr[i]>arr[j])

## {

## int temp=arr[i];

## arr[i]=arr[j];

## arr[j]=temp;

## }

## }

## }

## }

## 

## 

## for (int i=0; i<arr.length; i++)

## {

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

## }

## 

## 

## }

## 

## 

## 

## }

## 3)Max element(1 test case fail)

## Given a matrix, the task is to find the maximum element of each row.

## import java.io.\*;

## import java.util.\*;

## public class Solution {

## public static void maxelement(int no\_of\_rows, int[][] arr) {

## int i = 0;

## 

## // Initialize max to 0 at beginning

## // of finding max element of each row

## int max = 0;

## int[] result = new int[no\_of\_rows];

## while (i < no\_of\_rows) {

## for (int j = 0; j < arr[i].length; j++) {

## if (arr[i][j] > max) {

## max = arr[i][j];

## }

## }

## result[i] = max;

## max =0;

## i++;

## 

## }

## printArray(result);

## 

## }

## 

## // Print array element

## private static void printArray(int[] result) {

## for (int i =0; i<result.length;i++) {

## System.out.println(result[i]);

## }

## 

## }

## public static void main(String[] args) {

## /\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

## int[][] arr = new int[][] {{1, 2, 3},{1, 4, 9},{76, 34, 21}};

## // Calling the function

## maxelement(3, arr);

## }

## }

**Assignment 2)**

**1)Pair with sum**

import java.io.\*;

import java.util.\*;

public class Solution {

static void chkPair(int A[], int size, int x) {

for (int i = 0; i < (size - 1); i++) {

for (int j = (i + 1); j < size; j++) {

if (A[i] + A[j] == x) {

System.out.println("Pair Found" +

"(" + A[i] + ", " + A[j] + ")");

System.out.println("Valid pair exists");

}

}

}

System.out.println("Pair not found ");

}

public static void main(String[] args) {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

int A[] = { 5, 2, 6, 8, 1, 9};

int x = 12;

int size = A.length;

chkPair(A, size, x);

}

}

**2) Find duplicates within a range k in an array. Given an array and a positive number k, check whether the array contains any duplicate elements within the range k.If k is more than the array's size, the solution should check for duplicates in the complete array.**

**Input Format**

A[] = { 5, 6, 8, 2, 4, 6, 9 } k = 4

**Constraints**

.

**Output Format**

Output: Duplicates found

(element 6 repeats at distance 4 which is <= k)

import java.io.\*;

import java.util.\*;

public class Solution {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

String s =sc.nextLine();

int key = sc.nextInt();

String[] arr = s.split(", ");

int[] arr2=new int[arr.length-1];

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

{

arr2[i]=Integer.parseInt(arr[i]);

if(arr2[i]==key)

{

System.out.println("Duplicates found");

return;

}

}

System.out.println("Duplicates not found");

}

}

**3) Find the smallest missing element from a sorted array Given a sorted array of distinct non-negative integers, find the smallest missing element in it.**

**Input Format**

Input: A[] = [0, 1, 2, 6, 9, 11, 15]

**Output Format**

Output: The smallest missing element is 3

**Sample Input 0**

0, 1, 2, 6, 9, 11, 15

**Sample Output 0**

The smallest missing element is 3

import java.io.\*;

import java.util.\*;

public class Solution {

public static void main(String[] args) {

Scanner sc =new Scanner(System.in);

String s =sc.nextLine();

String arr[] =s.split(", ");

int[] arr2 =new int[arr.length];

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

{

arr2[i]=Integer.parseInt(arr[i]);

}

for(int i=0;i<arr2.length;i++)

{

if(arr2[i]!=i)

{

System.out.println("The smallest missing element is "+i);

break;

}

}

}

}

**4)Bubble Sort**

import java.io.\*;

import java.util.\*;

public class Solution {

public static void main(String[] args) {

Scanner sc =new Scanner(System.in);

String s =sc.nextLine();

String arr[]=s.split(", ");

int[] arr2=new int[arr.length];

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

{

arr2[i] = Integer.parseInt(arr[i]);

}

for(int i=0;i<arr2.length;i++)

{

for(int j=0 ;j<arr2.length-i-1;j++)

{

if(arr2[j] > arr2[j+1])

{

int temp=arr2[j];

arr2[j]=arr2[j+1];

arr2[j+1]=temp;

}

}

}

System.out.println(Arrays.toString(arr2));

}

}

5**) Reverse a doubly linked list. To reverse a doubly linked list using iteration or recursion.**

import java.io.\*;

import java.util.\*;

public class Solution {

static class Node

{

int data;

Node next, prev;

};

// function to get a new node

static Node getNode(int data)

{

// allocate space

Node new\_node = new Node();

new\_node.data = data;

new\_node.next = new\_node.prev = null;

return new\_node;

}

// function to insert a node at the beginning

// of the Doubly Linked List

static Node push(Node head\_ref, Node new\_node)

{

// since we are adding at the beginning,

// prev is always null

new\_node.prev = null;

// link the old list off the new node

new\_node.next = (head\_ref);

// change prev of head node to new node

if ((head\_ref) != null)

(head\_ref).prev = new\_node;

// move the head to point to the new node

(head\_ref) = new\_node;

return head\_ref;

}

// function to reverse a doubly linked list

static Node Reverse(Node node)

{

// If empty list, return

if (node == null)

return null;

// Otherwise, swap the next and prev

Node temp = node.next;

node.next = node.prev;

node.prev = temp;

// If the prev is now null, the list

// has been fully reversed

if (node.prev == null)

return node;

// Otherwise, keep going

return Reverse(node.prev);

}

// Function to print nodes in a given doubly

// linked list

static void printList(Node head)

{

while (head != null)

{

System.out.print(head.data +" -> ");

head = head.next;

}

System.out.print("null");

}

public static void main(String[] args) {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

Node head = null;

// Create doubly linked: 10<.8<.4<.2 /

head = push(head, getNode(1));

head = push(head, getNode(2));

head = push(head, getNode(3));

head = push(head, getNode(4));

head = push(head, getNode(5));

// Reverse doubly linked list

head = Reverse(head);

printList(head);

}

}

**6)Linked List Representation in Java**

import java.io.\*;

import java.util.\*;

class Solution

{

Node head;

static class Node

{

Node next;

int data;

Node(int x)

{

data=x;

}

}

void append(int key)

{

Node newNode = new Node(key);

Node n =head;

if(head==null)

{

head=newNode;

return ;

}

else

while(n.next!=null)

{

n=n.next;

}

n.next =newNode;

}

void display()

{

Node n =head;

while(n!=null)

{

System.out.print(n.data+" -> ");

n=n.next;

}

System.out.print("null");

}

public static void main(String[] args) {

Solution n1 =new Solution();

n1.append(1);

n1.append(2);

n1.append(3);

n1.append(4);

n1.display();

}

}

**7) Maximum Element**

You have an empty sequence, and you will be given  queries. Each query is one of these three types:

1 x -Push the element x into the stack.

2 -Delete the element present at the top of the stack.

3 -Print the maximum element in the stack.

import java.io.\*;

import java.util.\*;

public class Solution {

private static void getMaxElementFromStack()

{

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

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

Scanner sc = new Scanner(System.in);

int N = Integer.parseInt(sc.nextLine());

int temp = 0;

while(sc.hasNext())

{

String type[] = sc.nextLine().split(" ");

switch(type[0])

{

case "1":

temp = Integer.parseInt(type[1]);

stack.push(temp);

if(onlyMaxs.isEmpty() || onlyMaxs.peek() <= temp)

onlyMaxs.push(temp);

break;

case "2":

temp = stack.pop();

if(temp == onlyMaxs.peek())

onlyMaxs.pop();

break;

case "3":

System.out.println(onlyMaxs.peek());

}

N--;

}

while(N-- > 0)

System.out.println(onlyMaxs.peek());

}

public static void main(String[] args) {

getMaxElementFromStack();

}

}

**9)Balanced Bracket**

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int n = in.nextInt();

//int arr[] = new int[n];

for(int arr\_i=0; arr\_i < n; arr\_i++){

Stack st = new Stack();

//arr[arr\_i] = in.nextInt();

String ipSeq = in.next();

//System.out.println(ipSeq);

boolean match = true;

for(int ind=0; ind<ipSeq.length(); ind++){

char ch = ipSeq.charAt(ind);

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

st.push(ch);

}else if(st.isEmpty()){

match = false;

break;

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

if('('!=(char)st.pop()){

match = false;

break;

}

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

if('{'!=(char)st.pop()){

match = false;

break;

}

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

if('['!=(char)st.pop()){

match = false;

break;

}

}

}

if(match){

if(!st.isEmpty()){

System.out.println("NO");

}else{

System.out.println("YES");

}

}else{

System.out.println("NO");

}

}

}

}

**10)Balanced Paranthesis//only one test case pass**

import java.io.\*;

import java.util.\*;

public class Solution {

static int minParentheses(String p)

{

// maintain balance of string

int bal = 0;

int ans = 0;

for (int i = 0; i < p.length(); ++i) {

bal += p.charAt(i) == '(' ? 1 : -1;

// It is guaranteed bal >= -1

if (bal == -1) {

ans += 1;

bal += 1;

}

}

return bal + ans;

}

public static void main(String[] args) {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

String p = "()))";

// Function to print required answer

System.out.println(minParentheses(p));

}

}