//BST

class Node {

    int data;

    Node left,right;

    Node(int n){

        data = n;

    }

}

class BST {

    Node root;

    BST(int n){

        root = new Node(n);

    }

    void preOrder(){

        preOrderRecursively(root);

    }

    void preOrderRecursively(Node root){

        if(root==null){

            return;

        }

        System.out.println(root.data);

        preOrderRecursively(root.left);

        preOrderRecursively(root.right);

    }

     boolean isValid(){

        return isValidRec(root);

    }

    boolean isValidRec(Node root){

        if(root==null){

            return true;

        }

        if(root.left !=null ){

            if(root.data<root.left.data) {

                return false;

            }

        }

        if(root.right !=null ){

            if(root.data>root.right.data) {

                return false;

            }

        }

        return isValidRec(root.left) && isValidRec(root.right);

    }

}

public class Main {

    public static void main(String args[]){

        BST bst = new BST(10);

        bst.root.left = new Node(3);

        bst.root.right = new Node(15);

        bst.root.left.right = new Node(7);

        bst.root.left.right.right = new Node(8);

        System.out.println(bst.isValid());

    }

}

//BST

class Node {

    int data;

    Node left,right;

    Node(int n){

        data = n;

    }

}

class BST {

    Node root;

    BST(int n){

        root = new Node(n);

    }

    void preOrder(){

        preOrderRecursively(root);

    }

    void preOrderRecursively(Node root){

        if(root==null){

            return;

        }

        System.out.println(root.data);

        preOrderRecursively(root.left);

        preOrderRecursively(root.right);

    }

/\*    boolean isValid(){

    }\*/

}

public class Main {

    public static void main(String args[]){

        BST bst = new BST(10);

        bst.root.left = new Node(15);

        bst.root.right = new Node(3);

        bst.root.left.left = new Node(7);

        bst.root.left.right = new Node(8);

        bst.preOrder();

    }

}

//numbers

class Node {

    int data;

    Node next;

    Node(int n){

        data = n;

    }

}

class LinkedList {

    Node root;

    LinkedList(){

        root = null;

    }

    void addNode(int n){

        if(root==null){

            root = new Node(n);

            return;

        }

        Node iterator = root;

        while(iterator.next!=null){

            iterator = iterator.next;

        }

        iterator.next = new Node(n);

    }

    void show(){

        showRec(root);

    }

    void showRec(Node root) {

        if(root==null){

            return;

        }

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

        showRec(root.next);

    }

    void genNum(){

        Node iterator = root;

        int result = 0,k,i=0;

        while(iterator!=null){

            k = iterator.data;

            result = result + k \* (int)Math.pow(10,i);

            i++;

            iterator =  iterator.next;

        }

        System.out.println();

        System.out.println(result);

    }

}

public class Main {

    public static void main(String args[]){

        LinkedList ll = new LinkedList();

        ll.addNode(9);

        ll.addNode(3);

        ll.addNode(7);

        ll.addNode(6);

        ll.show();

        ll.genNum();

    }

}

class Team {

    String name;

    int age1, age2;

    Team(String n, int a1, int a2) {

        name = n;

        age1 = a1;

        age2 = a2;

    }

    boolean lessThan(Team t) {

        if(Math.abs(this.age2-this.age1) < Math.abs(t.age2-t.age1)) {

            return true;

        }

        else {

            return false;

        }

    }

}

public class Main {

    public static void main(String args[]) {

        Team[] teams = new Team[10];

        teams[0] =  new Team("Team 1",17,20);

        teams[1] =  new Team("Team 2",27,16);

        teams[2] =  new Team("Team 3",19,16);

        teams[3] =  new Team("Team 4",21,23);

        teams[4] =  new Team("Team 5",19,46);

        teams[5] =  new Team("Team 6",39,16);

        teams[6] =  new Team("Team 7",28,16);

        teams[7] =  new Team("Team 8",11,11);

        teams[8] =  new Team("Team 9",15,12);

        teams[9] =  new Team("Team 10",15,12);

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

            for(int j=0;j<teams.length-1;j++) {

                if(teams[j].lessThan(teams[j+1])){

                    Team temp = teams[j];

                    teams[j] = teams[j+1];

                    teams[j+1] = temp;

                }

            }

        }

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

            System.out.println(teams[i].name);

        }

    }

}

class Team {

    String name;

    int age1, age2;

    Team(String n, int a1, int a2) {

        name = n;

        age1 = a1;

        age2 = a2;

    }

}

public class Main {

    static boolean lessThan(Team t1, Team t2) {

        if(Math.abs(t1.age2-t1.age1) < Math.abs(t2.age2-t2.age1)) {

            return true;

        }

        else {

            return false;

        }

    }

    public static void main(String args[]) {

        Team[] teams = new Team[10];

        teams[0] =  new Team("Team 1",17,20);

        teams[1] =  new Team("Team 2",27,16);

        teams[2] =  new Team("Team 3",19,16);

        teams[3] =  new Team("Team 4",21,23);

        teams[4] =  new Team("Team 5",19,46);

        teams[5] =  new Team("Team 6",39,16);

        teams[6] =  new Team("Team 7",28,16);

        teams[7] =  new Team("Team 8",11,11);

        teams[8] =  new Team("Team 9",15,12);

        teams[9] =  new Team("Team 10",15,12);

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

            for(int j=0;j<teams.length-1;j++) {

                if(lessThan(teams[j],teams[j+1])){

                    Team temp = teams[j];

                    teams[j] = teams[j+1];

                    teams[j+1] = temp;

                }

            }

        }

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

            System.out.println(teams[i].name);

        }

    }

}

//depth of a binary tree

class Node {

    int data;

    Node left, right;

    Node(int n){

        data = n;

    }

}

public class Main {

    static int depthRecursive(Node root) {

        if(root==null) {

            return 0;

        }

        int leftDepth = depthRecursive(root.left);

        int rightDepth = depthRecursive(root.right);

        if(leftDepth>rightDepth) {

            return leftDepth + 1;

        }

        else {

            return rightDepth + 1;

        }

    }

    public static void main(String args[]) {

        Node root = new Node(19);

        root.left = new Node(10);

        root.right = new Node(10);

        root.left.left = new Node(12);

        root.left.right = new Node(8);

        root.left.left.right = new Node(3);

        root.left.left.right.left = new Node(13);

        int depth = depthRecursive(root);

        System.out.println(depth);

    }

}

//sorting a binary array

public class Main {

    public static void main(String args[]) {

      int[] arr = {1,0,0,1,0,0,1,1,0,1,1};

      int left = 0;

      int right = arr.length - 1;

      while(left<right){

          while(arr[left]==0 && left<right){

              left++;

          }

          while(arr[right]==1 && left<right){

              right--;

          }

          if(left<right) {

              arr[left]=0;

              arr[right]=1;

              left++;

              right--;

          }

      }

      for(int i=0;i<arr.length;i++){

          System.out.println(arr[i]);

      }

    }

}

// first and last occurrence of an element in a sorted array

public class FirstLastOccurrence {

public static int[] findFirstLastOccurrence(int[] arr, int K) {

int first = -1;

int last = -1;

int start = 0;

int end = arr.length - 1;

while (start <= end) {

int mid = (start + end) / 2;

if (arr[mid] == K) {

first = mid;

end = mid - 1;

} else if (arr[mid] < K) {

start = mid + 1;

} else {

end = mid - 1;

}

}

start = 0;

end = arr.length - 1;

while (start <= end) {

int mid = (start + end) / 2;

if (arr[mid] == K) {

last = mid;

start = mid + 1;

} else if (arr[mid] < K) {

start = mid + 1;

} else {

end = mid - 1;

}

}

return new int[]{first, last};

}

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 4, 4, 5, 6, 6, 7};

int K = 4;

int[] result = findFirstLastOccurrence(arr, K);

System.out.println("First Occurrence: " + result[0]);

System.out.println("Last Occurrence: " + result[1]);

}

}

// first and last occurrence of an element in a sorted array

public class Main {

    static int findFirst(int[] arr, int target){

     int left=0;

     int end = arr.length - 1;

     int result = -1;

     while(left<=end){

         int mid = left + (end-left)/2;

         if(arr[mid]==target){

             result = mid;

             end = mid - 1;

         }

         else if(arr[mid]<target) {

             left = mid+1;

         }

         else {

             right = mid - 1;

         }

     }

     return result;

    }

    static int findSecond(int[] arr, int target){

     int left=0;

     int end = arr.length - 1;

     int result = -1;

     while(left<=end){

         int mid = left + (end-left)/2;

         if(arr[mid]==target){

             result = mid;

             left = mid + 1;

         }

         else if(arr[mid]<target) {

             left = mid+1;

         }

         else {

             right = mid - 1;

         }

     }

     return result;

    }

    public static void main(String args[]) {

    }

}

//checking if a graph is cyclic

public class Main {

    static boolean isCyclic(int[][] graph){

        int n = graph.length;

        boolean[] visited = new boolean[n];

        boolean[] dfsStack = new boolean[n];

        for(int i=0; i<n;i++){

            if(!visited[i] && hasCycle(graph,visited,dfsStack,i)){

                return true;

            }

        }

        return false;

    }

    static boolean hasCycle(int[][] graph,boolean[] visited,boolean[] dfsStack, int node){

        visited[node] = true;

        dfsStack[node] = true;

        for(int i=0;i<graph[node].length;i++){

            if(graph[node][i]==1){

                if(!visited[i]){

                    if(hasCycle(graph,visited,dfsStack,i)){

                        return true;

                    }

                }

                else if(dfsStack[i]){

                    return true;

                }

            }

        }

        dfsStack[node] = false;

        return false;

    }

    public static void main(String args[]) {

      int v = 4;

      int[][] adj = new int[v][v];

      adj[0][1] = 1;

      adj[1][2] = 1;

      adj[0][3] = 1;

      System.out.println(isCyclic(adj));

    }

}

//Imposter Syndrome

class StudentNode {

    String name;

    int score;

    StudentNode nextStudent;

    StudentNode(String name, int score){

        this.name=name;

        this.score = score;

    }

}

class StudentList {

    StudentNode head;

    StudentList(String name, int score){

        head = new StudentNode(name,score);

    }

    void insertStudent(String name, int score){

        StudentNode iterator = head;

        while(iterator.nextStudent!=null){

            iterator = iterator.nextStudent;

        }

        iterator.nextStudent = new StudentNode(name,score);

    }

    void findImposters(){

        StudentNode iterator = head;

        StudentNode prevStudent = null;

        if(iterator.score<iterator.nextStudent.score){

            System.out.println(iterator.name);

        }

        prevStudent = iterator;

        iterator = iterator.nextStudent;

        while(iterator.nextStudent!=null){

            if(iterator.score<iterator.nextStudent.score && iterator.score<prevStudent.score){

                System.out.println(iterator.name);

            }

            prevStudent = iterator;

            iterator = iterator.nextStudent;

        }

        if(prevStudent.score > iterator.score){

            System.out.println(iterator.name);

        }

    }

}

public class Main {

    public static void main(String args[]) {

        StudentList list = new StudentList("A",80);

        list.insertStudent("B",85);

        list.insertStudent("C",70);

        list.insertStudent("D",90);

        list.insertStudent("E",87);

        list.insertStudent("F",95);

        list.insertStudent("G",94);

        list.findImposters();

    }

}