Day-8 (Exercise):

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q1.Write a program to insert an element at the end of the linked list. Get the values continuously from the user until the user enters a negative value.

Format:

Input:

The input should be numbers (until user enters a negative value).

Output:

The output should be the list elements in separate lines.

Example:

Input:

1

2

3

4

-1

Output:

1

2

3

4

Case 1

Input (stdin)

1

2

3

4

-1

Output (stdout)

1

2

3

4

Case 2

Input (stdin)

1

10

2

20

3

-1

Output (stdout)

1

10

2

20

3

Solution:

import java.util.LinkedList;

import java.util.Scanner;

public class InsertAtEndOfLinkedList {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

LinkedList<Integer> linkedList = new LinkedList<>();

System.out.println("Enter numbers (negative to stop):");

while (true) {

int num = scanner.nextInt();

if (num < 0) {

break;

}

linkedList.addLast(num);

}

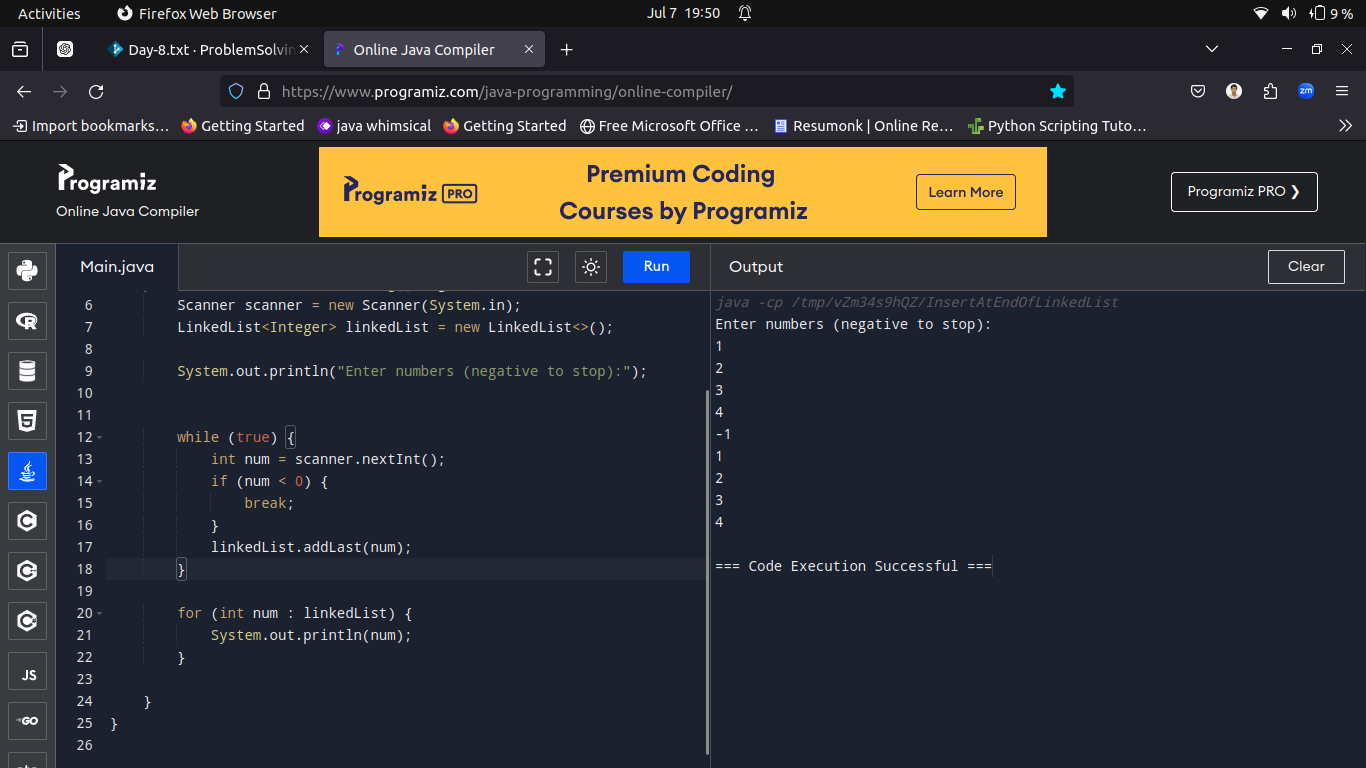
for (int num : linkedList) {

System.out.println(num);

}

}

}



-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q2.Write a program to delete an element at the beginning of the linked list. Get the values continuously from the user until the user enters a negative value.

Format:

Input:

The input should be numbers (until user enters a negative value).

Output:

The output should be the list of numbers in separate lines.

Example:

Input:

1

2

3

4

-1

Output:

2

3

4

Case 1

Case 2

Case 3

Input (stdin)

1

2

3

4

-10

Output (stdout)

2

3

4

Input (stdin)

5

50

6

60

-10

Output (stdout)

50

6

60

Input (stdin)

10

20

30

40

50

60

-1

Output (stdout)

20

30

40

50

60

Code Solution:

import java.util.LinkedList;

import java.util.Scanner;

public class DeleteAtBeginningOfLinkedList {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

LinkedList<Integer> linkedList = new LinkedList<>();

System.out.println("Enter numbers (negative to stop):");

while (true) {

int num = scanner.nextInt();

if (num < 0) {

linkedList.removeFirst();

break;

}

linkedList.addLast(num);

}

System.out.println("Output:");

for (int num : linkedList) {

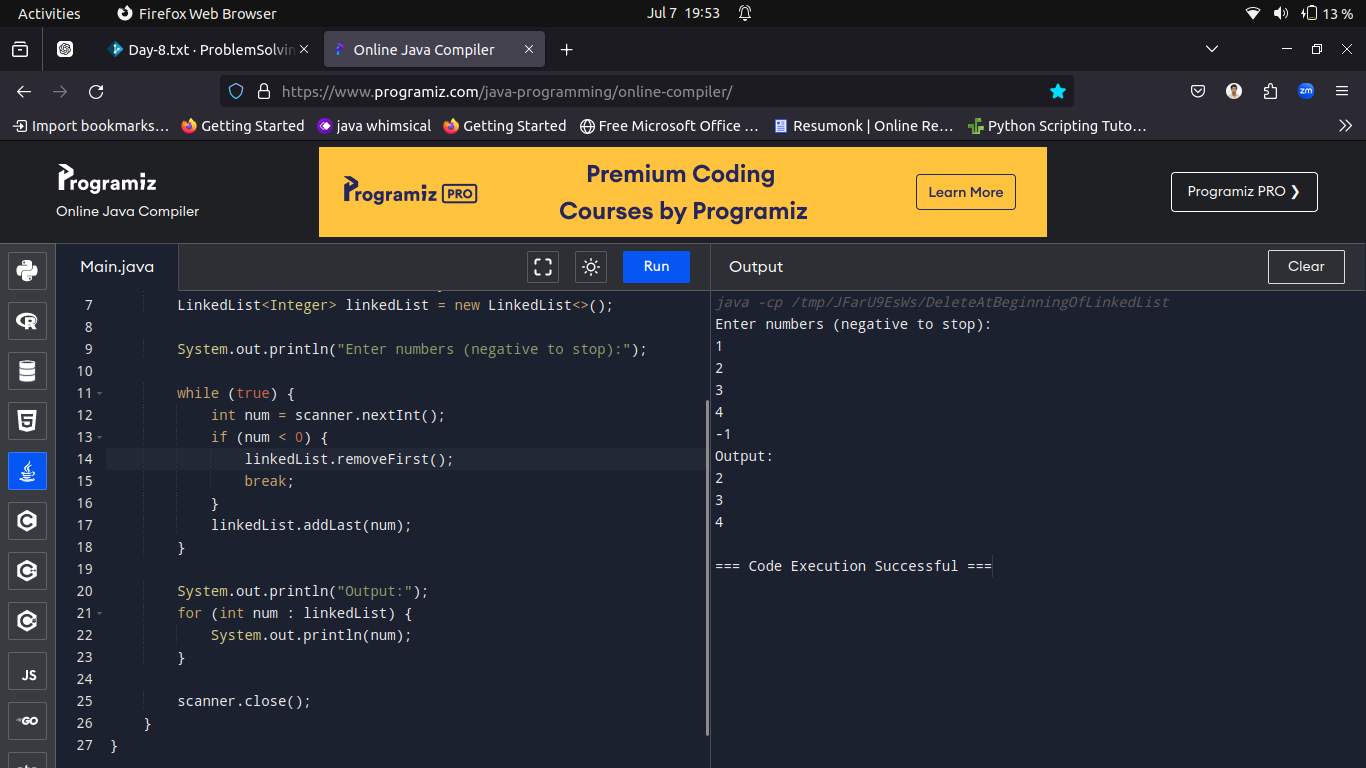
System.out.println(num);

}

scanner.close();

}

}



-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q3.Stack using linked list - Insertion

Write a Program to implement the stack using a linked list.

Example:

Input:

1

2

3

4

-1

Output:

4 3 2 1

Case 1

Case 2

Input (stdin)

1

2

3

4

-1

Output (stdout)

4 3 2 1

Input (stdin)

1

2

3

-1

Output (stdout)

3 2 1

Solution:

import java.util.Scanner;

import java.util.Stack;

public class StackUsingJavaUtilStack {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

System.out.println("Enter numbers (negative to stop):");

while (true) {

int num = scanner.nextInt();

if (num < 0) {

break;

}

stack.push(num);

}

System.out.println("Elements in reverse order:");

while (!stack.isEmpty()) {

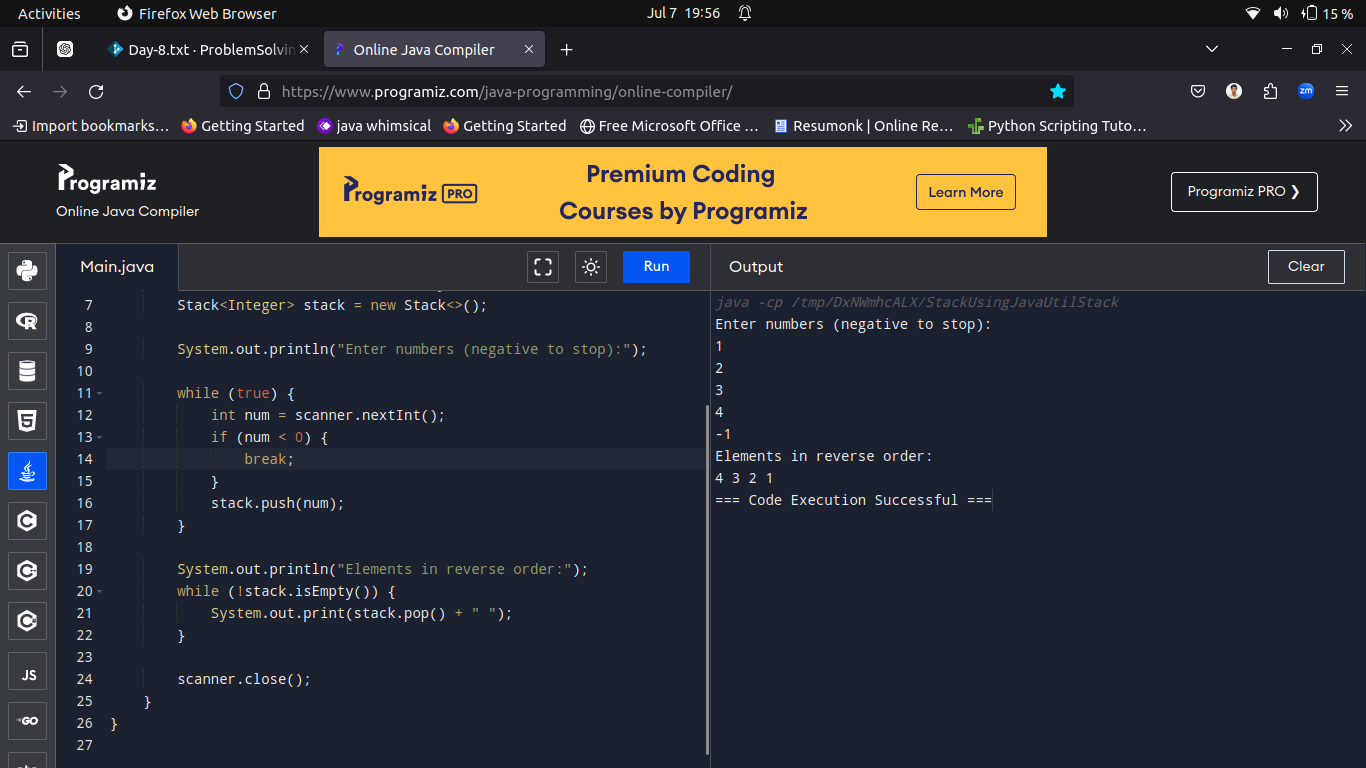
System.out.print(stack.pop() + " ");

}

scanner.close();

}

}



-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q4.Write a program to reverse the order of a queue. If the queue doesn't have a value, then print "Queue is empty".

Format:

Input:

The input consists of a list of integers, a negative value indicates the end of a queue.

Output:

Print the reversed queue.

Example:

Input:

1

2

3

4

-1

Output:

Before reversing:

1 2 3 4

After reversing:

4 3 2 1

Case 1

Case 2

Input (stdin)

1

2

3

4

-1

Output (stdout)

Before reversing:

1 2 3 4

After reversing:

4 3 2 1

Input (stdin)

1

2

-1

Output (stdout)

Before reversing:

1 2

After reversing:

2 1

Solution:

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

public class ReverseQueueSingleQueue {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

System.out.println("Enter numbers (negative to stop):");

while (true) {

int num = scanner.nextInt();

if (num < 0) {

break;

}

queue.add(num);

}

System.out.print("Before reversing:\n");

printQueue(queue);

reverseQueue(queue);

System.out.print("After reversing:\n");

printQueue(queue);

scanner.close();

}

private static void reverseQueue(Queue<Integer> queue) {

if (queue.isEmpty()) {

return;

}

int front = queue.remove();

reverseQueue(queue);

queue.add(front);

}

private static void printQueue(Queue<Integer> queue) {

if (queue.isEmpty()) {

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

return;

}

for (int num : queue) {

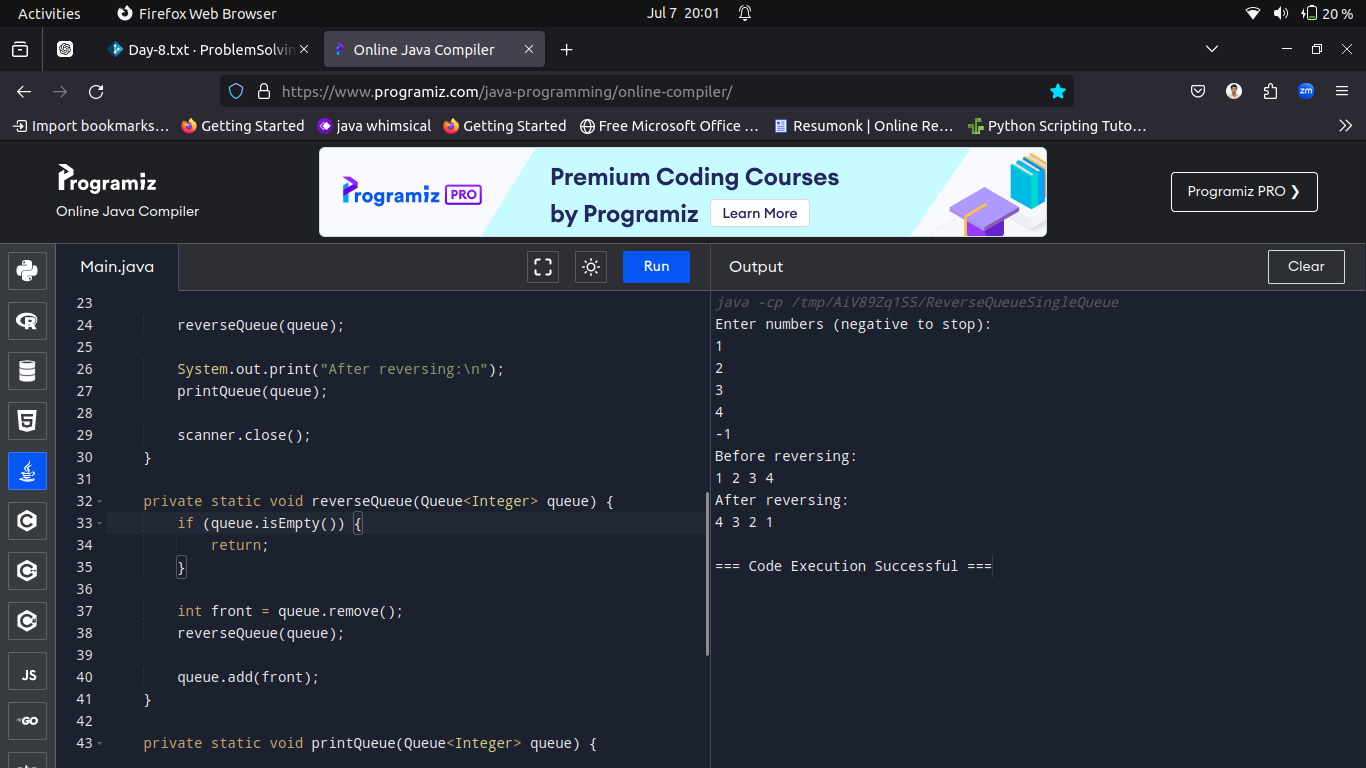
System.out.print(num + " ");

}

System.out.println();

}

}



-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q5.Given a string of balanced expression. Find whether it contains a redundant parenthesis or not. The redundant parenthesis is the expression surrounded by unnecessary or multiple brackets. Print ‘Yes’ if it's redundant, else ‘No’.

Format:

Input:

The input consists of expression.

Output:

The output consists of the result.

Example:

Input:

((a+b))

Output:

Yes

Case 1

Input (stdin)

((a+b))

Output (stdout)

Yes

Case 2

Input (stdin)

(a+b\*(c-d))

Output (stdout)

No

Solution:

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q6.Write a program to sum up all the elements in a tree.

Example:

Input:

6

3

1

4

2

-1

Output:

Sum of all nodes are 16

Case 1

Case 2

Input (stdin)

6

3

1

4

2

-1

Output (stdout)

Sum of all nodes are 16

Input (stdin)

1

2

3

4

5

-1

Output (stdout)

Sum of all nodes are 15

Solution:

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q7.

Odd occurring element

Given an array of integers where every element appears an even number of times except one element, which appears an odd number of times. Write a program to find that odd occurring element in O(log n) time. The equal elements must appear in pairs in the array, but there cannot be more than two consecutive occurrences of an element.

Example 1:

Input:

3

2 3 2

Output:

2

Explanation:

It doesn't have equal elements appear in pairs

Example 2:

Input:

7

1 1 2 2 2 3 3

Output:

2

Explanation:

It contains three consecutive instances of an element.

Example 3:

Input:

5

2 2 3 1 1

Output:

3

Explanation:

It contains two consecutive instances of two elements and one odd number of times.

Case 1

Input (stdin)

5

2 2 3 1 1

Output (stdout)

3

Case 2

Input (stdin)

11

2 2 3 3 2 2 4 4 3 1 1

Output (stdout)

3

Solution:

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q8.Write a program to implement a Binary searching algorithm for searching an element in an array.

Example:

Input:

4

23

34

43

54

23

Output:

23 is present at location 1

Case 1

Case 2

Input (stdin)

6

11

15

26

38

9

10

20

Output (stdout)

Not found

Input (stdin)

4

23

34

43

54

23

Output (stdout)

23 is present at location 1

Solution:

import java.util.Scanner;

public class BinarySearch {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int N = scanner.nextInt();

int[] arr = new int[N];

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

arr[i] = scanner.nextInt();

}

int target = scanner.nextInt();

int index = binarySearch(arr, target);

if (index != -1) {

System.out.println(target + " is present at location " + (index + 1));

} else {

System.out.println(target + " is not present in the array");

}

scanner.close();

}

public static int binarySearch(int[] arr, int target) {

int left = 0;

int right = arr.length - 1;

while (left <= right) {

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

if (arr[mid] == target) {

return mid;

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

left = mid + 1;

} else {

right = mid - 1;

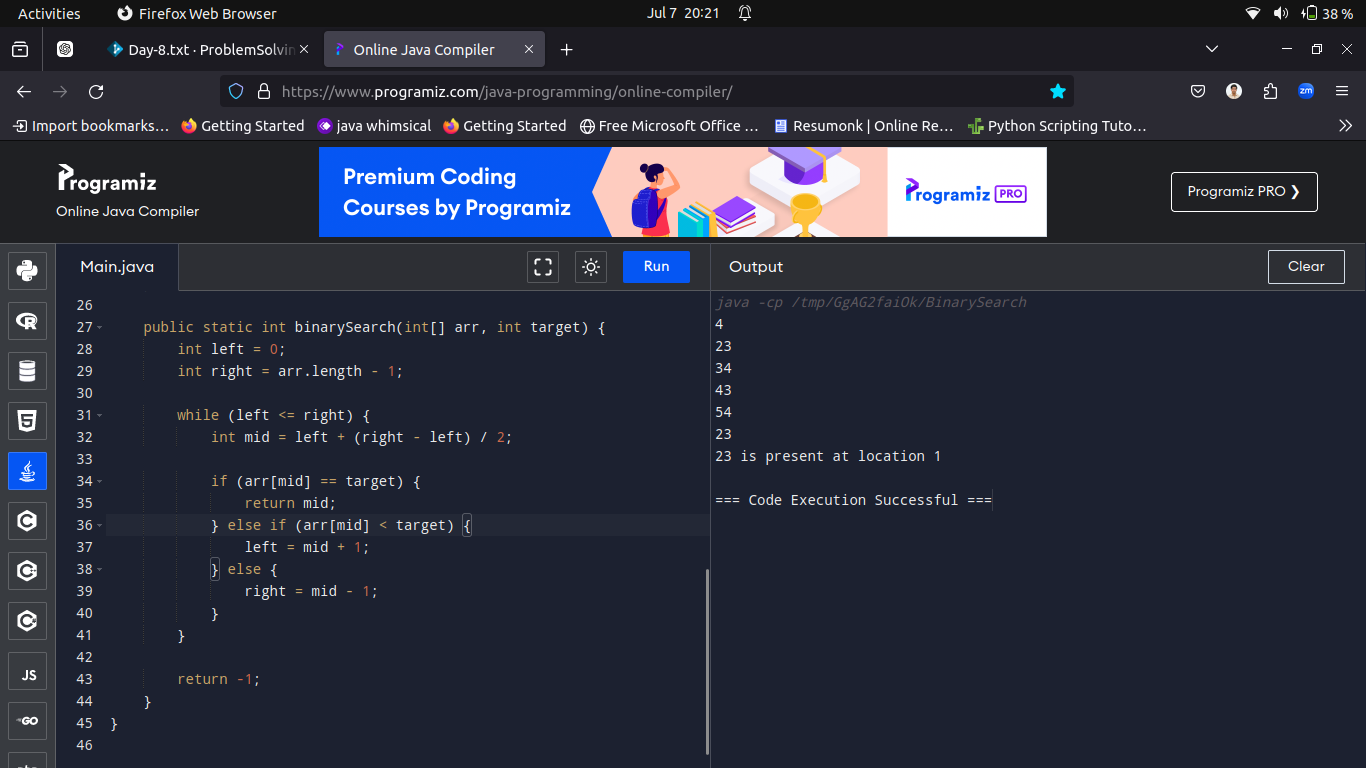
}

}

return -1;

}

}



-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q9.Move all zeroes to end

Write a program to move all zeroes to the end of a given integer.

Format:

Input:

The first line of input contains a number of test cases T. For each test case, the first line of input contains a single integer of 0's and 1's.

Output:

The output contains a single line integer value.

Constraints:

1 ≤ T ≤ 10

1 ≤ N ≤ 109

Example:

Input:

2

1010101

0000111

Output:

1111000

1110000

Case 1

Case 2

Input (stdin)

2

1010101

1000111

Output (stdout)

1111000

1111000

Input (stdin)

1

1100110

Output (stdout)

1111000

Solution:

import java.util.Scanner;

public class MoveZeroesToEnd {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int T = scanner.nextInt();

scanner.nextLine();

for (int t = 0; t < T; t++) {

System.out.println("Enter number : "+ (t+1));

String binaryString = scanner.nextLine();

String result = moveZeroesToEnd(binaryString);

System.out.println(result);

}

scanner.close();

}

public static String moveZeroesToEnd(String binaryString) {

int countZeroes = 0;

StringBuilder result = new StringBuilder();

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

char ch = binaryString.charAt(i);

if (ch == '1') {

result.append(ch);

} else if (ch == '0') {

countZeroes++;

}

}

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

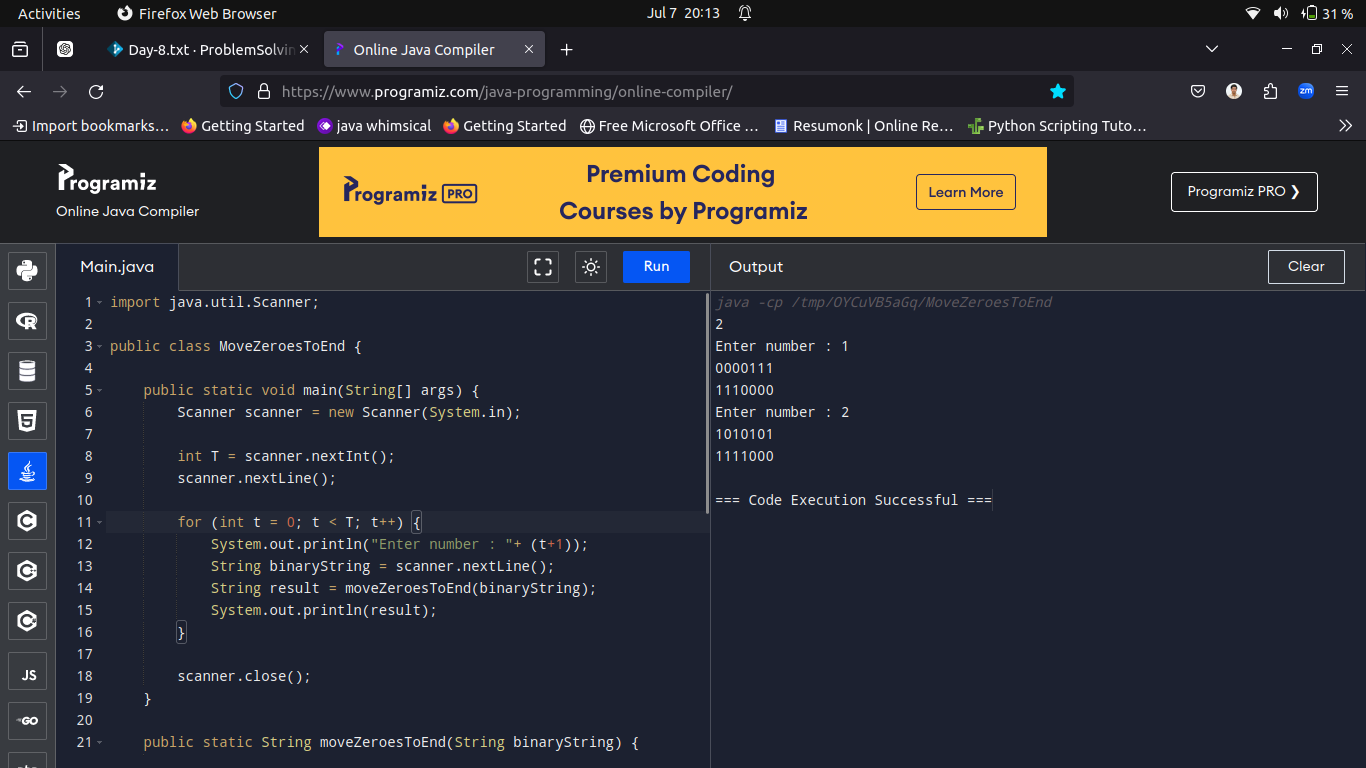
result.append('0');

}

return result.toString();

}

}



-----------------------------------------------------------------------------------------------------------------------------------------------------------

Q10.Write a program to implement a Selection Sort algorithm for sorting the elements of an array.

Format:

Input:

The first line of the input denotes the number of test cases 'T'. The first line of the test case is the size of an array and the second line consists of an array of elements.

Output:

The sorted array in ascending order is displayed to the user.

Constraints:

1 <= T <= 50

1 <= N <= 1000

1 <= arr[i] <= 1000

Example:

Input:

2

5

4 1 3 9 7

10

10 9 8 7 6 5 4 3 2 1

Output:

1 3 4 7 9

1 2 3 4 5 6 7 8 9 10

Explanation :

Sorted array is 1,3,4,7,9 for the test case 1.

Case 1

Case 2

Input (stdin)

2

5

4 1 3 9 7

10

10 9 8 7 6 5 4 3 2 1

Output (stdout)

1 3 4 7 9

1 2 3 4 5 6 7 8 9 10

Solution:

import java.util.Scanner;

public class SelectionSort {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int T = scanner.nextInt();

scanner.nextLine();

for (int t = 0; t < T; t++) {

System.out.println("Enter test case: "+(t+1));

int N = scanner.nextInt();

int[] arr = new int[N];

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

arr[i] = scanner.nextInt();

}

selectionSort(arr);

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

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

}

System.out.println();

}

scanner.close();

}

public static void selectionSort(int[] arr) {

int N = arr.length;

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

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

if (minIndex != i) {

int temp = arr[i];

arr[i] = arr[minIndex];

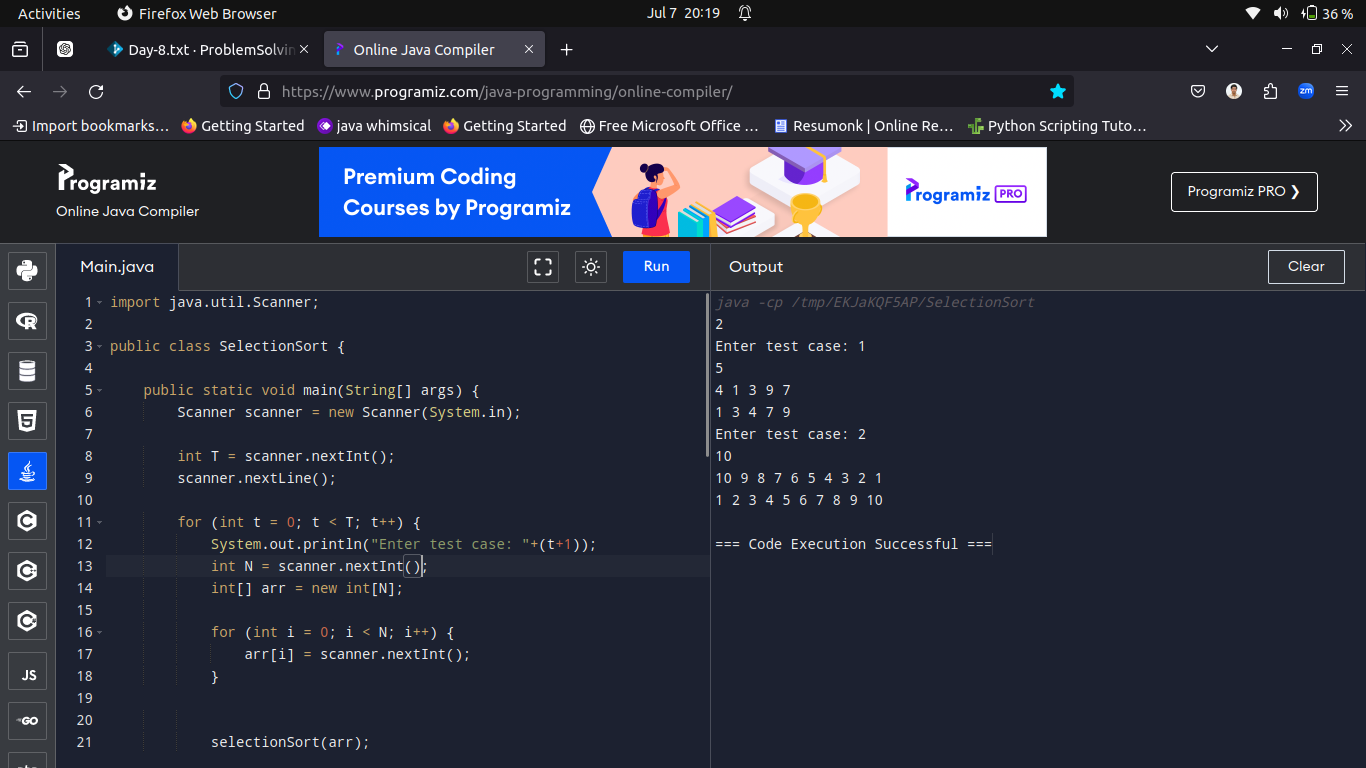
arr[minIndex] = temp;

}

}

}

}



-----------------------------------------------------------------------------------------------------------------------------------------------------------