## CHANDIGARH UNIVERSITY

## UNIVERSITY INSTITUTE OF NGINEERING

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



|  |  |
| --- | --- |
| **Submitted By: Submitted To:**  Vivek Kumar(21BCS8129) Mamta Punia(E12337) | |
| **Subject Name** | Competitive Coding - I |
| **Subject Code** | 20CSP-314 |
| **Branch** | Computer Science and Engineering |
| **Semester** | 5th |

**Lab MST-1**

**Student Name: Vivek Kumar UID: 21BCS8129**

**Branch: BE-CSE(LEET) Section/Group: WM-20BCS-616/A**

**Semester: 5th Date of Performance: 30/09/2022**

**Subject Name: Competitive coding - I Subject Code: 20CSP-314**

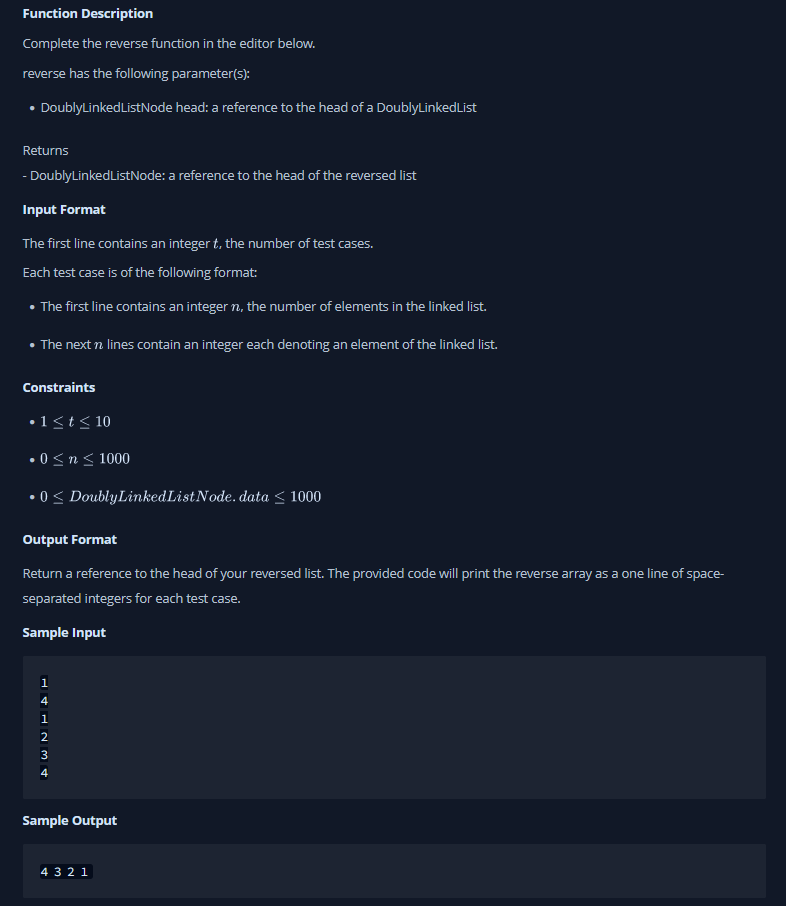
# **Fraudulent Activity Notifications:**

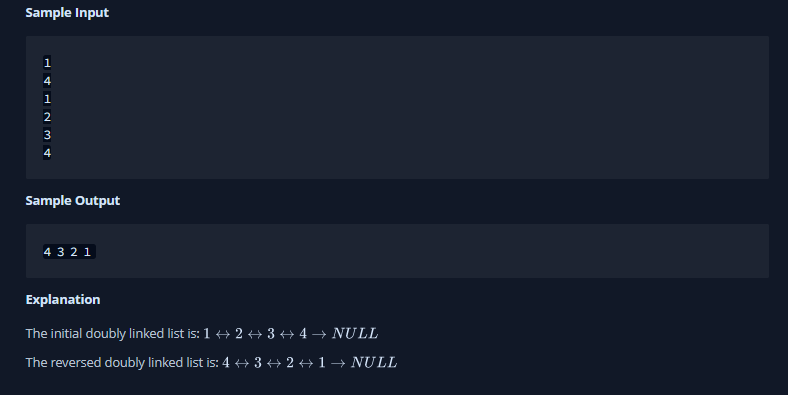
**1. Aim/Overview of the practical:**

Given the pointer to the head node of a doubly linked list, reverse the order of the nodes in place. That is, change the next and prev pointers of the nodes so that the direction of the list is reversed. Return a reference to the head node of the reversed list.

**Note:** The head node might be NULL to indicate that the list is empty.

**2. Task to be done/ Which logistics used:**





**3. Hardware and Software Requirements (For programming-based labs):**

* Laptop or Desktop
* Hacker-Rank Account

**4. Steps for experiment/practical/Code:**

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class DoublyLinkedListNode {

    public int data;

    public DoublyLinkedListNode next;

    public DoublyLinkedListNode prev;

    public DoublyLinkedListNode(int nodeData) {

        this.data = nodeData;

        this.next = null;

        this.prev = null;

    }

}

class DoublyLinkedList {

    public DoublyLinkedListNode head;

    public DoublyLinkedListNode tail;

    public DoublyLinkedList() {

        this.head = null;

        this.tail = null;

    }

    public void insertNode(int nodeData) {

        DoublyLinkedListNode node = new DoublyLinkedListNode(nodeData);

        if (this.head == null) {

            this.head = node;

        } else {

            this.tail.next = node;

            node.prev = this.tail;

        }

        this.tail = node;

    }

}

class DoublyLinkedListPrintHelper {

    public static void printList(DoublyLinkedListNode node, String sep, BufferedWriter bufferedWriter) throws IOException {

        while (node != null) {

            bufferedWriter.write(String.valueOf(node.data));

            node = node.next;

            if (node != null) {

                bufferedWriter.write(sep);

            }

        }

    }

}

class Result {

    /\*

     \* Complete the 'reverse' function below.

     \*

     \* The function is expected to return an INTEGER\_DOUBLY\_LINKED\_LIST.

     \* The function accepts INTEGER\_DOUBLY\_LINKED\_LIST llist as parameter.

     \*/

    /\*

     \* For your reference:

     \*

     \* DoublyLinkedListNode {

     \*     int data;

     \*     DoublyLinkedListNode next;

     \*     DoublyLinkedListNode prev;

     \* }

     \*

     \*/

    public static DoublyLinkedListNode reverse(DoublyLinkedListNode llist) {

    // Write your code here

        DoublyLinkedListNode oldNext = llist.next;

        llist.next = llist.prev;

        llist.prev = oldNext;

        if(oldNext != null) llist = reverse(oldNext);

        return llist;

    }

}

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        int t = Integer.parseInt(bufferedReader.readLine().trim());

        IntStream.range(0, t).forEach(tItr -> {

            try {

                DoublyLinkedList llist = new DoublyLinkedList();

                int llistCount = Integer.parseInt(bufferedReader.readLine().trim());

                IntStream.range(0, llistCount).forEach(i -> {

                    try {

                        int llistItem = Integer.parseInt(bufferedReader.readLine().trim());

                        llist.insertNode(llistItem);

                    } catch (IOException ex) {

                        throw new RuntimeException(ex);

                    }

                });

                DoublyLinkedListNode llist1 = Result.reverse(llist.head);

                DoublyLinkedListPrintHelper.printList(llist1, " ", bufferedWriter);

                bufferedWriter.newLine();

            } catch (IOException ex) {

                throw new RuntimeException(ex);

            }

        });

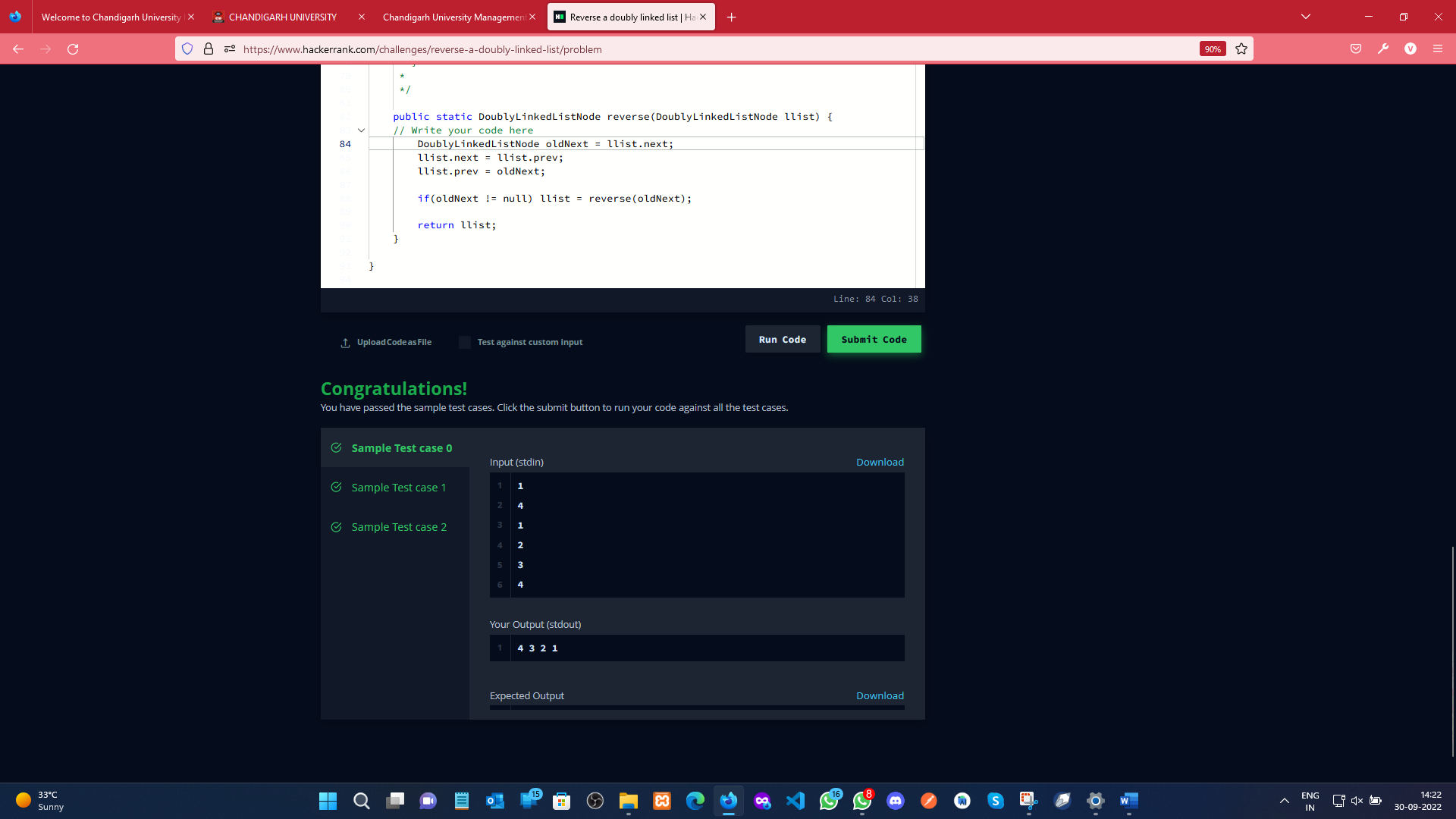
        bufferedReader.close();

        bufferedWriter.close();

    }

}

**5. Result/Output/Writing Summary:**



**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |
|  |  |  |  |