Lab 04 [CO3]

[Each method carries 5 marks]

Part-1: Doubly Linked List

Greetings Students. In this lab, we will play with Dummy Headed Doubly Circular Linked List first. If you want to read about this type of linked list then check **this file.**

In the first part of this lab, you have to implement a waiting room management system in an emergency ward of a hospital. Your program will serve a patient on a **first-come-first-serve basis.**

Solve the above problem using a **Dummy Headed Doubly Circular Linked List.**

1. You need to have a **Patient** class so that you can create an instance of it (patient) by assigning id(integer), name (String), age (integer), and bloodgroup (String).

2. Write a **WRM** (waiting room management) class that will contain the below methods.
    a. **RegisterPatient(id, name, age, bloodgroup):** This method will register a patient into your system. The method will create a Patient type object with the information received as parameter. It means this method will add a patient-type object to your linked list.
    b. **ServePatient():** This method calls a patient to provide hospital service to him/her. In this method, you need to ensure to serve the patient first who was registered first.
    c. **CancelAll():** This method cancels all appointments of the patients so that the doctor can go to lunch.
    d. **CanDoctorGoHome():** This method returns true if no one is waiting, otherwise, returns false.
    e. **ShowAllPatient():** This method prints all ids of the waiting patients in sequential order. It means the patient who got registered first, will come first, and so on.
    f. **ReverseTheLine():** This method reverses the patient line. It means the patient who got registered last, will come first, and so on.

3. Write a **Tester** code that will interact with users and take information about Patients. You will pass this information to **WRM** and create instances of **Patient**

in **WRM** and call the methods of **WRM** class. You just need to ensure your Tester code has completed all the properties mentioned in 4 no point.

4. Tester Code Options:
    a. Add Patient – print Success or Not
    b. Serve Patient – print Name of Patient being Served
    c. Show All patients – print all patient in sequence to serve
    d. Can Doctor go Home? – return yes or no
    e. Cancel all Appointment – print Success or Not
    f. ReverseTheLine - print Success or Not

Hints:

Usual Node class design in doubly linked list:

```
class DoublyNode:

    def __init__(self, elem, next, prev):
        self.elem = elem
        self.next = next # To store the next node's reference.
        self.prev = prev # To store the previous node's reference.
```

In your program your Patient class will work as the Node class for the Dummy Headed Doubly Circular Linked List and WRM class will work as that Linked List.

Part 2: Stack

NOTE:

- **YOUR CODE SHOULD WORK FOR ANY VALID INPUTS. [Make changes to the Sample Inputs and check whether your program works correctly]**
- **ALL YOUR METHODS/TASKS SHOULD BE OUTSIDE Stack CLASS**

1. Diamond Count:

Faisal is working in a diamond mine, trying to extract the highest number of diamonds “< >”. A “<” followed by “>” forms one new diamond. He must exclude all the sand particles found (denoted by “.”) and **unpaired** “<”, “>” in this process to extract new diamonds.

You need to solve the above problem using **Stack class**. You cannot use other methods than pop(), peek(), push(), isEmpty() methods of Stack.

Complete the function **count_diamond** which will take an object of Stack and a string and then return the number of diamonds that can be extracted using the mentioned process.

Sample Input String	Sample Output
.<...<<..>>....>....>>>.	3
<<<..<.....<<<<....>	1
<..><.<..>>	3

Explanation:
For an input “.<...<<..>>....>....>>>.” three diamonds are formed. The first is taken from <..> resulting “.<...<>....>....>>>.” The second diamond <> is then removed, leaving “.<.....>....>>>.” The third diamond <> is then removed, leaving at the end “.....>>>.” without the possibility of extracting new diamonds. Hence, 3 diamonds have been extracted.

2. Tower of Blocks:

A kid is trying to make a tower using blocks with numbers or characters written on them. He puts a block on the top of another to make the tower. As for removing the blocks, he removes the topmost block if needed. He is afraid that removing other blocks from the middle in one go will result in the collapse of his tower.

Now his friend wants your nth block from the top of your tower. He is willing to give him the nth block from the top, preserving the relative position of others.

Following his tower build process, solve this scenario using **Stack** class. You can only use the stack methods.
Hints: You can create a temporary object of Stack class.

In the following examples, consider the rightmost element to be the topmost element of the stack.

Sample Tower	Sample Output	Explanation
Stack: 4 19 23 17 5 n: 2	Stack: 4 19 23 5	In this tower, the 2nd block from the top is the block with 17. After removing the block the stack looks like 4 19 23 5
Stack: 73 85 15 41 n: 3	Stack: 73 15 41	In this tower, the 3rd block from the top is the block with 85. After removing the block the stack looks like 73 15 41

3. Stack Reverse:

Consider that a Stack class has been implemented containing the push(element), pop(), peek() and isEmpty() functions.

Complete the function **conditional_reverse()** which will take an object of Stack class as an input that contains some integer values. **The function returns a new stack** which will contain the values in reverse order from the given stack **except the consecutive one's.** You cannot use any other data structure except Stack.

Note: The Stack class implements a singly linked list-based Stack hence overflow is not possible. The pop() and peek() functions return None in case of the underflow.

In the following example, consider the rightmost element to be the topmost element of the stack.

Sample Input Stack (Rightmost is the top)	Output Reversed Stack (Rightmost is the top)	Explanation
Stack: 10, 10, 20, 20, 30, 10, 50 Top = 50	New Stack: 50, 10, 30, 20, 10 Top = 10	Consecutive 20 and 10 are not present in the output reversed stack