Doubly Linked List

You do not need to write the DoublyNode class, createList()/printList()/count()/nodeAt() etc. functions unless specified in the problem.

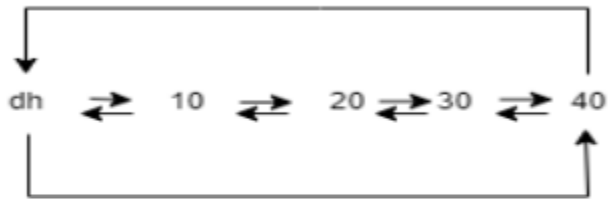
Just design the function specified in the problem.

No need to write driver code.

This document is a compilation of previous semester's exam (midterm/final) problems and lab quiz problems. Try to practice relevant problems from online judges as well.

1.

You are given a dummy headed doubly circular linked list and a block of code. The list is like below:




```
for i in range(5):
    for j in range(i):
        n1 = dh.next
        n2 = n1.next
        n3 = dh.prev
        dh.next = n2
        n2.prev = dh
        n3.next = n1
        n1.next = dh
        n1.prev = n3
        dh.prev = n1
```

Draw the resulting list (The list you will find after the nested loop) for each value **i** in your answer script. Mention the value of **i** and the resulting list, no need to show the intermediate states.

2.

You are working on a project where you need to manage a list of clients. Each client is represented as a node in a dummy headed doubly linked list. Each client node contains information about the client's ID .

Now, your task is to perform a specific operation to maintain a list of clients. You need to implement a function called **Removes_client(head).** This function will remove the client if the client id is divisible by the length of the linked list

- No need to write the Node class. Just assume Node class is there with three instance variables; ***client_id, prev*** and ***next***.
- You may want to **write a length(head)** method/function to use it in this task.

Sample Input	Sample Output
DH ⇄ 17 ⇄ 15 ⇄ 12 ⇄ 41 ⇄ 66 ⇄ 67 ⇄ None **Removes_client(head).**	**DH ⇄ 17 ⇄ 15 ⇄ 41 ⇄ 67 ⇄ None** Explanation: Length of the linked list is 6. Client id is 17; 17 % 6 = 5 so you won't remove the client. Client id is 15; 15 % 6 = 3 so you won't remove the client. Client id is 12; 12 % 6 =0 so you will remove the client. Client id is 41; 41 % 6 = 5 so you won't remove the client. Client id is 66; 66 % 6 = 0 so you will remove the client. Client id is 67; 67 % 6 = 1 so you won't remove the client

3.

You are working on a project where you need to manage a list of clients. Each client is represented as a node in a dummy headed doubly linked list. Each client node contains information about the client's ID .

Now, your task is to perform a specific operation to maintain a list of clients. You need to implement a function called **Removes_client(head).** This function will remove the client if the client id is not divisible by the length of the linked list

- No need to write the Node class. Just assume Node class is there with three instance variables; ***client_id, prev*** and ***next***.
- You may want to **write a length(head)** method/function to use it in this task.

Sample Input	Sample Output
DH ⇄ 18 ⇄ 15 ⇄ 12 ⇄ 41 ⇄ 66 ⇄ 67 ⇄ None **Removes_client(head).**	**DH ⇄ 18 ⇄ 12 ⇄ 66 ⇄ None** Explanation: Length of the linked list is 6. Client id is 18; 18 % 6 = 0 so you won't remove the client. Client id is 15; 15 % 6 = 3 so you will remove the client. Client id is 12; 12 % 6 =0 so you won't remove the client. Client id is 41; 41 % 6 = 5 so you will remove the client. Client id is 66; 66 % 6 = 0 so you won't remove the client. Client id is 67; 67 % 6 = 1 so you will remove the client.

4.

You are given a **dummy headed doubly circular** linked list *X*, and a non-dummy headed singly linear linked list *A*. You need to insert *A* at the *k* th index of *X*. The resulting linked list will also be dummy headed doubly circular.
Write a function **insert_list(head_X, head_A, k)**. Just completing this function will suffice.
Note: Assume that Node class is given

Sample Input	Sample Output
X = DH ⇔ 2 ⇔ 3 ⇔ 1 ⇔ 9 A = 4 -> 8 k = 2	DH ⇔ 2 ⇔ 3 ⇔ 4 ⇔ 8 ⇔ 1 ⇔ 9
X = DH ⇔ 2 ⇔ 3 ⇔ 1 ⇔ 9 A = 5 -> 4 -> 6 k = 0	DH ⇔ 5 ⇔ 4 ⇔ 6 ⇔ 2 ⇔ 3 ⇔ 1 ⇔ 9

5.

You are given the head of a **dummy headed doubly circular** linked list. You need to remove some nodes from this linked list so that no two consecutive nodes have exactly the same element.
Write a function **multi_delete(head)**. Just completing this function will suffice.
Note: Assume that Node class is given.

Sample Input	Sample Output
DH ⇔ 2 ⇔ 3 ⇔ 3 ⇔ 1 ⇔ 1 ⇔ 1 ⇔ 9	DH ⇔ 2 ⇔ 3 ⇔ 1 ⇔ 9
DH ⇔ 2 ⇔ 3 ⇔ 1 ⇔ 9	DH ⇔ 2 ⇔ 3 ⇔ 1 ⇔ 9

6.

A teacher gives a child a set of toy boxes arranged in a circle, each placed in a specific position. As a challenge, the teacher asks the child to reverse the order of boxes between two specific positions **m** and **n**, while keeping the rest of the arrangement unchanged. Now, your task is to help the child fix the arrangement. The boxes are represented as a **Dummy Headed Circular Doubly Linked List**, where each node corresponds to a box.

You are given the **head** of the list and two integers **m** and **n** ($1 \le m < n \le$ length of the list). **Reverse** the nodes from position m to n without affecting the rest of the list, and return the **modified list's head**. Consider that the **DoublyNode** class, **createDLL()**, **printDLL()**, **nodeAt()** methods/functions are already provided.

Sample Function Call	Sample Returned Result
head => 10 ⇌ 40 ⇌ 30 ⇌ 20 ⇌ 50, m = 2, n = 4 **reverseDLLBetweenTwoPositions(head, m, n)**	**Head of the modified linked list =>** 10 ⇌ 20 ⇌ 30 ⇌ 40 ⇌ 50

Note: *Solve this task in* ***constant space complexity****, meaning you must modify the linked list* ***in-place*** *without creating a new one. m and n will never have the same value, and the linked list will always contain at least two nodes.*

***nodeAt(head, idx):** returns the node at the given index in a Dummy Headed Circular Doubly Linked List or None if out of bounds.

7.

A teacher gives a child a set of toy boxes arranged in a circle, each labeled with a **unique number**. As a challenge, the teacher asks the child to reverse the order of boxes between two specific boxes with given values **x** and **y**, while keeping the rest of the arrangement unchanged. Now, your task is to help the child organize the boxes. The toy boxes are represented as a **Dummy Headed Circular Doubly Linked List**, where each node corresponds to a box.

You are given the **head** of the list and two distinct values **x** and **y** (where x appears before y in the list). **Reverse** the nodes from value x to value y without affecting the rest of the list, and return the **modified list's head**. Consider that the **DoublyNode** class, **createDLL()**, **printDLL()**, **getNodeByValue()** methods/functions are already provided.

Sample Function Call	Sample Returned Result
head => 10 ⇄ 20 ⇄ 30 ⇄ 90 ⇄ 40 ⇄ 50, x = 20, y = 40 **reverseDLLBetweenTwoValues(head, x, y)**	**Head of the modified linked list =>** 10 ⇄ 40 ⇄ 90 ⇄ 30 ⇄ 20 ⇄ 50

Note: *Solve this task in* ***constant space complexity****, meaning you must modify the linked list* ***in-place*** *without creating a new one. x will always appear before y in the list, and the linked list will always contain at least two nodes.*

***getNodeByValue(head, value):** returns the node containing the given value in a Dummy Headed Circular Doubly Linked List, or None if the value is not found.

8.

Rearrange a Doubly Linked List with Odd and Even Elements

Given a doubly linked list, rearrange it so that all odd elements appear before all even elements, while maintaining the relative order of the nodes.

The first node of the resultant modified doubly linked list should always be an odd number if there is at least one odd element. Otherwise, the order remains the same.

You can assume the `Node` class is already defined, with attributes `elem`, `next`, and `prev`.

Note: You can only use linked list data structure

Sample Input	Sample Output
5 ⇄ 2 ⇄ 8 ⇄ 1 ⇄ 4 ⇄ 7	5 ⇄ 1 ⇄ 7 ⇄ 2 ⇄ 8 ⇄ 4
7 ⇄ 9 ⇄ 2 ⇄ 4	7 ⇄ 9 ⇄ 2 ⇄ 4
6 ⇄ 2 ⇄ 8 ⇄ 10 ⇄ 12	6 ⇄ 2 ⇄ 8 ⇄ 10 ⇄ 12

9.

Rearrange a Doubly Linked List with even and odd Elements

Given a doubly linked list, rearrange it so that all even elements appear before all odd elements, while maintaining the relative order of the nodes.

The first node of the modified resultant doubly linked list should always be an even number if there is at least one even element. Otherwise, the order remains the same.

You can assume the `Node` class is already defined, with attributes `elem`, `next`, and `prev`.

Note: You can only use linked list data structure

Sample Input	Sample Output
5 ⇄ 2 ⇄ 8 ⇄ 1 ⇄ 4 ⇄ 7	2 ⇄ 8 ⇄ 4 ⇄ 5 ⇄ 1 ⇄ 7
7 ⇄ 9 ⇄ 2 ⇄ 4	2 ⇄ 4 ⇄ 7 ⇄ 9
5 ⇄ 9 ⇄ 13 ⇄ 11 ⇄ 15	5 ⇄ 9 ⇄ 13 ⇄ 11 ⇄ 15

10.

Question: Check if a Doubly Linked List is a Palindrome

You are given a dummy-headed doubly circular linked list. Implement a function to check if the linked list is a palindrome. A list is a palindrome if the elements read the same forward and backward.

- The function should return `true` if it is a palindrome, otherwise `false`.
- You can access `elem`, `next`, and `prev` of a node directly. No need to write the node class code.
- No need to write the driver class code.
- You cannot modify the list while checking.

Sample Input	Sample Output
DH ↔ 1 ↔ 2 ↔ 3 ↔ 2 ↔ 1	True
DH ↔ 1	True
DH	True
DH ↔ 4 ↔ 5 ↔ 6 ↔ 7 ↔ 8	False

You only need to complete the following functions. You don't have to write anything else.

Java	Python

`public boolean isPalindrome(Node head) {`
`// TODO: Implement your solution`
`}`
	`def isPalindrome(head):`
	`# TODO: Implement your solution`

11.

Question: Find a Pair with a Target Sum in a Sorted Doubly Linked List

You are given a sorted dummy-headed doubly circular linked list and a target sum, X. Implement a function that returns any first pair of nodes `(a, b)` such that `a.elem + b.elem == X`.

- If multiple pairs exist, return the first found pair. If no such pair exists, return None.
- The function should traverse from both forward and backward.
- You can access `elem`, `next`, and `prev` of a node directly. No need to write the node class code.
- No need to write the driver class code.

Sample Input	Sample Output
DH ↔ 1 ↔ 2 ↔ 3 ↔ 4 ↔ 6 ↔ 8 ↔ 9 **Target: 10**	**(1, 9)**
DH ↔ 1 ↔ 3 ↔ 5 ↔ 7 ↔ 9 ↔ 10 **Target: 12**	**(3, 9)**
DH ↔ 1 ↔ 2 ↔ 4 ↔ 5 ↔ 7 **Target: 20**	**None**
DH **Target: 20**	**None**

Hint: Use two pointers. One starting from the smallest element (`head.next`) and another from the largest (`head.prev`). Move the front pointer forward if the sum is too small and the back pointer backwards if the sum is too large. Stop when a pair is found or pointers cross.

You only need to complete the following functions. You don't have to write anything else.

Java	Python

`public Node[] findPair(Node head, int target) {`
`// TODO: Implement your solution`
`}`
	`def findPair(head, target):`
	`# TODO: Implement your solution`

12.

Tasks from Book - Practice Sheet 4 - Doubly Linked List.pdf

13.

Tasks from Lab 2 - Lab 2 - Singly & Doubly LinkedList.docx