1-Consider the implementation of CircularlyLinkedList.addFirst, in Code Fragment 3.16. The else body at lines 39 and 40 of that method relies on a locally declared variable, newest. Redesign that clause to avoid use of any local variable.?

public void addFirst(T element) {

Node<T> newest = new Node<>(element);

if (isEmpty()) {

newest.next = newest;

} else {

newest.next = tail.next;

tail.next = newest;

}

tail = newest;

size++;

}

2-Give an implementation of the size( ) method for the CircularlyLinkedList class, assuming that we did not maintain size as an instance variable.

public class CircularlyLinkedList<T> {

private Node<T> tail;

private int size;

// ... other methods for the CircularlyLinkedList class

public int size() {

if (tail == null) {

return 0;

}

int count = 1; // Start with 1 for the tail node

Node<T> current = tail.next;

while (current != tail) {

count++;

current = current.next;

}

return count;

}

private static class Node<T> {

T data;

Node<T> next;

Node(T data) {

this.data = data;

this.next = null;

}

}

}

3- Implement the equals( ) method for the CircularlyLinkedList class, assuming that two lists are equal if they have the same sequence of elements, with corresponding elements currently at the front of the list.

public class CircularlyLinkedList<T> {

private Node<T> tail;

private int size;

// ... other methods for the CircularlyLinkedList class

public boolean equals(CircularlyLinkedList<T> otherList) {

if (size != otherList.size) {

return false;

}

if (tail == null && otherList.tail == null) {

return true;

}

Node<T> currentThis = tail.next;

Node<T> currentOther = otherList.tail.next;

while (currentThis != tail && currentOther != otherList.tail) {

if (!currentThis.data.equals(currentOther.data)) {

return false;

}

currentThis = currentThis.next;

currentOther = currentOther.next;

}

return currentThis.data.equals(currentOther.data); // Check the last element

}

private static class Node<T> {

T data;

Node<T> next;

Node(T data) {

this.data = data;

this.next = null;

}

}

}

4- Suppose you are given two circularly linked lists, L and M. Describe an algorithm for telling if L and M store the same sequence of elements (but perhaps with different starting points).

function sameSequence(L, M):

if L is empty and M is empty:

return true

currentL = L.head

currentM = M.head

count = 0

while count < L.size and count < M.size:

if currentL.data is not equal to currentM.data:

return false

currentL = currentL.next

currentM = currentM.next

count = count + 1

return true

5-Given a circularly linked list L containing an even number of nodes, describe how to split L into two circularly linked lists of half the size.

function splitCircularLinkedList(L):

if L is empty or L contains an odd number of nodes:

return

L1 = L.head

hare = L.head

tortoise = L.head

while hare.next is not equal to L.head and hare.next.next is not equal to L.head:

hare = hare.next.next

tortoise = tortoise.next

midpoint = tortoise

lastNodeL1 = L1

while lastNodeL1.next is not equal to L.head:

lastNodeL1 = lastNodeL1.next

lastNodeL1.next = L.head

L2 = midpoint.next

lastNodeL2 = L2

while lastNodeL2.next is not equal to L.head:

lastNodeL2 = lastNodeL2.next

lastNodeL2.next = midpoint

return L1, L2

6- Implement the clone( ) method for the CircularlyLinkedList class.

public class CircularlyLinkedList<T> {

private Node<T> tail;

private int size;

// ... other methods for the CircularlyLinkedList class

public CircularlyLinkedList<T> clone() {

CircularlyLinkedList<T> clonedList = new CircularlyLinkedList<>();

if (isEmpty()) {

return clonedList;

}

Node<T> current = tail.next;

while (current != tail) {

clonedList.addLast(current.data);

current = current.next;

}

clonedList.addLast(tail.data); // Add the last element

return clonedList;

}

private static class Node<T> {

T data;

Node<T> next;

Node(T data) {

this.data = data;

this.next = null;

}

}

}

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