## Homework

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, null);

if (size == 0) {

newest.next = newest; // Link to itself circularly

tail = newest;

} else {

newest.next = tail.next;

tail.next = newest;

}

size++;

}

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

public int size() {

if (tail == null) {

return 0;

}

int count = 1;

Node<T> current = tail;

while (current.next != tail) {

count++;

current = current.next;

}

return count;

}

1. 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 boolean equals(CircularlyLinkedList<T> otherList) {

if (this.size() != otherList.size()) {

return false;

}

Node<T> thisCurrent = this.tail.next;

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

while (thisCurrent != this.tail) {

if (!thisCurrent.element.equals(otherCurrent.element)) {

return false;

}

thisCurrent = thisCurrent.next;

otherCurrent = otherCurrent.next;

}

return true; // All elements are equal

}

1. 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).

Traverse through one list and check if the elements match the sequence of the other list. One way to do this is by starting at each node of one list and comparing it with the nodes of the other list.

1. 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.

Traverse through the list to find the middle node, then set the next pointer of the last node to the middle node to create two separate circularly linked lists

1. Implement the clone( ) method for the CircularlyLinkedList class.

public CircularlyLinkedList<T> clone() {

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

if (tail == null) {

return clonedList; // Return an empty list

}

Node<T> current = tail.next;

do {

clonedList.addLast(current.element);

current = current.next;

} while (current != tail.next);

return clonedList;

}