## Topics

1. Implement Node Class

class Node<T> {

T data;

Node<T> next;

public Node(T data) {

this.data = data;

this.next = null;

    }

}

1. Implement CircularlyLinkedList Class

public class CircularLinkedList<T> {

private Node<T> tail;

}

1. Implement Basic Methods of CircularlyLinkedList

* isEmpty()

public boolean isEmpty() {

return tail == null;

}

* size()

public int size() {

if (tail == null) {

return 0;

}

int count = 1;

Node<T> current = tail.next;

while (current != tail) {

count++;

current = current.next;

}

 return count;

}

* first()

public T first() {

if (isEmpty()) {

throw new NoSuchElementException();

}

return tail.next.data;

}

* last()

public T last() {

if (isEmpty()) {

throw new NoSuchElementException();

}

return tail.data;

}

* addFirst()

public void addFirst(T item) {

Node<T> newNode = new Node<>(item);

if (tail == null) {

tail = newNode;

tail.next = newNode;

} else {

newNode.next = tail.next;

tail.next = newNode;

    }

}

* addLast()

public void addLast(T item) {

addFirst(item); // في القوائم الدائرية، يمكن إضافة إلى النهاية عن طريق إضافة إلى البداية

}

* removeFirst()

public T removeFirst() {

if (isEmpty()) {

throw new NoSuchElementException();

}

if (tail.next == tail) { // حالة عنصر واحد

T data = tail.data;

tail = null;

return data;

}

T data = tail.next.data;

tail.next = tail.next.next;

  return data;

}

* rotate()

public void rotate() {

if (tail != null) {

tail = tail.next;

    }

}

## 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 item) {

Node<T> newNode = new Node<>(item);

if (tail == null) {

tail = newNode;

tail.next = newNode;

} else {

// بدلاً من استخدام متغير مؤقت، نعيد ترتيب الروابط مباشرة

newNode.next = tail.next;

tail.next = newNode;

 }

    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.next;

while (current != 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.

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

    }

}

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).

class CircularLinkedList {

Node head;

}

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.

public boolean equals(CircularLinkedList other) {

if (this == other) return true;

if (other == null) return false;

if (this.size() != other.size()) return false;

Node temp1 = this.head;

Node temp2 = other.head;

do {

if (temp1.data != temp2.data) return false;

temp1 = temp1.next;

temp2 = temp2.next;

} while (temp1 != this.head);

  return true;

}

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

public CircularLinkedList clone() {

if (head == null) return null;

CircularLinkedList newList = new CircularLinkedList();

Node newHead = new Node(head.data);

newList.head = newHead;

Node temp = head.next, newTemp = newHead;

while (temp != head) {

newTemp.next = new Node(temp.data);

newTemp = newTemp.next;

temp = temp.next;

}

newTemp.next = newHead;

return newList;

}