## Topics

1. Implement Node Class
2. Implement CircularlyLinkedList Class
3. Implement Basic Methods of CircularlyLinkedList

* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()
* rotate()

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

public class CircularlyLinkedList<E> {

private static class Node<E> {

private E element;

private Node<E> next;

public Node(E element, Node<E> next) {

this.element = element;

this.next = next;

}

public E getElement() {

return element;

}

public void setElement(E element) {

this.element = element;

}

public Node<E> getNext() {

return next;

}

public void setNext(Node<E> next) {

this.next = next;

}

}

private Node<E> tail = null;

private int size = 0;

public CircularlyLinkedList() {}

public int size() {

if (tail == null) return 0;

int count = 1;

Node<E> current = tail.getNext();

while (current != tail) {

count++;

current = current.getNext();

}

return count;

}

public boolean isEmpty() {

return tail == null;

}

public E first() {

if (isEmpty()) return null;

return tail.getNext().getElement();

}

public E last() {

if (isEmpty()) return null;

return tail.getElement();

}

public void rotate() {

if (tail != null) tail = tail.getNext();

}

public void addFirst(E e) {

if (size == 0) {

tail = new Node<>(e, null);

tail.setNext(tail);

} else {

tail.setNext(new Node<>(e, tail.getNext()));

}

size++;

}

public void addLast(E e) {

addFirst(e);

tail = tail.getNext();

}

public E removeFirst() {

if (isEmpty()) return null;

Node<E> head = tail.getNext();

if (head == tail) {

tail = null;

} else {

tail.setNext(head.getNext());

}

size--;

return head.getElement();

}

public String getAll() {

if (isEmpty()) return "";

StringBuilder all = new StringBuilder();

Node<E> current = tail.getNext();

for (int i = 0; i < size; i++) {

all.append(current.getElement()).append(" ");

current = current.getNext();

}

return all.toString();

}

@Override

public boolean equals(Object o) {

if (o == this) return true;

if (!(o instanceof CircularlyLinkedList<?>)) return false;

CircularlyLinkedList<?> other = (CircularlyLinkedList<?>) o;

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

Node<E> current1 = this.tail != null ? this.tail.getNext() : null;

Node<?> current2 = other.tail != null ? other.tail.getNext() : null;

for (int i = 0; i < this.size(); i++) {

if (!current1.getElement().equals(current2.getElement())) {

return false;

}

current1 = current1.getNext();

current2 = current2.getNext();

}

return true;

}

public CircularlyLinkedList<E>[] split() {

if (size % 2 != 0) throw new IllegalStateException("List size must be even.");

CircularlyLinkedList<E> firstHalf = new CircularlyLinkedList<>();

CircularlyLinkedList<E> secondHalf = new CircularlyLinkedList<>();

Node<E> slow = tail.getNext();

Node<E> fast = tail.getNext();

for (int i = 0; i < size / 2 - 1; i++) {

slow = slow.getNext();

}

// Create first half

firstHalf.tail = slow;

firstHalf.tail.setNext(tail.getNext());

firstHalf.size = size / 2;

// Create second half

secondHalf.tail = tail;

secondHalf.tail.setNext(slow.getNext());

secondHalf.size = size / 2;

// Break the connection to make two separate lists

slow.setNext(firstHalf.tail.getNext());

tail = null; // Original list no longer valid

return new CircularlyLinkedList[]{firstHalf, secondHalf};

}

@Override

public CircularlyLinkedList<E> clone() {

CircularlyLinkedList<E> clone = new CircularlyLinkedList<>();

if (this.isEmpty()) return clone;

Node<E> current = this.tail.getNext();

for (int i = 0; i < this.size(); i++) {

clone.addLast(current.getElement());

current = current.getNext();

}

return clone;

}

public static void main(String[] args) {

java.util.Scanner scan = new java.util.Scanner(System.in);

CircularlyLinkedList<String> l = new CircularlyLinkedList<>();

int choice;

while (true) {

System.out.println("1 add first 2 add last 3 rotate 4 remove first 5 size 6 is the list empty -1 exit");

System.out.println("Input your choice:");

choice = scan.nextInt();

switch (choice) {

case 1:

System.out.println("Input an element:");

l.addFirst(scan.next());

System.out.println(l.first() + " was added successfully");

break;

case 2:

System.out.println("Input an element:");

l.addLast(scan.next());

System.out.println(l.last() + " was added successfully");

break;

case 3:

l.rotate();

System.out.println("List element was rotated");

break;

case 4:

System.out.println(l.removeFirst() + " was removed");

break;

case 5:

System.out.println("List size is: " + l.size());

break;

case 6:

System.out.println("Is the list empty? " + l.isEmpty());

break;

case -1:

System.out.println("Goodbye");

System.exit(0);

}

System.out.println("List elements are: " + l.getAll());

}

}

}