## 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.

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.
2. 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).

### Algorithm

1. **Check Sizes**:
   * If L and M have different sizes, they cannot store the same sequence. Return false.
2. **Check for Empty Lists**:
   * If both lists are empty, they trivially store the same sequence. Return true.
3. **Find a Potential Starting Point**:
   * Start by comparing L's head element (i.e., the element after L's tail) to every element in M. Find the first position in M that matches L's head element. If no match is found, return false.
4. **Verify the Sequences**:
   * Starting from the matching position in M, compare elements in L and M one by one, wrapping around the circular structure as needed.
   * If all elements match, return true. Otherwise, return false.
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.

### Algorithm

1. **Check Preconditions**:
   * Verify that L contains an even number of nodes. If not, return an error or handle the odd-sized list case.
2. **Identify the Midpoint**:
   * Traverse L to find the midpoint (the node at position size/2\text{size} / 2size/2).
3. **Split into Two Lists**:
   * Create a new circularly linked list L1 starting from the head (the node after L's tail) up to the midpoint.
   * Create another circularly linked list L2 starting from the node after the midpoint up to L's tail.
4. **Adjust Links**:
   * Update L1's tail to point to its own head to make it circular.
   * Update L2's tail to point to its own head to make it circular.
5. **Return L1 and L2**:
   * Return the two new circularly linked lists.
6. Implement the clone( ) method for the CircularlyLinkedList class.