## Topics

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

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

## Homework

1. Describe a method for finding the middle node of a doubly linked list with header and trailer sentinels by “link hopping,” and without relying on explicit knowledge of the size of the list. In the case of an even number of nodes, report the node slightly left of center as the “middle.”
2. Give an implementation of the size( ) method for the DoublyLinkedList class, assuming that we did not maintain size as an instance variable.
3. Implement the equals( ) method for the DoublyLinkedList class.
4. Give an algorithm for concatenating two doubly linked lists L and M, with header and trailer sentinel nodes, into a single list L′.

### Algorithm:

1. **Check if M is empty**:
   * If list M is empty, no change is needed, and list L′ will just be L.
   * If list M is not empty, proceed to the next steps.
2. **Link the last node of L to the first node of M**:
   * Set the next pointer of the last node of L (which is L.trailer.prev) to point to the first node of M (which is M.header.next).
   * Set the prev pointer of the first node of M (which is M.header.next) to point to the last node of L (which is L.trailer.prev).
3. **Link the last node of M to the trailer of L**:
   * Set the next pointer of the last node of M (which is M.trailer.prev) to point to the trailer of L (which is L.trailer).
   * Set the prev pointer of the trailer of L (which is L.trailer) to point to the last node of M (which is M.trailer.prev).
4. **Update the size of list L'**:
   * The size of the concatenated list L′ will be the size of L plus the size of M. Update L.size by adding the size of M.
5. **Clear list M**:
   * After concatenation, set the header and trailer of list M to point to each other (i.e., M.header.next = M.trailer and M.trailer.prev = M.header) to reset M as an empty list.
   * Set the size of M to zero.
6. Our implementation of a doubly linked list relies on two sentinel nodes, header and trailer, but a single sentinel node that guards both ends of the list should suffice. Reimplement the DoublyLinkedList class using only one sentinel node.
7. Implement a circular version of a doubly linked list, without any sentinels, that supports all the public behaviors of the original as well as two new update methods, rotate( ) and rotateBackward.
8. Implement the clone( ) method for the DoublyLinkedList class.