Data Structures and Algorithms

Lab 04 – Linked Lists

Exercises/Tasks:

1. Write a program that implements all the methods of a doubly linked list, as mentioned below:
    - **addToFront**: Adds a new node with the given data to the front of the linked list.
    - **getFrontItem**: Returns the data stored in the first node (front) of the linked list.
    - **removeFrontItem**: Removes the first node (front) from the linked list.
    - **addToBack**: Adds a new node with the given data to the end (back) of the linked list.
    - **getBackItem**: Returns the data stored in the last node (back) of the linked list.
    - **removeBackItem**: Removes the last node (back) from the linked list.
    - **find**: Checks if the given key is present in the linked list.
    - **Remove**: Removes the node with the given key from the linked list, if present.
    - **isListEmpty**: Checks if the linked list is empty.
    - **addKeyBeforeNode**: Adds a new node with the given key before the node containing the specified data in the linked list.
    - **addKeyAfterNode**: Adds a new node with the given key after the node containing the specified data in the linked list.
    - **printAll**: Prints all the values in the linked list.

    Also, implement the main method to show/test how the different operations are performed on the list.

2. Extend the doubly linked list mentioned in the question above by adding the "**tail**" to it. Then, update the methods **addToBack, removeBackItem**, and **printInReverseOrder** to see if the efficiency increases.
3. Implement a basic (singly) Circular Linked List with the following operations:
    - Insert at the beginning
    - Insert at the end
    - Delete from the beginning
    - Delete from the end
    - Display the list

    Also, implement the main method to show/test the different operations.

4. A linked list can contain a cycle when there is a node in the list that has a reference pointing back to a previous node in the list, creating a loop. This can occur when the next pointer of a node in the list points to a node that is already part of the list or a previous node in the sequence.
   For example, the following list contains a cycle:
   1 -> 2 -> 3 -> 4 -> 2

   Here, the **next** pointer of **node 4** points to **node 2** (hence a cycle).

   Similarly, the following list does not contain a cycle:
   1 -> 2 -> 3 -> 4 -> NULL

   Now, extend/update the linked list created in the previous task and add a method **hasCycle()** that tells if the linked list contains a cycle or not.
5. Write a method to find (and return) the middle element of a linked list (whether singly or doubly).