Implement Circular Queue Using Linked List LeetCode

This program implements a circular queue data structure using a linked list. It defines a **CircularQueue** class with methods for enqueuing elements, dequeuing elements, checking if the queue is empty, getting the front and rear elements, getting the size of the queue, and displaying its elements. The circular queue is implemented using a linked list where the last element points back to the first element, making it circular. The program demonstrates the usage of the **CircularQueue** class by performing various circular queue operations.

1. **CircularQueue()** - Constructor to initialize an empty circular queue.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This constructor initializes an empty circular queue by setting both the **front** and **rear** pointers to **nullptr**. It's a simple operation with constant time and space complexity.
2. **void enQueue(int val)** - Enqueues an element to the circular queue.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method adds an element to the circular queue at the rear end by creating a new node, updating the **rear** pointer to the new node, and making the queue circular by linking the **rear** node to the **front** node. It has constant time and space complexity.
3. **int deQueue()** - Dequeues an element from the front of the circular queue.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method removes the front element from the circular queue by updating the **front** pointer to the next node and unlinking the previous front node. It also handles the case when there's only one element in the queue. It has constant time and space complexity.
4. **int getFront()** - Retrieves the front element of the circular queue.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method returns the value of the **front** node, which represents the front element of the circular queue. It has constant time and space complexity.

5. **int getRear()** - Retrieves the rear element of the circular queue.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method returns the value of the **rear** node, which represents the rear element of the circular queue. It has constant time and space complexity.
6. **bool isEmpty()** - Checks if the circular queue is empty.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method checks if the **rear** pointer is **nullptr**, which indicates that the circular queue is empty. It has constant time and space complexity.
7. **int getSize()** - Returns the size (number of elements) of the circular queue.
    - **Time Complexity: O(N), where N is the number of elements in the circular queue.**
    - **Space Complexity: O(1)**
    - **Explanation:** This method traverses the circular queue from **front** to **rear** to count the number of elements. It has a time complexity proportional to the number of elements in the queue and constant space complexity.
8. **void display()** - Displays the elements in the circular queue.
    - **Time Complexity: O(N), where N is the number of elements in the circular queue.**
    - **Space Complexity: O(1)**
    - **Explanation:** This method iterates through the circular queue, starting from **front** and stopping when it reaches **front** again, to display all elements. It has a time complexity proportional to the number of elements in the queue and constant space complexity.