Implement Circular Queue Using Array LeetCode

This program implements a circular queue data structure. It defines a **CircularQueue** class with methods for enqueuing elements, dequeuing elements, checking if the queue is empty or full, getting the front and rear elements, getting the size of the queue, and displaying its elements. The circular queue is implemented using an array, and it supports circular incrementing of the front and rear indices. The program demonstrates the usage of the **CircularQueue** class by performing various circular queue operations.

1. **CircularQueue(int k)** - Constructor to initialize the circular queue with a given size.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(k), where k is the size of the circular queue.**
    - **Explanation:** This constructor allocates memory for the circular queue array and initializes the front, rear, and length variables. The time complexity is constant, and the space complexity depends on the size of the circular queue.
2. **void enQueue(int value)** - 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 index and updates the rear index using circular incrementation. It has constant time and space complexity.
3. **int deQueue()** - Dequeues an element from the circular queue.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method removes an element from the circular queue at the front index and updates the front index using circular incrementation. 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 element at the front index 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 element at the rear index 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 length of the circular queue is zero, indicating that it is empty. It has constant time and space complexity.
7. **bool isFull()** - Checks if the circular queue is full.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method checks if the length of the circular queue is equal to its size, indicating that it is full. It has constant time and space complexity.
8. **int getSize()** - Returns the size (number of elements) of the circular queue.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method returns the length of the circular queue, representing its size. It has constant time and space complexity.
9. **void display()** - Displays the elements in the circular queue.
    - **Time Complexity: O(k), where k is the size of the circular queue.**
    - **Space Complexity: O(1)**
    - **Explanation:** This method iterates through the elements in the circular queue and displays them. The time complexity is linear with respect to the size of the queue, and the space complexity is constant.
10. **~CircularQueue()** - Destructor to free the dynamically allocated memory.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This destructor deallocates the memory used by the circular queue array. It has constant time and space complexity.