**Question**

A class CQueue is defined to implement a circular queue. Define the class CQueue with methods to check if the queue is full or empty, to add and remove elements, and to display the status of the queue.

Class name: CQueue  
Data member/instance variable:  
SIZE: to store the size of the circular queue  
front: to keep track of the front index of the queue  
rear: to keep track of the rear index of the queue  
items[]: an array to store the elements of the queue

Member functions/methods:  
CQueue(): constructor to initialize the data members  
boolean isFull(): returns true if the queue is full, otherwise false  
boolean isEmpty(): returns true if the queue is empty, otherwise false  
void enQueue(int element): adds an element to the queue if it is not full  
int deQueue(): removes and returns an element from the queue if it is not empty  
void display(): displays the status of the circular queue

Specify the class CQueue giving the details of the constructor, boolean isFull(), boolean isEmpty(), void enQueue(int element), int deQueue(), and void display().

**Algorithm**

Start

1.Define a class `CQueue` with the following instance variables:

- `SIZE`: an integer to store the size of the circular queue.

- `front`: an integer to keep track of the front index of the queue.

- `rear`: an integer to keep track of the rear index of the queue.

- `items[]`: an array to store the elements of the queue.

2.Define a constructor for the class `CQueue`:

- Initialize `front` to -1.

- Initialize `rear` to -1.

3.Define a method `isFull()` for the class `CQueue`:

- Return true if `front` is 0 and `rear` is `SIZE - 1`.

- Return true if `front` is `rear + 1`.

- Otherwise, return false.

4.Define a method `isEmpty()` for the class `CQueue`:

- Return true if `front` is -1.

- Otherwise, return false.

5. Define a method `enQueue(int element)` for the class `CQueue`:

- If the queue is full, print "Queue is full".

- Otherwise, if `front` is -1, set `front` to 0.

- Update `rear` to `(rear + 1) % SIZE`.

- Store `element` in `items[rear]`.

- Print "Inserted " followed by the element.

6.Define a method `deQueue()` for the class `CQueue`:

- If the queue is empty, print "Queue is empty" and return -1.

- Otherwise, retrieve the element from `items[front]`.

- If `front` is equal to `rear`, reset `front` and `rear` to -1.

- Otherwise, update `front` to `(front + 1) % SIZE`.

- Return the retrieved element.

7.Define a method `display()` for the class `CQueue`:

- If the queue is empty, print "Empty Queue".

- Otherwise, print the front index.

- Print the items in the queue from `front` to `rear`.

- Print the rear index.

8.Define a `main` method for the class `CQueue`:

- Create an object of the class `CQueue`.

- Attempt to dequeue an element (which will fail because the queue is empty).

- Enqueue elements 1 through 5.

- Attempt to enqueue element 6 (which will fail because the queue is full).

- Display the status of the queue.

- Dequeue an element and print the deleted element if it is not -1.

- Display the status of the queue.

- Enqueue element 7.

- Display the status of the queue.

- Attempt to enqueue element 8 (which will fail because the queue is full).

End

**Variable Description**

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data Type** | **Purpose** |
| SIZE | int | To store the size of the array |
| front | int | To store the front index of the queue |
| rear | int | To store the rear index of the queue |
| items[] | int[] | To store the items of the queue |
| elem | int | To store the element returned by deque() method |