Circular queue

Introduction:

Circular queue is a linear data structure. It follows FIFO principle. In circular queue the last node is connected back to the first node to make a circle. Elements are added at the rear end and the elements are deleted at front end of the **queue**.

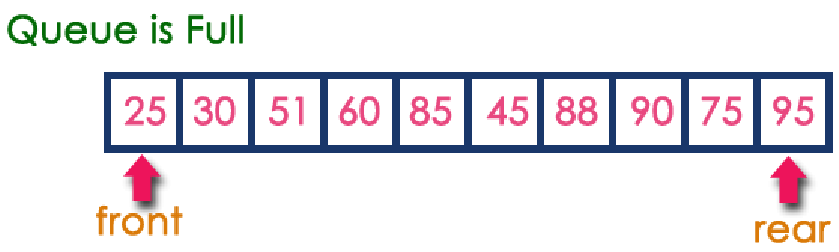
Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (front and rear). That means, we can insert at both front and rear positions and can delete from both front and rear positions.

Definition

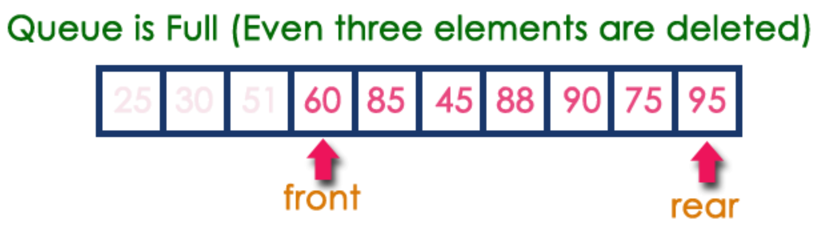
In a normal Queue Data Structure, we can insert elements until queue becomes full. But once if queue becomes full, we cannot insert the next element until all the elements are deleted from the queue.

For example consider the queue below...

After inserting all the elements into the queue.



Now consider the following situation after deleting three elements from the queue...



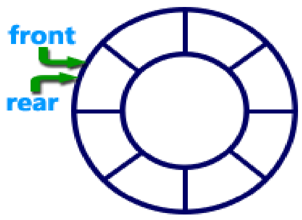
This situation also says that Queue is Full and we cannot insert the new element because, '**rear**' is still at last position. In above situation, even though we have empty positions in the queue we cannot make use of them to insert new element. This is the major problem in normal queue data structure. To overcome this problem we use circular queue data structure.

What is Circular Queue?

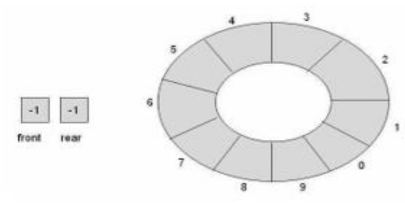
A Circular Queue can be defined as follows...

Circular Queue is also a linear data structure, which follows the principle of **FIFO**(First In First Out), but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behave like a circular data structure.

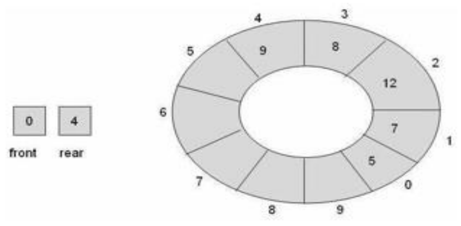
Graphical representation of a circular queue is as follows...



Empty queue:

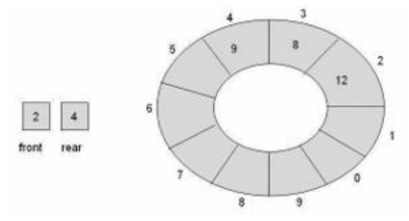


Enqueuing five items:

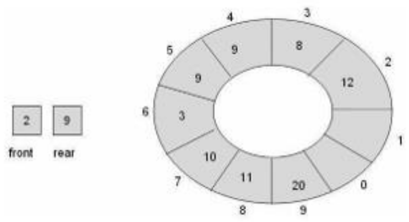


39

Dequeuing two items:



Enqueuing five items:



Application of Circular Queue

Below we have some common real-world examples where circular queues are used:

1. Computer controlled **Traffic Signal System** uses circular queue.
2. CPU scheduling and Memory management.

Comparison between Queue and Circular Queue:

BASIS FOR COMPARISON	LINEAR QUEUE	CIRCULAR QUEUE
Basic	Organizes the data elements and instructions in a sequential order one after the other.	Arranges the data in the circular pattern where the last element is connected to the first element.
Order of task execution	Tasks are executed in order they were placed before (FIFO).	Order of executing a task may change.
Insertion and deletion	The new element is added from the rear end and removed from the front.	Insertion and deletion can be done at any position.

Implementation of Circular Queue

To implement a circular queue data structure using array, we first perform the following steps before we implement actual operations.

- **Step 1:** Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.
- **Step 2:** Declare all **user defined functions** used in circular queue implementation.
- **Step 3:** Create a one dimensional array with above defined SIZE (**int cQueue[SIZE]**)
- **Step 4:** Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**)
- **Step 5:** Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on circular queue.

EnQueue(value) - Inserting value into the Circular Queue

In a circular queue, enQueue() is a function which is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at **rear** position. The enQueue() function takes one integer value as parameter and inserts that value into the circular queue. We can use the following steps to insert an element into the circular queue...

- **Step 1:** Check whether **queue** is **FULL**. (**(rear == SIZE-1 && front == 0) || (front == rear+1)**)
- **Step 2:** If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.
- **Step 3:** If it is **NOT FULL**, then check **rear == SIZE - 1 && front != 0** if it is **TRUE**, then set **rear = -1**.
- **Step 4:** Increment **rear** value by one (**rear++**), set **queue[rear]** = **value** and check '**front == -1**' if it is **TRUE**, then set **front = 0**.

DeQueue() - Deleting a value from the Circular Queue

In a circular queue, deQueue() is a function used to delete an element from the circular queue. In a circular queue, the element is always deleted from **front** position. The deQueue() function doesn't take any value as parameter. We can use the following steps to delete an element from the circular queue...

- **Step 1:** Check whether **queue** is **EMPTY**. (**front == -1 && rear == -1**)
- **Step 2:** If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
- **Step 3:** If it is **NOT EMPTY**, then display **queue[front]** as deleted element and increment the **front** value by one (**front ++**). Then check whether **front == SIZE**, if it is **TRUE**, then set **front = 0**. Then check whether both **front - 1** and **rear** are equal (**front -1** == **rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

Display() - Displays the elements of a Circular Queue

We can use the following steps to display the elements of a circular queue...

- **Step 1:** Check whether **queue** is **EMPTY**. (**front == -1**)
- **Step 2:** If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.
- **Step 3:** If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front**'.
- **Step 4:** Check whether '**front <= rear**', if it is **TRUE**, then display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i <= rear**' becomes **FALSE**.
- **Step 5:** If '**front <= rear**' is **FALSE**, then display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until'**i <= SIZE - 1**' becomes **FALSE**.
- **Step 6:** Set **i** to **0**.
- **Step 7:** Again display '**cQueue[i]**' value and increment **i** value by one (**i++**). Repeat the same until '**i <= rear**' becomes **FALSE**.

Home works

1. Write a C++ program to implement queue using array?
2. Write a C++ program to implement circular queue using arrays?