**Practical No: 13(E)**

**Practical Title:** Perform operations on Circular queue.

**Aim:** Pizza parlor accepting maximum M orders. Orders are served in first come first served basis. Order once placed cannot be cancelled. Write C++ program to simulate the system using **circular queue** using array

**Pre-requisite:**

* + Knowledge of Circular Queue
  + Types of Circular queue
  + Knowledge of Singly and double Circular queue and different operations.

**Objective:** To simulate circular queue with functions to add and delete elements from Circular queue.

**Input :**

* Accept the order
* Elements in queue

**Outcome:**

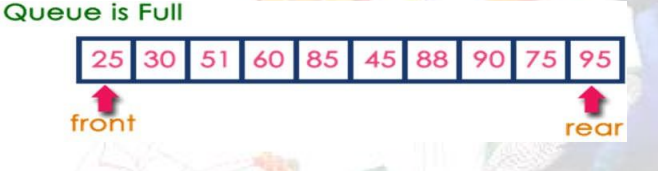
* + Result of Circular queue with functions to accept or cancel Pizza order.

**Theory:**

Why Circular Queue?

In a normal Queue Data Structure, we can insert elements until queue becomes full. But once if queue becomes full, we can’t 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 can’t insert the new element because, 'rear' is still at last

position. In above situation, even though we have empty positions in the queue we can’t 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.

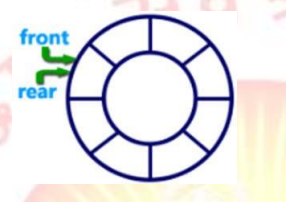
• What is a Circular Queue?

A Circular Queue can be defined as follows...

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out)

principle and the last position is connected back to the first position to make a circle.

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



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.

**Algorithms :**

Write your own algorithms

**Flowchart :**

Draw flowchart for above algorithms

**Conclusion:** Hence we have learned how to implement Circular Queue and perform various operations on it.

**Question Bank:**

1 .What is Circular queue?

2. Types of Circular queue

3. What is double ended queue?

4. How to insert the new node in Circular Link List?

5. How to delete the node from front of Circular Link List ?..!+