



# Batch: A3 Roll No.: 16010421075 Experiment

**No.: 4**

**Aim:** Write a menu driven program to implement a static cicular queue using supporting following operations.

1. Create empty queue,
2. Insert an element on the queue,
3. Delete an element from the queue,
4. Display front element
5. Display all elements of the queue.



**Resources Used:** Turbo C/ C++/JAVA editor and compiler.



# Theory:

**What is Linear Queue?**

A linear queue is an ordered list in which insertion and deletion happen two different ends. The insertion happens from the *rear* and the deletion takes place at the *front*. It works with the FIFO concept i.e. first in first out. Basic operations of Queue are enqueue, dequeue, isempty, etc.

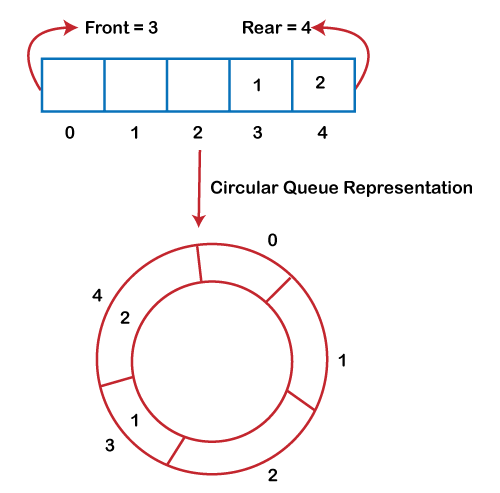
# What is a Circular Queue?

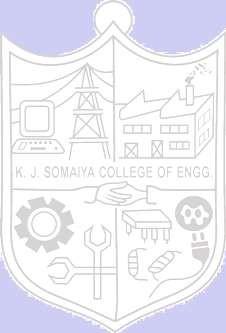
A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a ***Ring Buffer***.

# Why was the concept of the circular queue introduced?

There was one limitation in the array implementation of Linear Queue:

If the rear reaches to the end position of the Queue then there might be possibility that some vacant spaces are left in the beginning which cannot be utilized. So, to overcome such limitations, the concept of the circular queue was introduced.



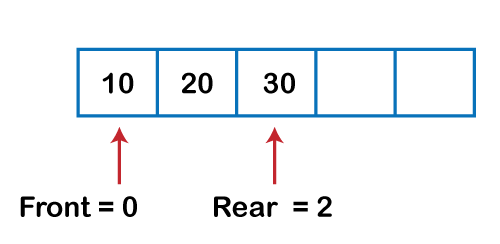
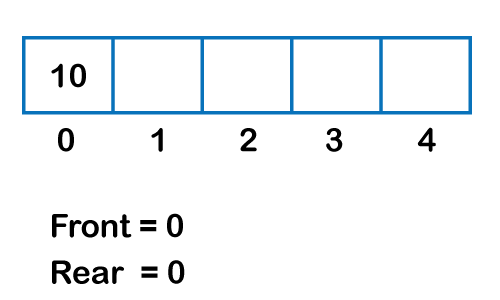
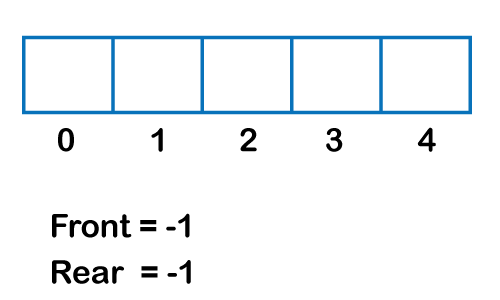
As we can see in the above image, the rear is at the last position of the Queue and front is pointing somewhere rather than the 0th position. In the above array, there are only two elements and other three positions are empty. The rear is at the last position of the Queue; if we try to insert the element then it will show that there are no empty spaces in the Queue. There is one solution to avoid such wastage of memory space by shifting both the elements at the left and adjust the front and rear end accordingly. It is not a practically good approach because shifting all the elements will consume lots of time. The efficient approach to avoid the wastage of the memory is to use the circular queue data structure.

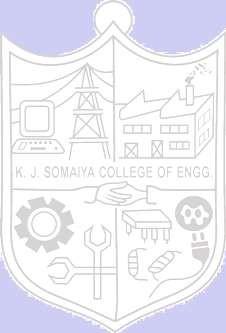
# Operations on Circular Queue

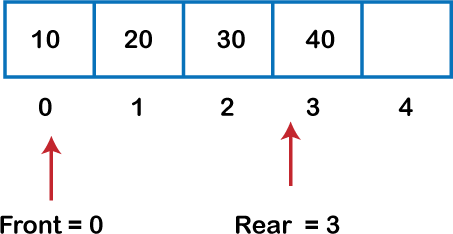
The following are the operations that can be performed on a circular queue:

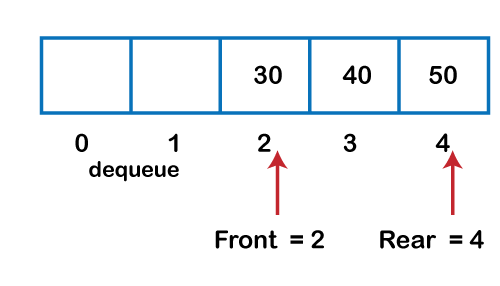
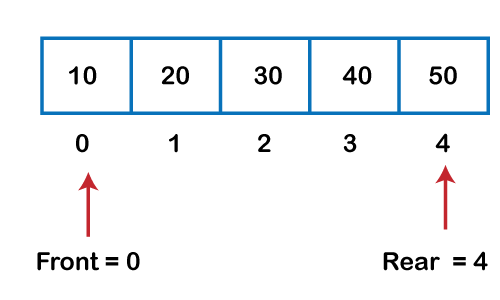
* **Front:** It is used to get the front element from the Queue.
* **Rear:** It is used to get the rear element from the Queue.
* **Enqueue(value):** This function is used to insert the new value in the Queue. The new element is always inserted from the rear end.
* **Dequeue():** This function deletes an element from the Queue. The deletion in a Queue always takes place from the front end.

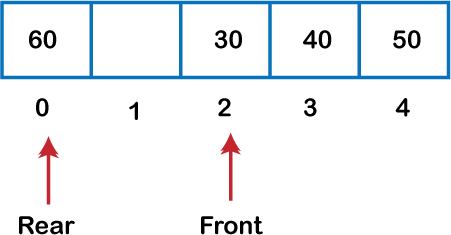
# Let's understand the enqueue and dequeue operation through the diagrammatic representation.

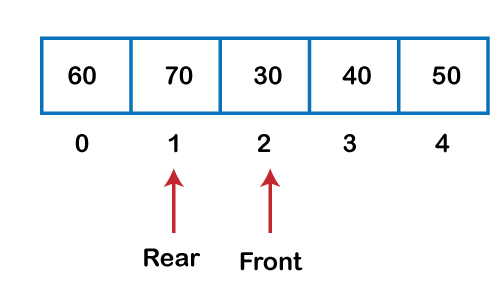


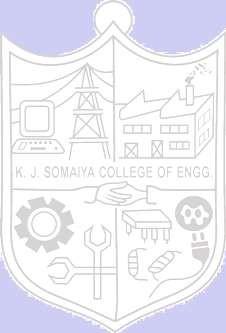










*Steps for Implementing Queue Operations*

Enqueue(value) and [Dequeue()](https://www.simplilearn.com/tutorials/data-structure-tutorial/dequeue-in-data-structure) are the primary operations of the queue, which allow you to manipulate the data flow. These functions do not depend on the number of elements inside the queue or its size; that is why these operations take constant execution time, i.e., O(1) [time- complexity]. Here, you will deal with steps to implement queue operations:

# Enqueue(x) Operation

You should follow the following steps to insert (enqueue) a data element into a circular queue - Step 1: Check if the queue is full (Rear + 1 % Maxsize = Front)

Step 2: If the queue is full, there will be an Overflow error

Step 3: Check if the queue is empty, and set both Front and Rear to 0

Step 4: If Rear = Maxsize - 1 & Front != 0 (rear pointer is at the end of the queue and front is not at 0th index), then set Rear = 0

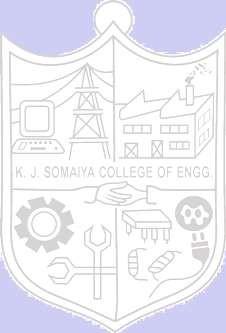
Step 5: Otherwise, set Rear = (Rear + 1) % Maxsize

Step 6: Insert the element into the queue (Queue[Rear] = x) Step 7: Exit

Now, you will explore the Enqueue() operation by analyzing different cases of insertion in the

circular queue:

# Dequeue() Operation

Obtaining data from the queue comprises two subtasks: access the data where the front is pointing and remove the data after access. You should take the following steps to remove data from a circular queue -

Step 1: Check if the queue is empty (Front = -1 & Rear = -1)

Step 2: If the queue is empty, Underflow error

Step 3: Set Element = Queue[Front]

Step 4: If there is only one element in a queue, set both Front and Rear to -1 (IF Front = Rear, set Front = Rear = -1)

Step 5: And if Front = Maxsize -1 set Front = 0 Step 6: Otherwise, set Front = Front + 1

Step 7: Exit

# Applications of Circular Queue

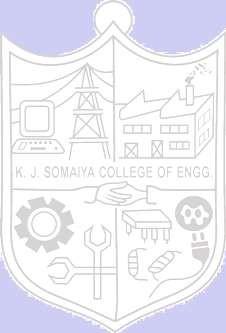
**The circular Queue can be used in the following scenarios:**

* **Memory management:** The circular queue provides memory management. As we have already seen that in linear queue, the memory is not managed very efficiently. But in case

of a circular queue, the memory is managed efficiently by placing the elements in a location which is unused.

* **CPU Scheduling:** The operating system also uses the circular queue to insert the processes and then execute them.
* **Traffic system:** In a computer-control traffic system, traffic light is one of the best examples of the circular queue. Each light of traffic light gets ON one by one after every jinterval of time. Like red light gets ON for one minute then yellow light for one minute and then green light. After green light, the red light gets ON.



**Activity:** Students are expected to implement circular queue using **array.**

# NOTE : All functions should be able to handle boundary(exceptional) conditions.



**Results:** A program implementing solution depicting the correct behaviour of circular queue and capable of handling all possible exceptional conditions and the same is reflecting clearly in the output.

#include <stdio.h>

#include<stdlib.h>

#define MAX 50

int a[MAX];

int rear = - 1;

int front = - 1;

int main()

{

    int choice;

    while (1)

    {

        printf("\n0.Create a queue \n");

        printf("1.Insert element to queue \n");

        printf("2.Delete element from queue \n");

        printf("3.Display front element \n");

        printf("4.Display all elements of queue \n");

        printf("5.Quit \n");

        printf("Enter your choice : ");

        scanf("%d", &choice);

        switch (choice)

        {

            case 0:

            create();

            break;

            case 1:

            equeue();

            break;

            case 2:

            dequeue();

            break;

            case 3:

            display();

            break;

            case 4:

            full();

            break;

            case 5:

            exit(1);

            default:

            printf("Wrong choice \n");

        }

    }

}

void create()

{

    printf("Queue is created");

}

void equeue()

{

    int add\_item;

    if (rear == MAX - 1)

    printf("Queue Overflow \n");

    else

    {

        front = 0;

        printf("Inset the element in queue : ");

        scanf("%d", &add\_item);

        rear = rear + 1;

        a[rear] = add\_item;

    }

}

void dequeue()

{

    if (front == - 1 || front > rear)

    {

        printf("Queue Underflow \n");

        return ;

    }

    else

    {

        printf("Element deleted from queue is : %d\n", a[front]);

        front = front + 1;

    }

}

void display()

{

    printf("%d",a[front]);

}

void full()

{

    int i;

    if (front == - 1)

        printf("Queue is empty \n");

    else

    {

        printf("Queue is : \n");

        for (i = front; i <= rear; i++)

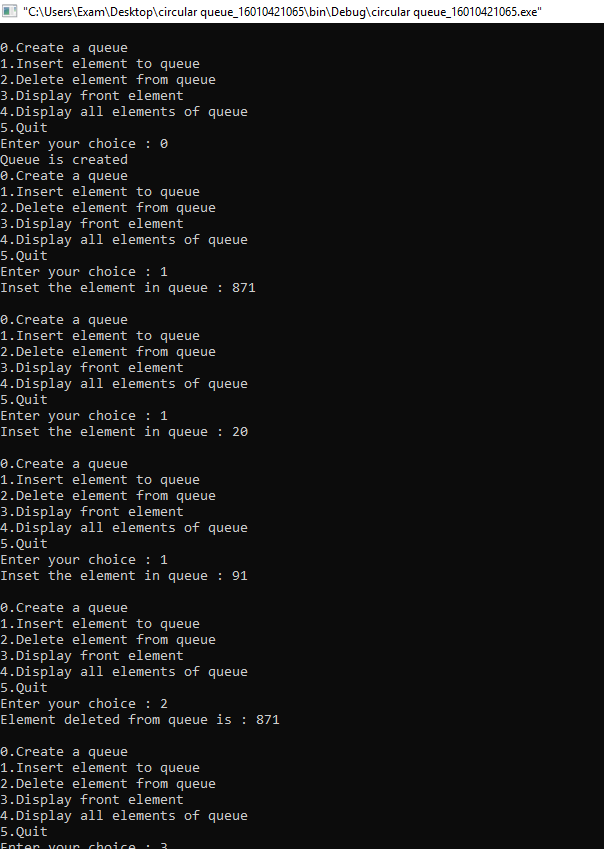
            printf("%d", a[i]);

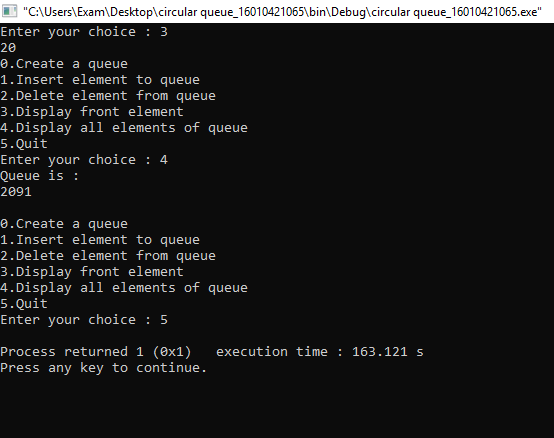
        printf("\n");

    }

}

Output

****

****



# Outcome: CO2: Demonstrate the implementation of Static Circular Queue



**Conclusion:** The circular queue has more advantages than a linear queue. Other advantages of circular queue are:

* + **Easier for insertion-deletion:** In the circular queue, elements can be inserted easily if there are vacant locations until it is not fully occupied, whereas in the case of a linear queue insertion is not possible once the rear reaches the last index even if there are empty locations present in the queue.
  + **Efficient utilization of memory:** In the circular queue, there is no wastage of memory as it uses the unoccupied space, and memory is used properly in a valuable and effective manner as compared to a linear queue.
  + **Ease of performing operations:** In the linear queue, **FIFO** is followed, so the element inserted first is the element to be deleted first. This is not the scenario in the case of the circular queue as the rear and front are not fixed so the order of insertion-deletion can be changed, which is very useful.



.

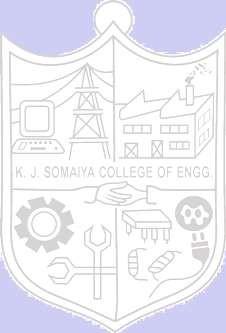
# Grade: AA / AB / BB / BC / CC / CD /DD:



**Signature of faculty in-charge with date :**



# References:

**Books/ Journals/ Websites:**

* Y. Langsam, M. Augenstin and A. Tannenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002.