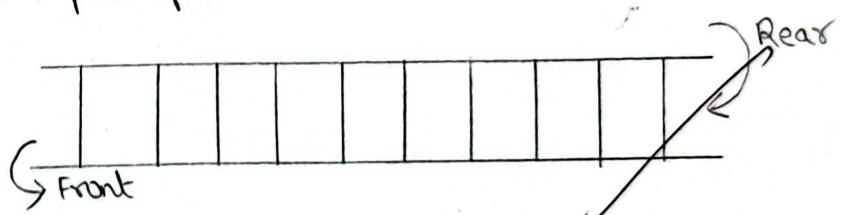
Introduction

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. The insertion is performed at one end and deletion is performed at another end. In an queue data structure, the insertion operation is performed at a position which is known as 'rear'. In queue data structure, the insertion and deletion operations are performed based on FIFO (first In First Out) principle.



A real world enample of queue can be a single lane one way road, where the vechile enters first, enists first.

Point-to-Point on Queue

→ It is a non-primitive linear data structure that stores different data items temporarily and performs different operations on those data items on the FIFO (first-in-first-out) sequence.

→ Data can be inserted only at one end, called the rear, and deleted from the other end, called the front.

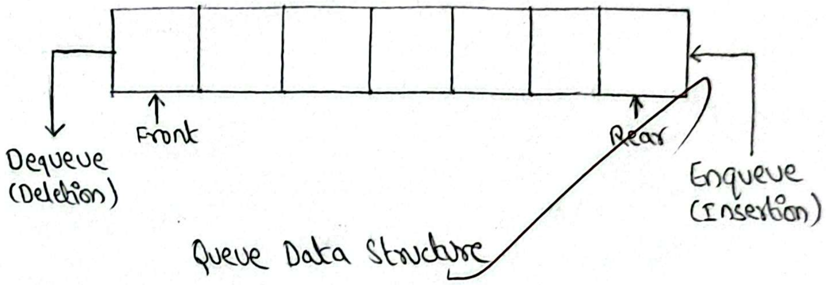
→ As there are two ends for insertions and deletions operations, the first added item will be firstly removed, so called First-In-First-Out type of list.

→ A real-world enample of queue can be a single-lane one-way road, where the vechicle enters first, enists first or the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

Queue as an ADT

→ A queue contains data elements of any particular type with a finite sequence in which the following operations can be carried out:

1) Enqueue or Insertion : which inserts an element at the end(rear) of the queue.
2) Dequeue or deletion : which deletes an element at the start(front) of the queue.



Queue Implementation

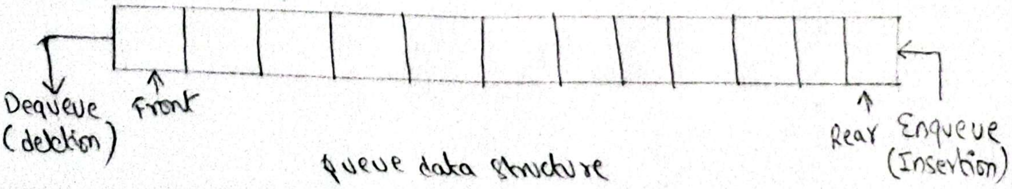
1) Static Implementation using array
2) Dynamic Implementation using linked list

Types of Queue

1) Linear queue
2) Circular queue
3) Priority queue
4) Double ended queue

1) Linear queue

→ It is a homogenous collection of elements which stores data in the linear fashion and performs insertions and deletion of items from two ends, rear and front respectively.



→ for the enque operation (when the queue is not full), we increase the value of Rear inden and place the new element in the position to by REAR.

→ for the dequeue operation (when the queue is not empty), we return the value pointed to by FRONT and increase the FRONT inden.

→ Empty of queue : If FRONT = -1 or FRONT > REAR, it indicates queue is empty (Underflow condition).

→ Full of the queue : If REAR = MAXSIZE - 1, it indicates queue is full (overflow condition).

Note :- Front always points to the first Element.

Algorithm :- Enqueue

→ Let Mansize be the manimum size of the linear queue, Front and Rear are two pointers that indicates the first and last data elements in the queue.

```
1) Initialize Front = Rear = -1
2) Check the overflow condition:
       If (rear == MAXSIZE-1)
           Print "Queue is full" and Enist
3) If front = -1 and Rear = -1, set Front = Rear = 0
4) Otherwise, Increment rear by 1 ie rear = rear + 1
5) Read the element as "item" to be inserted in the
   queue.
6) Set Queue [Rear] = item
7) Stop
```

Algorithm :- Dequeue

```
1) Check the underflow condition
        If (front == -1 or front > Rear)
            Print "Queue is empty" and enist
        Else
            Set item = Queue [front]
2) If (Front == Rear), reinitialize front :=-1, rear :=-1
3) Otherwise, front = front+1
4) Stop
```

Algorithm :- Displaying data items

```
1) Check for the empty of queue.
        if Front :=-1, then print "Queue is empty" and enist.
2) for (i = Rear; i <= front; i--)
        Print Queue [i]
3) Stop
```