جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم المن السيبراني

Department of Cyber Security

Subject: Data Structure

Class: Second

Lecturer: Msc :Muntather AL-mussawee

Lecture: (3)

Queue

Study Year: 2025-2026

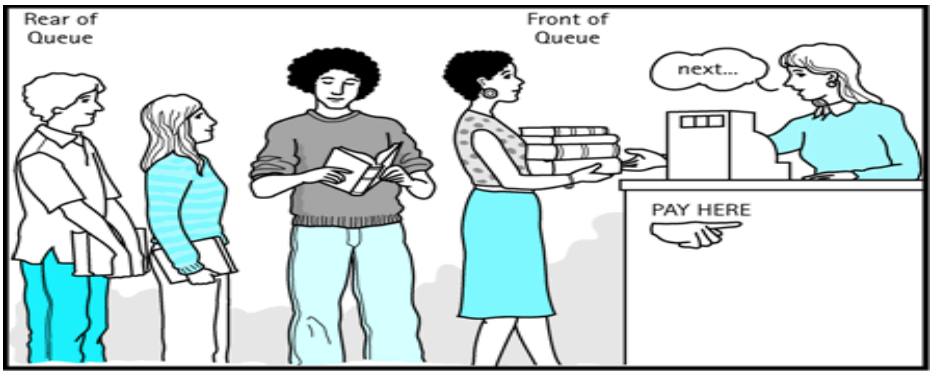
QUEUE

A queue is linear data structure and collection of elements. A queue is another special kind of list, where items are inserted at one end called the **rear** and deleted at the other end called the **front**. The principle of queue is a **"FIFO" or "First-in-first-out"**.

Queue is an abstract data structure. A queue is a useful data structure in programming. **It is similar to the ticket queue outside a cinema hall**, where the first person entering the queue is the first person who gets the ticket.

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first.

More real-world examples can be seen as queues at the ticket windows and bus-stops and our college library.



The operations for a queue are analogues to those for a stack; the difference is that the insertions go at the end of the list, rather than the beginning.

Operations on QUEUE:

A queue is an object or more specifically an abstract data structure (ADT) that allows the following operations:

- **Enqueue or insertion:** which inserts an element at the end of the queue.
- **Dequeue or deletion:** which deletes an element at the start of the queue.

Queue operations work as follows:

1. Two pointers called FRONT and REAR are used to keep track of the first and last elements in the queue.
2. When initializing the queue, we set the value of FRONT and REAR to 0.
3. On enqueueing an element, we increase the value of REAR index and place the new element in the position pointed to by REAR.
4. On dequeueing an element, we return the value pointed to by FRONT and increase the FRONT index.
5. Before enqueing, we check if queue is already full.
6. Before dequeuing, we check if queue is already empty.
7. When enqueing the first element, we set the value of FRONT to 1.
8. When dequeing the last element, we reset the values of FRONT and REAR to 0.

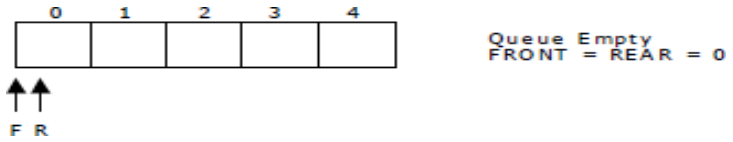
Representation of Queue (or) Implementation of Queue:

The queue can be represented in two ways:

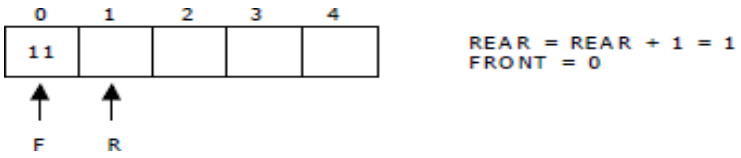
1. Queue using Array
2. Queue using Linked List

1. Queue using Array:

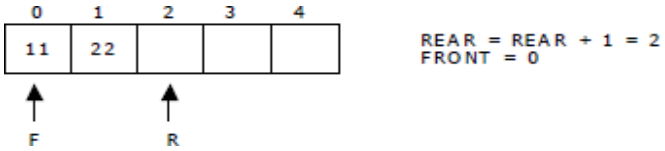
Let us consider a queue, which can hold maximum of five elements. Initially the queue is empty.



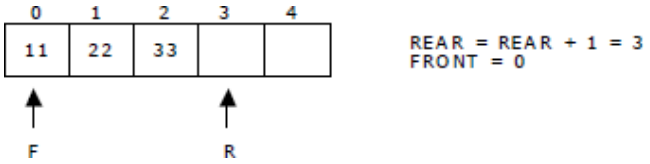
Now, insert 11 to the queue. Then queue status will be:



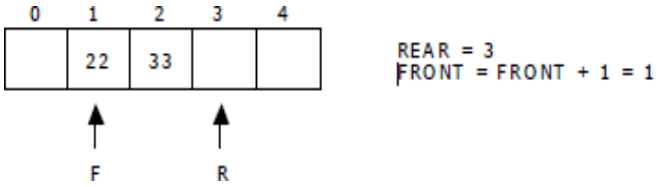
Next, insert 22 to the queue. Then the queue status is:



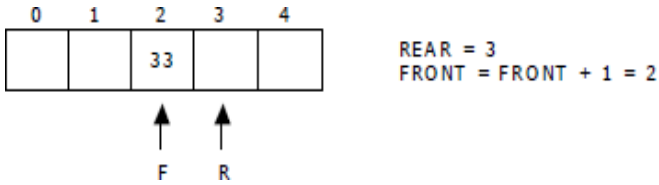
Again insert another element 33 to the queue. The status of the queue is:



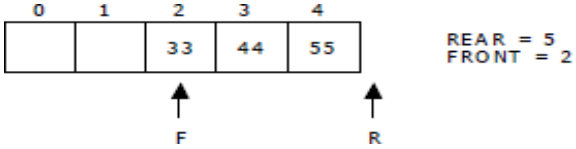
Now, delete an element. The element deleted is the element at the front of the queue.So the status of the queue is:



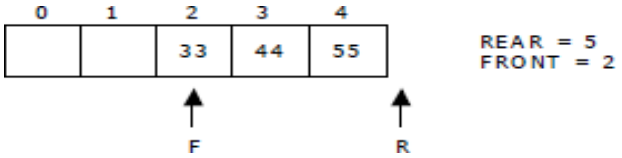
Again, delete an element. The element to be deleted is always pointed to by the FRONT pointer. So, 22 is deleted. The queue status is as follows:



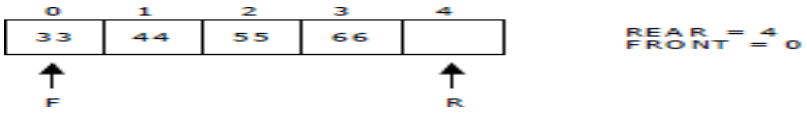
Now, insert new elements 44 and 55 into the queue. The queue status is:



Next insert another element, say 66 to the queue. We cannot insert 66 to the queue as the rear crossed the maximum size of the queue (i.e., 5). There will be queue full signal. The queue status is as follows:



Now it is not possible to insert an element 66 even though there are two vacant positions in the linear queue. To overcome this problem the elements of the queue are to be shifted towards the beginning of the queue so that it creates vacant position at the rear end. Then the FRONT and REAR are to be adjusted properly. The element 66 can be inserted at the rear end. After this operation, the queue status is as follows:



This difficulty can overcome if we treat queue position with index 0 as a position that comes after position with index 4 i.e., we treat the queue as a **circular queue.**

Queue operations using array:

a.enqueue() or insertion():which inserts an element at the end of the queue.

```
void insertion()
{
   if(rear==max)
        printf("\n Queue is Full");
   else
   {
         printf("\n Enter no %d:",j++);
         scanf("%d",&queue[rear++]);
    }
}
```

Algorithm: Procedure for insertion():

Step-1:START
Step-2: if rear==max then
Write 'Queue is full'
Step-3: otherwise
3.1: read element 'queue[rear]'
Step-4:STOP

b.dequeue() or deletion(): which deletes an element at the start of the queue.

```
void deletion()
{
    if(front==rear)
   {
      printf("\n Queue is empty");
   }
   else
    {
      printf("\n Deleted Element is
              %d",queue[front++]);
           x++;
    } }
```

Algorithm: procedure for deletion():

Step-1:START
Step-2: if front==rear then
Write' Queue is empty'
Step-3: otherwise
3.1: print deleted element
Step-4:STOP

c.dispaly(): which displays an elements in the queue.

```
void deletion()
{
    if(front==rear)
   {
      printf("\n Queue is empty");
   }
         else
         {
            for(i=front; i<rear; i++)
            {
               printf("%d",queue[i]);
               printf("\n");
            }
         }
}
```

Algorithm: procedure for deletion():

Step-1:START
Step-2: if front==rear then
Write' Queue is empty'
Step-3: otherwise
3.1 : for i=front to rear then
3.2 : print 'queue[i]'
Step-4:STOP

2. Queue using Linked list:

We can represent a queue as a linked list. In a queue data is deleted from the front end and inserted at the rear end. We can perform similar operations on the two ends of alist. We use two pointers *front* and *rear* for our linked queue implementation.
The linked queue looks as shown in figure:

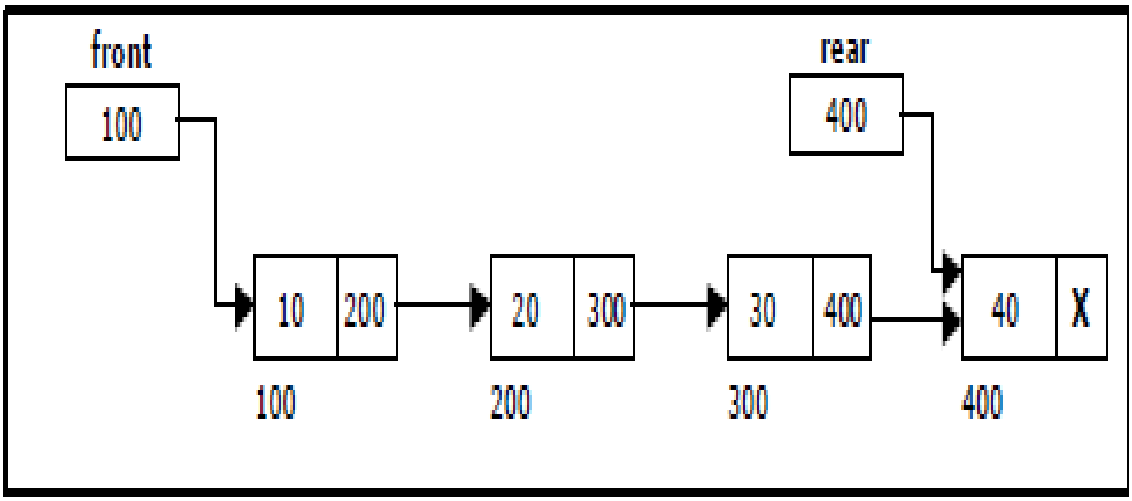


Figure ; Linked Queue representation

Applications of Queue:

1. It is used to schedule the jobs to be processed by the CPU.
2. When multiple users send print jobs to a printer, each printing job is kept in the printing queue. Then the printer prints those jobs according to first in first out (FIFO) basis.
3. Breadth first search uses a queue data structure to find an element from a graph.