Book

A Simplified Approach
to
Data Structures

Prof.(Dr.)Vishal Goyal, Professor, Punjabi University Patiala

Dr. Lalit Goyal, Associate Professor, DAV College, Jalandhar

Mr. Pawan Kumar, Assistant Professor, DAV College, Bhatinda

Shroff Publications and Distributors

Edition 2014

QUEUES

Contents for Today’s Lecture

- Introduction to Queue
- Operations on the Queue
- Memory Representation

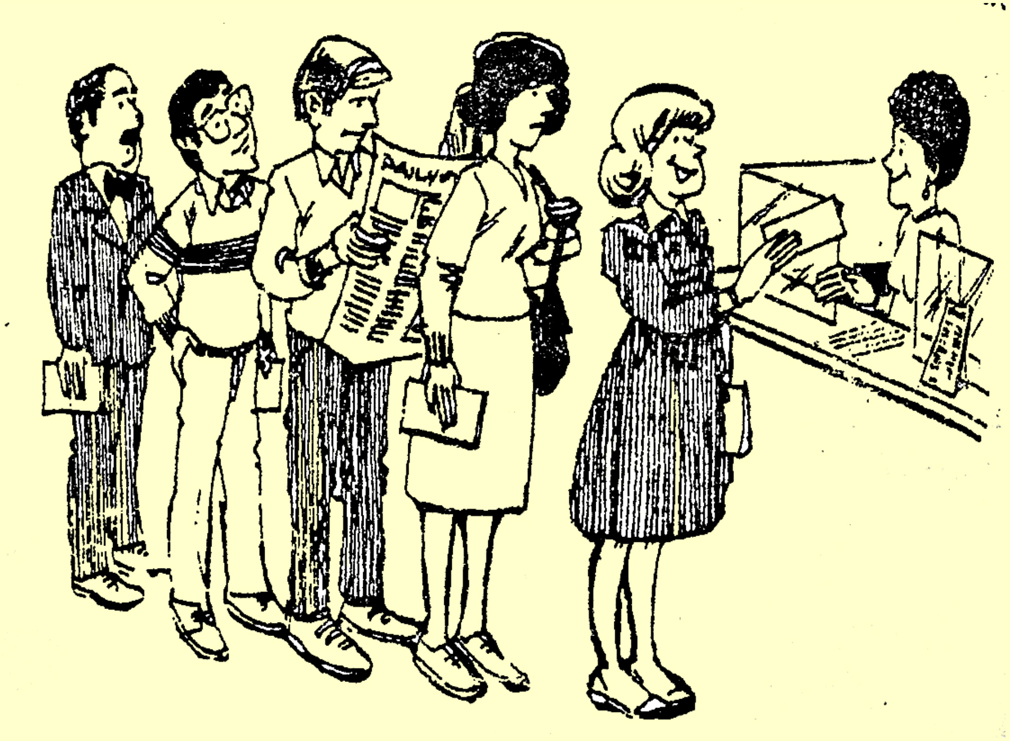
Introduction

- **Queue** is a linear data structure.
- Queue has two ends Front and Rear.
- Element can be added at R**ear** of the queue and the element can be removed from the Front end of the queue.
- The elements of a queue are processed in the same order as they were added into the queue.
- Queues are also known as **FIFO (First In Order First Out)** list or FCFS (**First Come First Serve basis) list**

For Example,

Queue at the ticket counter of railway station, bank, post office, bill deposit counter etc.

Introduction (continued)



People waiting in a Queue

Queue is a very important data structure as it has various applications in programming(system programming as well as application programming).

Operations on the Queue

Two basic operations which are performed on queue are:

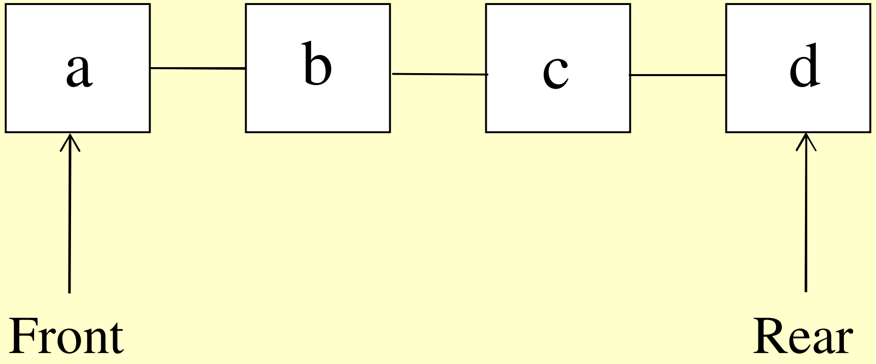
- Insertion
- Deletion

Insertion operation refers to addition of a new element at the **Rear** of the queue. An attempt to insert an element in a filled queue (having no space) results in a state called **overflow** condition.

Deletion operation refers to the removal of an element from the **Front** of the queue. An attempt to delete an element from the empty queue(having no element) is known as **underflow** condition.

Operations on the Queue (continued)

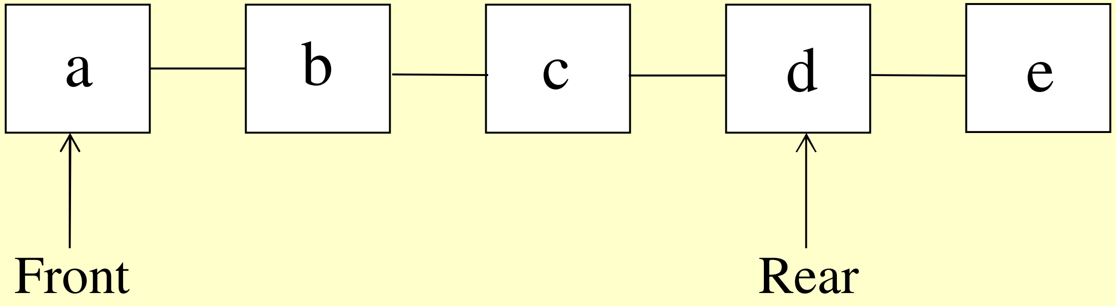
Consider a list of four elements **(a, b, c, d)** where **a** is the front element and **d** is rear element.



A queue with Four Elements

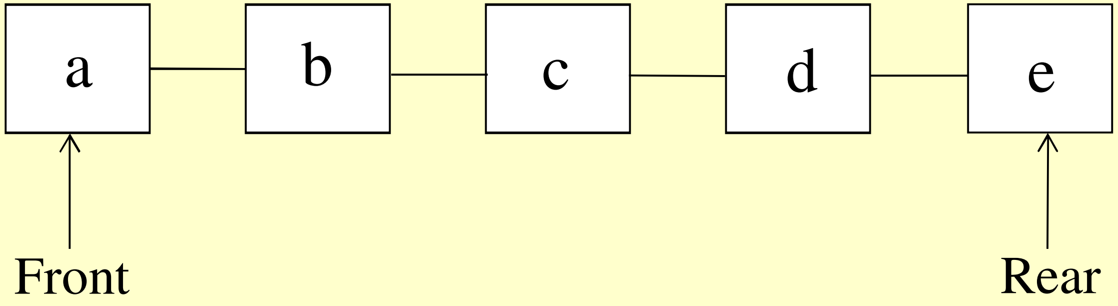
Operations on the Queue (continued)

New element **e** will be inserted at the rear end, here, after the element **d** as shown in figure below:



Inserting an element in the queue

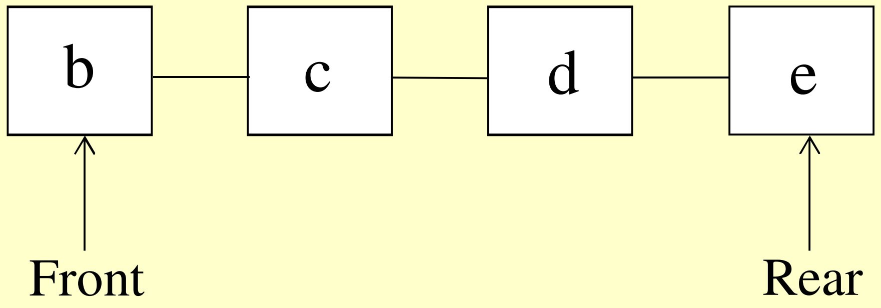
Only element at the front end can be deleted from the queue. Here, the element **a** will be deleted from the queue as shown:



Deleting an element from the queue

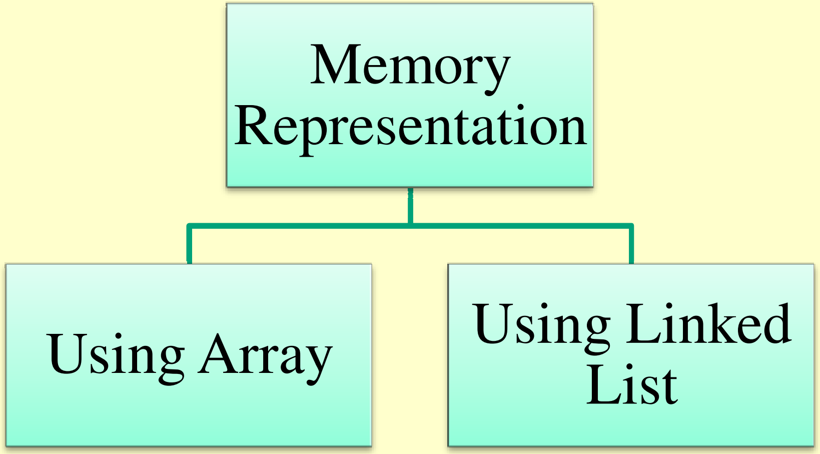
Operations on the Queue (continued)

Another element that can be deleted from the queue is **b** as shown below:



Deleting another element from the queue

Memory Representation of Queue



Memory Representation

Using Array

Using Linked List

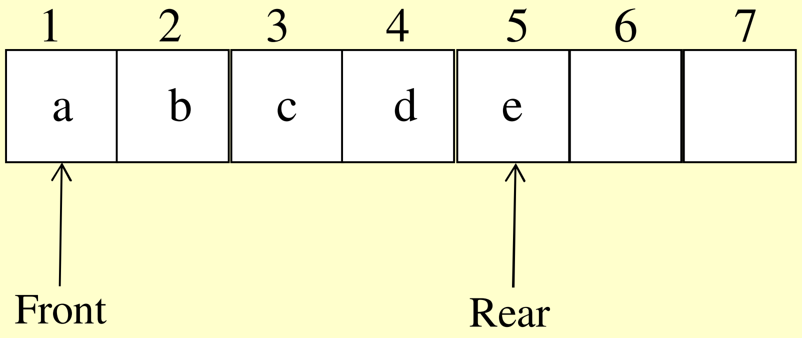
Array Representation of Queue

- The elements of the queue must be of same type (homogenous).
- Maximum size of the queue must be defined before implementing it as array is static data structure.
- Queue grows and shrinks over time but an array has constant size.
- **First In First Out (FIFO)** order must be maintained using two variables **Front** and **Rear.**

Array Representation of Queue (Cont..)

Insertion

The new element can be added at the **Rear** end after incrementing the variable **Rear**



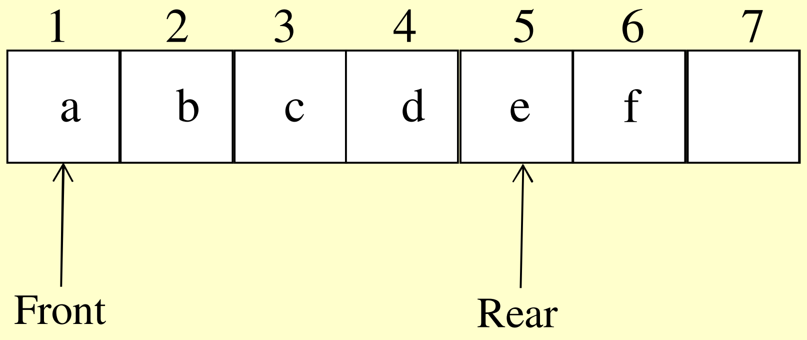
1	2	3	4	5	6	7
a	b	c	d	e		

Front

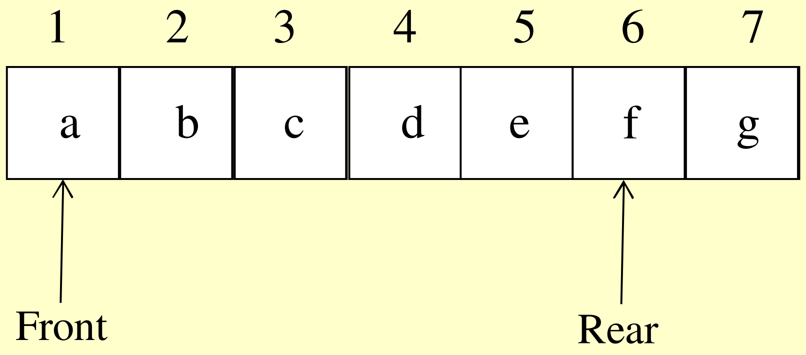
Rear

Queue having 5 elements

Array Representation of Queue (Cont..)



Inserting an element f at index 6 in the queue

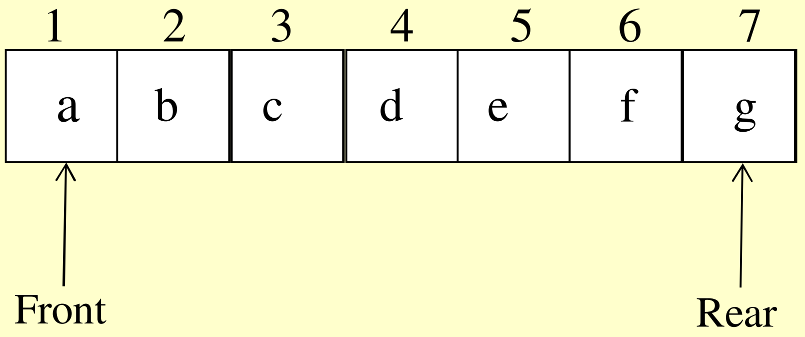


Inserting an element g at index 7 in the queue

Array Representation of Queue (Cont..)

Deletion

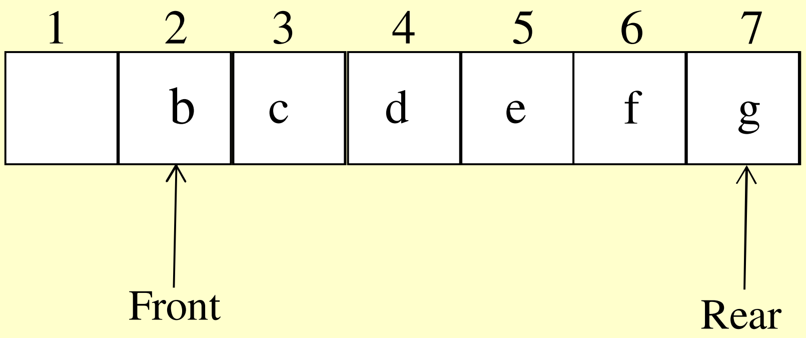
The only element at the front of the queue can be removed and variable **Front** of the queue will be incremented by one.



1	2	3	4	5	6	7
a	b	c	d	e	f	g

Deletion of an element from the queue

Array Representation of Queue (Cont..)



1	2	3	4	5	6	7
	b	c	d	e	f	g

Front

Rear

Deletion of another element from the queue

Array Representation of Queue (Cont..)

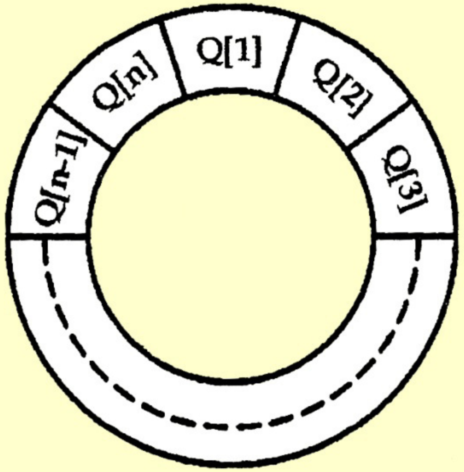
In the above mentioned queue, the front positions start vacating during the deletion process. To make full use of space, two cases arise,

- Shift all the elements in the left after each deletion position.
- Use circular array to implement queue termed as **circular queue**.

Shifting elements in the front positions is not efficient in terms of time, so the circular queue is very efficient option.

Circular Queue

- An array in the form of circle is used.
- After the last index, there it the turn of first index making it circular.



A Circular Array of Size *'n'*

Operations on Circular Queue

- **Insertion**
- **Deletion**

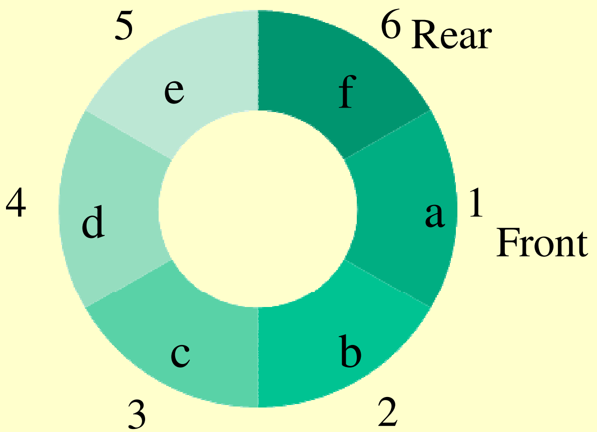
Insertion in Circular Queue

- Before inserting an element, the **overflow** condition must be checked.
- If last indexed position is occupied, element will be inserted at the first index.

Insertion in Circular Queue

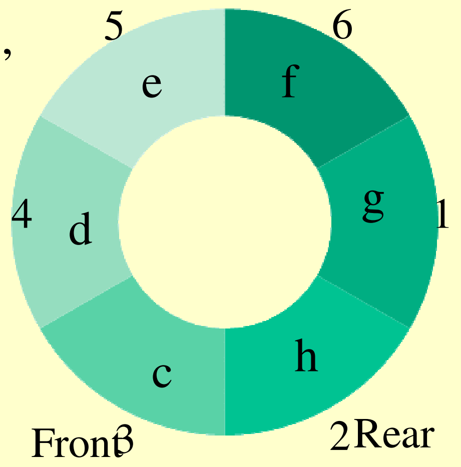
Insertion of an element ‘Data’ into the circular queue. The size of the Queue is ‘n’ i.e. ‘n’ number of elements can be accommodated in the Queue. Here, lower index is taken as ‘1’ and upper index is taken as ‘n’.

```
Step 1:   If Front = 1 and Rear = n Then
              Print “Queue is full, Overflow Condition”
              Exit
          [End If]
```



Front = 1   Rear = 6

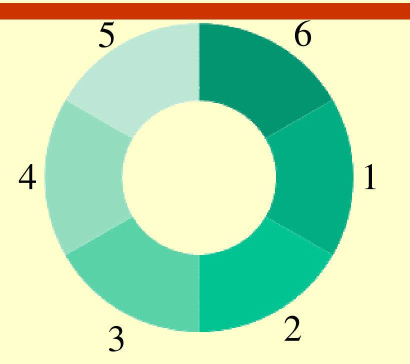
```
Step 2:   If Front = Rear + 1 Then
                     Print “Queue is full, Overflow Condition”
                     Exit
          [End If]
```



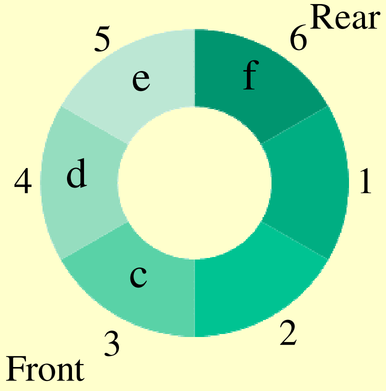
Front = Rear + 1

Insertion in Circular Queue

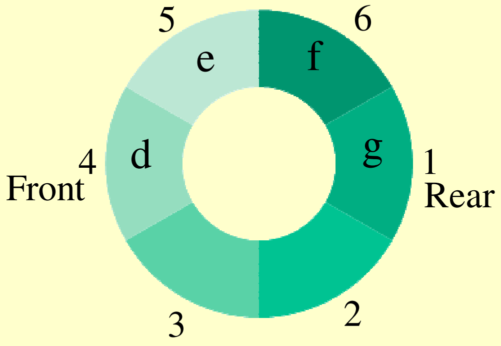
```
Step 3:   If Rear = Null Then
              Set Front = 1 and Rear = 1

          If Rear = n Then
              Set Rear = 1

          Else
              Set Rear = Rear + 1
          [End If]
Step 4:   Set Q[Rear] = Data
Step 5:   Exit
```



Front = 0 Rear = 0



Rear = 1



Rear = Rear + 1

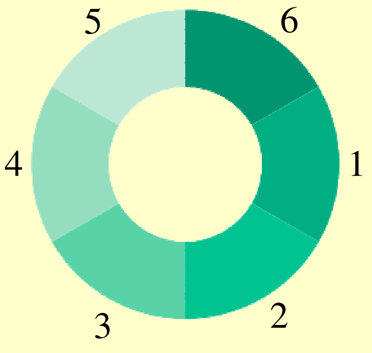
Deletion in Circular Queue

- Before deleting an element, the **underflow** condition must be checked.
- If **Front** is reached at last index, after deletion **Front** will refer to the first index.

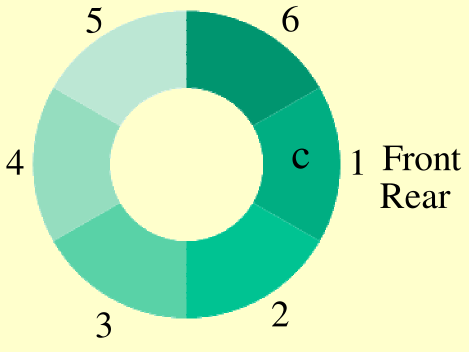
Deletion in Circular Queue

Deleting an Element from the Queue. The size of the Queue is ‘n’ i.e. ‘n’ number of elements can be accommodated in the Queue. Here, lower index is taken as ‘1’ and upper index is taken as ‘n’.

```
Step 1:    If **Front = Null** Then
               Print: “Queue is empty, Underflow Condition”
               Exit
           [End If]

Step 2:    Set **Data = Q[Front]**

Step 3:    If **Front = Rear** Then
               Set **Front = Null** and **Rear = Null**
```



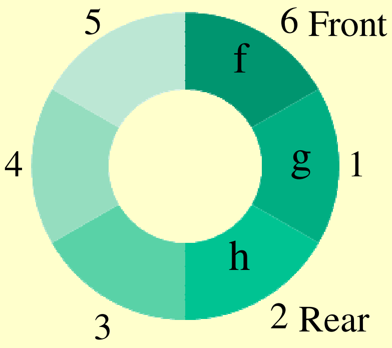
Front = 0   Rear = 0



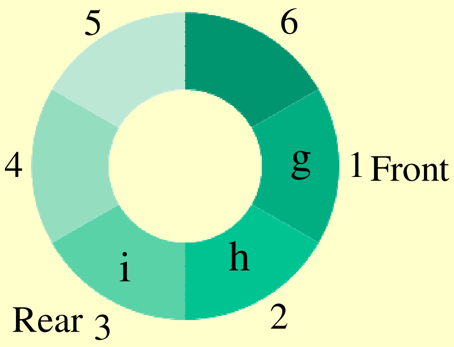
Front= Rear

Deletion in Circular Queue

```
        Else If **Front = n** Then
            Set **Front = 1**

        Else
            Set **Front = Front + 1**
        [End If ]

Step 4: Exit
```



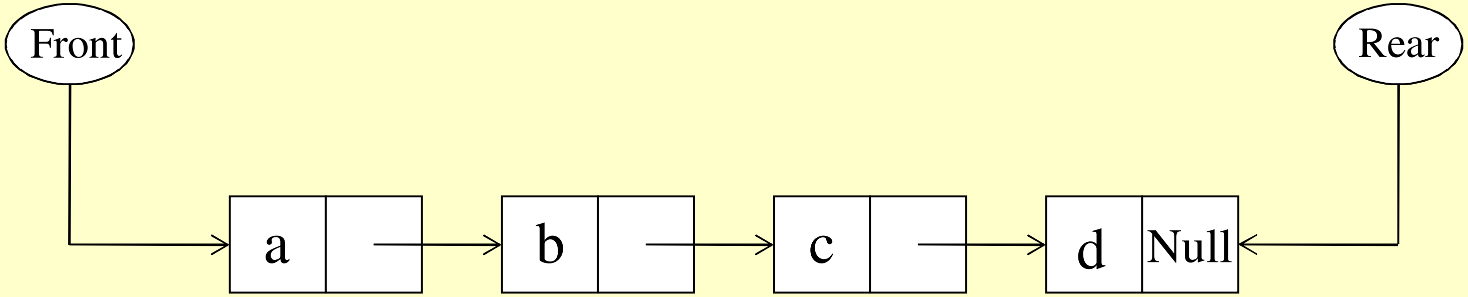
Front= 6



Front= 1

Linked List Representation of Queue

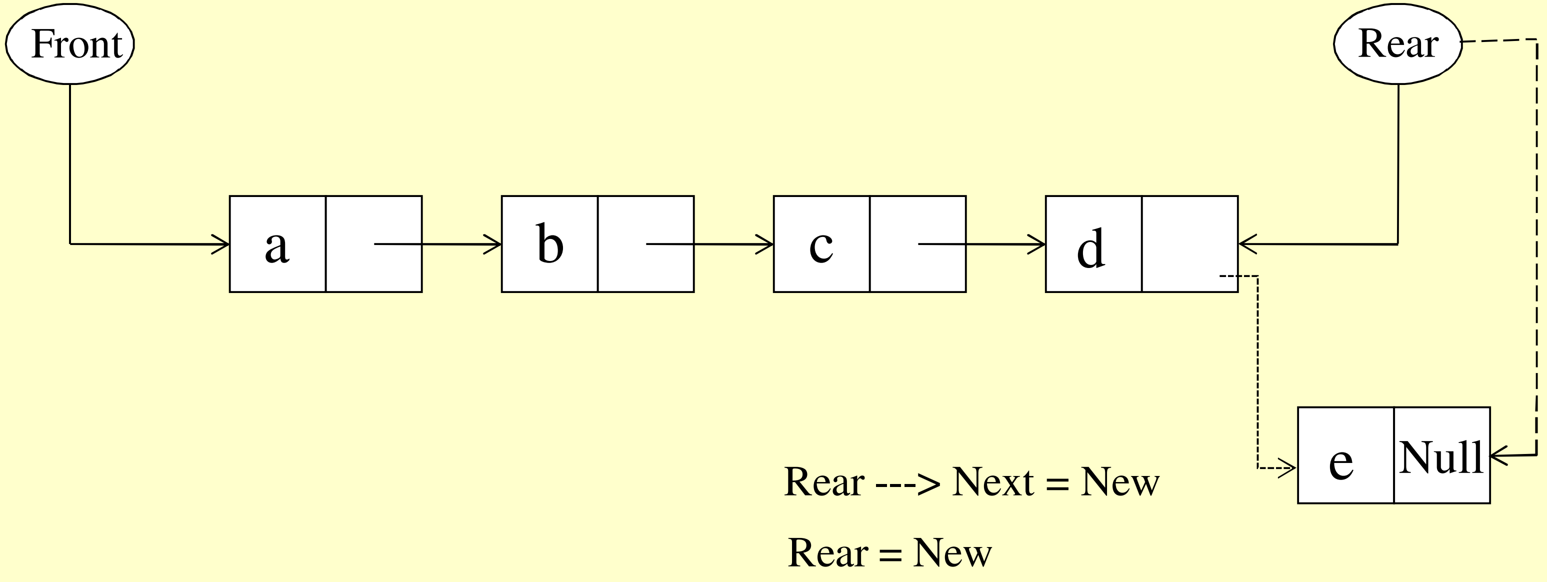
- The elements of the queue may be of different type (hetrogenous).
- Maximum size of the queue may be changed at run time (Dynamic data structure).
- **First In First Out (FIFO)** order must be maintained using two pointer variables **Front** and **Rear.**
- **Front** holds the address of the first node and the Rear holds the address of the last node of the linked list.



A Queue Maintained using a Linked List

Insertion in Queue using Linked List

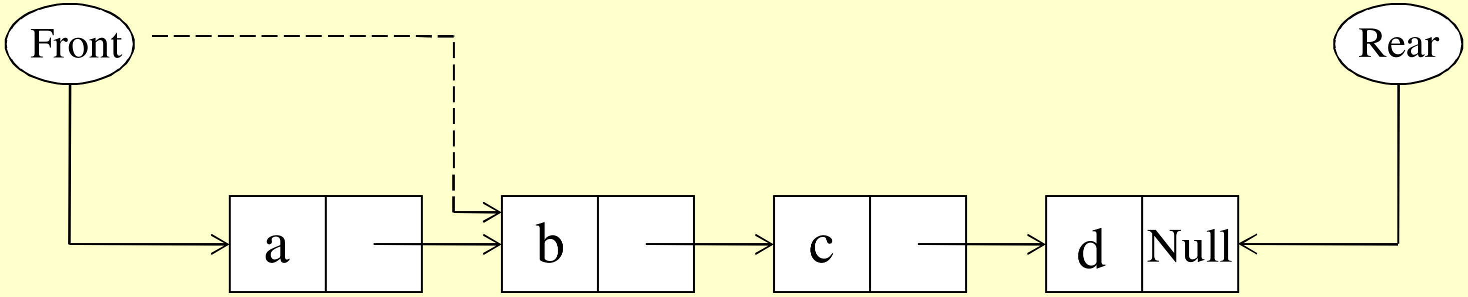
The **insertion** of a new element **e** in the above shown queue can be shown as in figure below:



This insertion of an element 'e' in the queue

Deletion in Queue using Linked List

- Deletion of node pointed by Front variable can be done.
- After deletion, Front will point to 2nd node.



Front = Front → Next

Deletion of an element from the Queue

Insertion in Queue using Linked List

Insert an element ‘**Data**’ in queue having variable ‘**Front**’ which contains the address of 1st element of the queue and variable ‘**Rear**’ which contains the address of last element of the queue.

```
Step 1:   If Free = Null Then
                Print: “No Free Space Available for Insertion”
                Exit
           [End If]
Step 2:   Allocate memory to node New
                Set New = Free and Free = Free -> Next
Step 3:   Set New-> Info = Data and New -> Next = Null
Step 4:   If Rear = Null Then
                Set Front = New and Rear = New
          Else
                Set Rear -> Next = New and Rear = New
          [End If]
Step 5:   Exit
```

Deletion in Queue using Linked List

Deletes an element from a queue having a variable **Front** which contains the address of 1st element of the queue and variable **Rear** which contains the address of last element of the queue.

```
Step 1:    If Front = Null Then
                Print " Queue is Empty"
                Exit
           [End If]
Step 2:    Set Data = Front -> Info, Temp = Front
Step 3:    If Front = Rear Then
                Set Front = Null and Rear = Null
           Else
                Set Front = Front -> Next
           [End If]
Step 5:    Deallocate memory taken by node Temp
           Set Temp -> Next = Free, Free = Temp
Step 6:    Exit
```