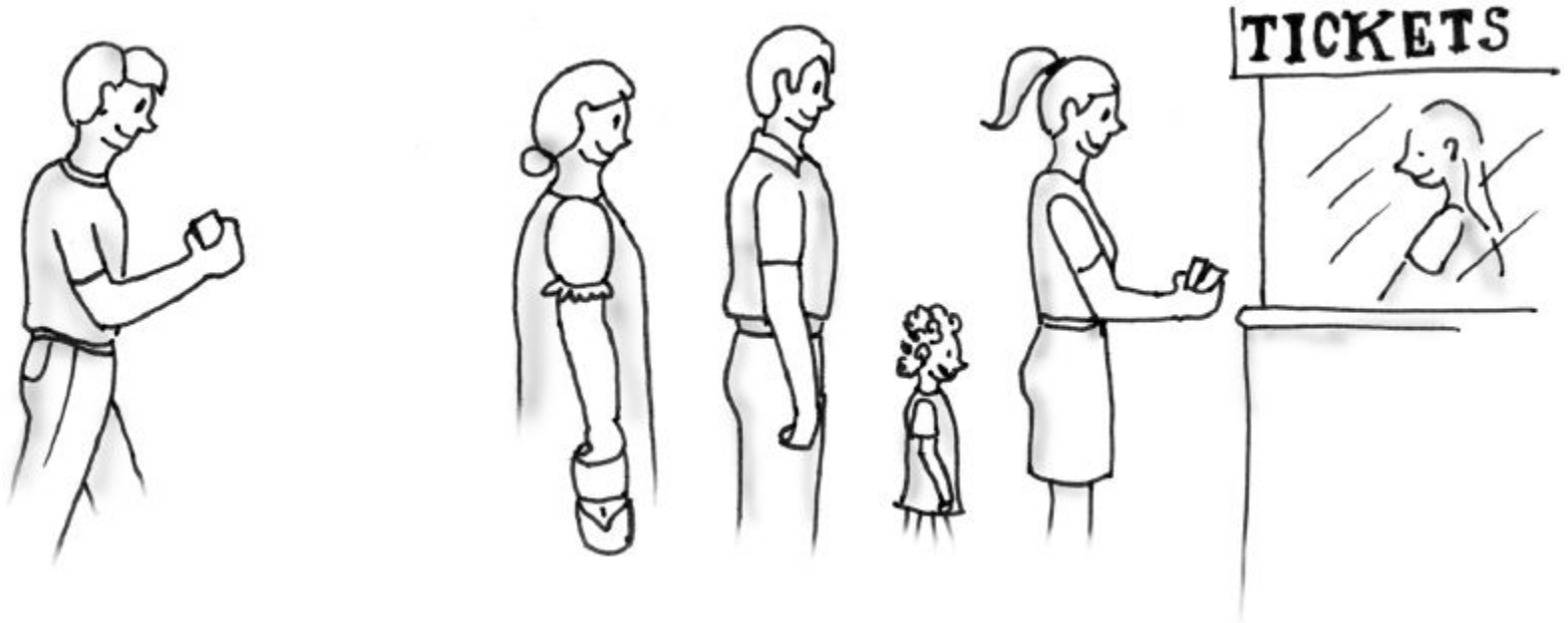
Data Structure
(Queue)
CSE-207

Md. Manowarul Islam, Dept. of CSE, Jagannath University, Dhaka-1100.

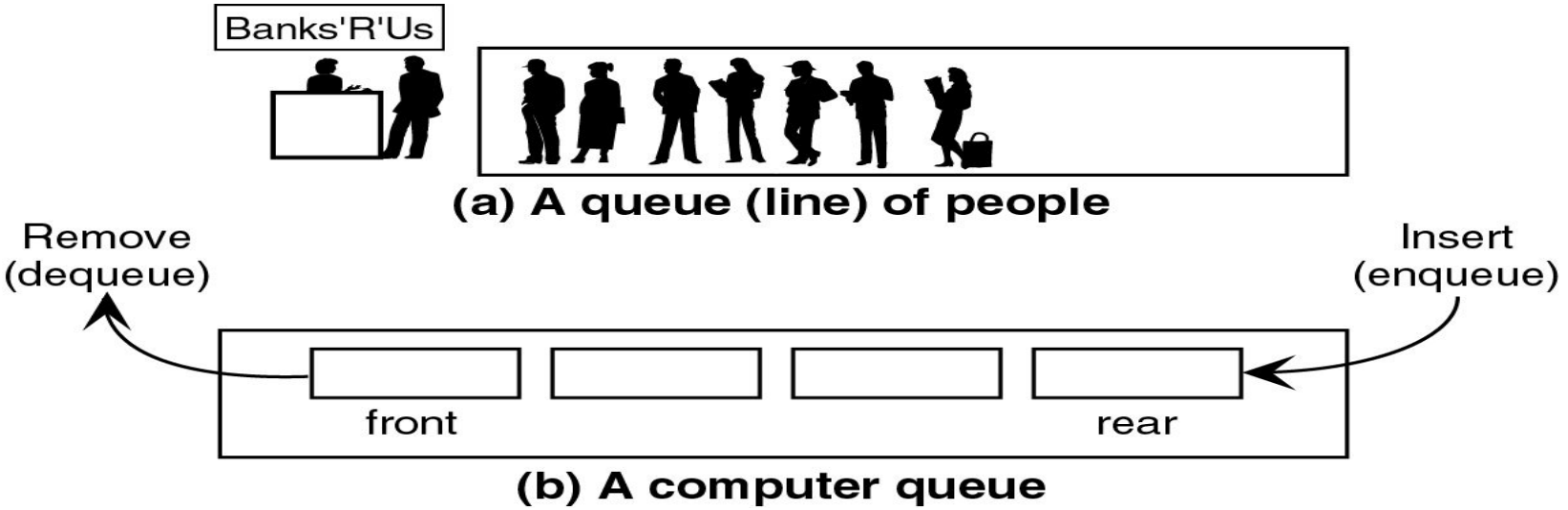
Simulating a Waiting Line

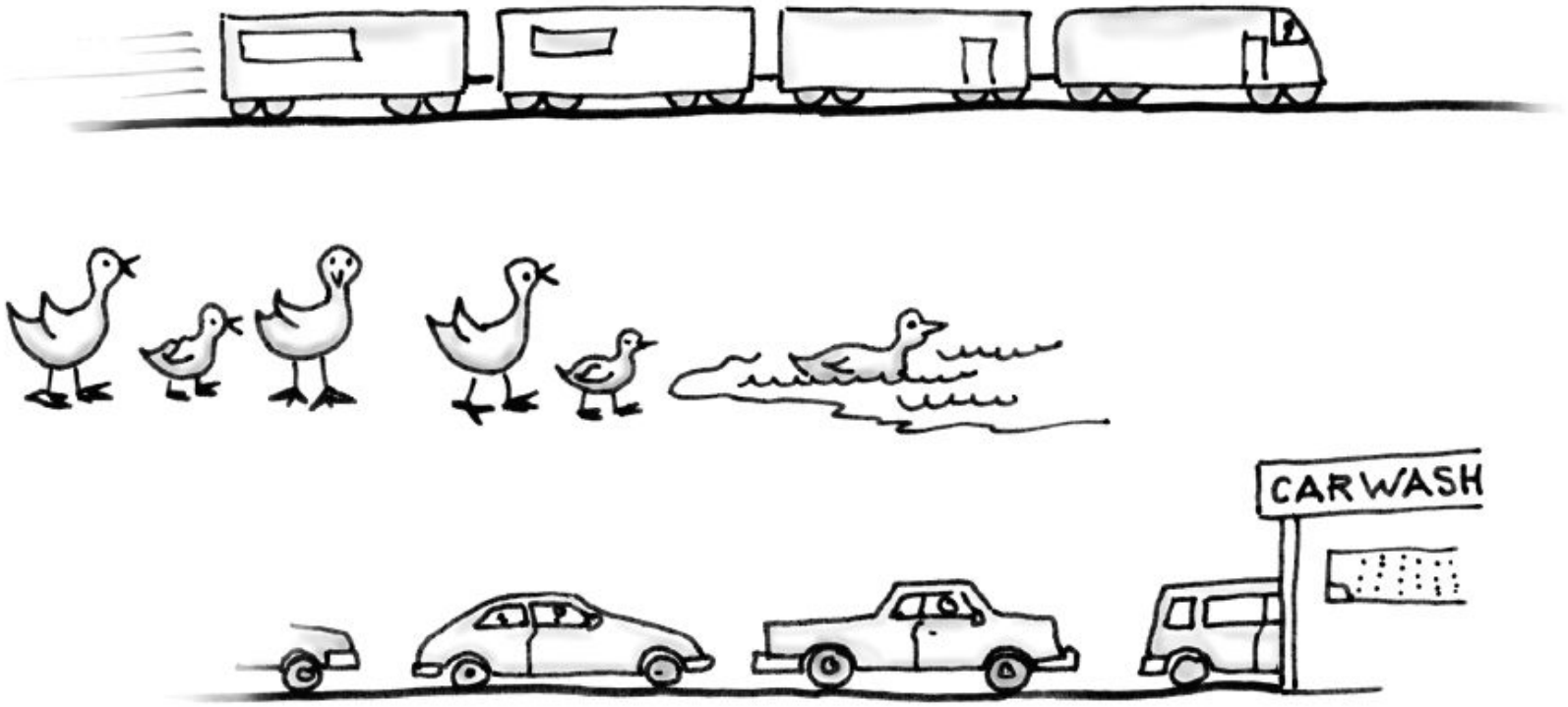


A line, or queue, of people

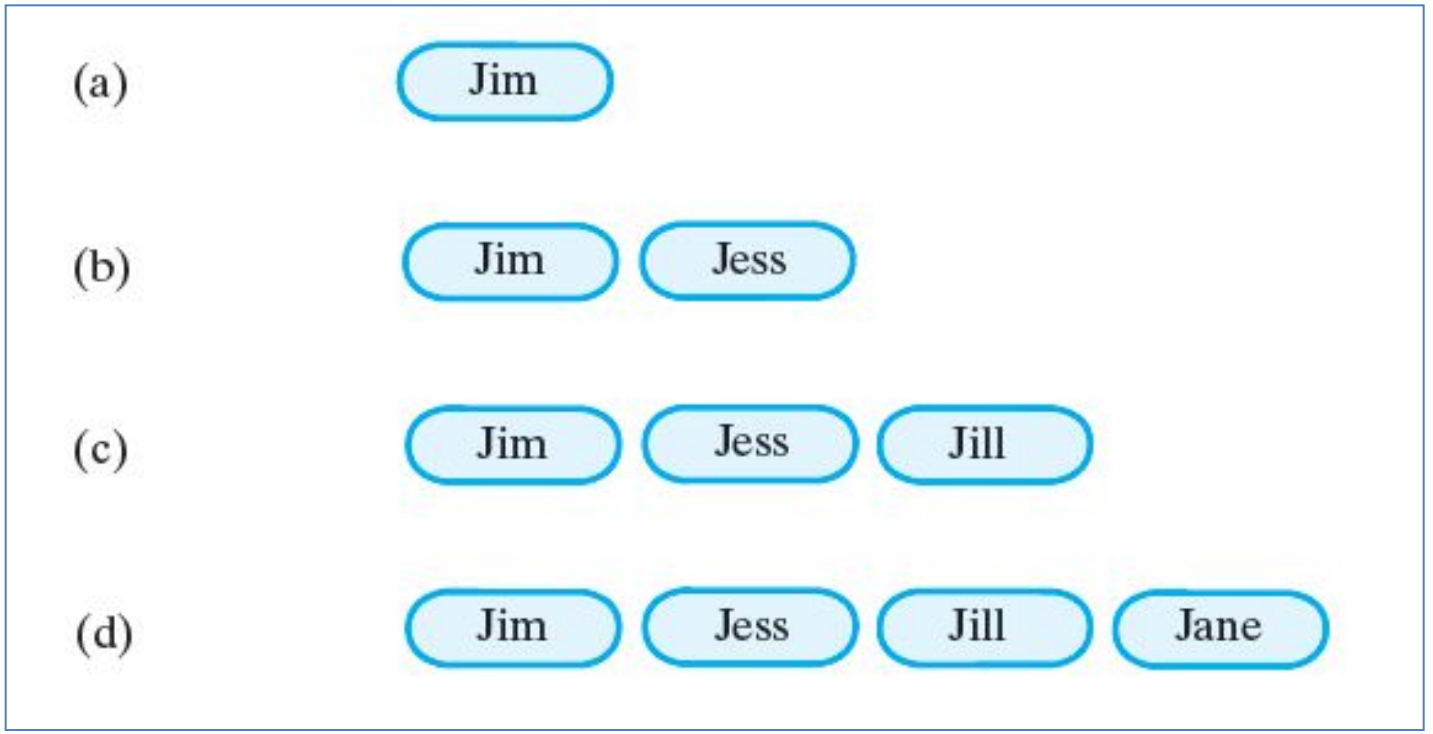
Queue

- It is a linear data structure consisting of list of items.
- In queue, data elements are added at one end, called the rear and removed from another end, called the front of the list.

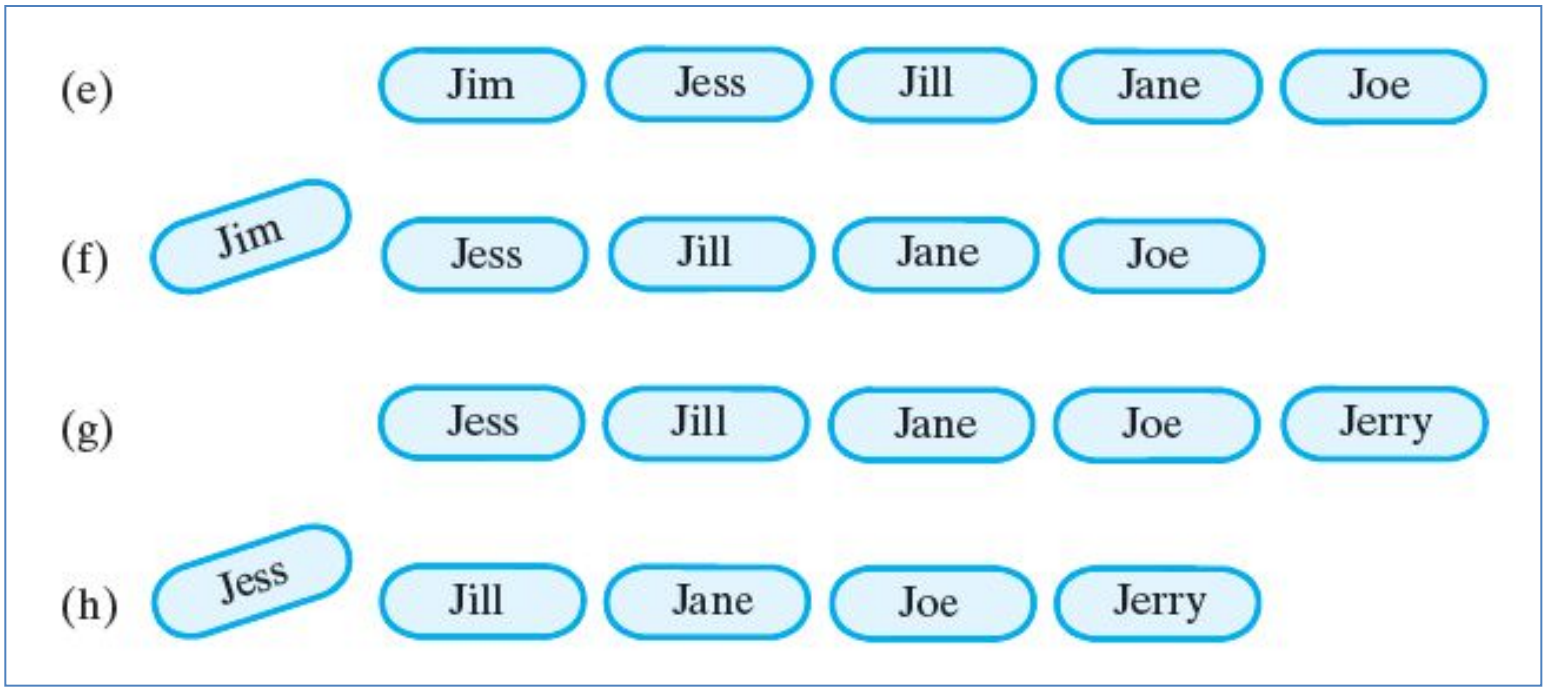




Some everyday queues



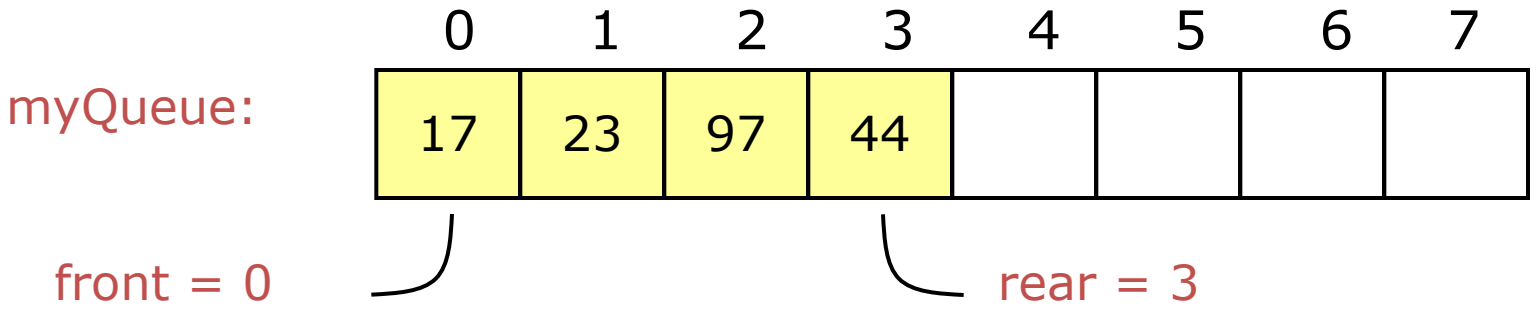
A queue of strings after (a) enqueue adds *Jim*; (b) enqueue adds *Jess*; (c) enqueue adds *Jill*; (d) enqueue adds *Jane*;



A queue of strings after (e) enqueue adds *Joe*; (f) dequeue retrieves and removes *Jim*; (g) enqueue adds *Jerry*; (h) dequeue retrieves and removes *Jess*;

Queue

- A queue is a first in, first out (FIFO) data structure
- This is accomplished by inserting at one end (the rear) and deleting from the other (the front)



- **To insert:** put new element in location 4, and set rear to 4
- **To delete:** take element from location 0, and set front to 1

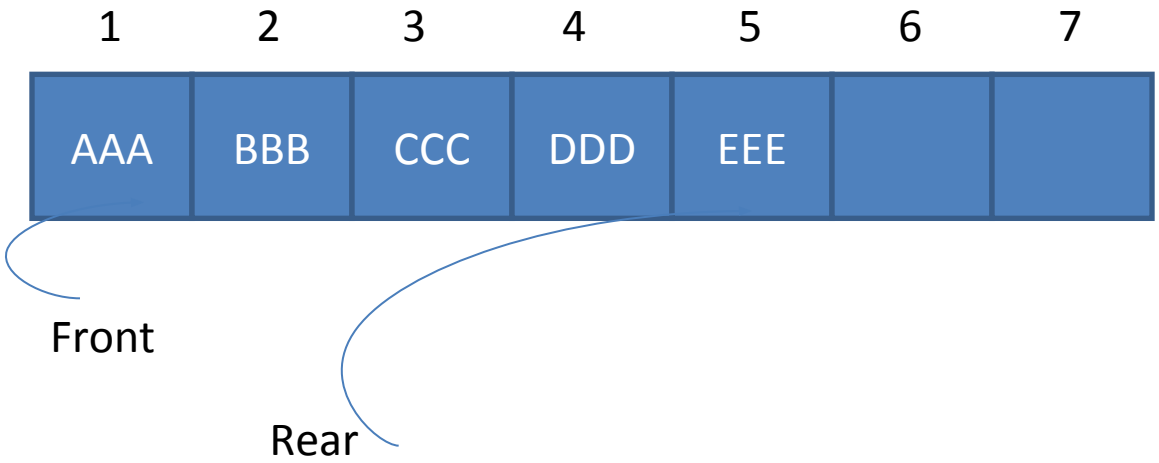
Queue

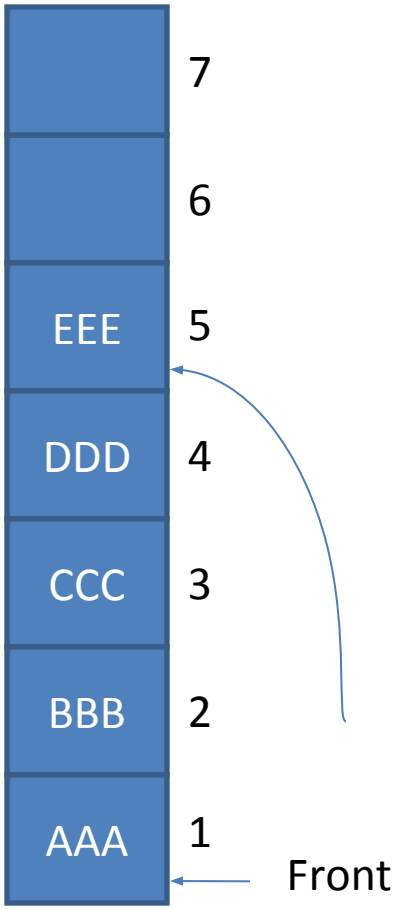
- Two basic operations are associated with queue:
- "Insert" operation is used to insert an element into a queue.
  - Called *enqueue* operation
- "Delete" operation is used to delete an element from a queue.
  - Called *dequeue* operation

Queue

Example:

Queue: AAA, BBB, CCC, DDD, EEE





Queue

- rear is the location in which the data element is to be inserted.
- front is the location from which the data element is to be removed.
- When queue is empty
  - rear=-1
  - front=-1
- Adding an element in queue will increase the value of rear
  - rear=rear+1;
- Removing an element in queue will increase the value of front
  - front=front+1;

Array implementation of queue

Queue empty rear=-1 front=-1

Array implementation of queue

Queue empty rear=-1 front=-1

Add 10, rear=0 front=0

Array implementation of queue

Queue empty rear=-1 front=-1

Add 10, rear=0 front=0

Add 20, rear=1, front=0

Array implementation of queue

Queue empty rear=-1 front=-1

Add 10, rear=0 front=0

Add 20, rear=1, front=0

Add 30, rear=2, front=0

Array implementation of queue

Queue empty rear=-1 front=-1

Add 10, rear=0 front=0

Add 20, rear=1, front=0

Add 30, rear=2, front=0

Add 40, rear=3, front=0

Array implementation of queue

Add 40, rear=3, front=0

Remove, rear=3, front=1

Array implementation of queue

Add 40, rear=3, front=0

Remove, rear=3, front=1

Remove, rear=3, front=2

Array implementation of queue

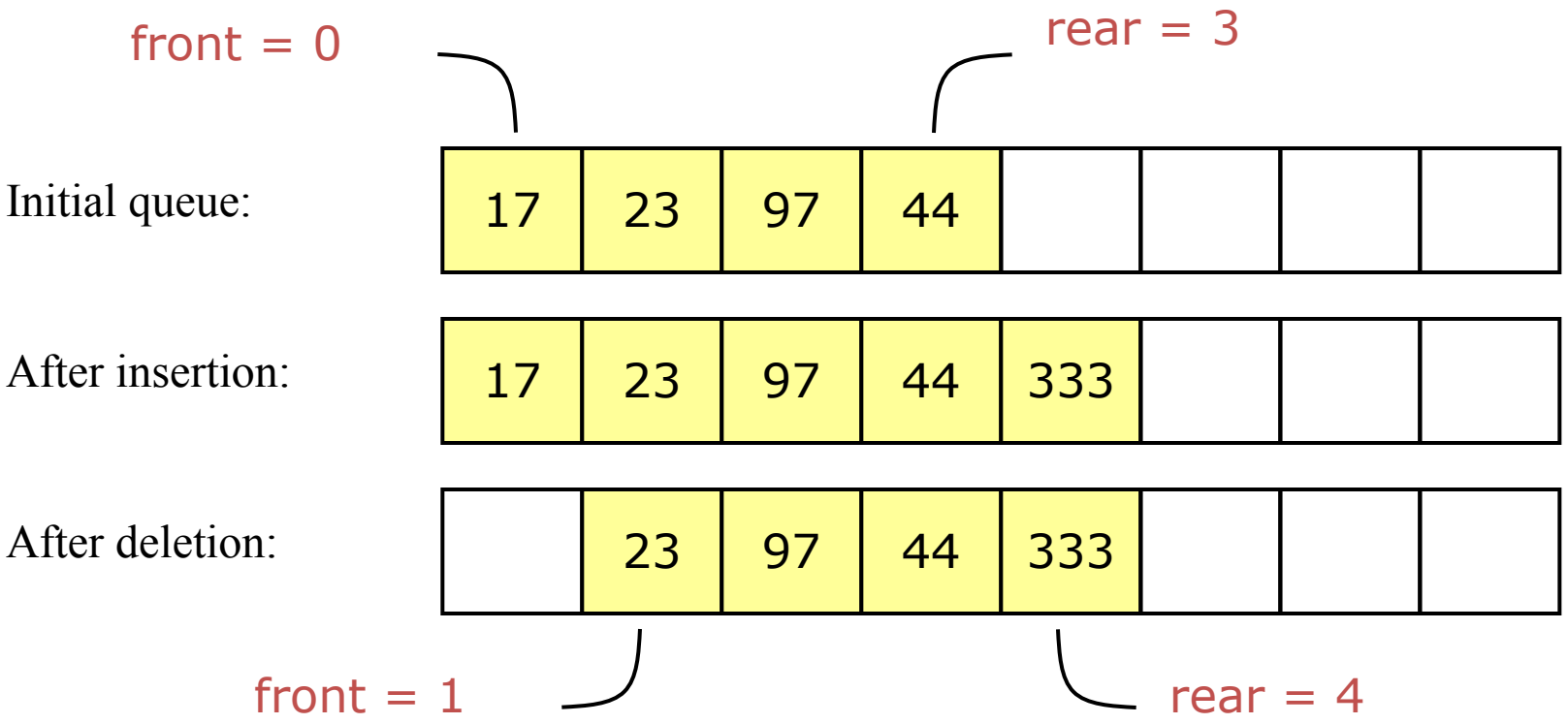
Add 40, rear=3, front=0

Remove, rear=3, front=1

Remove, rear=3, front=2

Add 60, rear=4, front=2

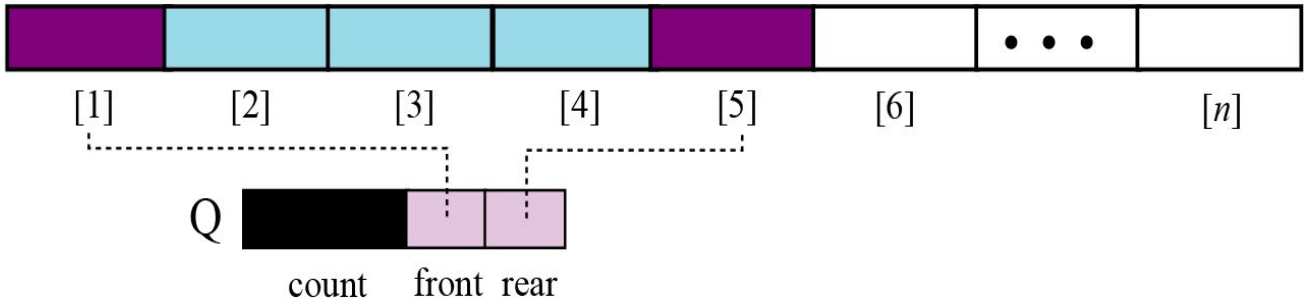
Queue



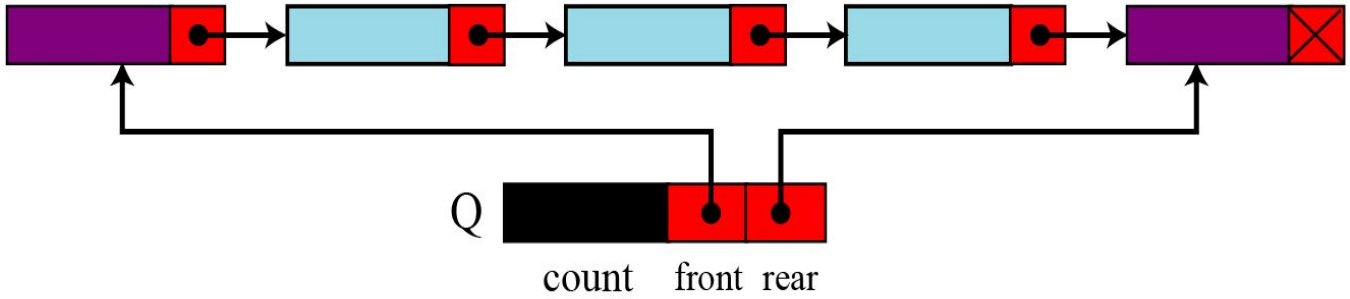
- Notice how the array contents “crawl” to the right as elements are inserted and deleted
- This will be a problem after a while!

Queue

b. Array implementation



c. Linked list implemenation



Insert Operations in Linear Queue

- Rear is the location in which the data element is to be inserted.
- Front is the location from which the data element is to be removed.
- Here N is the maximum size of the Queue

```
1. If Rear = N then Print: Overflow and Return.  /*…Queue already filled..*/
2. Set Rear := Rear +1
3. Set Queue[Rear] := Item
4. Return.
```

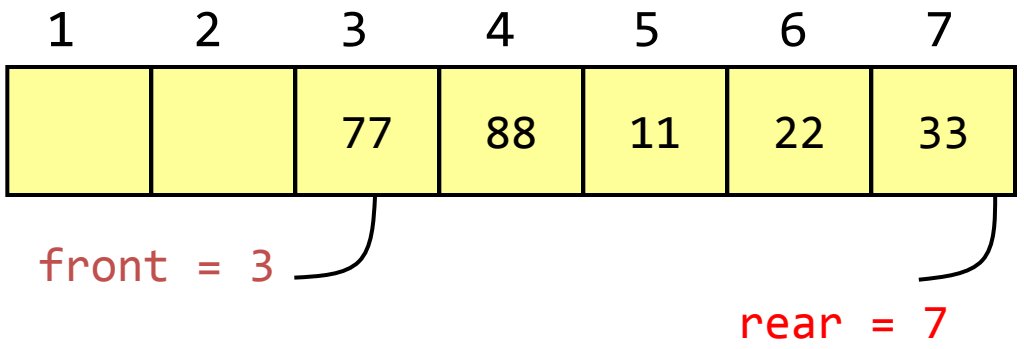
Delete Operations in Linear Queue

- Rear is the location in which the data element is to be inserted.
- Front is the location from which the data element is to be removed.
- Here N is the maximum size of the Queue

```
1. If Front = N+1   then Print: Underflow and Return.
/*…Queue Empty*/
2. Set  Item := Queue[Front]
3. Set Front := Front + 1
4. Return.
```

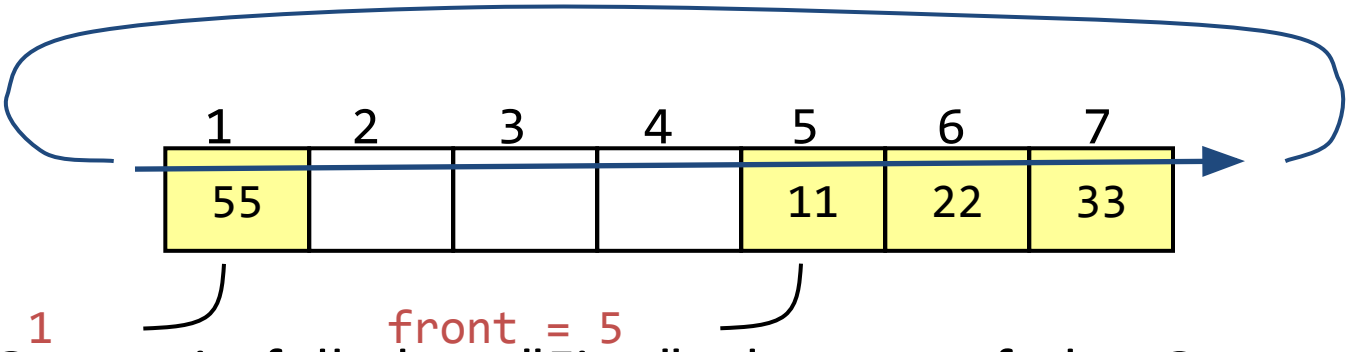
Problem

- Once the queue is full, rear has already reached the Queue's rear most position
- Even though few elements from the front are deleted and some occupied space is relieved,
- it is not possible to add anymore new elements

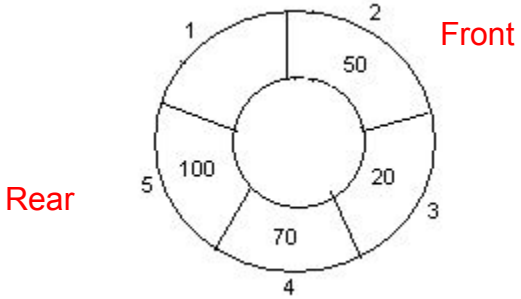


Circular queue

- We can treat the array holding the queue elements as circular (joined at the ends)



- Once the Queue is full the "First" element of the Queue becomes the "Rear" most element, if and only if the "Front" has moved forward



Drawback of Linear Queue

- Once the queue is full, even though few elements from the front are deleted and some occupied space is relieved, it is not possible to add anymore new elements, as the rear has already reached the Queue's rear most position.

Circular Queue

- This queue is not linear but circular.
- Its structure can be like the following figure:
- In circular queue, once the Queue is full the "First" element of the Queue becomes the "Rear" most element, if and only if the "Front" has moved forward. otherwise it will again be a "Queue overflow" state.

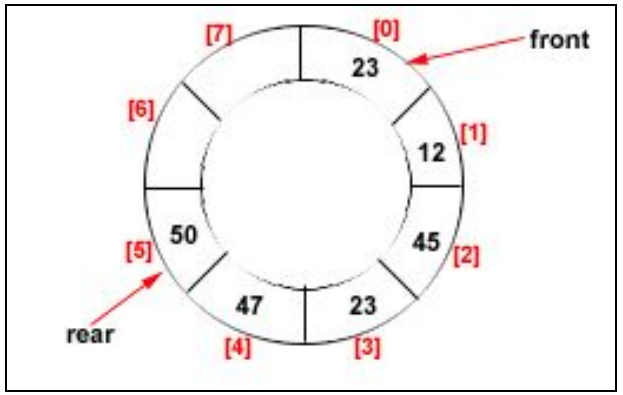


Figure: Circular Queue having Rear = 5 and Front = 0

For Insert Operation

- Rear is the inserting location
- Front is the removing location.
- Here N is the maximum size of the Cqueue
- Initailly Rear = 0 and Front = 0.

1. If Front = 0 and Rear = 0
   then Set Front := 1 and go to step 4.
2. If Front =1 and Rear = N or Front = Rear + 1
   then Print: “Circular Queue Overflow” and Return.
3. If Rear = N then Set Rear := 1 and go to step 5.
4. Set Rear := Rear + 1
5. Set CQueue [Rear] := Item.
6. Return

For Delete Operation

- Rear is the inserting location
- Front is the removing location.
- Here N is the maximum size of the Cqueue
- Front element is assigned to Item, initially, Front = 1.

```
1. If Front = 0 then
        Print: “Circular Queue Underflow” and Return.      /*..Delete
without Insertion
2. Set Item := CQueue [Front]
3. If Front = N then Set Front = 1 and Return.
4. If Front = Rear then Set Front = 0 and Rear = 0 and Return.
5. Set Front := Front + 1
6. Return.
```

For Delete Operation

Delete-Circular-Q(CQueue, Front, Rear, Item)

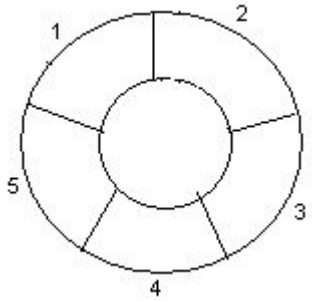
Here, CQueue is the place where data are stored. Rear represents the location in which the data element is to be inserted and Front represents the location from which the data element is to be removed. Front element is assigned to Item. Initially, Front = 1.

1. If Front = 0 then

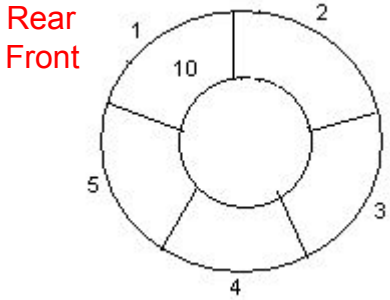
   Print: “Circular Queue Underflow” and Return. /*..Delete without Insertion
2. Set Item := CQueue [Front]
3. If Front = N then Set Front = 1 and Return.
4. If Front = Rear then Set Front = 0 and Rear = 0 and Return.
5. Set Front := Front + 1
6. Return.

Circular queue

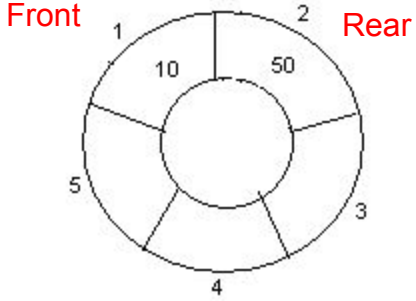
1. Initially, Rear = 0, Front = 0.



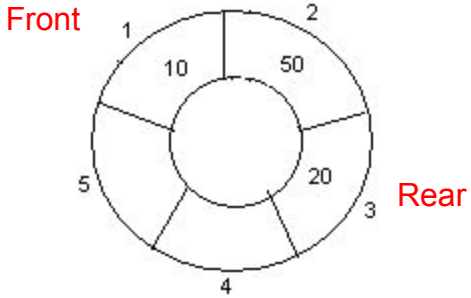
2. Insert 10, Rear = 1, Front = 1.



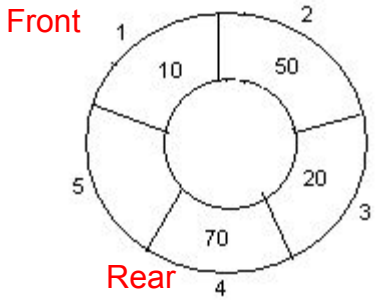
3. Insert 50, Rear = 2, Front = 1.



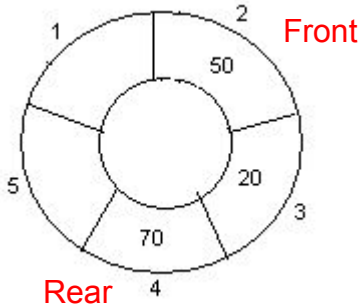
4. Insert 20, Rear = 3, Front = 1.



5. Insert 70, Rear = 4, Front = 1.

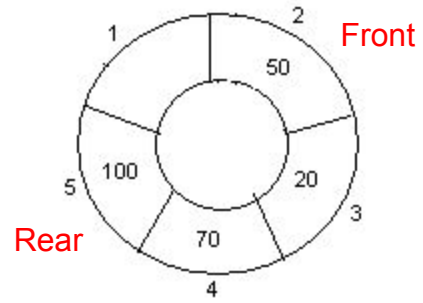


6. Delete front, Rear = 4, Front = 2.

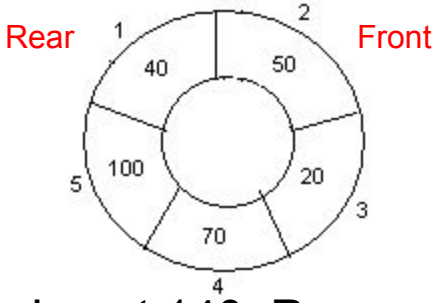


Circular queue

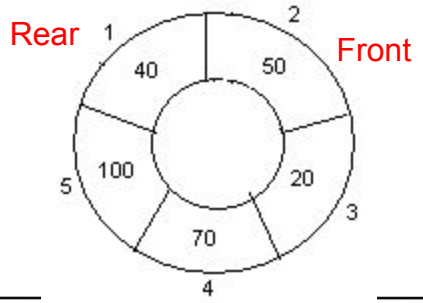
7. Insert 100, Rear = 5, Front = 2.



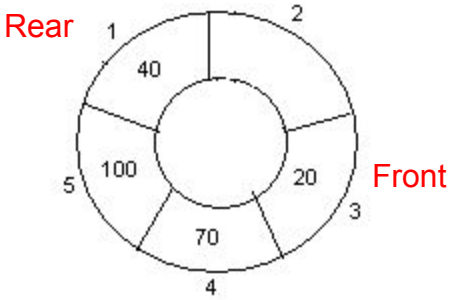
8. Insert 40, Rear = 1, Front = 2.



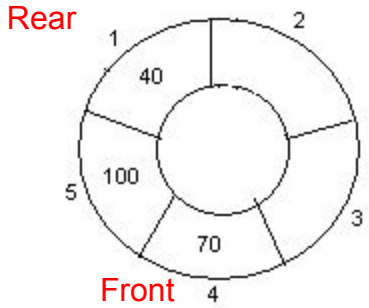
9. Insert 140, Rear = 1, Front = 2.
As Front = Rear + 1, so Queue overflow.



10. Delete front, Rear = 1, Front = 3.



11. Delete front, Rear = 1, Front = 4.



12. Delete front, Rear = 1, Front = 5.