DATA STRUCTURES

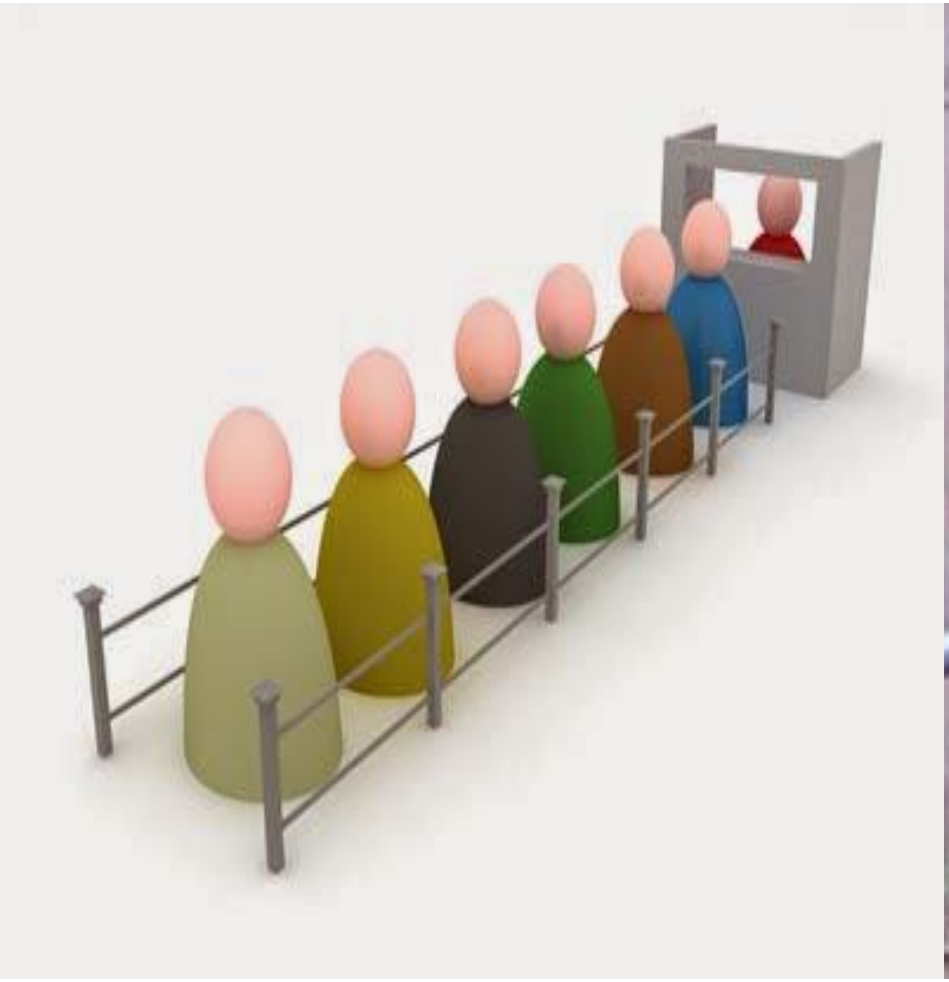
UNIT-2

Queue Data Structure

Dr G.KALYANI

Topics

- **Introduction to Queues**
- Operations on Queues
- Queue ADT
- Queue Implementation
- Applications of Queues
- Types of Queues



Queue Data Structure

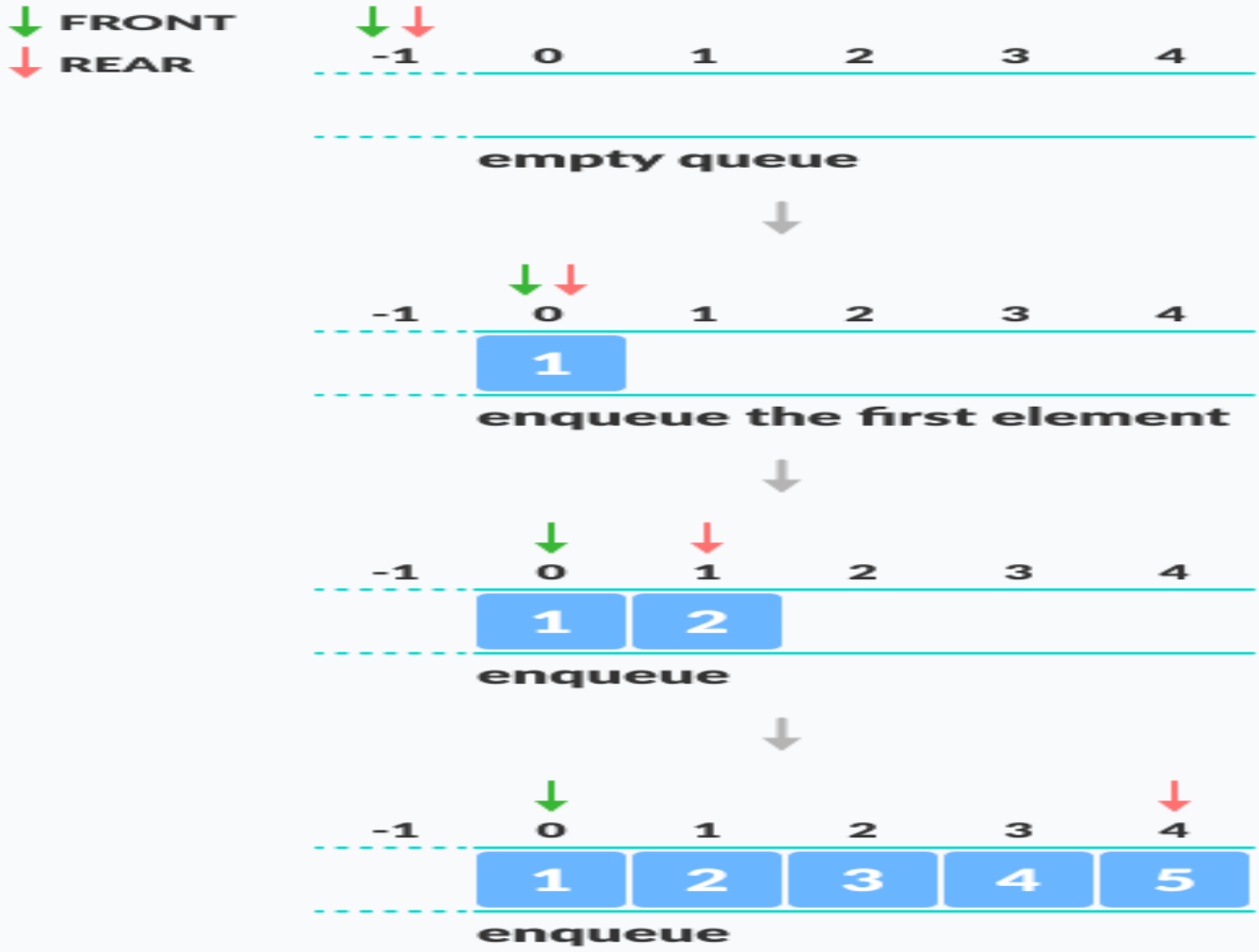
- A **Queue** is an ordered collection of items.
- Queue is a linear data structure which follows a particular order in which the operations are performed.
- Addition of new items and the removal of existing items takes place at different ends.
- The end at which insertion is performed referred as "Rear."
- The end at which deletion is performed referred as "Front."
- The order may be FIFO(First In First Out).

Topics

- Introduction to Queues
- **Operations on Queues**
- Queue ADT
- Queue Implementation
- Applications of Queues
- Types of Queues

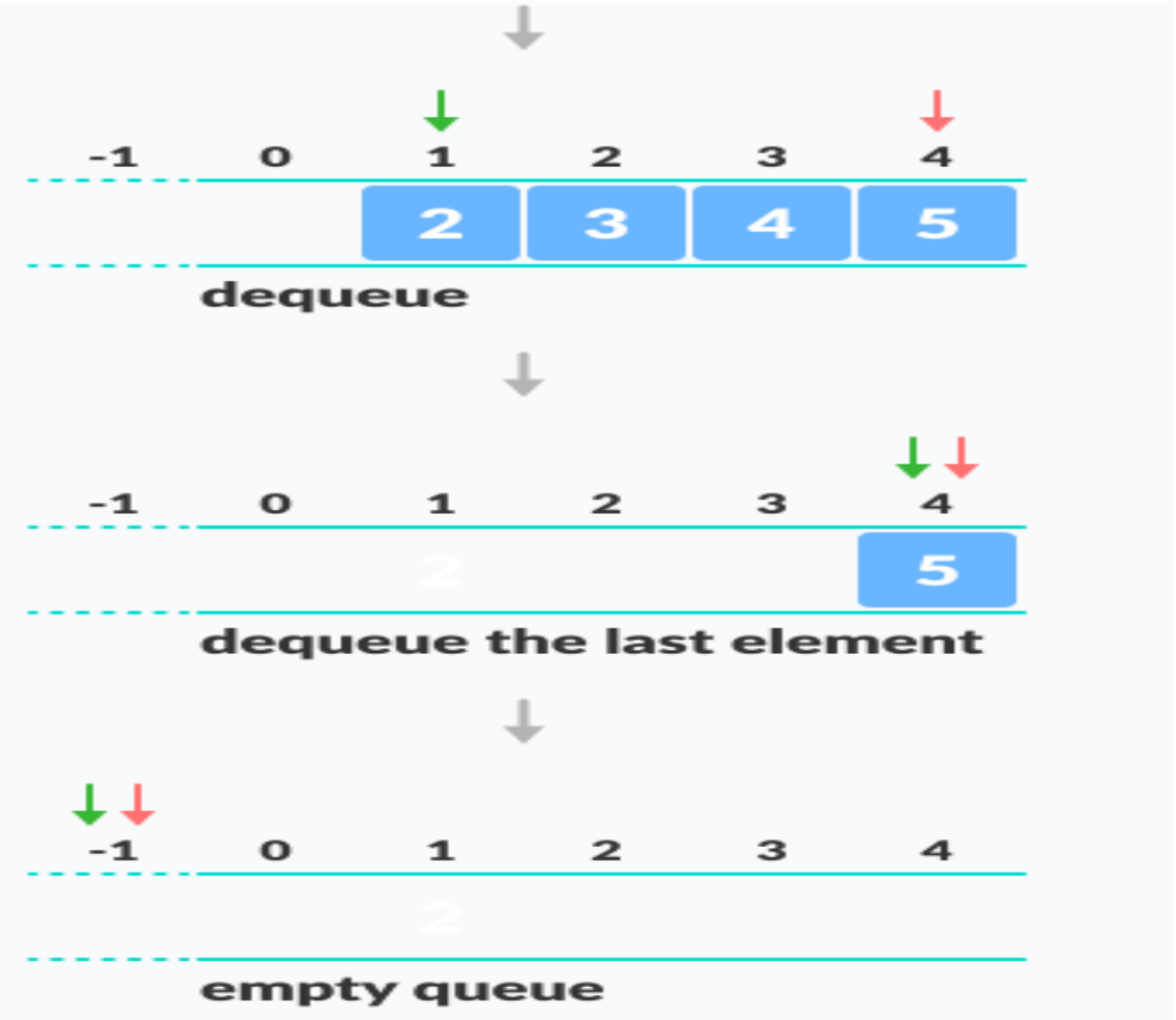
Operations on Queue

- **Enqueue or Insertion or Addition**



Operations on Queue

- **Dequeue or Deletion or Removal**



Operations on Queue

- **Is Empty**
  - Check whether queue is empty or not
- **Is Full**
  - Check whether queue is full or not

Operations on Queue

Example: Consider the following queue (linear queue).

Rear = 4 and Front = 1 and N = 7

10	50	30	40			
1	2	3	4	5	6	7

(1) Insert 20. Now Rear = 5 and Front = 1

10	50	30	40	20		
1	2	3	4	5	6	7

(2) Delete Front Element. Now Rear = 5 and Front = 2

	50	30	40	20		
1	2	3	4	5	6	7

(3) Delete Front Element. Now Rear = 5 and Front = 3

		30	40	20		
1	2	3	4	5	6	7

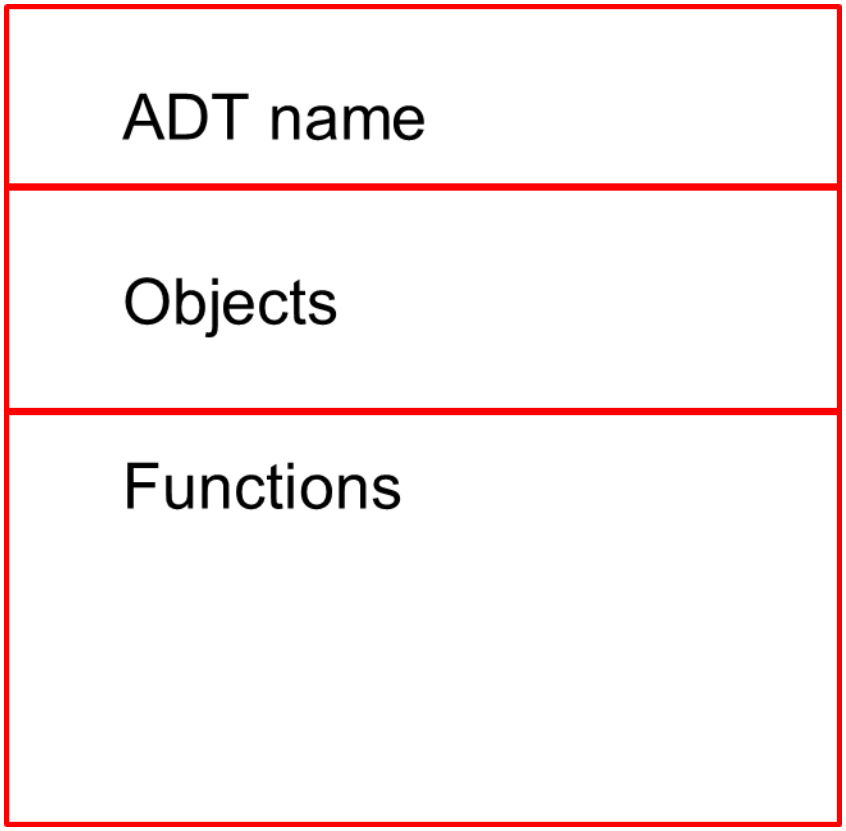
(4) Insert 60. Now Rear = 6 and Front = 3

		30	40	20	60	
1	2	3	4	5	6	7

Topics

- Introduction to Queues
- Operations on Queues
- **Queue ADT**
- Queue Implementation
- Applications of Queues
- Types of Queues

Queue ADT



Queue ADT

- **ADT Queue**

  Objects: a finite ordered list with zero or more elements

  Functions:

  - **Create(max Queue size):** creates a new queue of max queue size that is empty.
  - **Enqueue(queue, item):** adds a new item to the rear position of the queue.
  - **Dequeue(queue):** removes the item from the front position of the queue.
  - **Is Full(queue):** tests to see whether the queue is full. It returns a Boolean value.
  - **Is Empty(queue):** tests to see whether the queue is empty. It returns a Boolean value.

Topics

- Introduction to Queues
- Operations on Queues
- Queue ADT
- **Queue Implementation**
- Applications of Queues
- Types of Queues

Queue Implementation using Arrays

- Define Max size with some value.
- Declare an array Queue of your preferred type with Max size.
- Declare variables front and rear of integer type and initialize to -1(because initially queue empty).

Enqueue Operation

- The process of placing a new data element onto queue is known as a enqueue Operation.
- Enqueue operation involves a series of steps −

  Step 1 − Check is the queue full.

  Step 2 − If the queue is full, produces an error and exit.

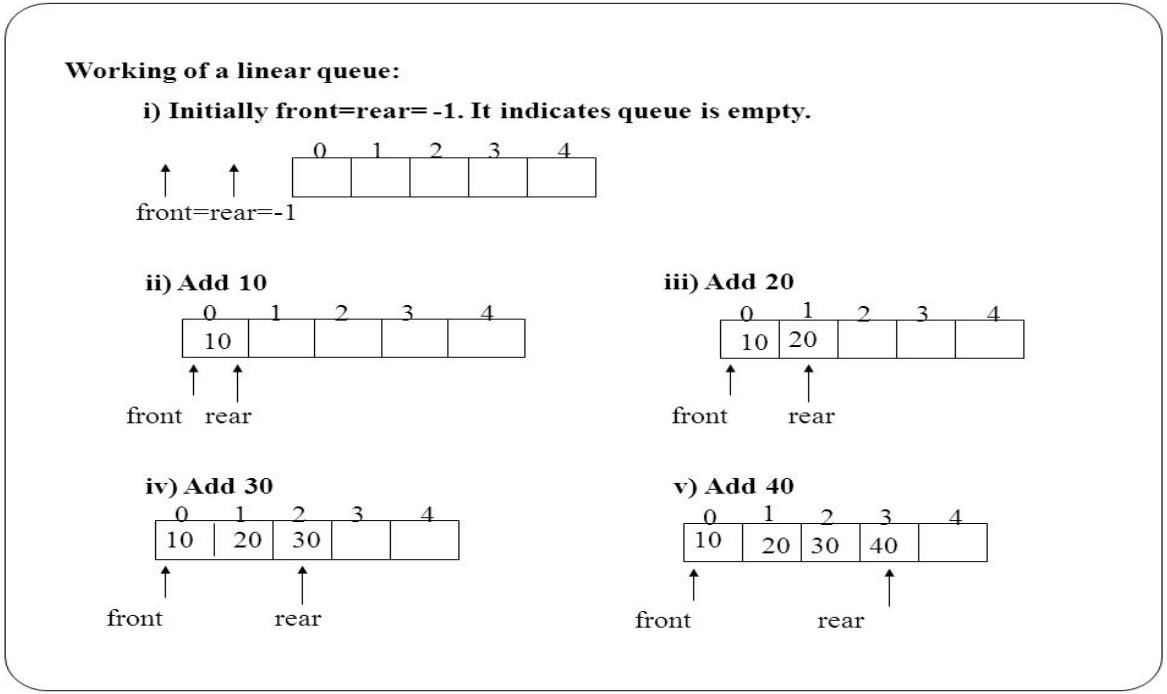
  Step 3 − If the queue is not full

  If queue not empty then increments rear to next empty space.

  If queue empty then increment front & rear to next empty space.

  Step 4 − Adds data element to the queue location, where rear is pointing.

  Step 5 − end.



Enqueue Operation

```
Enqueue(int queue[], int item)
{
        if(Is Full(queue))
                Printf("queue overflow")
        else
        {
                if(Is empty(queue))
                        front++; rear++;
                else rear++;
                queue[rear]=item;
        }
}
```

we can not perform enqueue operation

we can perform enqueue operation

Dequeue Operation

- A Pop operation may involve the following steps:

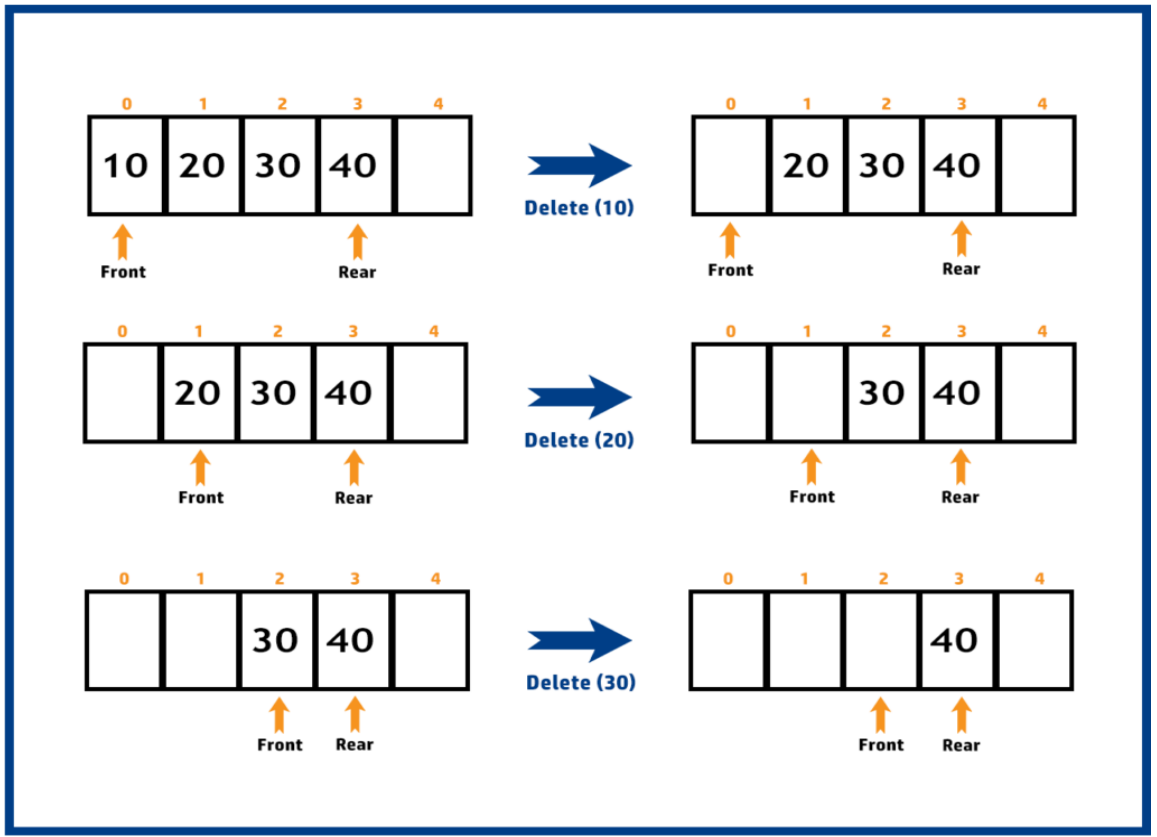
  Step 1 – Check is the queue empty.

  Step 2 – If the queue is empty, produces an error and exit.

  Step 3 – If the queue is not empty, take the element at front position

  Step 4 – increment the value of front by 1.

  Step 5 – end.



Dequeue Operation

```
Dequeue()
{
        if(Is Empty(queue))
                Printf("queue is empty");
        else
        {
                a=queue[front];
                front++;
                Printf(" deleted element is a ");
        }
}
```

In this state we can't perform dequeue operation

queue not empty we can perform dequeue operation

Is Full Operation

- Checks whether the queue is full or not.
- If rear is equal to the max size-1 then queue is full.

```
bool Is Full(queue)
{
    if( rear == Max Size-1)
          return true;
    else
          return false;
 }
```

Is Empty Operation

- Checks whether the queue is empty or not.
- If front and rear are equal to the initialized value -1 then queue is empty.

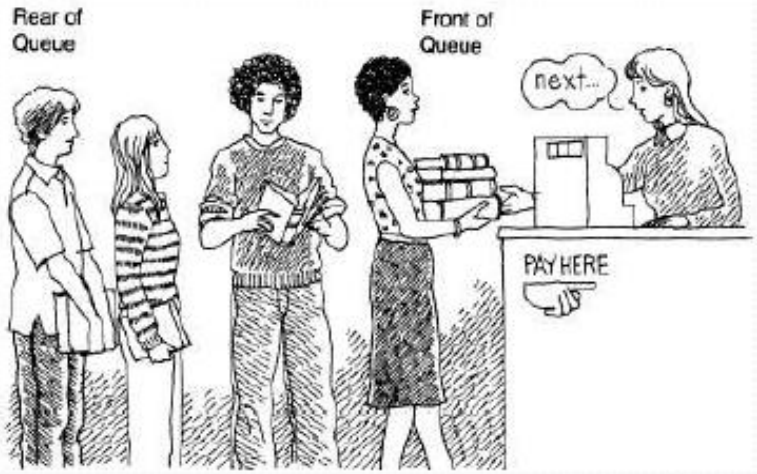
```
bool is Empty(queue)
{
    if((front == -1 and rear==-1)or(front>rear))
        return true;
    else
        return false;
}
```

Topics

- Introduction to Queues
- Operations on Queues
- Queue ADT
- Queue Implementation
- **Applications of Queues**
- Types of Queues

Applications of Queues

- Real world applications
  - Cashier line in any store.
  - Waiting on hold for tech support.
  - people on an escalator.
  - Checkout at any book store.



Applications of Queues

- Applications related to computer science:

1. When data is transferred asynchronously between two processes. eg. IO Buffers.
2. When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
3. In recognizing palindrome.
4. In shared resources management.
5. Keyboard buffer.
6. Round robin scheduling.
7. Job scheduling.
8. Simulation

Topics

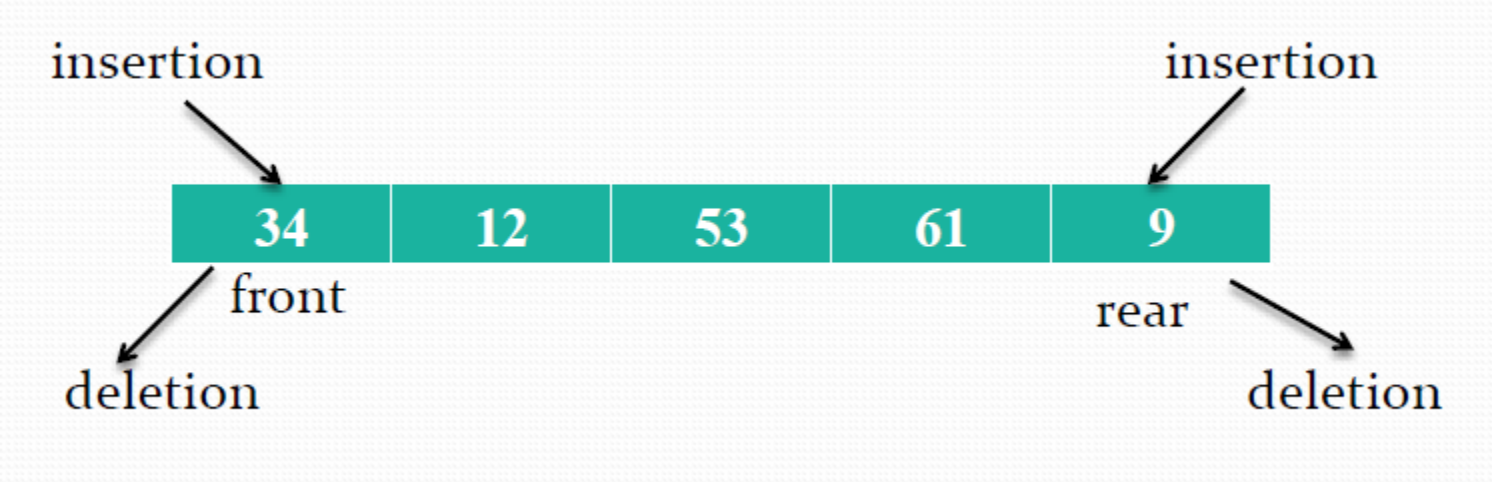
- Introduction to Queues
- Operations on Queues
- Queue ADT
- Queue Implementation
- Applications of Queues
- **Types of Queues**

Types of Queues

- **Three Types of queues**
  - 1. Deque
  - 2. Priority Queue
  - 3. Circular Queue

Deque

- Deque stands for double ended queue.
- Elements can be inserted or deleted at either end.



Insertion into Deque at front

WHEN ONE ELEMENT IS ADDED
LETS SAY 10,

FRONT= REAR = 0

INSERT 12 AT FRONT.

REAR = 0 FRONT=5

INSERT 21 AT REAR

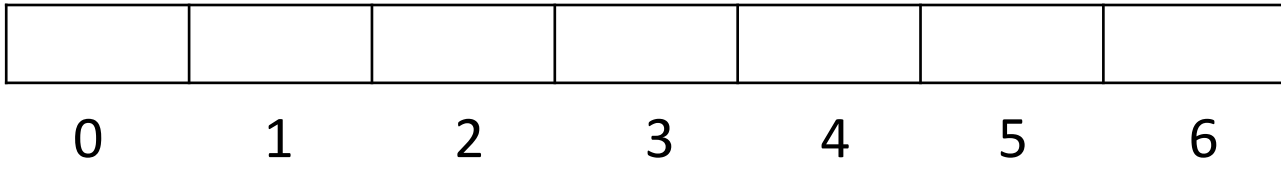
REAR = 1 FRONT= 4

NOW INSERT 14 AT FRONT

REAR = 0 FRONT= 4

Double Ended Queue Example

- Perform the following operations on double ended queue of size 7:



front = rear = -1;

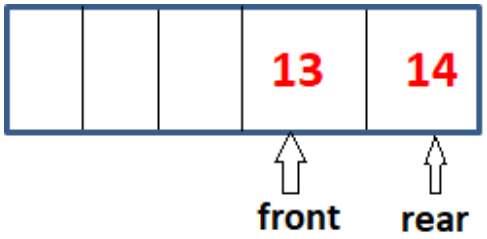
- Insert 10 at front
- Insert 20 at rear
- Insert 30 at front
- Insert 40 at rear
- Delete at front
- Insert 50 at rear
- Insert 60 at front
- Delete at rear
- Delete at front
- Insert 70 at front
- Insert 80 at front
- Delete at rear
- Delete at front

Priority Queue

- It is collection of elements where elements are stored according to the their priority levels.
- Inserting and removing of elements from queue is decided by the priority of the elements.
- An element of the higher priority is processed first.
- Two element of same priority are processed on first-come-first-served basis.

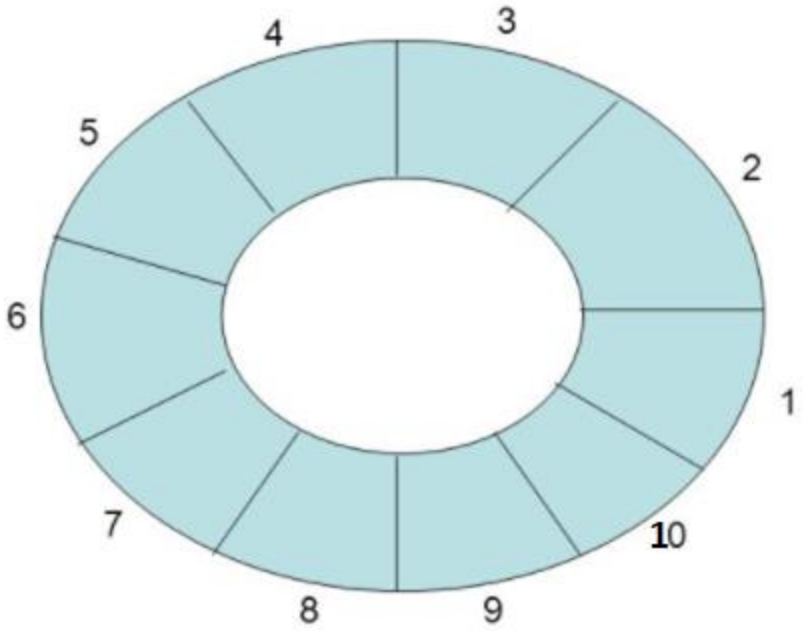
Circular Queue

- Circular queue are used to remove the drawback of simple queue.
- **Drawback of 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.



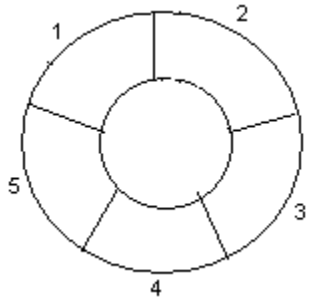
Circular Queue

- Front and rear are initialized to 0

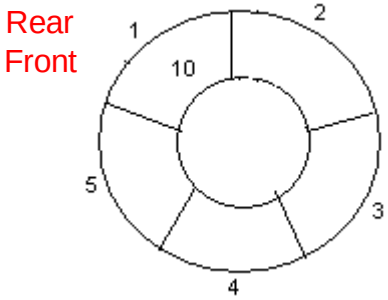


Example 1: Consider the following circular queue with N = 5.

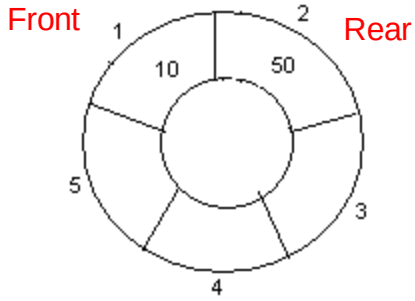
1. Initially, Rear = 0, Front = 0.



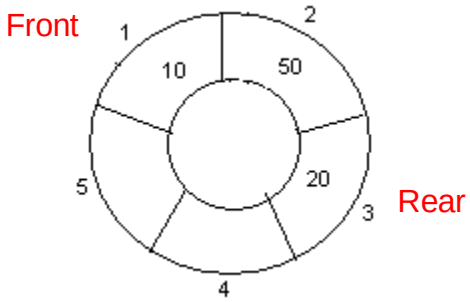
2. Insert 10, Rear = 1, Front = 1.



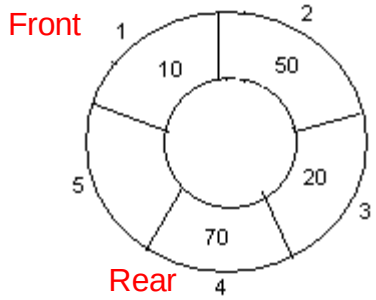
3. Insert 50, Rear = 2, Front = 1.



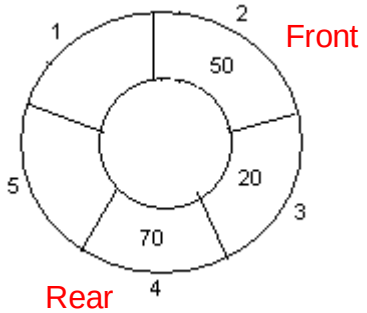
4. Insert 20, Rear = 3, Front = 0.



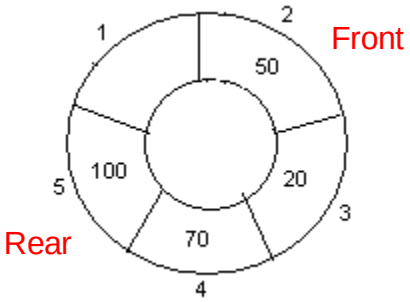
5. Insert 70, Rear = 4, Front = 1.



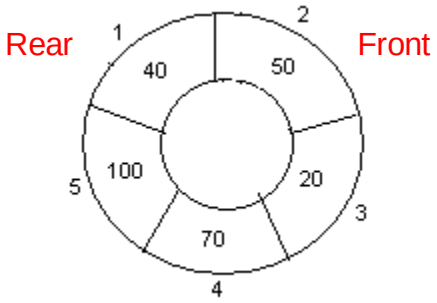
6. Delete front, Rear = 4, Front = 2.



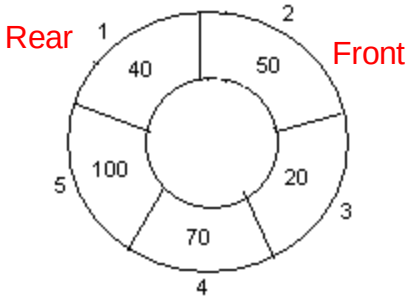
7. Insert 100, Rear = 5, Front = 2.



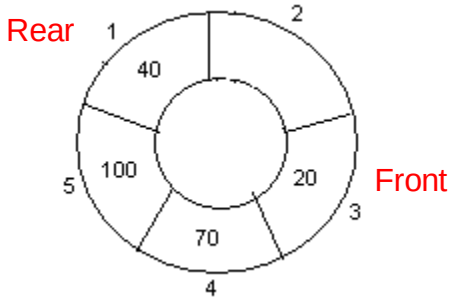
8. Insert 40, Rear = 1, Front = 2.



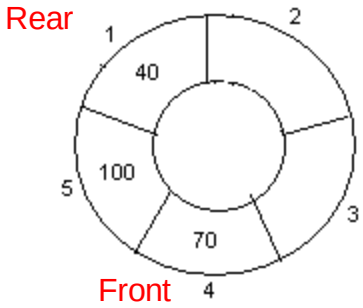
9. Insert 140, Rear = 1, Front = 2.
As Front = Rear + 1, so Queue overflow.



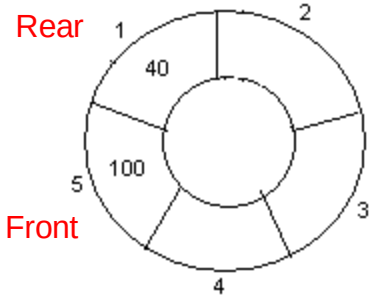
10. Delete front, Rear = 1, Front = 3.



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



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



Circular Queue Example 2

- Perform the following operations on circular queue of size 6:

- Insert 10,20,30,40,50
- Delete
- Insert 60
- Insert 70
- Insert 80
- Delete, Delete, Delete, Delete, Delete, Delete
- Insert 90
- Insert 100

Insertion into Circular Queue

```
Algorithm insert_CQ (CQueue, Element)
{
    If  (Is Full(CQueue))
        Write “Circular Queue Overflow” and Return;
    else If (Front == 0 and Rear == 0)
            Front = 1; Rear=1;
            CQueue [Rear] = Element;
        else If (Rear == N)
                Rear = 1;
                CQueue [Rear] = Element;
            else Rear = Rear + 1;
                CQueue [Rear] = Element;
}
```

Deletion in Circular Queue

```
Algorithm Deletion_CQ()
{
    If (Is Empty(CQueue))
        Write "Circular Queue Underflow" and Return;
    Else
        Item = CQueue [Front];
        If (Front == Rear)
            Front = 0; Rear = 0; Return;
        else If (front==n)
            Front = 1; Return;
        else
            Front = Front + 1;
}
```

Is Full Operation

- Checks whether the circular queue is full or not.

```
bool Is Full(CQueue)
{
    if((Front ==1 and Rear == n) or (Front == (Rear + 1)))
        return true;
    else
        return false;
 }
```

Is Empty Operation

- Checks whether the queue is empty or not.
- If front and rear are equal to the initialized value 0 then circular queue is empty.

```
bool Is Empty(CQueue)
{
    if((Front == 0) and (Rear==0))
        return true;
    else
        return false;
 }
```

Summary

- Queue is a linear data structures.
- Items inserted into the queue at the rear and deleted from the queue at the front.
- Queues can be implemented using an array or using a linked list.
- Linear or simple queues are having problem of memory re-usage.
- Circular queues to reuse the memory efficiently.
- Like stacks, queues have many applications.

LEARNING

DATA STRUCTURE & ALGORITHM

IS IMPORTANT