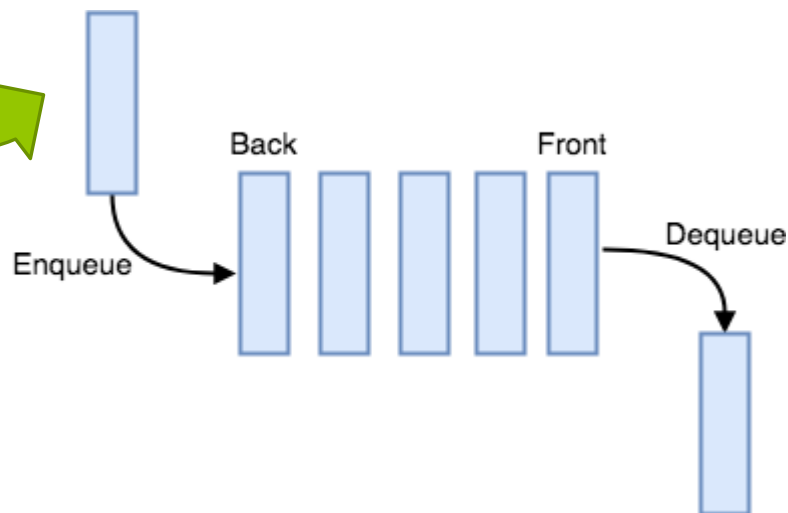


05-09-2023

Queues

Queue

- Ordered list of elements in which we
 - Add elements only at one end, called Rear end of the queue
 - Delete elements only at the other end, called Front end of the queue.



Queue Example

Accessing printer in multiuser environment-

- ❑ If a printer is in process and more than one user wants to access the printer then
- ❑ it maintains the queue for user requesting access and serves in FIFO manner for giving access.

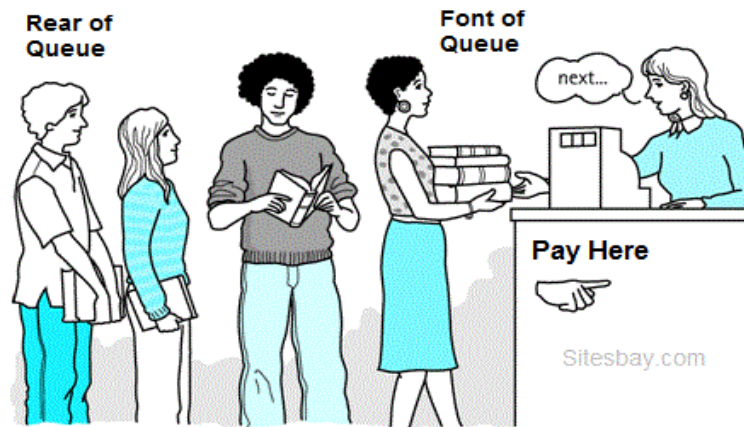
Queue

- Our software queues have counterparts in real world queues.
- We wait in
 - queues to buy pizza,
 - to enter movie theaters,
 - to drive on a turnpike, and
 - to ride on a roller coaster.



Prof. Shweta Dhawan Chachra

Queue



Real Life Example of Queue : Library Counter

- Library Counter is another best Real Life Example of Queue; here first person visits first on counter for issuing book and exit first.
- A common example of queue is movie theatre ticket counter,
 - the first person who stand in front of ticket window take ticket first and remove from line and
 - new person always stand in line from end.

Typical uses of queues are in simulations and operating systems

- ◉ **Ready Queue:-** Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.
- ◉ **Buffer:-** Computer systems must often provide a “holding area” for messages between two internal processes or programs, or between two systems over a network.
 - ◉ This holding area is usually called a “buffer” and is often implemented as a queue, because we want the message time order to be retained.

Vidya Chaur
Station

Aditi Agam Chhavi Priyansh

$f = -1$
 $r = -1$

1st

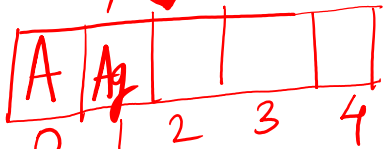
$f = 0$
 $r = 0$

$f = 0$

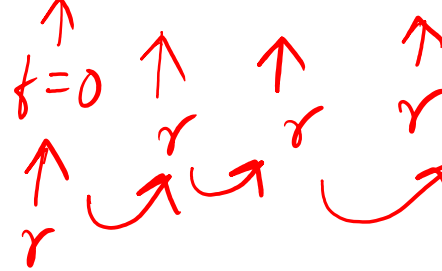
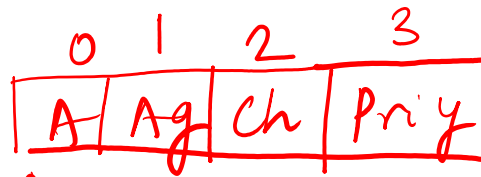
$f = -1$
front

rear = -1

empty



underflow
arr[5]



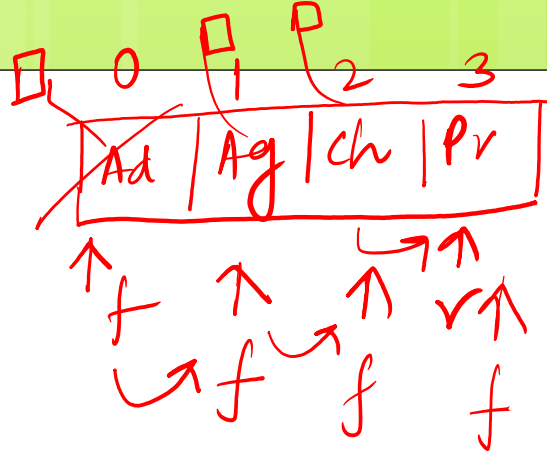
$r = r + 1$

arr[0] = "Aditi"

arr[1] =

Queue is full/
Overflow

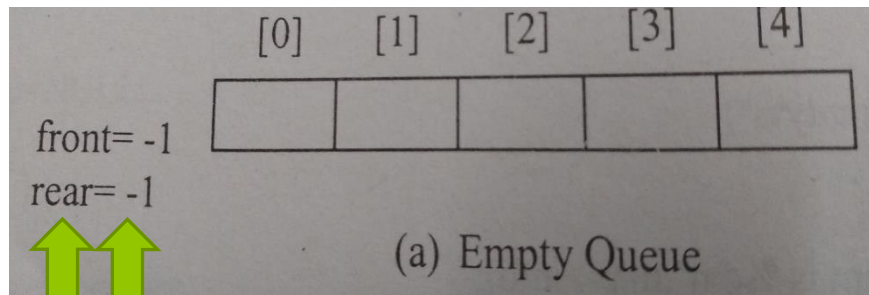
$r = ? \text{ MAX} - 1$



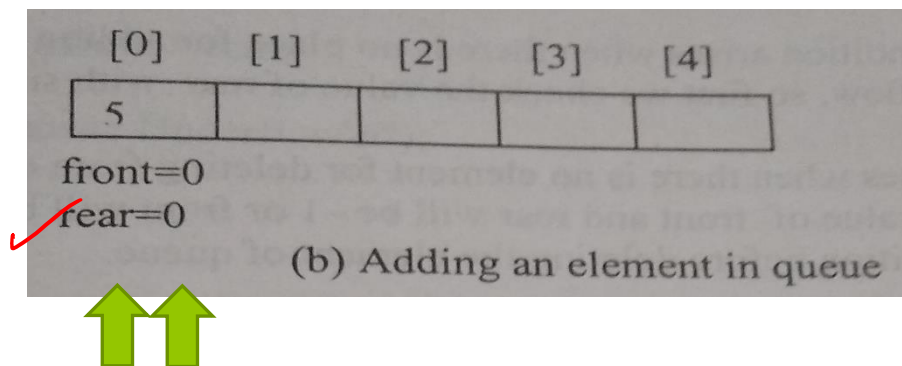
$$f = f + 1$$

Dequeue/Deleting

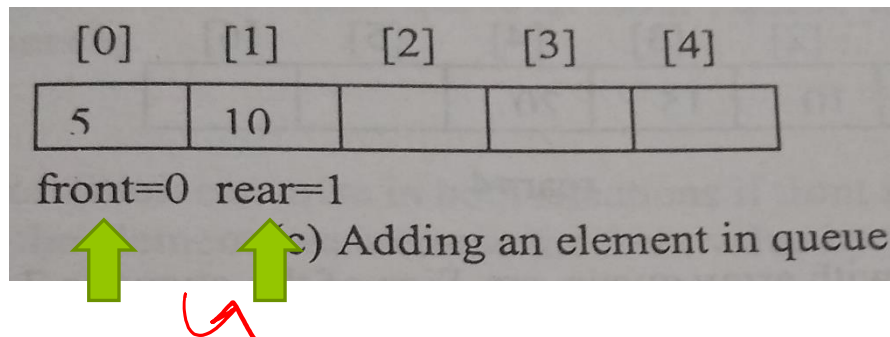
Working of Queue



Initially Queue is empty



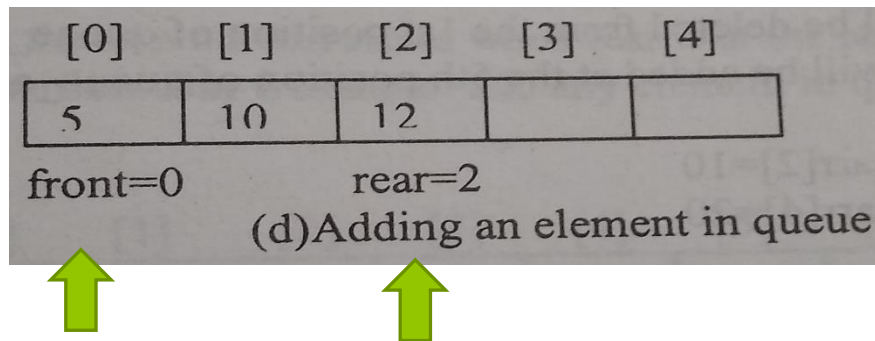
Adding 1st element,
both front and rear
pointing to the First
element



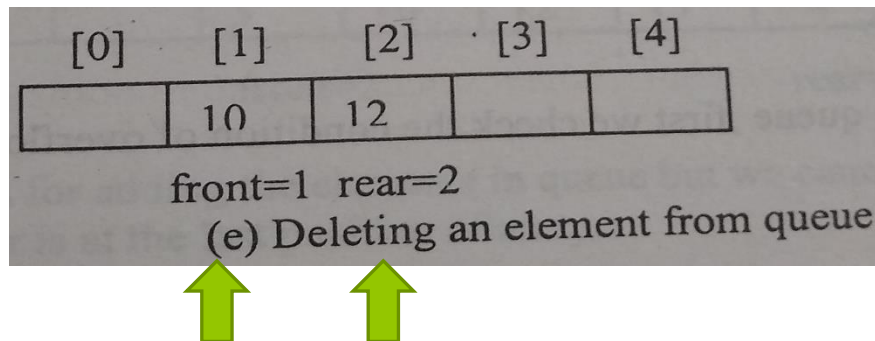
Adding 2nd element,
1) Rear gets
incremented,
2) Insertion takes place
at rear end.

Prof. Shweta Dhawan Chachra

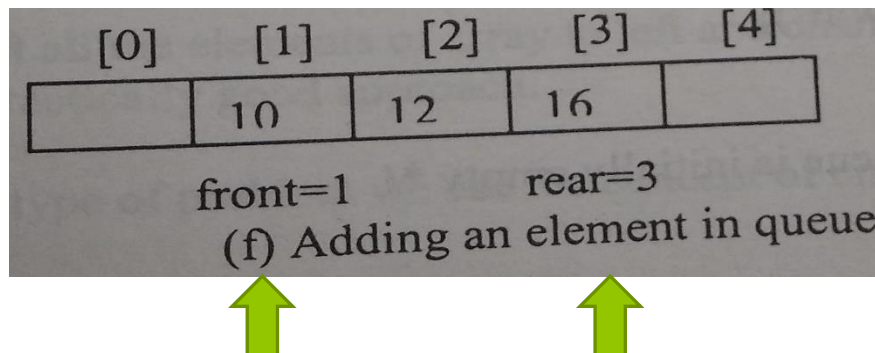
Working of Queue



Adding 3rd element,
Rear gets incremented,
As Insertion takes place
at rear end.

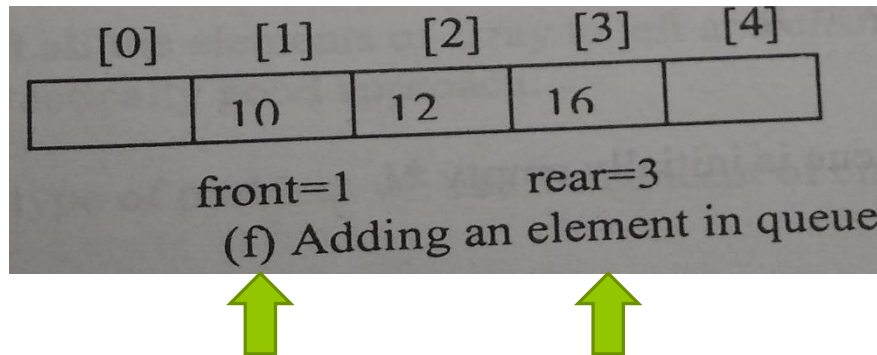


Deleting an element,
1) Deletion takes place
at Front end.
2) Front gets
incremented,



Adding other element,
Rear gets incremented,
As Insertion takes place
at rear end.

Working of Queue



Delete 10
Delete 12
Delete 16

05-09-2023

Array Representation of Queue

Queue Implementation using Arrays

Two Variables Needed-

- **Rear**

- Keeps the status of **Last element added** in the queue

- **Front**

- Keeps the status of **First element** of the queue

Conditions-

- **Queue Overflow condition,**
 - There is no place for adding elements in queue.
 - So need to **check the Value of rear with the size of array**

```
if (rear==MAX-1)
{
    printf("Queue Overflow\n")
}
```

Conditions-

- **Queue Underflow condition,**
 - If there is no element in queue.
 - Either:
 - The value of **front and rear will be -1 or**
 - **Front will be greater than rear.**

```
if (front == -1 || front > rear)
{
    printf("Queue Underflow\n");
    return ;
}
```

Queue Implementation using Arrays

```
#define MAX 5  
int queue_arr[MAX];  
int rear = -1;  
int front = -1;
```


Insert Operation

```
insert()
{
    int added_item;
    if (rear==MAX-1)
        printf("Queue Overflow\n");
    else
    {
        if (front==-1)                /*If queue is initially empty */
            front=0;

        printf("Input the element for adding in queue : ");
        scanf("%d", &added_item);

        rear=rear+1;
        queue_arr[rear] = added_item ;

    }
}/*End of insert()*/
```

- 1) Rear gets incremented,
- 2) Insertion takes place at rear end.

Delete Operation

```
del()
{
    if (front == -1 || front > rear)
    {
        printf("Queue Underflow\n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n",
queue_arr[front]);
        front=front+1;
    }
}/*End of del() */
```

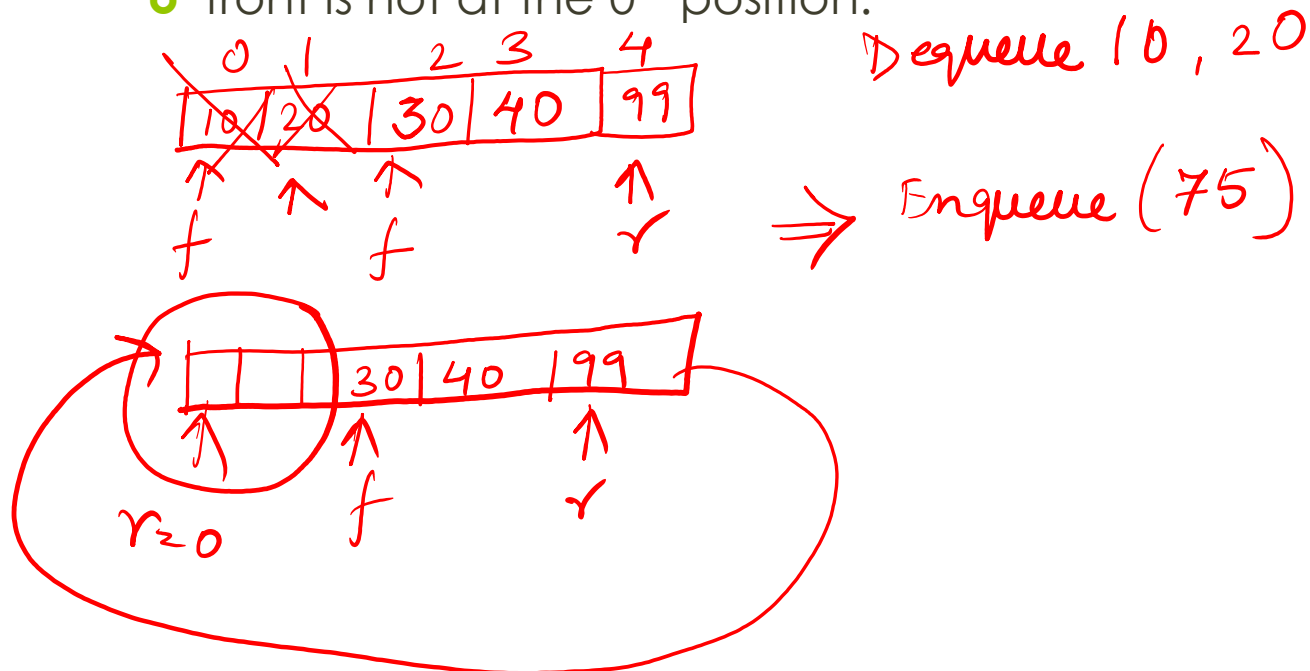
- 1) Deletion takes place at Front end.
- 2) Front gets incremented,

Display Operation

```
display()
{
    int i;
    if (front == -1)
        printf("Queue is empty\n");
    else
    {
        printf("Queue is :\n");
        for(i=front; i<= rear; i++)
            printf("%d ",queue_arr[i]);
        printf("\n");
    }
}/*End of display() */
```

Problem-

- A situation arises when
 - rear is at the last position of array and
 - front is not at the 0th position.

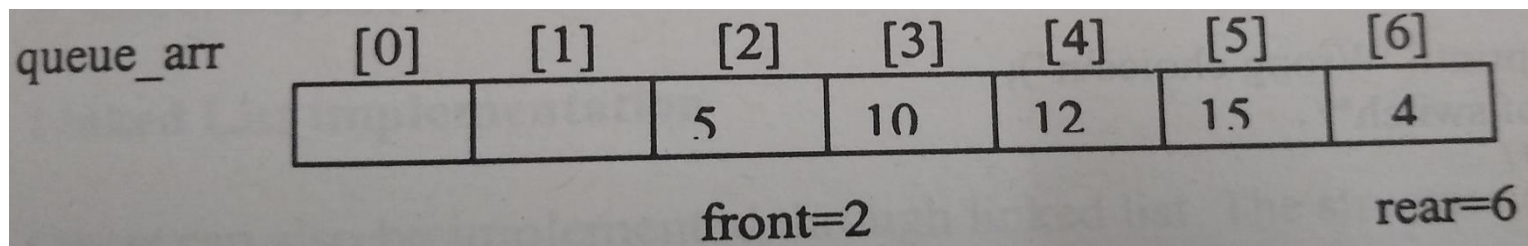


- A situation arises when
 - rear is at the last position of array and
 - front is not at the 0th position.



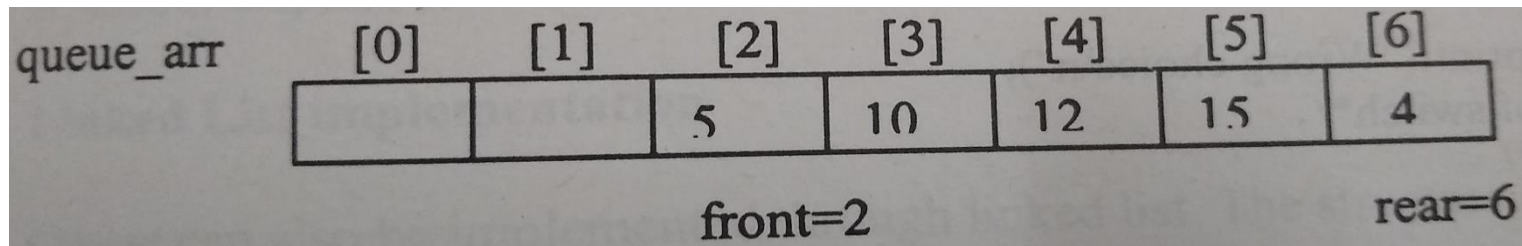
Problem-

- There are 2 spaces for adding elements in queue
- But we cannot add any element in queue because rear is at the $n-1$ th position/last position of the array.



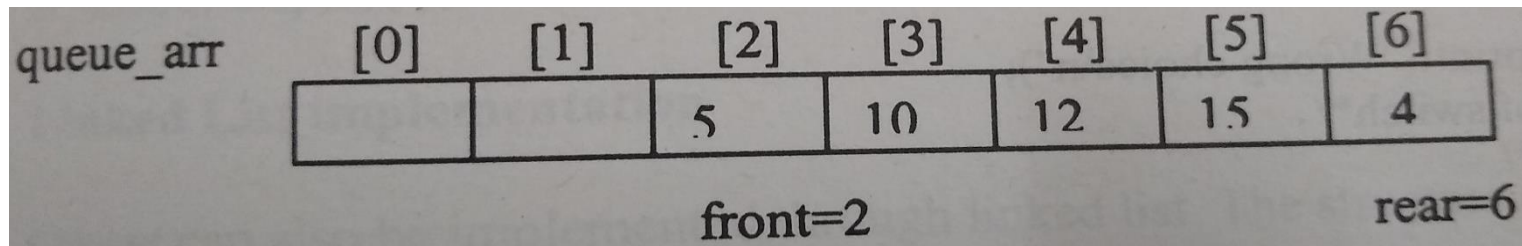
Solution-

- 1st Approach
 - Shift all elements of array to left and change the position of front and rear but it is not practically good approach



Solution-

- Other approach-
 - Use the concept of Circular queue**



ISRO | ISRO CS 2017 | Question 53

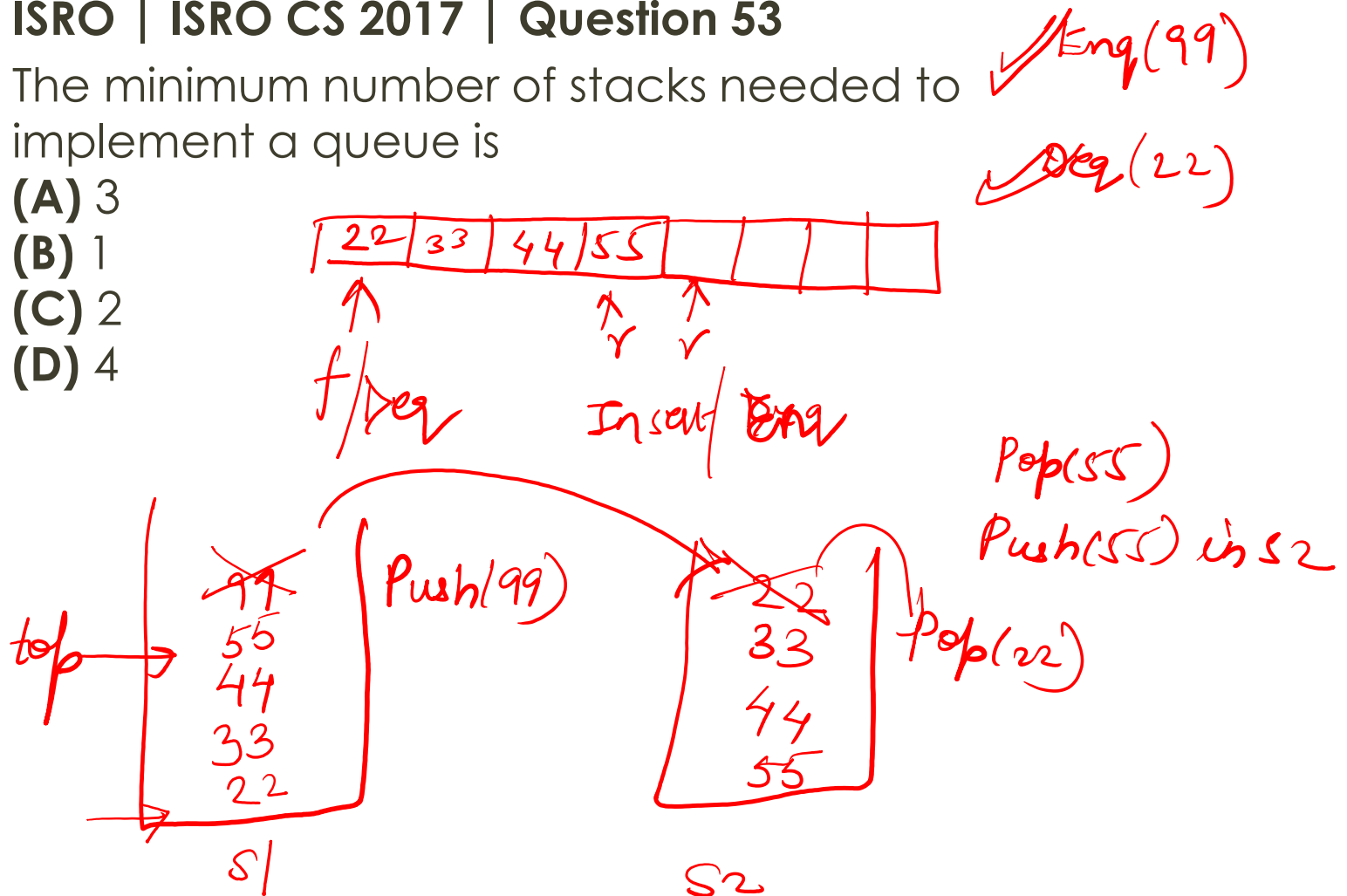
The minimum number of stacks needed to implement a queue is

- (A) 3
- (B) 1
- (C) 2
- (D) 4

ISRO | ISRO CS 2017 | Question 53

The minimum number of stacks needed to implement a queue is

- (A) 3
- (B) 1
- (C) 2
- (D) 4



ISRO | ISRO CS 2017 | Question 53

The minimum number of stacks needed to implement a queue is

- (A) 3
- (B) 1
- (C) 2
- (D) 4

Answer: (C)

Keep 2 stacks, let's call them inbox and outbox.

Enqueue:

Push the new element onto inbox

Dequeue:

If outbox is empty, refill it by popping each element from inbox and pushing it onto outbox

Pop and return the top element from outbox

05-09-2023

Linked List Representation of Queues

Linked List Representation of Queues

- Queue can also be implemented through linked list.
- The structure of the node will be as:

```
struct node  
{  
    int data;  
    struct node*link;  
}
```

Queue Implementation using Singly Linked List

Two Pointers Needed-

- **Front**

- Will point to the 1st node of the linked list

- **Rear**

- Will point to the last node of linked list

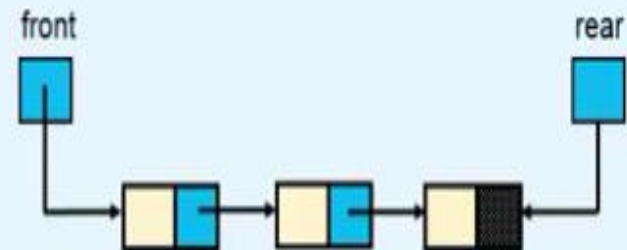
Linked implementation of Queue

Beginning of list Front end of queue

End of list Rear end of queue

Insertion

Add a node at the end of the list



Deletion

Delete a node from the beginning of the list

Working of Queues

Empty Queue

front



rear

Insert 27


front



rear

Insert 36

front

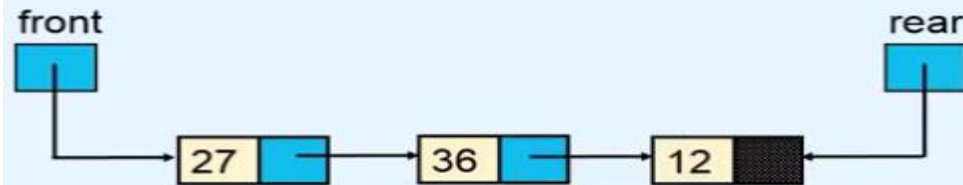


rear

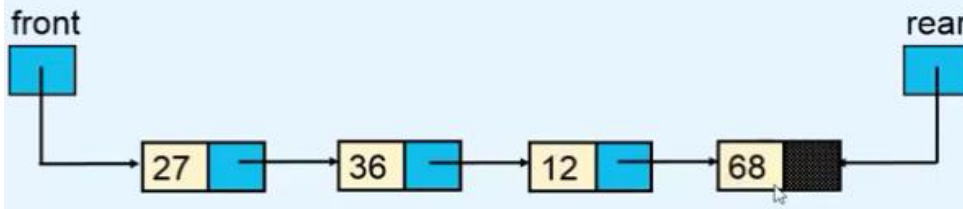
Prof. Shweta Dhawan Chachra

Working of Queues

Insert 12

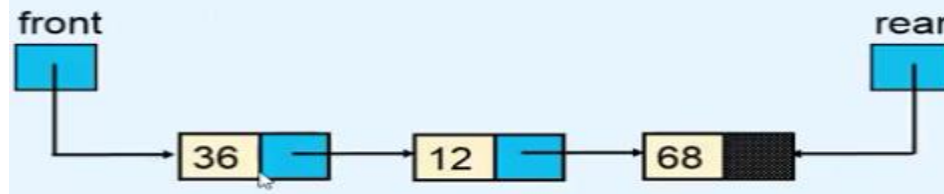


Insert 68



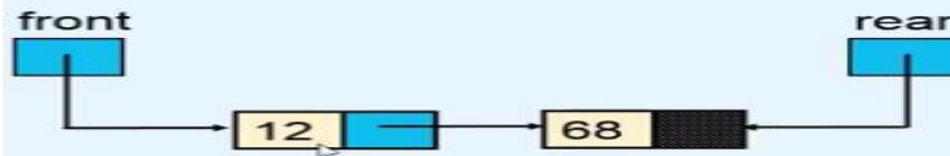
Working of Queues

Delete



Deleted element = 27

Delete

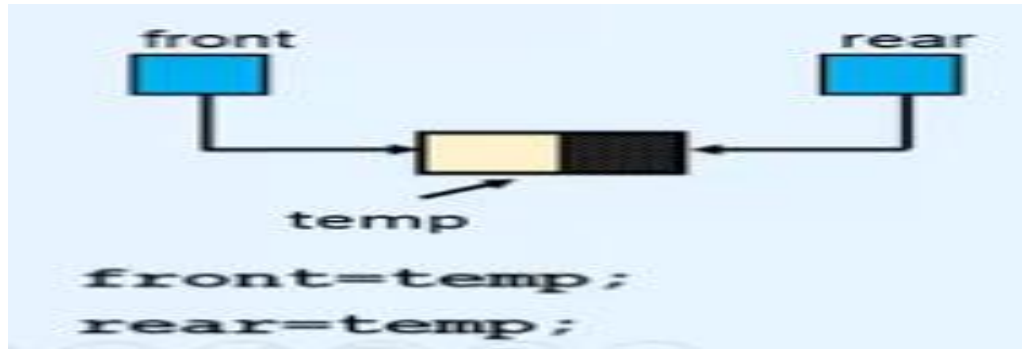


Deleted element = 36

Insert Operation

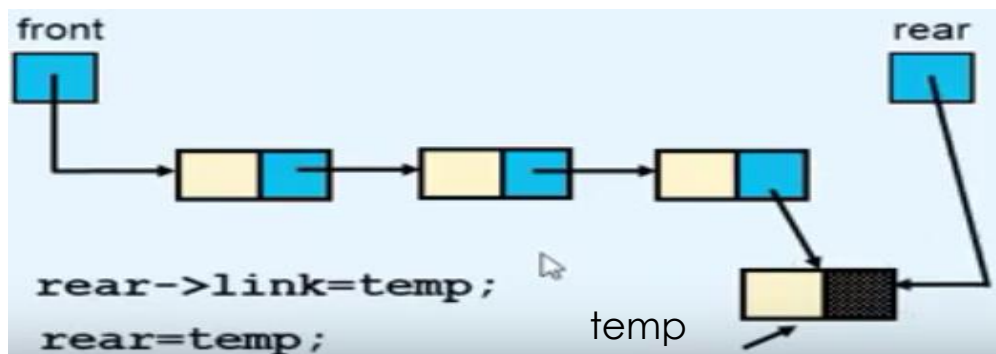
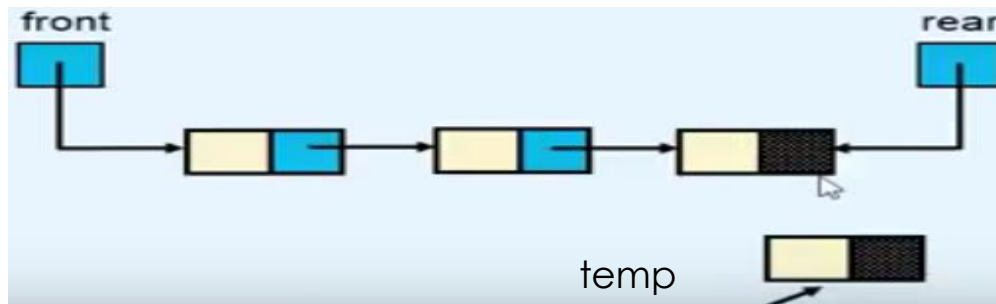
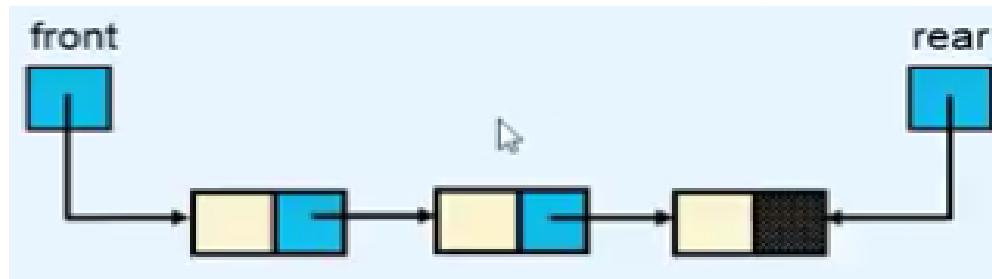
- For Insertion in Queue,
 - We add element at the end of the linked list.

Insert Operation



- If the linked list is empty,
front=NULL
rear=NULL
- After insertion, Both rear and front will point to the newly inserted node

Insert Operation



If the linked list is not empty,

- Rear will point to the newly inserted node

Insert Operation

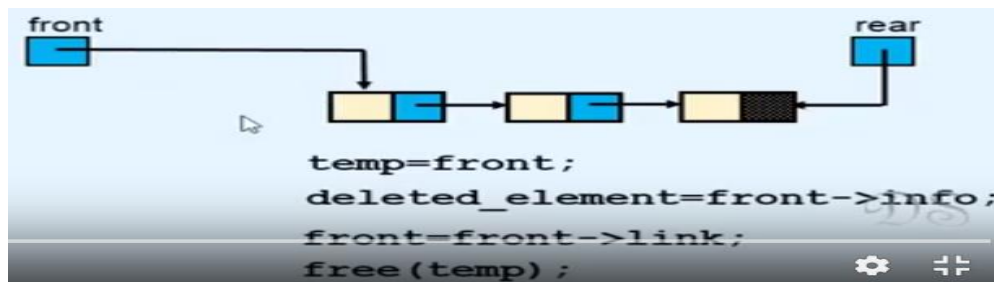
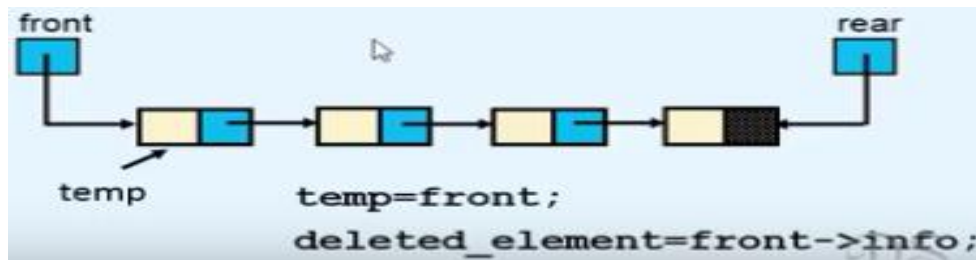
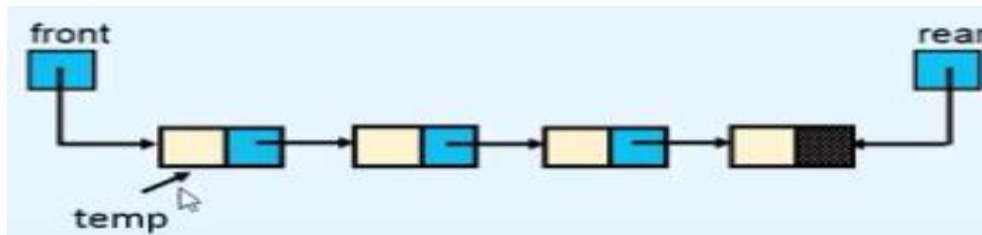
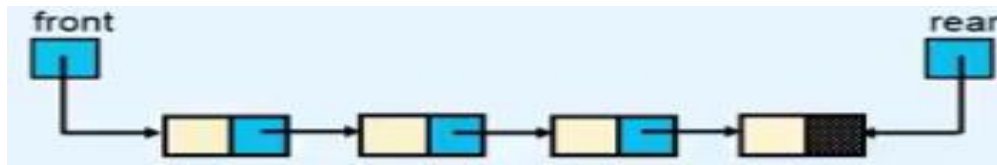
```
insert()
{
    struct node *tmp;
    int added_item;
    tmp = (struct node *)malloc(sizeof(struct node));
    printf("Input the element for adding in queue :");
    scanf("%d",&added_item);

    tmp->info = added_item;
    tmp->link=NULL;
    if(front==NULL)                                /*If Queue is empty*/
        front=tmp;
        rear=tmp;
    else
        rear->link=tmp;
        rear=tmp;
/*End of insert()*/
```

Delete Operation

- For Deletion in Queue,
 - We delete the first node of the linked list

Deletion Operation



Delete Operation

```
del()
{
    struct node *tmp;
    if(front == NULL)
        printf("Queue Underflow\n");
    else
    {
        tmp=front;
        printf("Deleted element is %d\n",tmp->info);
        front=front->link;
        free(tmp);
    }
}/*End of del()*/
```

Display Operation

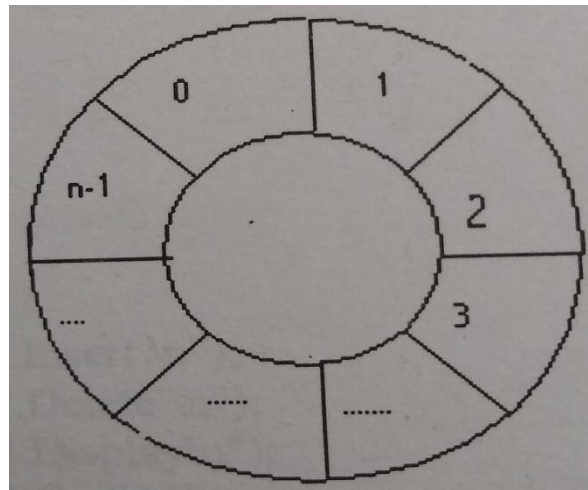
```
display()
{
    struct node *ptr;
    ptr = front;
    if(front == NULL)
        printf("Queue is empty\n");
    else
    {
        printf("Queue elements :\n");
        while(ptr != NULL)
        {
            printf("%d ",ptr->info);
            ptr = ptr->link;
        }
        printf("\n");
    }/*End of else*/
}/*End of display()*/
```

05-09-2023

Circular Queue

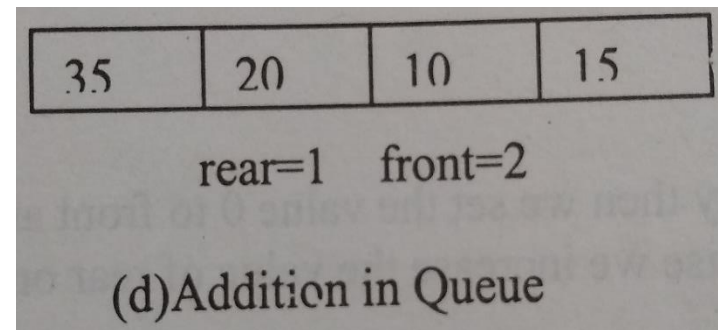
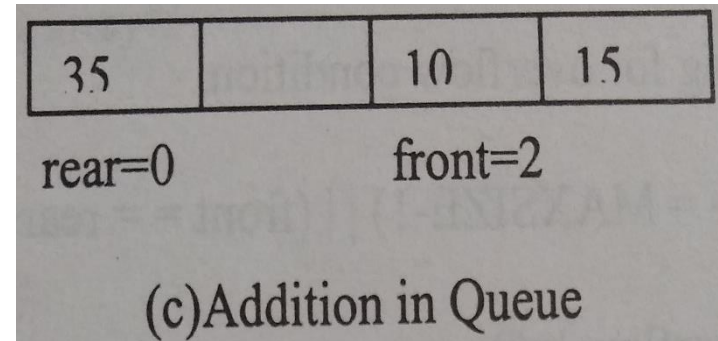
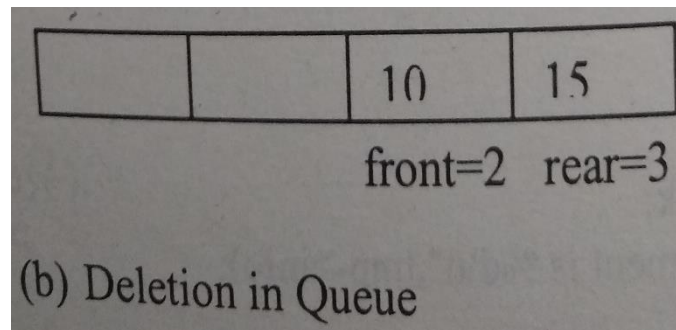
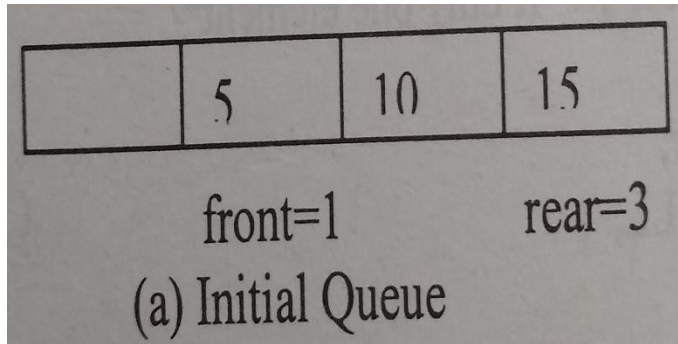
Circular Queue

- As in a circle, after last element, first element occurs
- Similarly we assume that after last element of queue, the 1st element will occur

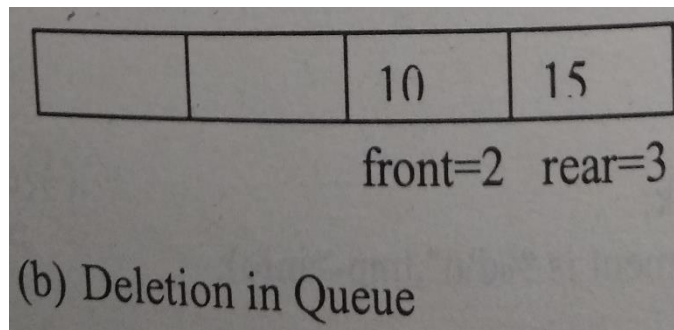
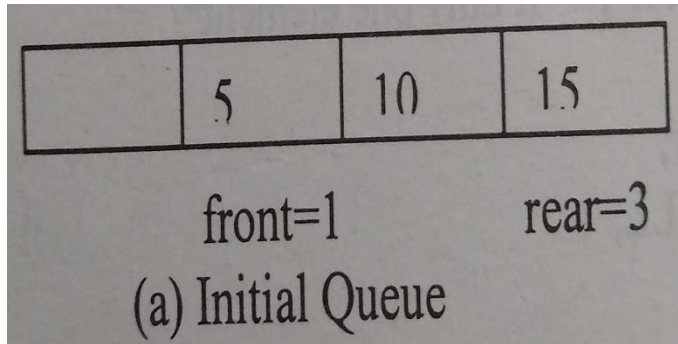


- After $n-1$ the element, 0th element occurs.

Working of Circular Queue



Working of Circular Queue



Working of Circular Queue

| | | | |
|-----------------------|--|---------|----|
| 35 | | 10 | 15 |
| rear=0 | | front=2 | |
| (c) Addition in Queue | | | |

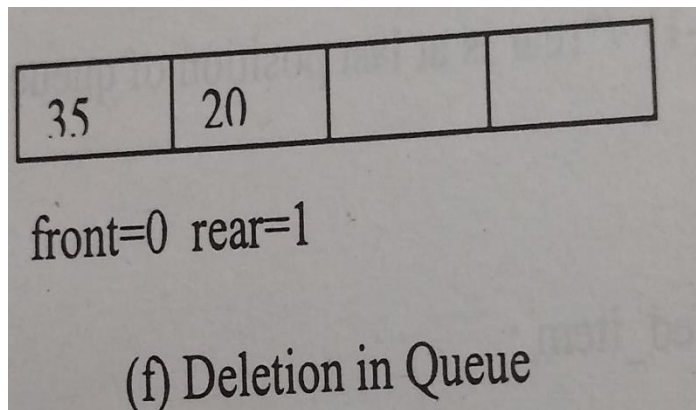
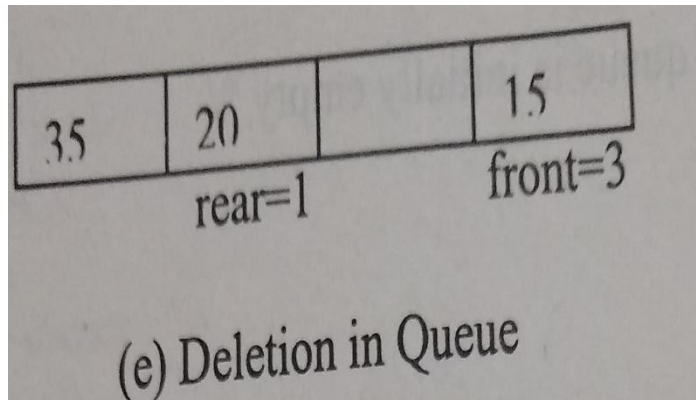
| | | | |
|-----------------------|---------|----|----|
| 35 | 20 | 10 | 15 |
| rear=1 | front=2 | | |
| (d) Addition in Queue | | | |

CIRCULAR EFFECT

**Rear comes and stands
just behind front**

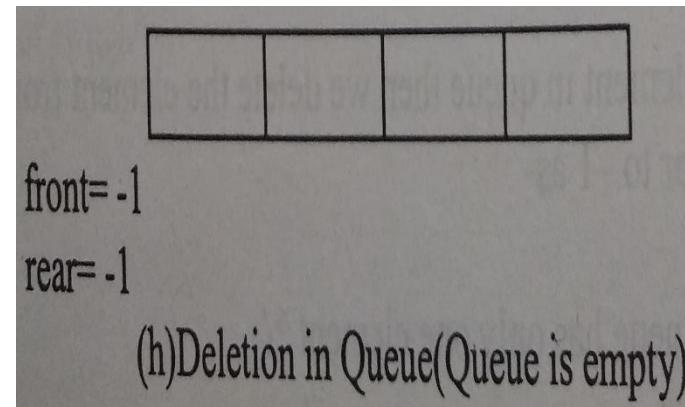
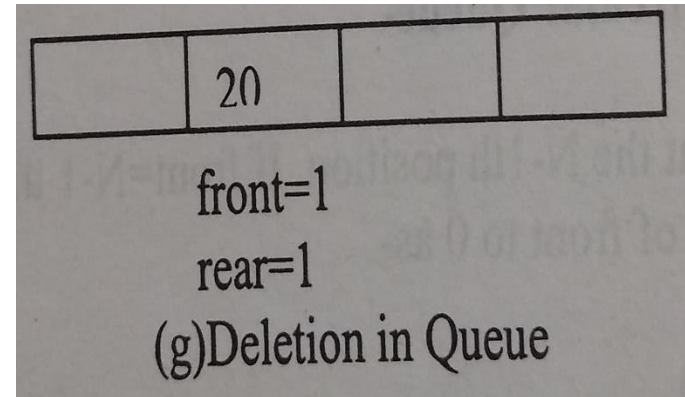
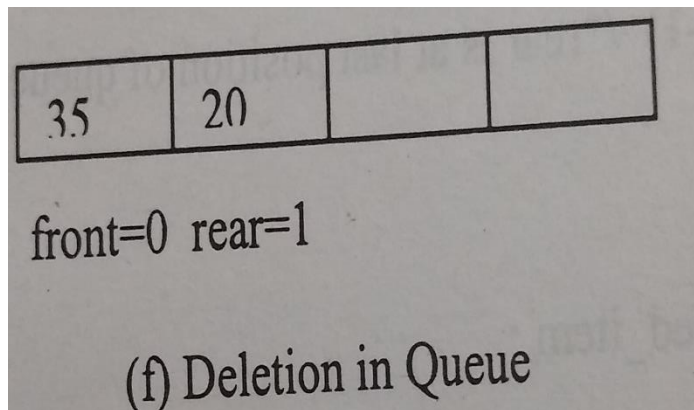
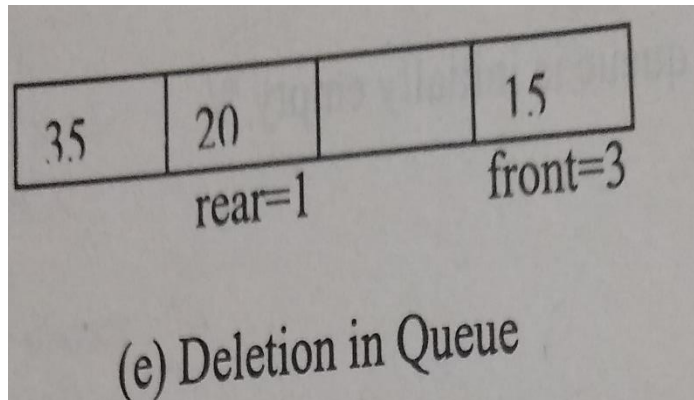
??

Working of Circular Queue



CIRCULAR EFFECT

Working of Circular Queue



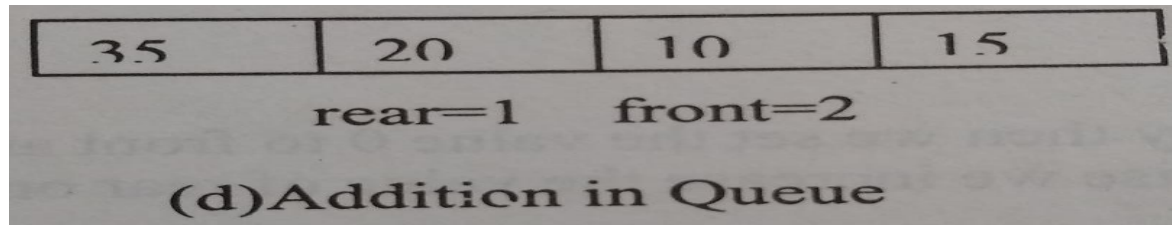
Circular Queue Implementation using Arrays

```
# define MAX 5  
int cqueue_arr[MAX];  
int front = -1;  
int rear = -1;
```

05-09-2023

Insert Operation In Circular Queue

Overflow Condition in Circular Queue



If $\text{Front} = \text{Rear} + 1$

CIRCULAR EFFECT

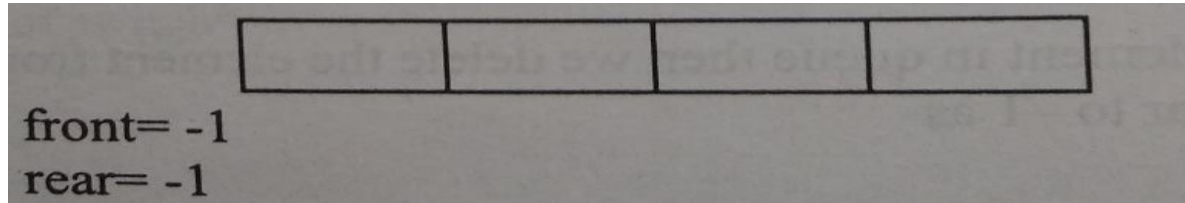
Or

If $\text{Front} = 0$ and $\text{Rear} = \text{Maxsize} - 1$

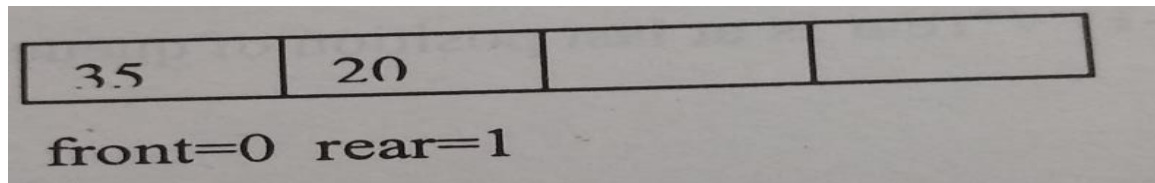
Overflow Condition in Circular Queue

```
if((front == 0 && rear == MAX-1) || (front == rear+1))  
{  
    printf("Queue Overflow \n");  
    return;  
}
```

Add operation in Circular Queue



- If Queue is initially empty,
 - Then we set front = 0 and rear = 0
 - Then add the element in the queue



- Otherwise
 - we increase the values of rear only and
 - then the element will be added in queue

Prof. Shweta Dhawan Chachra

Add operation in Circular Queue/**Circular Effect**

- If $\text{Rear} = N-1$, then we set the values of $\text{Rear} = 0$
 - add the element at the 0th position of the array
- otherwise element will be added
 - same as in simple queue

Insert Operation

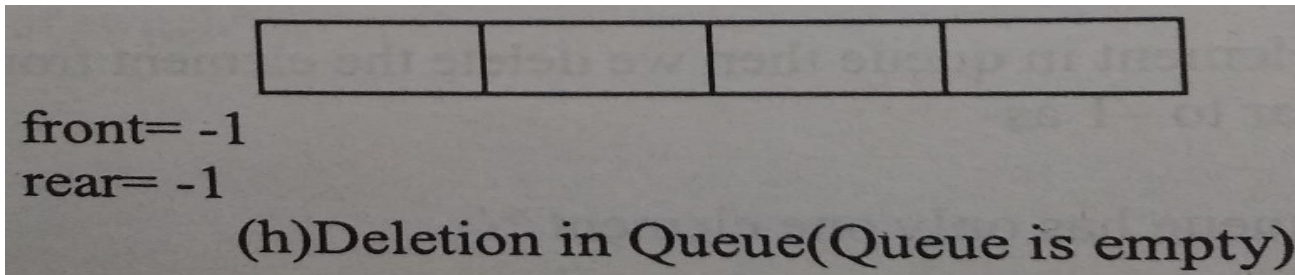
```
insert()
{
    int added_item;
    if((front == 0 && rear == MAX-1) || (front == rear+1))
    {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1) /*If queue is empty */
    {
        front = 0;
        rear = 0;
    }
```

```
else
    if(rear == MAX-1)/*rear is at last position of queue */
        rear = 0;
    else
        rear = rear+1;
    printf("Input the element for insertion in queue :");
    scanf("%d", &added_item);
    cqueue_arr[rear] = added_item ;
}/*End of insert()*/
```


05-09-2023

Delete Operation In Circular Queue

Underflow Condition in Circular Queue

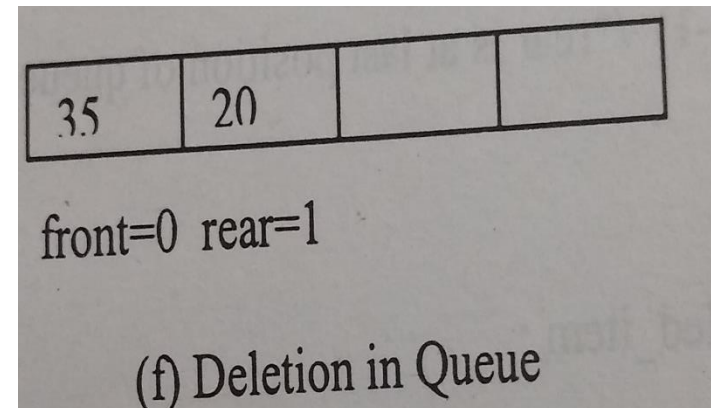
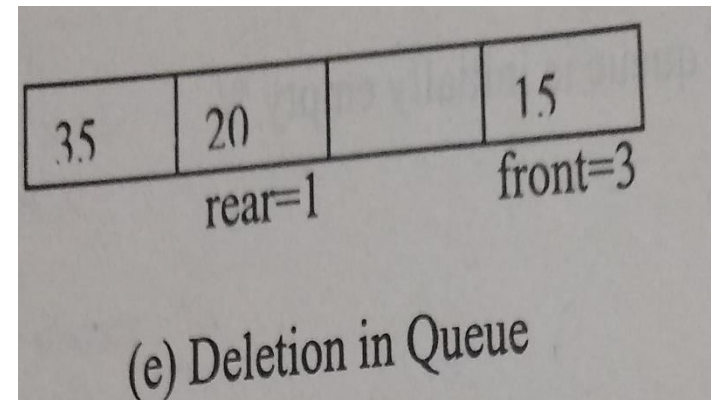


If Front = -1

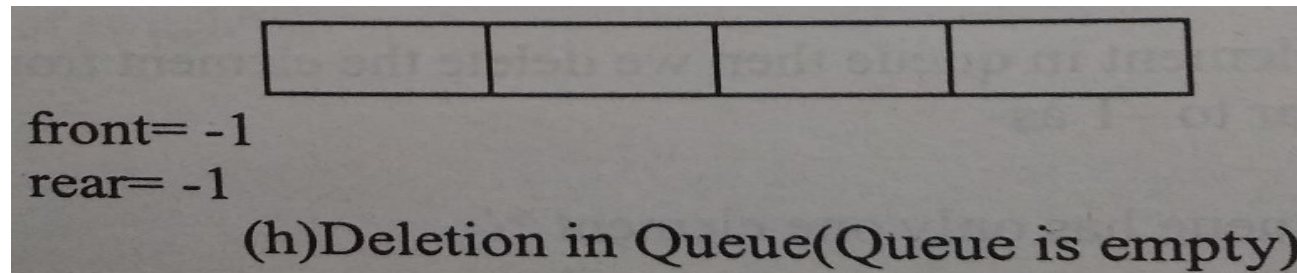
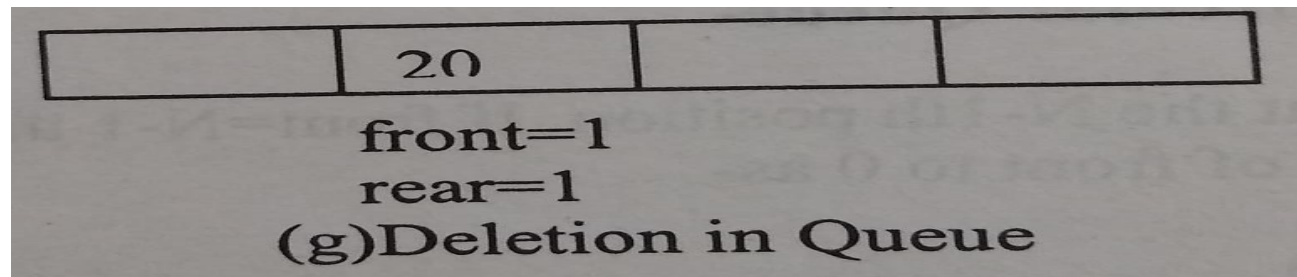
Queue is empty

Delete operation in Circular Queue/**Circular Effect**

- If $\text{Front} = N-1$, then
 - we delete the element and
 - set the values of $\text{Front} = 0$
- otherwise element will be deleted
- same as in simple queue



If there is only One element in Circular Queue



- Whenever front and rear are equal and value is other than -1,
 - It means only one element is left in the Queue,
- So , On deletion, the Queue becomes empty,
 - Thus Both Front and Rear become -1

Delete Operation

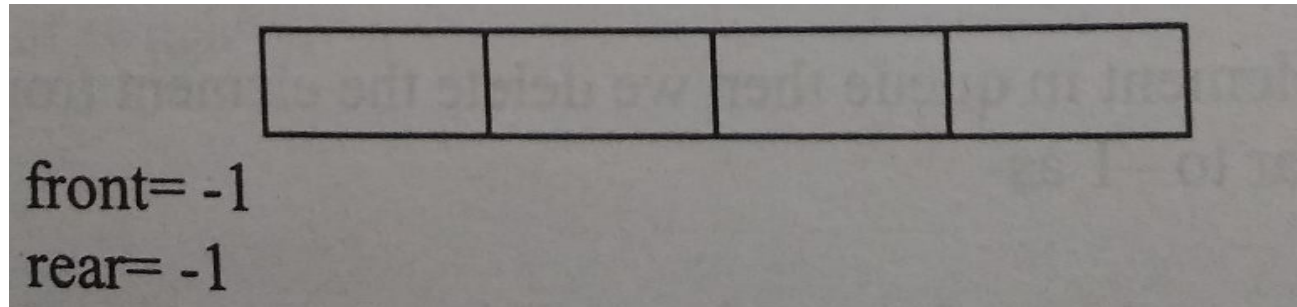
```
del()
{
    if (front == -1)
    {
        printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from
queue is : %d\n",cqueue_arr[front]);
```

```
    if(front == rear) /* queue has
only one element */
    {
        front = -1;
        rear=-1;
    }
    else
        if(front == MAX-1)
            front = 0;
        else
            front = front+1;
}/*End of del() */
```

05-09-2023

Display Operation In Circular Queue

Display operation in Circular Queue

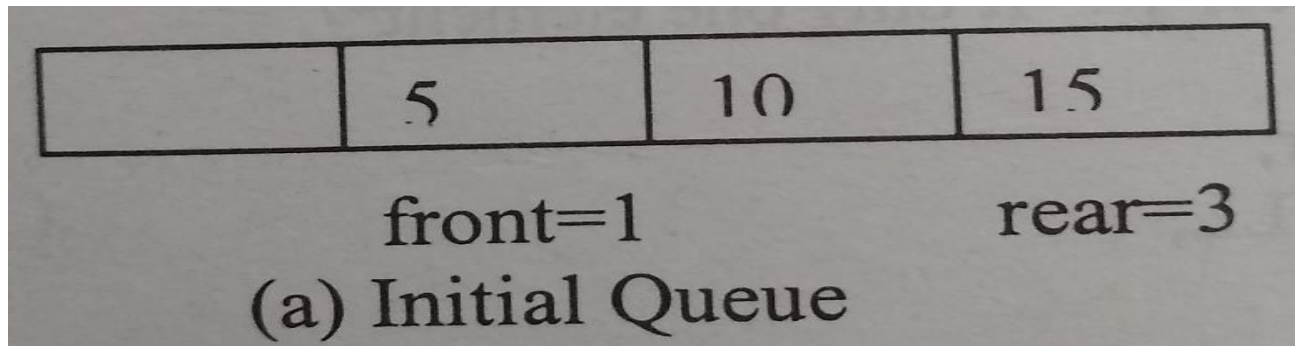


Case 1:

- First check Queue Underflow condition i.e. Queue is empty-

```
int front_pos = front, rear_pos = rear;  
if(front == -1)  
{  
    printf("Queue is empty\n");  
    return;  
}
```

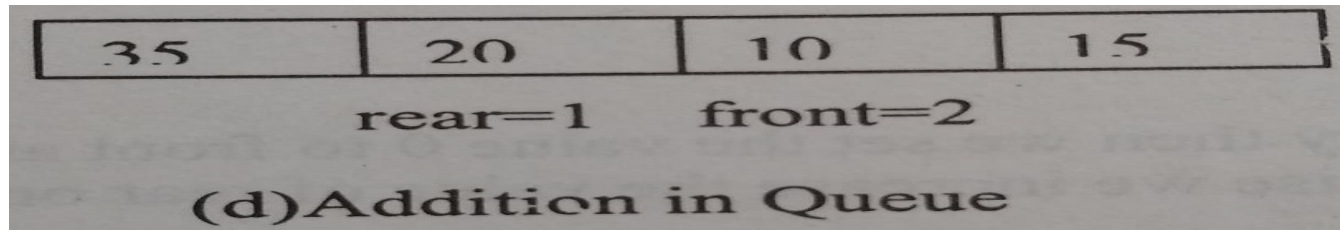
Display operation in Circular Queue



Case 2:

```
printf("Queue elements :\n");
if( front_pos <= rear_pos )
    while(front_pos <= rear_pos)
    {
        printf("%d ",cqueue_arr[front_pos]);
        front_pos++;
    }
```


Display operation in Circular Queue



Case 3:

```

else
{
    while(front_pos <= MAX-1)
    {
        printf("%d ",cqueue_arr[front_pos]);
        front_pos++;
    }
    front_pos = 0;
    while(front_pos <= rear_pos)
    {
        printf("%d ",cqueue_arr[front_pos]);
        front_pos++;
    }
}
/*End of else */

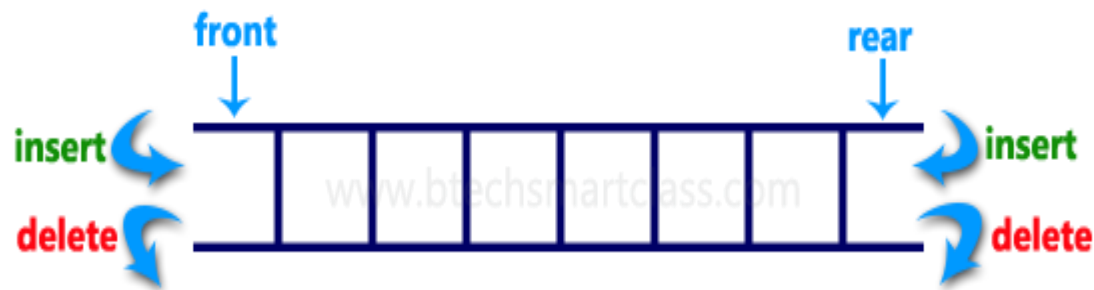
```

05-09-2023

Double ended Queue

Double ended Queue

- Dequeue
- Double Ended Queue
- We can add or delete the element from both sides.



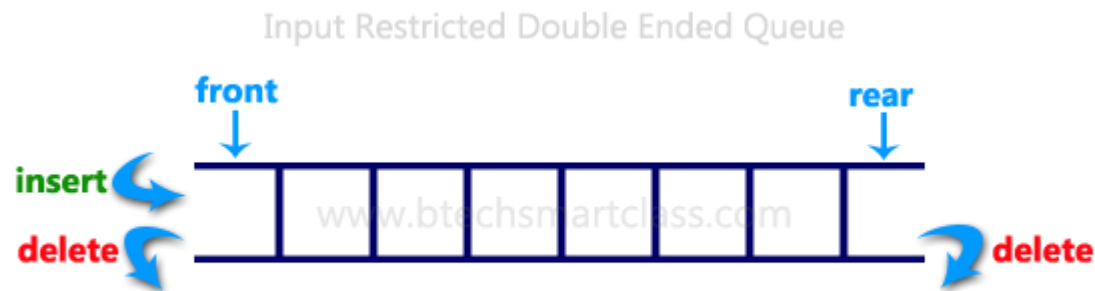
Double ended Queue

- Dequeue can be of 2 types-
 1. Input Restricted
 2. Output restricted

Double ended Queue

- Input Restricted Dequeue-

- Element can be added at only one end but element can be deleted from both sides.



- Output restricted Dequeue

- Element can be added from both sides but deletion is allowed only at one end



Double ended Queue

- Dequeue can be of 2 types-

1. Input Restricted-

- Right Input Restricted
- Left Input Restricted

- 2) Output restricted

- Right Output Restricted
- Left Output Restricted

Double ended Queue

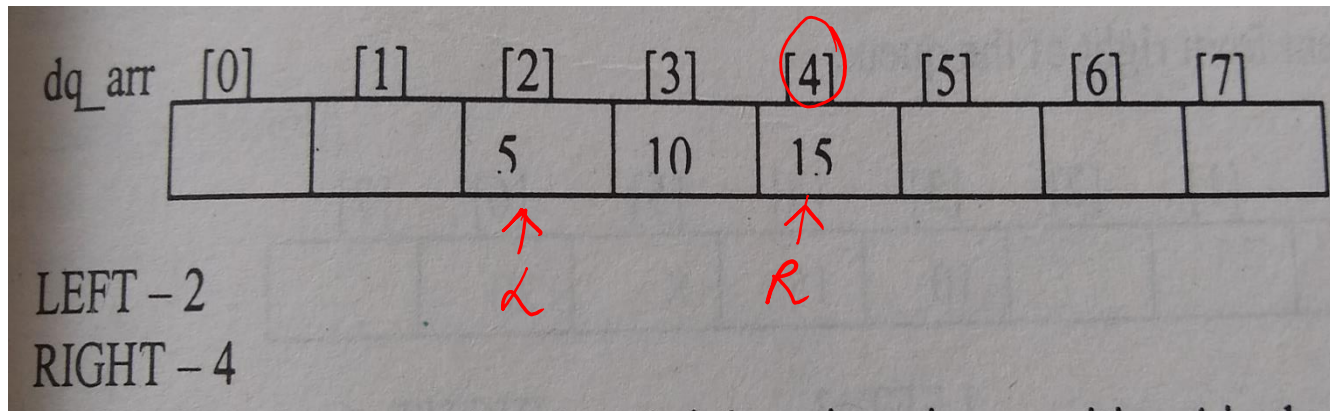
- Two Pointer needed-
 - Left
 - Right
- Left –indicates the left position of the queue
- Right –indicates the right position of the queue

Working of Dequeue

- We assume Circular array for addition and deletion operations
- Lets take an Right Input restricted Dequeue
- We can add element only on the right side of the queue but
- We can delete from both sides

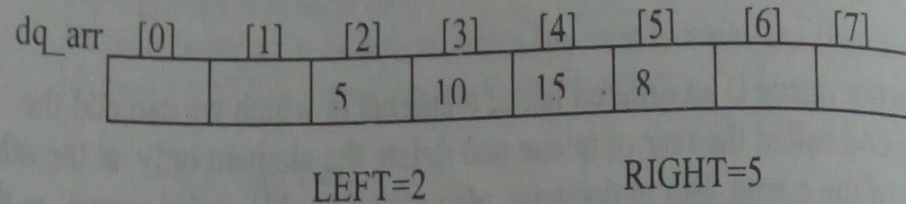
Double ended Queue

- Example of Queue



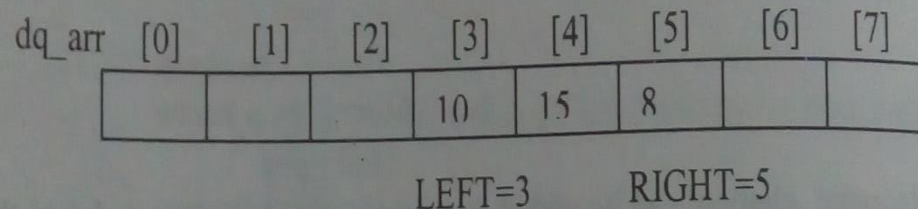
Working of Dequeue

1. Add the element 8 in the queue



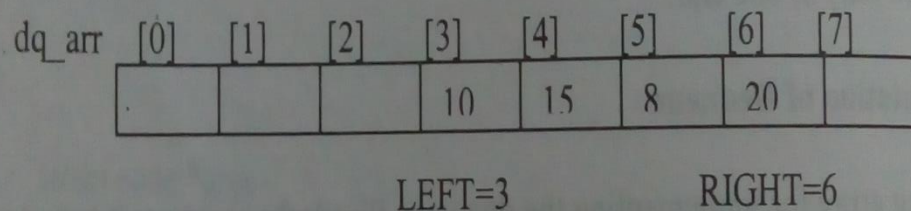
Right=Right+1
Add element at
Right

2. Delete the element from left of the queue



Delete element
from Left
Left=Left+1

3. Add the element 20 in the queue

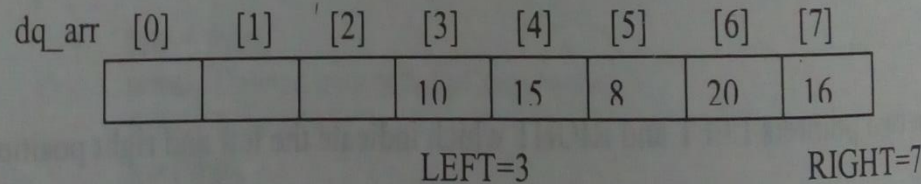


Right=Right+1
Add element at
Right

Prof. Shweta Dhawan Chachra

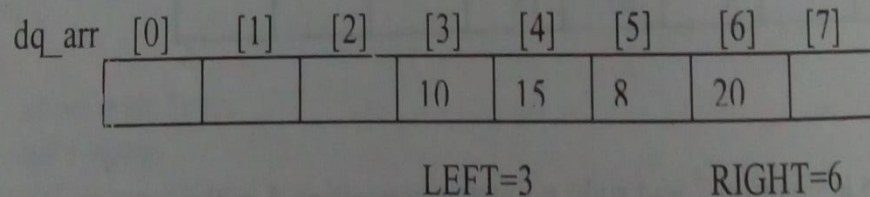
Working of Dequeue

4. Add the element 16 in the queue



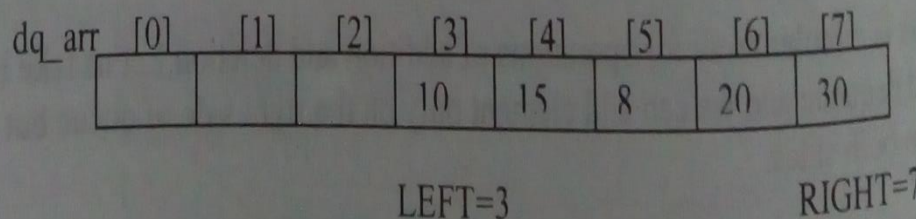
Right=Right+1
Add element at Right

5. Delete the element from right of the queue



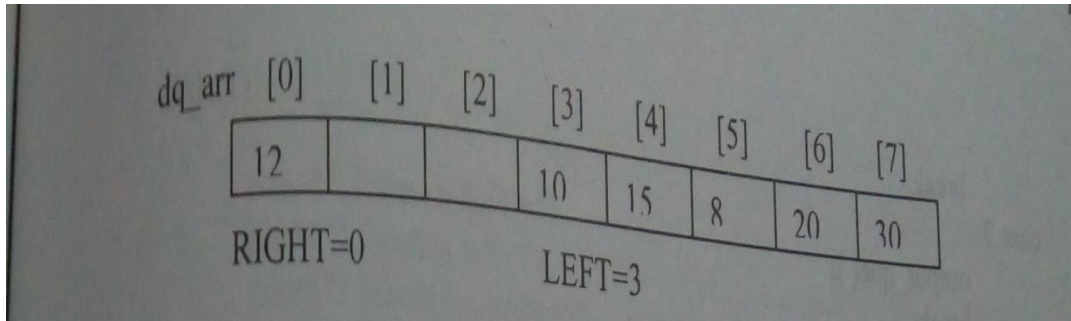
Delete element from Right
Right=Right-1

6. Add the element. 30 in the queue

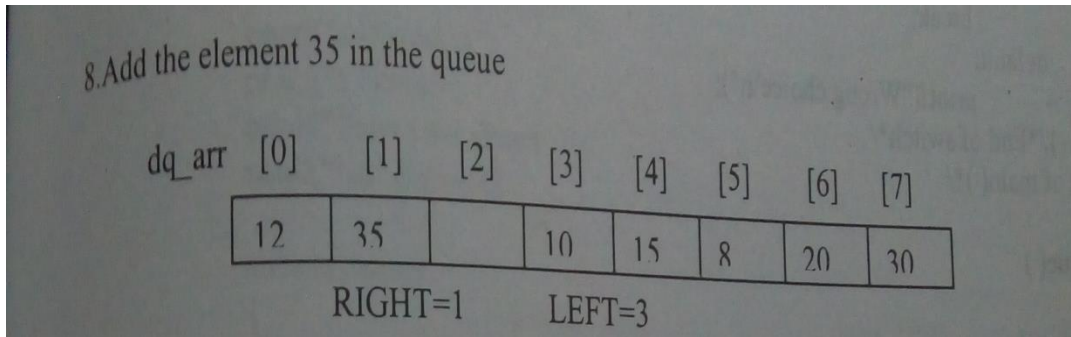


Right=Right+1
Add element at Right

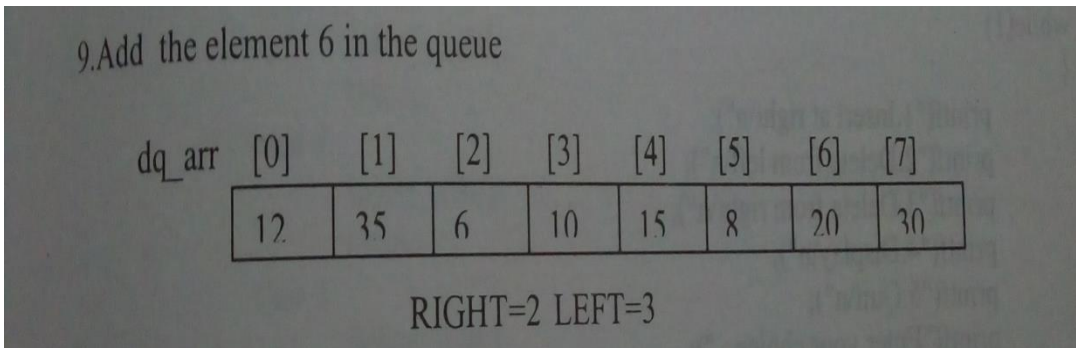
Working of Dequeue



Right=MAX-1
So Right=0
Add element at Right



Right=Right+1
Add element at Right



Right=Right+1
Add element at Right

Insert Operation

9. Add the element 6 in the queue

| | | | | | | | | |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|
| dq_arr | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
| | 12 | 35 | 6 | 10 | 15 | 8 | 20 | 30 |

RIGHT=2 LEFT=3

To check if Queue is Full-

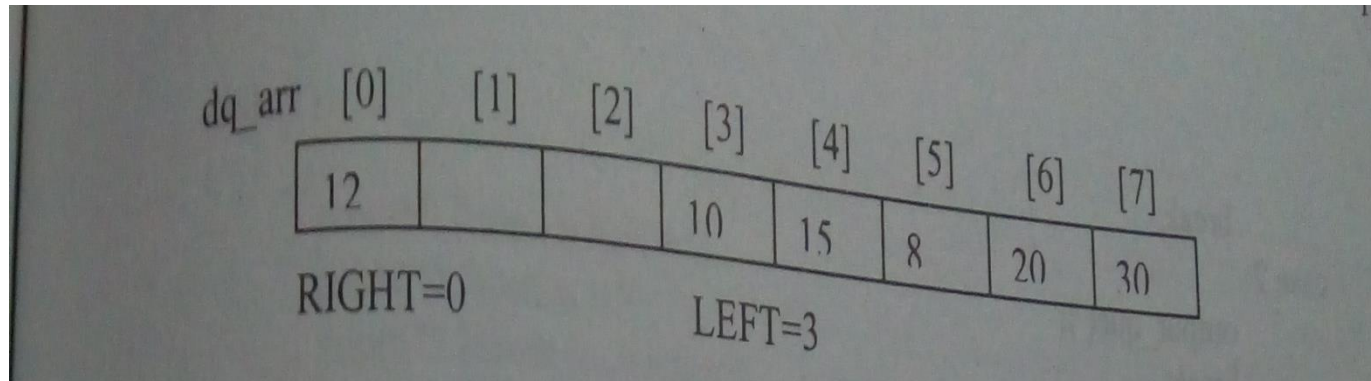
```
if((left == 0 && right == MAX-1) || (left == right+1))  
{  
    printf("Queue Overflow\n");  
    return;  
}
```

Insert Operation

If initially Queue is empty, then both the pointers are incremented

```
if (left == -1) /* if queue is initially empty */  
{  
    left = 0;  
    right = 0;  
}
```

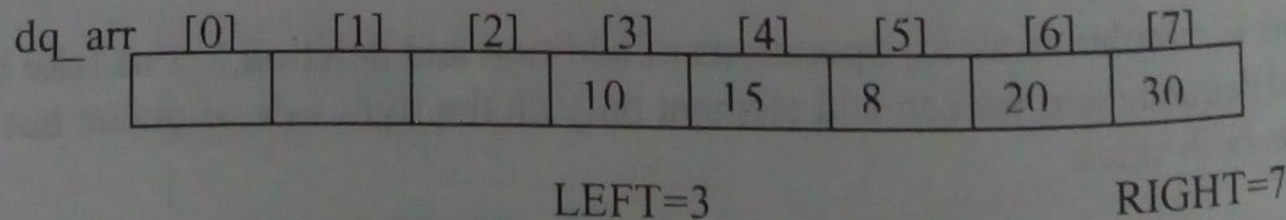
Insert Right Operation/Circular Effect-



- If `right == MAX-1` , then we set the values of `right = 0`
- add the element at the 0th position of the array

Insert Right Operation/Circular Effect-

6. Add the element. 30 in the queue



- otherwise element will be added
 - same as in simple queue, $\text{right} = \text{right} + 1$

Insert Right Operation

```
insert_right()
{
    int added_item;
    if((left == 0 && right == MAX-1)
    || (left == right+1))
    {
        printf("Queue
Overflow\n");
        return;
    }
    if (left == -1) /* if queue is
initially empty */
    {
        left = 0;
        right = 0;
    }
```

```
else
    if(right == MAX-1) /*right is at last
position of queue */
        right = 0;
    else
        right = right+1;
    printf("Input the element for adding
in queue : ");
    scanf("%d", &added_item);
    deque_arr[right] = added_item ;
}/*End of insert_right()*/
```

Insert Left Operation/Circular Effect-

- If $\text{left} == 0$, then we set the values of $\text{left} = \text{MAX}-1$
 - add the element at the $\text{MAX}-1$ th position of the array
- otherwise element will be added
 - same as in $\text{left} = \text{left}-1$

Insert Left Operation

```

insert_left()
{
    int added_item;
    if((left == 0 && right == MAX-1)
    || (left == right+1))
    {
        printf("Queue
Overflow \n");
        return;
    }
    if (left == -1)/*If queue is initially
empty*/
    {
        left = 0;
        right = 0;
    }
    else
        if(left== 0)
            left=MAX-1;
        else
            left=left-1;
        printf("Input the element for adding
in queue : ");
        scanf("%d", &added_item);
        deque_arr[left] = added_item ;
    }/*End of insert_left()*/

```

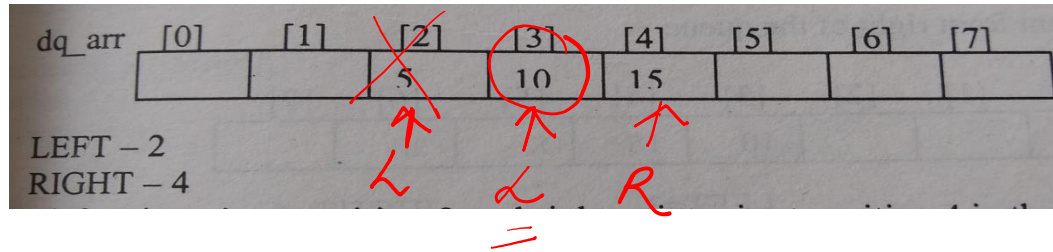
Delete Operation

To check if Queue is Empty-

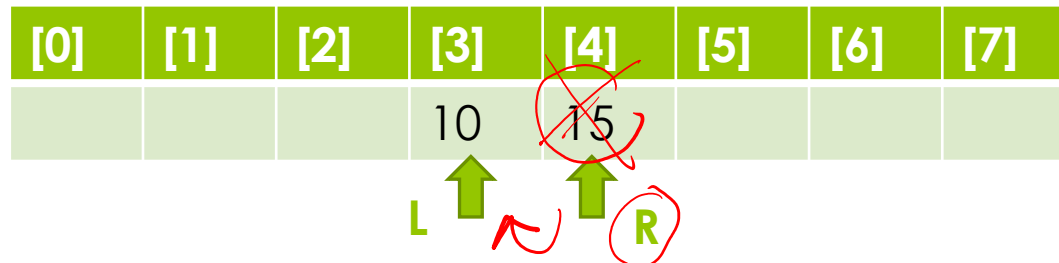
```
if (left == -1)
{
    printf("Queue Underflow\n");
    return ;
}
```

Delete Operation

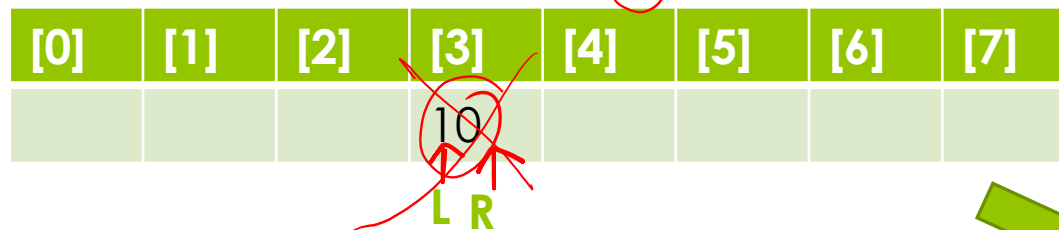
To check if Queue has only one element-



Left=2
Right=4



Delete at Left
 \checkmark Left = Left + 1
 Left = 3
 Right = 4



Delete at Right
 Left = 3
 \checkmark Right = Right - 1
 Right = 3


$L = -1$
 $R = -1$ } set as in CQ
 as in SQ

Left == Right
 Prof. Shweta Dhawan Chachra

Delete Operation

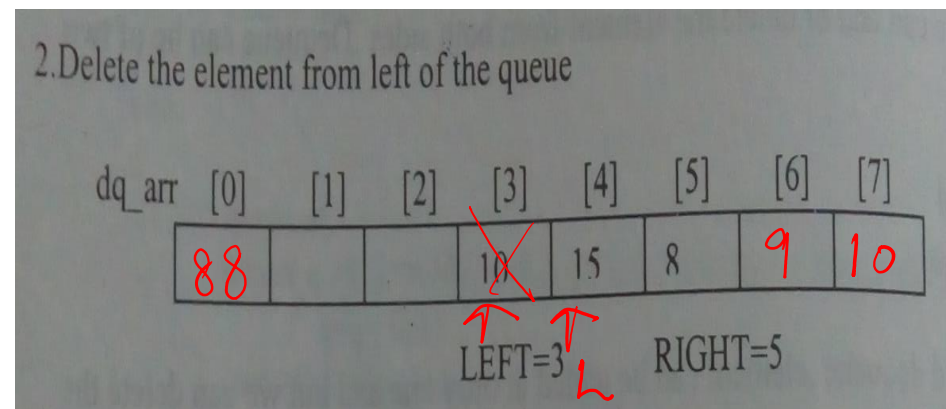
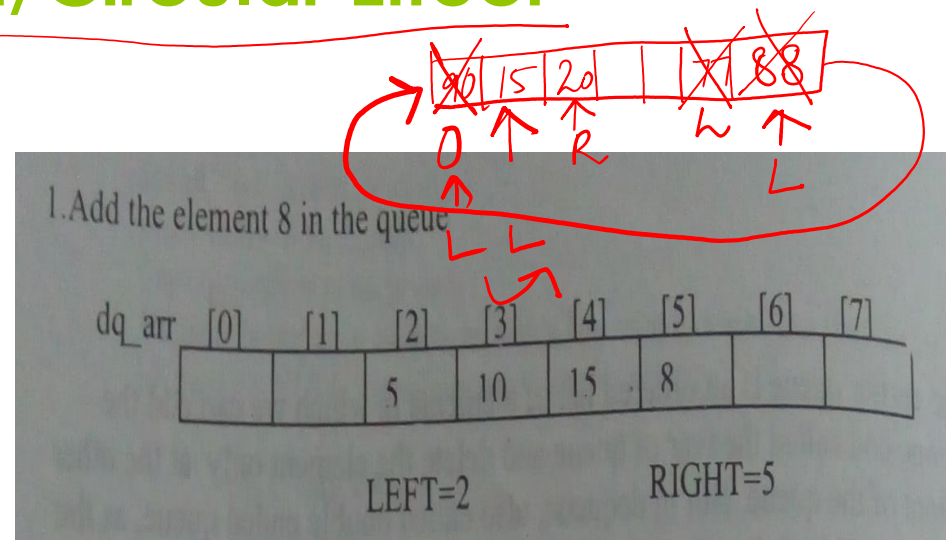
To check if Queue has only one element-

```
if(left == right) /*queue has only one element*/  
{  
    left = -1;  
    right=-1;  
}
```



Delete Left Operation/Circular Effect-

- If left == MAX-1 , then
 - Delete the element and
 - Set the values of left = 0
- otherwise element will be deleted
- And left=left+1 ✓✓



Delete Left Operation

```
delete_left()
{
    if (left == -1)
    {
        printf("Queue
Underflow\n");
        return ;
    }
    printf("Element deleted from
queue is : %d\n",deque_arr[left]);
```

```
if(left == right) /*Queue has only one
element */
{
    left = -1;
    right=-1;
}
else
    if(left == MAX-1)
        left = 0;
    else
        left = left+1;
}/*End of delete_left()*/
```


Delete Right Operation/Circular Effect-

- If $\text{right} == 0$, then
 - Delete the element and
 - Set the values of $\text{right} = \text{MAX}-1$
- otherwise element will be deleted
- And $\text{right} = \text{right}-1$



4. Add the element 16 in the queue

dq_arr [0] [1] [2] [3] [4] [5] [6] [7]

| | | | | | | | |
|--|--|--|----|----|---|----|----|
| | | | 10 | 15 | 8 | 20 | 16 |
|--|--|--|----|----|---|----|----|

LEFT=3
RIGHT=7

5.Delete the element from right of the queue

dq_arr [0] [1] [2] [3] [4] [5] [6] [7]

| | | | | | | | |
|--|--|--|----|----|---|----|--|
| | | | 10 | 15 | 8 | 20 | |
|--|--|--|----|----|---|----|--|

LEFT=3
RIGHT=6

Delete Right Operation

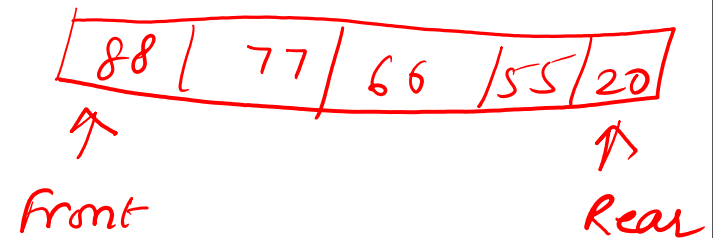
```
delete_right()
{
    if (left == -1)
    {
        printf("Queue
Underflow\n");
        return ;
    }
    printf("Element deleted from queue is
: %d\n",deque_arr[right]);

    if(left == right) /*queue has only one
element*/
    {
        left = -1;
        right=-1;
    }
    else
        if(right == 0)
            right=MAX-1;
        else
            right=right-1;
}/*End of delete_right() */
```

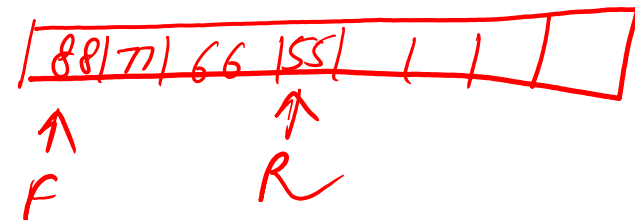
Display Operation/Circular Effect-

sq, overflow

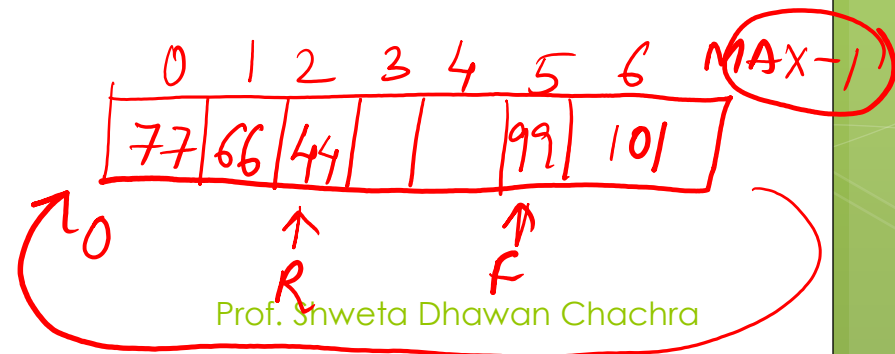
- Same as Circular Queue



(I)



(II)



Prof. Shweta Dhawan Chachra

Display Operation

```

display_queue()
{
    int front_pos = left, rear_pos = right;
    if(left == -1)
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements :\n");
    if( front_pos <= rear_pos )
    {
        while(front_pos <= rear_pos)
        {
            printf("%d", deque_arr[front_pos]);
            front_pos++;
        }
    }
    else
    {
        while(front_pos <= MAX-1)
        {
            printf("%d", deque_arr[front_pos]);
            front_pos++;
        }
        front_pos = 0;
        while(front_pos <= rear_pos)
        {
            printf("%d", deque_arr[front_pos]);
            front_pos++;
        }
    }
    /*End of else */
    printf("\n");
    /*End of display_queue() */
}

```

05-09-2023

Priority Queue

Priority Queue

- A priority queue is a data structure used for storing a set S of elements, based on a key value, which denotes the priority of that element.

Priority Queue

- *The priority determines the order in which they exit the queue.*
- *Every element of queue has some priority and based on that priority it will be processed.*

Priority Queue

- Element of higher priority will be processed/removed before the element that has low priority
- With 2 elements of same priority, FIFO rule will be the tie breaker
 - The element that comes first in the queue , will be processed first.

Priority Queue

Generally

- Smaller Numbers => Higher Priorities
- Bigger Numbers => Lower Priority

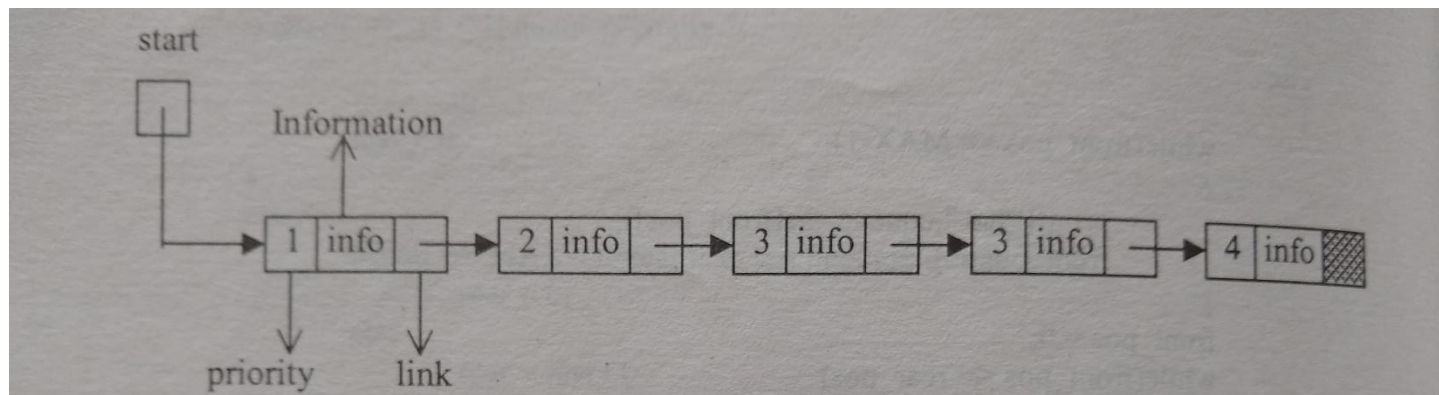
Priority Queue

- In computer implementation, Priority Queue is used In CPU scheduling algorithm.
- Processes with Higher priority are allocated the CPU first.

Linked List Implementation of Priority Queue

- Structure of Linked List-

```
struct pq{  
    int priority;  
    int data;  
    struct pq *link;  
}
```



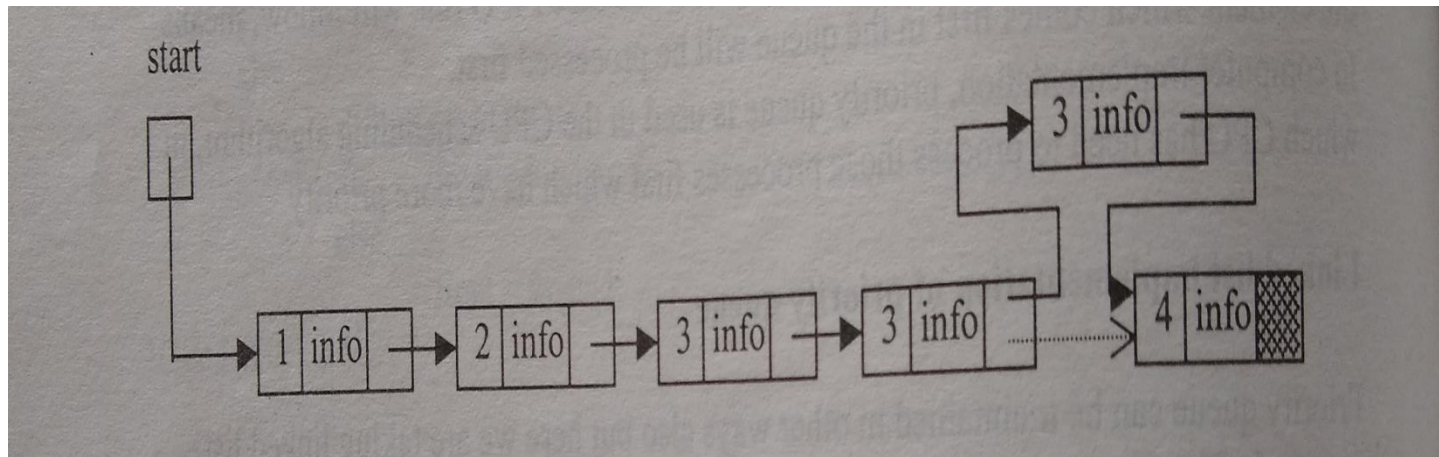
Operations in Priority Queue

Similar to other Queues, Priority Queues also have :

- **Add Operation**
- **Deletion Operation**

Add Operation in Priority Queue

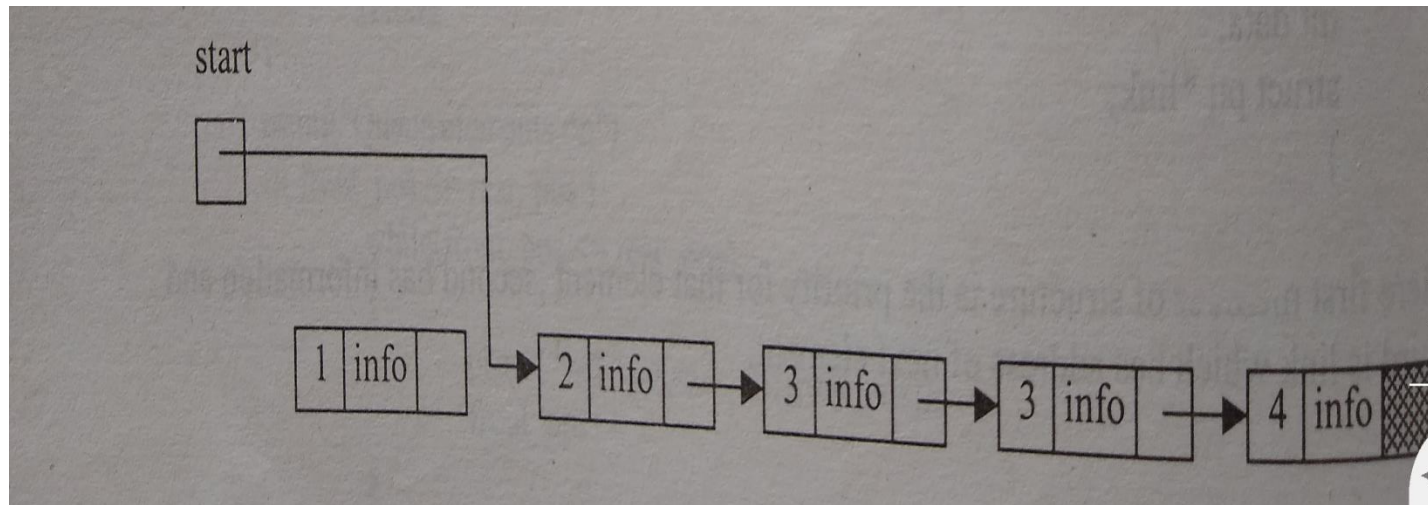
- Same as Insertion in Sorted Linked List
- Insert the new element on the basis of priority of element
- **The new element will be inserted before the element which has less priority than new element**



Prof. Shweta Dhawan Chachra

Deletion Operation in Priority Queue

- Deletion of First element of the linked list because it has more priority than other elements of Queue



Applications of Priority Queue

- 1) CPU Scheduling
- 2) Graph algorithms like Dijkstra's shortest path algorithm, Prim's Minimum Spanning Tree, etc
- 3) All queue applications where priority is involved.

extra

GATE | GATE-CS-2016 (Set 1) | Question 20

A queue is implemented using an array such that ENQUEUE and DEQUEUE operations are performed efficiently. Which one of the following statements is CORRECT (n refers to the number of items in the queue)?

- (A) Both operations can be performed in $O(1)$ time
- (B) At most one operation can be performed in $O(1)$ time but the worst case time for the other operation will be $\Omega(n)$
- (C) The worst case time complexity for both operations will be $\Omega(n)$
- (D) Worst case time complexity for both operations will be $\Omega(\log n)$

GATE | GATE-CS-2016 (Set 1) | Question 20

A queue is implemented using an array such that ENQUEUE and DEQUEUE operations are performed efficiently. Which one of the following statements is CORRECT (n refers to the number of items in the queue)?

- (A) Both operations can be performed in $O(1)$ time
- (B) At most one operation can be performed in $O(1)$ time but the worst case time for the other operation will be $\Omega(n)$
- (C) The worst case time complexity for both operations will be $\Omega(n)$
- (D) Worst case time complexity for both operations will be $\Omega(\log n)$

Answer: (A)