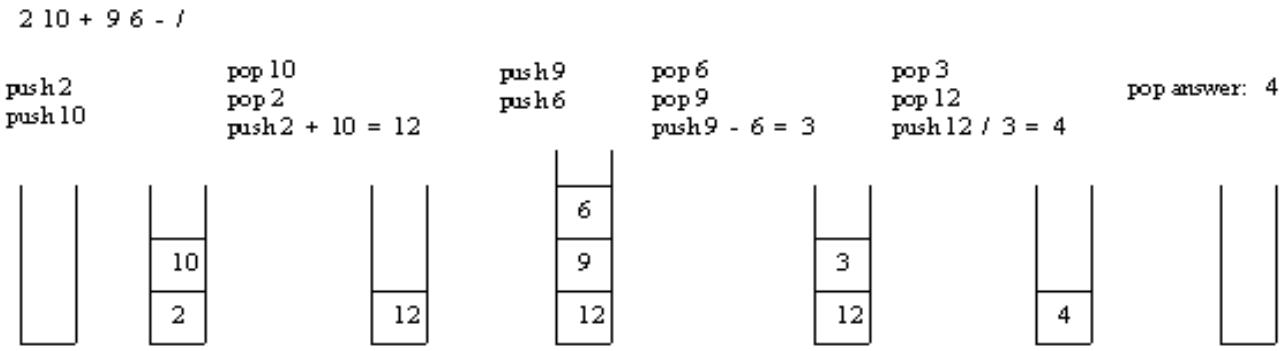
the reading symbol (if it is * do multiplication, etc) using the two popped operands and push the result back on to the stack.

4. Repeat steps 2 & 3 till the postfix expression completes.
5. Finally! perform a pop operation and display the popped value as final result.



Queues

- Queue is also a linear data structure in which the insertion and deletion operations are performed at two different ends as it is opposed to stacks.
- The insertion is performed at one end (Called as **enqueue**) and deletion is performed at the other end (called as **dequeue**).
- To perform insertion and deletion we used two pointers **rear** and **front**.
- Insertion operation is performed at a position pointed by '**rear**' and the deletion operation is performed at a position pointed by '**front**'.

In queue data structure, the insertion and deletion operations are performed based on **FIFO (First In First Out)** principle.

Applications of Queue

Due to the fact that queue performs actions **on first in first out** basis which is quite fair for the ordering of actions. There are various applications of queues discussed as below.

1. Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
2. Queues are used in asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.
3. Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.
4. Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.
5. Queues are used in operating systems for handling interrupts.

Operations on Queue:

Mainly the following four basic operations are performed on queue:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

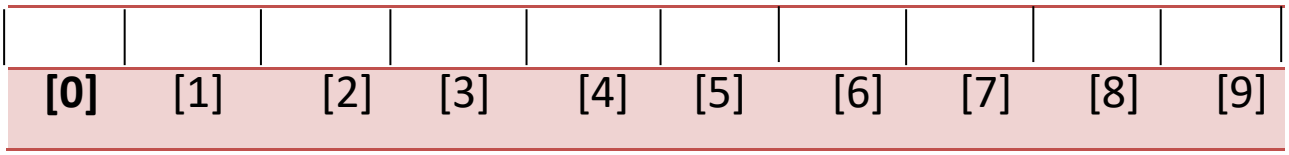
Dequeue: Removes an item from the queue. The items are popped

in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.
Front: Get the front item from queue.
Rear: Get the last item from queue.

Example of a queue:

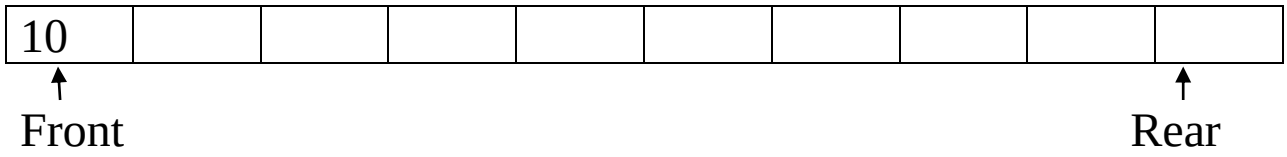
Initially, rear = -1 and front = -1

Queue is empty:



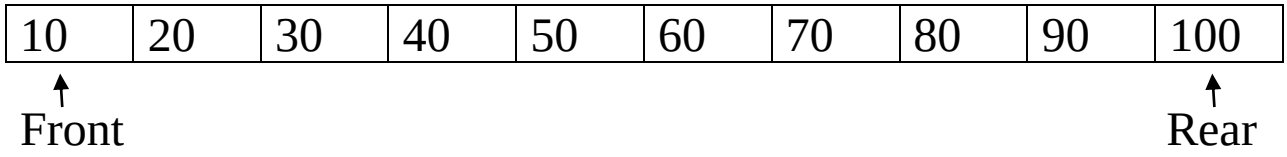
Inserting after first element:

(Rear = Front = 0)

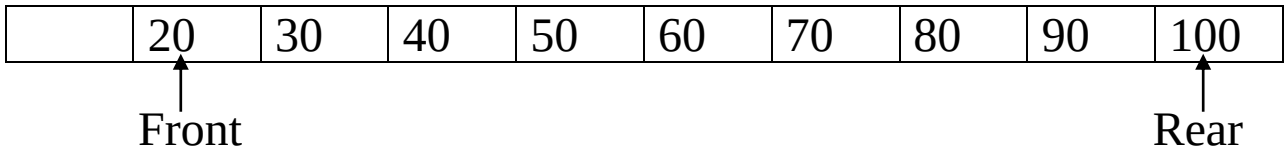


Inserting after last element (queue is full):

Rear=Max-1 (Queue is Full)

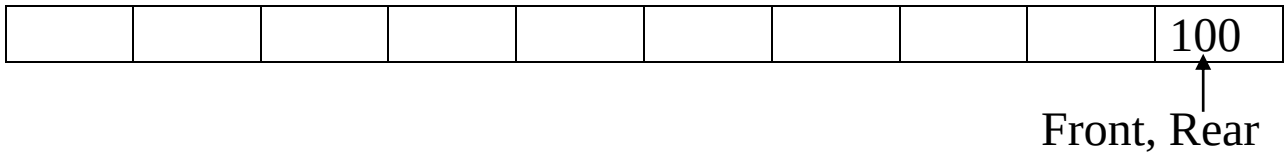


Delete: {10} is deleted



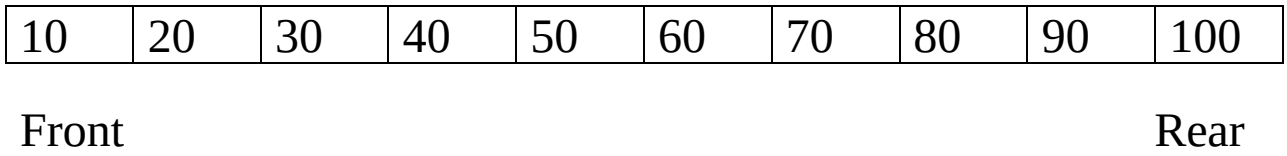
Deleted elements are

{10,20, 30, 40, 50, 60, 70, 80, 90}

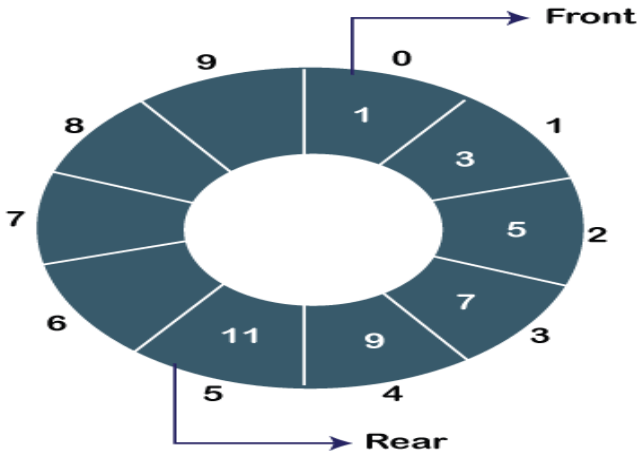


Types of Queue:

1. Simple or Normal Queue: Where the insertion will happen from position zero to position Max. We cannot use the deleted positions to insert elements again.



2. Circular Queue: Last location is connected to the first location, when an element is inserted in the last position and if the first location is free then we can insert another element in the first location.

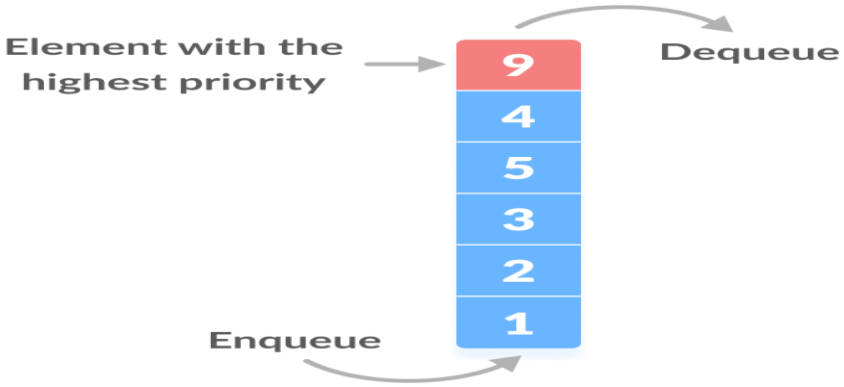


3. Deque (Doubly Ended Queue): It is a Queue where the insertion and deletion operations are performed at both ends.

10	20	30	40	50	60	70	80	90	100

Front 1 Rear 1 Rear2 Front 2

4. Priority Queue: In priority queues the items are associated with a value called priority. When you perform delete, first delete the highest priority one before deleting any lower priority item.



Implementation of Queue

Queue data structure can be implemented in two ways. They are as follows...

- **Using Array**
- **Using Linked List**

Implementation of simple queues using arrays:

Enqueue operation:

- **enQueue(Queue, front, rear, Element Type) - Inserting value into the queue**
- In a queue, the new element is always inserted at **rear** position.
- We can use the following steps to insert an element into the queue...
- **Step 1 -** Check whether **queue** is **FULL**.

  (**rear == SIZE-1**)

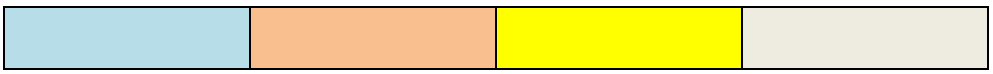
- **Step 2 -** If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.
- Step 3 - If rear = -1 then set rear =0, front = 0; otherwise increment **rear** value by one.
- **Step 4 -** set **queue[rear]** = **value.**

```
Step - 1: IF rear = Max-1
            Display “Queue Full”
            Go to step-4
Step – 2: IF front = -1  ; SET front = 0
Step – 3: rear++
            queue[rear] = value
Step - 4: End
```

Example:

The list of elements to be inserted into Queue: 10, 30, 40, 50

Initially Rear = -1 Front = -1

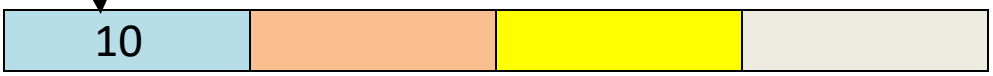


[0] [1] [2] [3]

Queue is empty, for insertion; both rear and front are set to position 0 for first element.

After inserting 10;

Front, Rear



[0] [1] [2] [3]

After inserting 30;

Rear =rear+1;

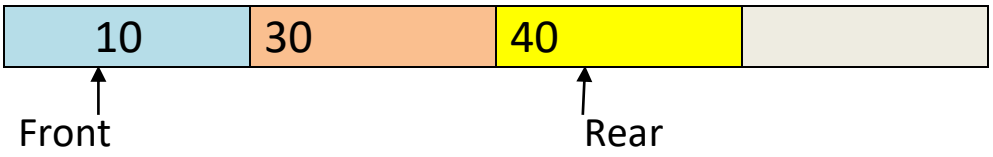
Queue[rear]=30

Front Rear

After inserting 40;

Rear =rear+1;

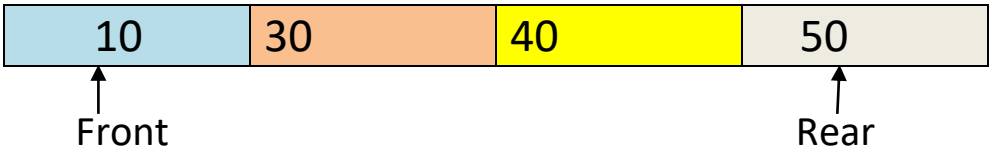
Queue[rear]=40



After inserting 50;

Rear =rear+1;

Queue[rear]=50



deQueue operation:

deQueue (Queue, int, int) – Deletes the element from the queue

In a queue, the element is always deleted from **front** position.

We can use the following steps to delete an element from the queue...

Step 1 - Check whether **queue** is **EMPTY**. (**front == rear == - 1**)

Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY! Deletion is not possible!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then display **queue[front]** as deleted element.

Now increment the **front** value by one (**front++**).

Step 4 - Check if **front** > **rear**, if it is **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

Step - 1: IF front = -1

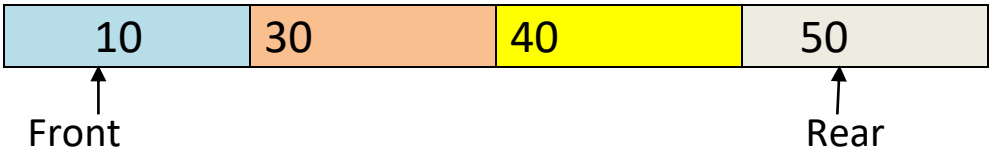
Display “Queue Empty”

Go to step-4

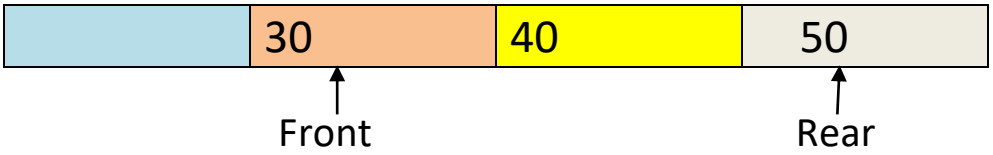
Step – 2: ele = queue[front]

Step – 3: front++ , queue[rear] = value

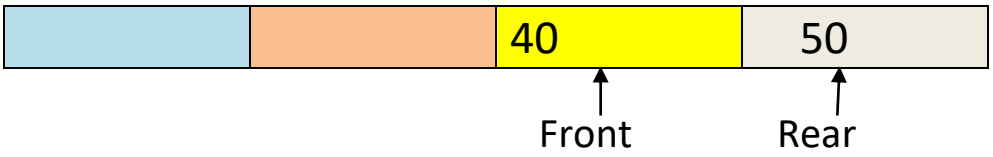
Step - 4: if front > rear then set front and rear to -1.



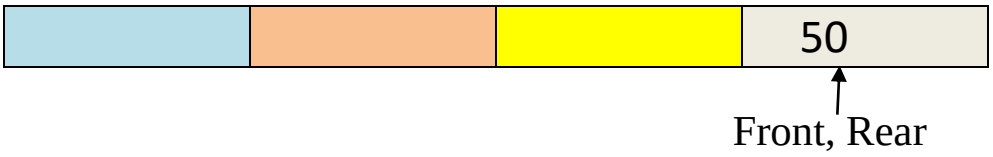
{10} is deleted



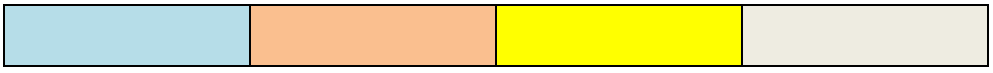
{30} is deleted



{40} is deleted



{50} is deleted (Empty queue) [First in first out principle]



Displaying queue elements:

display(Queue, int, int) - Displays the elements of a Queue

We can use the following steps to display the elements of a queue...

Step 1 - Check whether **queue** is **EMPTY**. (**front == rear==-1**)

Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front**.

Step 4 - Display '**queue[i]**' value and increment '**i**' value by one (**i++**).

Step 5 - Repeat the step 4 until '**i**' value reaches to **rear** (**i** <= **rear**).

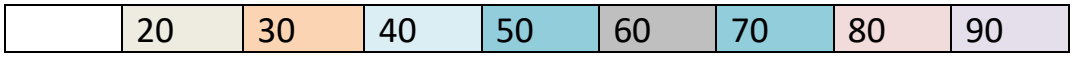
Source code:

```
void display()
{   int i;
    if(front == -1)
      printf("\n QUEUE IS EMPTY");
     else

      {    for(i = front;   i <= rear;   i++)
      printf("\t %d", queue[i]);
       }
}
```

Circular Queue

- For example if we want to insert an element value is 100.
- Since rear==Max(9), it is not possible to insert element, though empty slots are available.



Front Rear

- To address this issue, circular queue is used, where the elements are connected in a circular manner as shown in fig.
- **A circular queue is a linear data structure follows FIFO (First In First Out) principle and the last location is connected back to the first location to make a circle.**

Conditions : queue is full and empty:

- *Queue is empty*

  Front = ***rear*** = -1

- Queue is full

  front = = 0 and ***rear*** == MAX – 1

  or

  front==rear+1

Operations

- enQueue
  - Insert element into queue at rear position
- deQueue
  - To delete an element from front position of queue
- Display
  - To display the list of elements from front to rear ends.

enQueue

- The enQueue() function inserts that value into the circular queue.
- We can use the following steps to insert an element into the circular queue...

Step 1 - Check whether **queue** is **FULL**.

(**(rear == SIZE-1 && front == 0) || (front == rear+1)**)

Step 2 - If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT FULL**, then check if **rear == SIZE - 1** then set **rear = 0** otherwise **increment rear**. // rear = (rear + 1) %size

Step 4 - set **queue[rear]** = **value**

Step 5 – Check if '**front == -1**' if it is **TRUE**, then set **front = 0**.

Step -1 : IF ((front = 0 and rear = MAX-1)
or front = rear-1)

Display "Queue Full"

Go to step-4

Step – 2: if rear = size-1 then set rear = 0,
otherwise rear = rear +1

Step – 3: queue[rear] = value. If front = -1
then set front =0.

Step - 4: End

deQueue

- The deQueue() function deletes the first inserted element from the circular queue.
- We can use the following steps to delete an element from the circular queue...

Step 1 - Check whether **queue** is **Empty**.

(rear == -1 && front -1)

Step 2 - If it is Empty, then display **"Queue is Empty!!! Deletion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then display **queue[front]** as deleted element .

Step 4 - Check whether both **front** and **rear** are equal (**front** == **rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**'.
Check whether **front == SIZE-1**, if it is **TRUE**, then set **front = 0.**

step - 1 : IF ((front = -1 and rear = -1)
Display " Queue Full"

Go to step-4

step – 2: Assign queue[front] to ele.

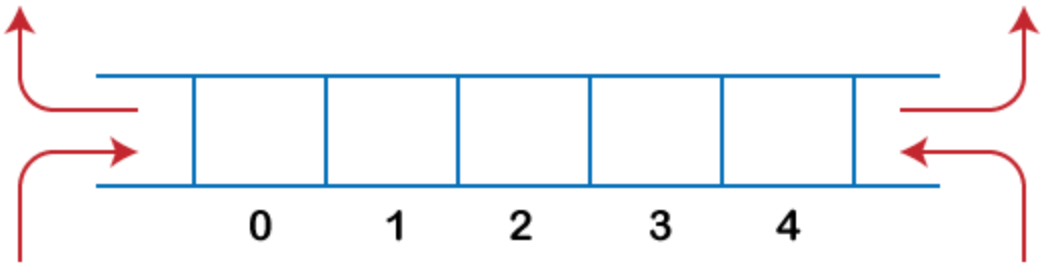
Step – 3: if front = rear then set front = rear = -1.

if front = Size-1 then set front = 0.

Step - 4: End

Deque

The dequeue stands for **Double Ended Queue**. In the queue, the insertion takes place from one end while the deletion takes place from another end. The end at which the insertion occurs is known as the **rear end** whereas the end at which the deletion occurs is known as **front end**.



Deque is a linear data structure in which the insertion and deletion operations are performed from both ends. We can say that deque is a generalized version of the queue.

Operations on Deque

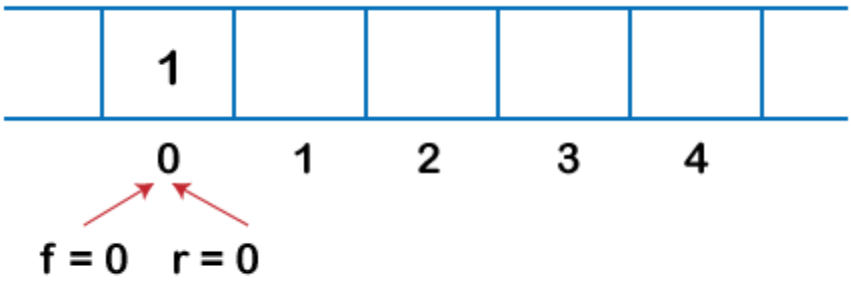
The following are the operations applied on deque:

- **Insert at front**
- **Delete from end**
- **insert at rear**
- **delete from rear**

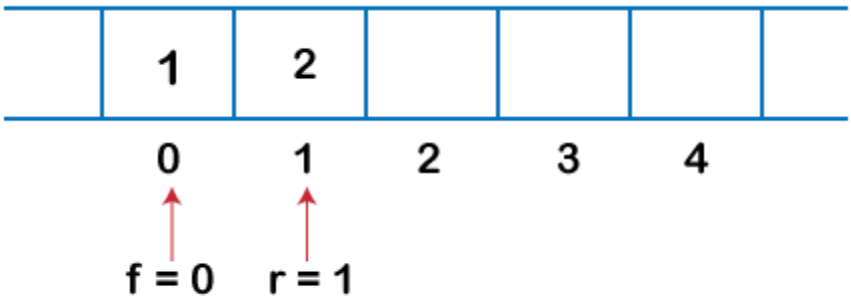
Other than insertion and deletion, we can also perform **peek** operation in deque. Through **peek** operation, we can get the **front** and the **rear** element of the dequeue.

Enqueue operation

1. Initially, we are considering that the deque is empty, so both front and rear are set to -1, i.e., **f = -1** and **r = -1**.
2. As the deque is empty, so inserting an element either from the front or rear end would be the same thing. Suppose we have inserted element 1, then **front is equal to 0,** and the **rear is also equal to 0.**

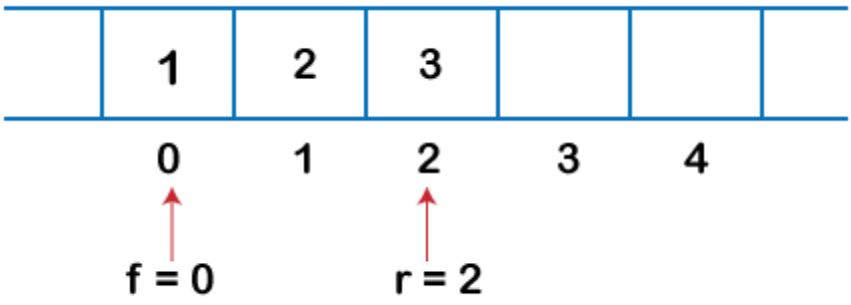


3. Suppose we want to insert the next element from the rear. To insert the element from the rear end, we first need to increment the rear, i.e., **rear=rear+1**. Now, the rear is pointing to the second element, and the front is pointing to the first element.

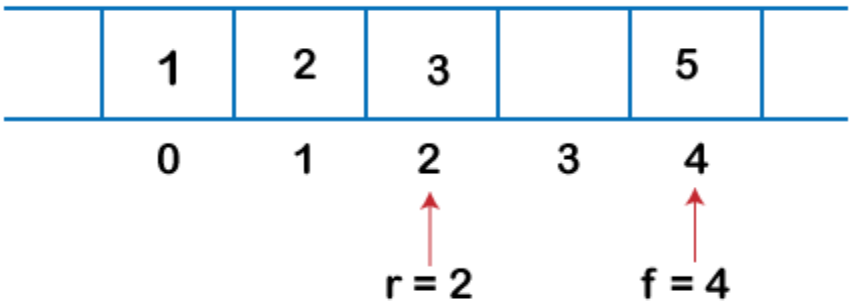


4. Suppose we are again inserting the element from the rear end. To insert the element, we will first increment the rear, and now rear

points to the third element.



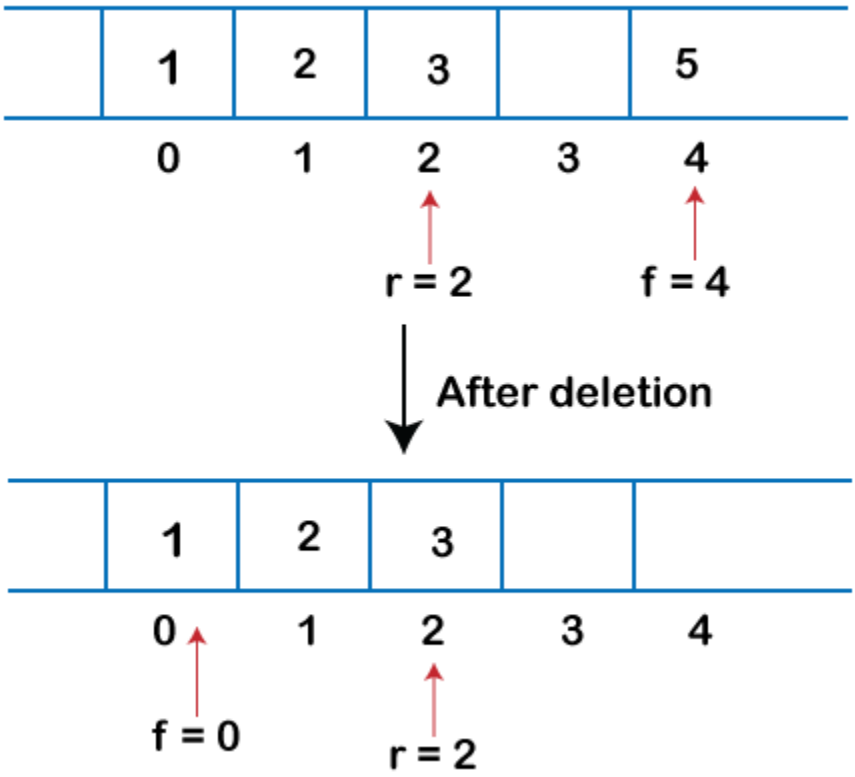
5. If we want to insert the element from the front end, and insert an element from the front, we have to decrement the value of front by 1. If we decrement the front by 1, then the front points to -1 location, which is not any valid location in an array. So, we set the front as **(n -1),** which is equal to 4 as n is 5. Once the front is set, we will insert the value as shown in the below figure:



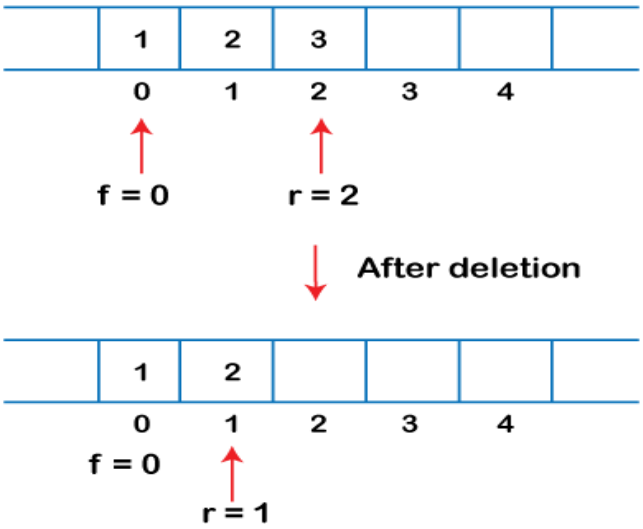
Dequeue Operation

1. If the front is pointing to the last element of the array, and we want to perform the delete operation from the front. To delete any element from the front, we need to set **front=front+1**. Currently, the value of the front is equal to 4, and if we increment the value of front, it becomes 5 which is not a valid index. Therefore, we conclude that if front points to the last element,

then front is set to 0 in case of delete operation.

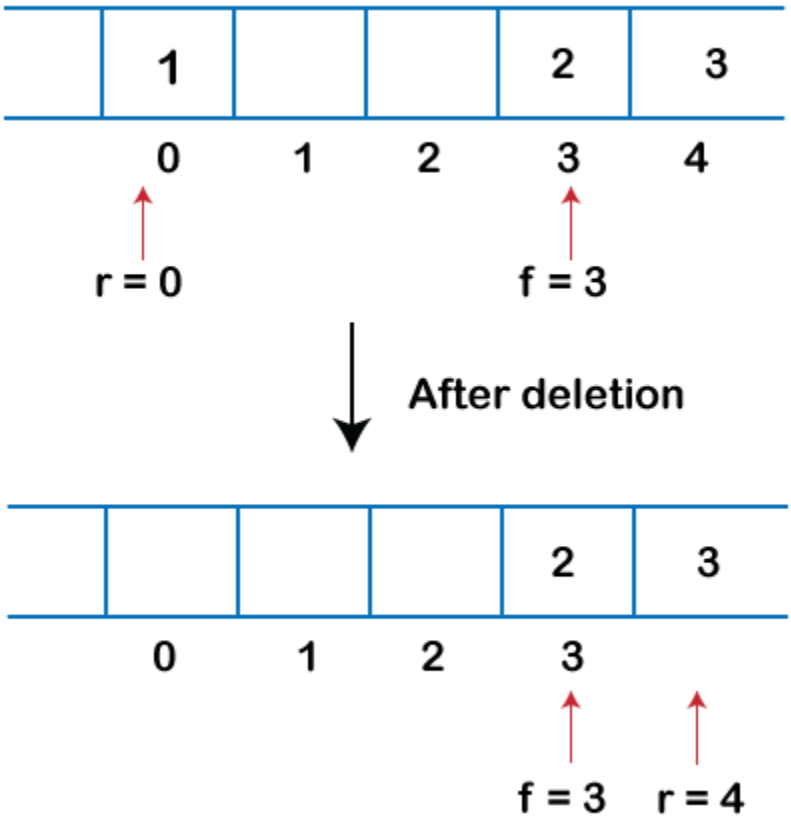


2. If we want to delete the element from rear end then we need to decrement the rear value by 1, i.e., **rear=rear-1** as shown in the below figure:



3. If the rear is pointing to the first element, and we want to delete the element from the rear end then we have to set:

Rear =n-1 where **n** is the size of the array as shown in the below figure:



Queue/Deque/Circular Queue:

- **Insert** : **O(1) (enQueue)**
- **Remove**: **O(1) (deQueue)**
- **Size**: **O(1)**
- **Display :O(n)**

We can implement a stack by using two queues where one operation take constant time and the other takes linear time.

Algorithm for linked list representation of a simple queue:

For insertion:

- **Step 1:** Allocate the space for the new node PTR
- **Step 2:** SET PTR -> DATA = VAL
- **Step 3:** IF FRONT = NULL
  SET FRONT = REAR = PTR
  SET FRONT -> NEXT = REAR -> NEXT = NULL
  ELSE
  SET REAR -> NEXT = PTR
  SET REAR = PTR
  SET REAR -> NEXT = NULL
  [END OF IF]
- **Step 4:** END

For deletion:

- **Step 1:** IF FRONT = NULL
  Write " Underflow "
  Go to Step 5
  [END OF IF]
- **Step 2:** SET PTR = FRONT
- **Step 3:** SET FRONT = FRONT -> NEXT
- **Step 4:** FREE PTR
- **Step 5:** END