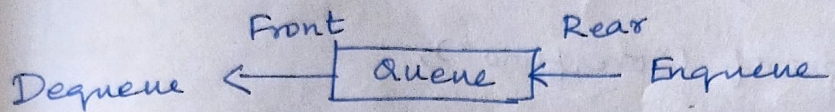
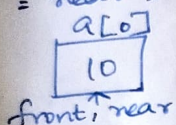
Queue.

Queue.

Queue is a linear data structure - FIFO (First In First Out) principle, insertion is performed at rear end and deletion is performed at front end.



front = rear = -1 → No elements in Queue.

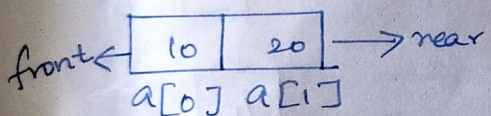


Inserting 1st element in Queue.

front = rear = 0 | Front ++, rear ++

Inserting 2nd element in queue.

rear = rear + 1.



Implementation of Queue.

* Array implementation of Queue.
* Linked list implementation of Queue.

Operations of Queue.

① Enqueue - Inserting an element in queue.
- only in rear end insertion can be done. rear = rear + 1.

② Dequeue - deleting an element in queue
- front end → insertion is possible
front = front + 1

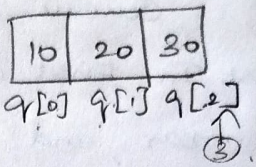
Overflow - ~~deleting~~ inserting an element in Queue which is full is called overflow

Underflow - deleting an element from empty Queue is called Underflow.

Array implementation of Queue.

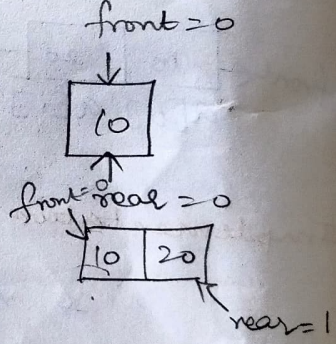
front = rear = queue = -1 → empty Queue.

2 > 3



Insertion

```
int queue[10];
if (rear >= maxsize)
        printf ("overflow");
else
    if (front == rear == -1)        // empty Queue
    {
        front = rear = 0;
    }
    else                            // more than 1 element in queue
        rear = rear+1
    queue [rear] = 10;
```

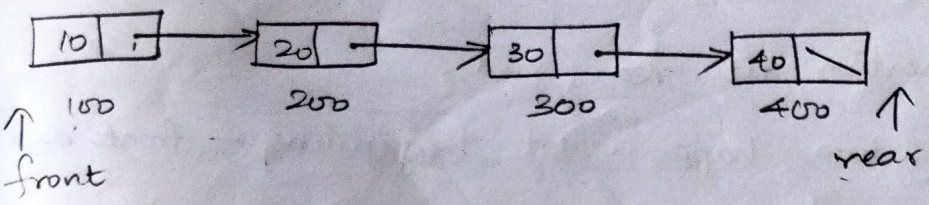


Deletion

```
if (front == -1)
        Printf (" Underflow ");
else  val = queue[front];
      if(front == rear)
            front = rear = -1;
      else
            front = front +1;
Printf ("The element popped is %d", val);
```

Linked list Implementation of Queue.

* Array Implementation cannot be used in large scale applications.



* front → address of the starting element of Queue
  rear → address of the last element of Queue.

* front & rear pointing = Null ← Queue is empty.

Routine:

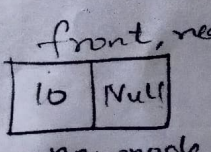
```
Struct node
{
    int data;
    Struct node * next;
};
Struct node *front = Null;
Struct node *rear = Null;
```

Insertion / Enqueue

```
Struct node *newnode;
newnode = malloc (sizeof (Struct node));
```

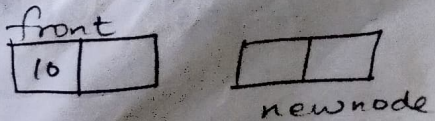
Queue is empty

```
newnode → data = 10;
newnode → next = NULL;
if (front == Null)
{
    front = rear = newnode;
}
```

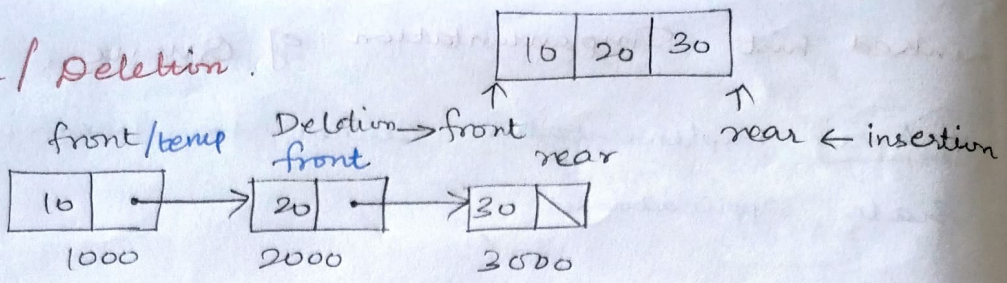


Queue contains more than one element.

```
rear → next = newnode;
rear = newnode;
rear → next = null;
```



Dequeue / Deletion.



Insertion at rear end.

Deletion happens at beginning — front end

↳ routine - delete @ beginning

Queue has more than 1 element

```
temp = front;
front = front -> next
free (temp);
```

Queue is empty

```
if (front == Null)
    printf("Underflow");
```

* Deletion - removes first element inserted in the queue.

Case 1:

* if list is empty, check the condition front == Null and printf Underflow, which means when Queue is empty, we try to delete an element it is called underflow.

Case 2:

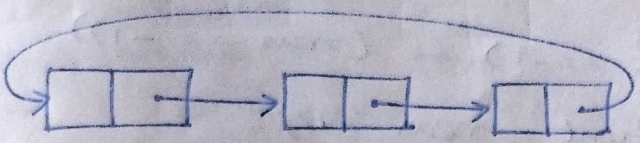
* If Queue has more than 1 element, change the front pointer and free the 1st pointed element using temporary (temp) variable

Types of Queue.

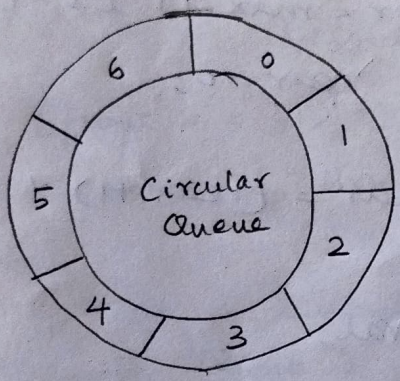
1. Simple / Linear Queue
2. Circular Queue
3. Priority Queue
4. Double ended Queue (Deque)

Circular Queue.

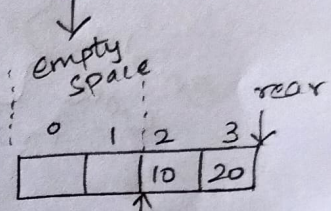
* Nodes are represented in circular manner.
* last element is connected to the first element
* It is also known as Ring Buffer



* Advantage: memory utilization.



Cannot be used in simple queue.



[Simple Queue]

Enqueue operation.

* 2 Scenario for inserting an element when Queue is not full.

rear != max - 1

↓

rear = (rear+1) % max

front != 0 && rear = max - 1
[empty spaces are there]

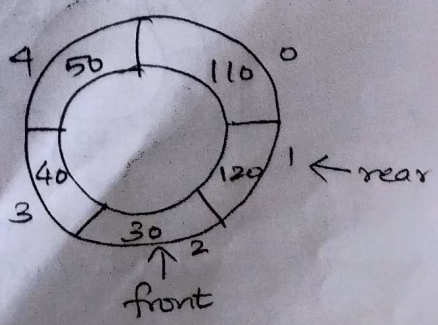
rear = 0 ← New element is inserted.

* 2 Scenario – element cannot be inserted.

front == 0 && rear = max - 1

Queue is full

front = rear + 1



Insertion - Routine.

```
if ((rear+1) % max) = front
        printf ("overflow");
```
} → Queue is full.

```
else
    if ( (front == -1) && (rear == -1))
        front = rear = 0;
```
← Empty Queue

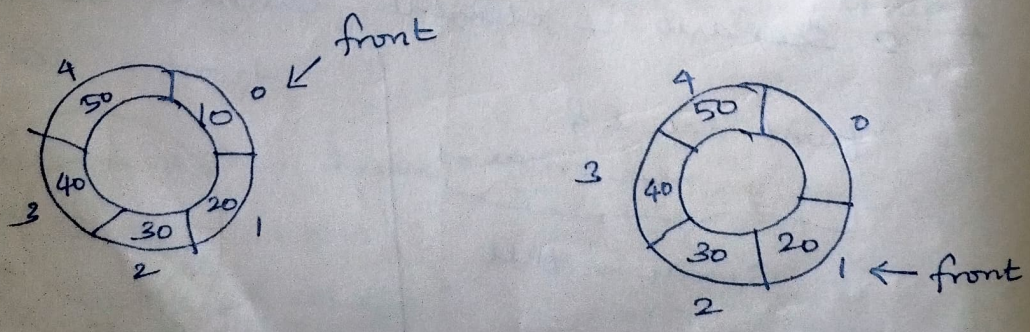
```
else
    if ((rear = max-1) && (front != 0))
        rear = 0;
    else
        rear = (rear+1) % max;
```
← empty space

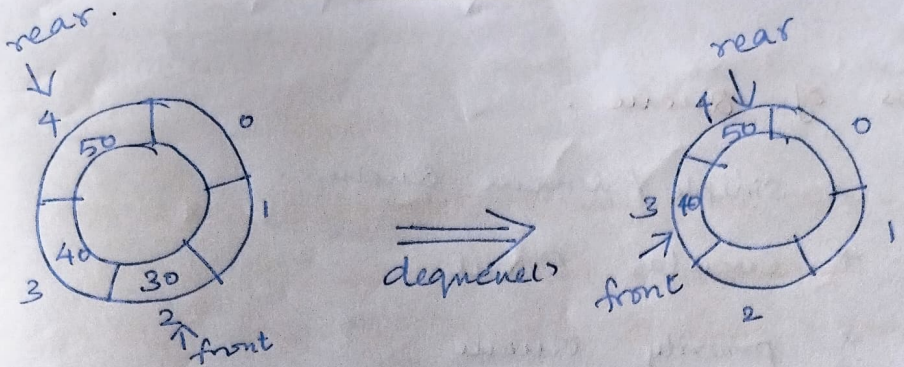
```
Queue [rear] = val;
```

Dequeue operation.

* Queue is empty → front = rear = -1
* deleting an element from Queue → front = front + 1
* 1 element in Queue ⇒ front = rear

Example:





Empty Spaces are there

Routine for deletion.

```
dequeue ()
{
    if (front == -1)
        printf (" underflow ");
    else
        val = Queue [front];
        if (front == rear)
            front = rear = -1;
        else
            if (front = max - 1)
                front = 0;
            else
                front = front + 1
}
printf (" %d is the deleted element ", val);
```

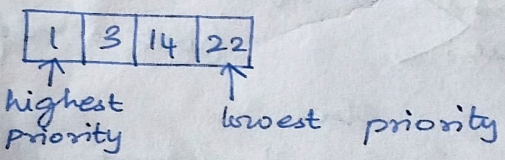
only 1 element in Queue

Types of Queue.

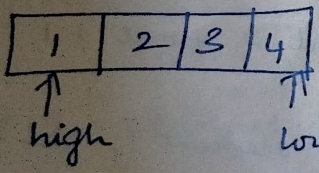
* Simple / linear Queue
* Circular Queue
* priority Queue
* Double ended queue.

priority Queue.

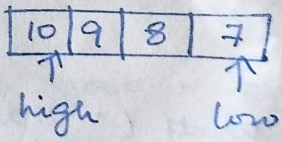
* Values are arranged based on their priority values.
* Elements with highest priority are removed before the elements with lowest priority.
* Each element has a priority.



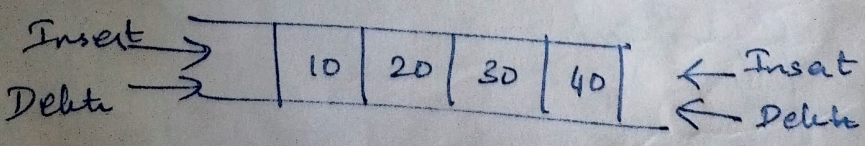
Ascending order priority Queue



Descending order priority Queue



Double Ended Queue (Deque)



Insertion & deletion can be performed at both ends. It doesn't follow FIFO principle.