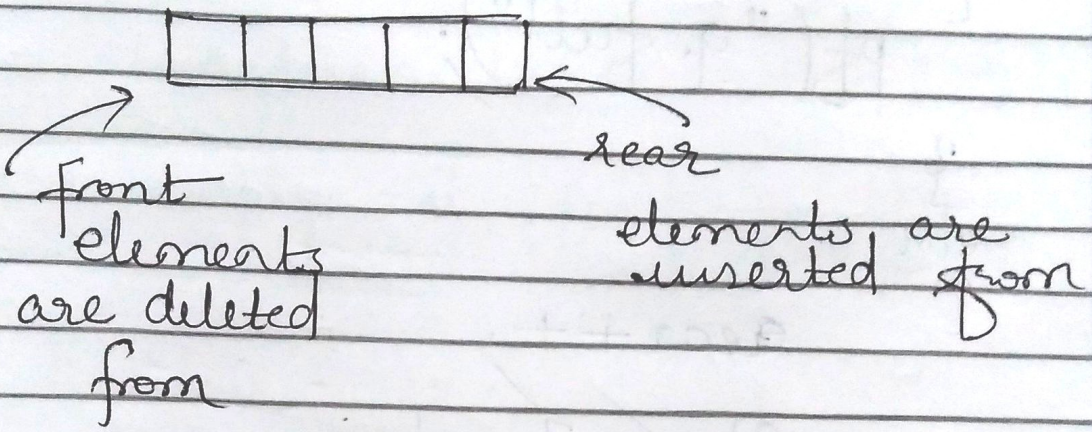
Queue

Works on First In First Out (FIFO) principle.

There are two ends



Insertion of element → enqueue
Deletion of element → dequeue

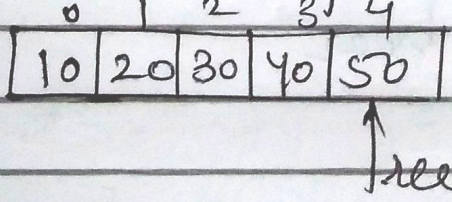
Initially front = rear = −1

enqueue(10);

element

rear = rear + 1
a[rear] = x

While enqueue you have to ensure that queue is not full.

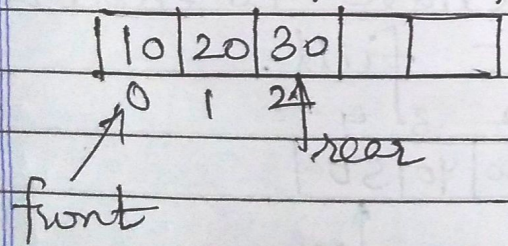


enquee(60) ⟹

```
if(rear == SIZE-1)
{
    q.full
}
```

```
void enqueue(ele)
{
    if(rear == SIZE-1)
    {
        ptf("q. full");
    }
    else
    {
        rear++;
        a[rear] = ele;
        if(front == -1)  // if this is first insertion
            front = 0;
    }
}
```

Deletion from front end. predondition



queue should not be empty

```
ele = a[front];
front = front + 1;
return ele
```

dequeue() ⟹ 10
dequeue() ⟹ 20
dequeue() ⟹ 30

```
ele = a[front];
if(front == rear)
    front = rear = -1;
else
    front = front + 1;
return ele
```

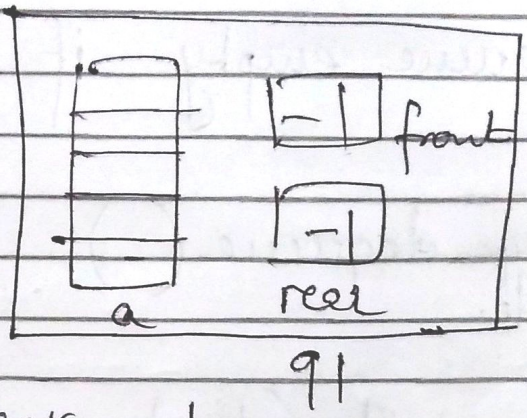
queue empty .if (front == -1)
return true.

```
datatype dequeue()
of ele.
{
    if (front == -1)
    {
        q.empty
        return -1;
    }
    else
    {
        ele = a[front];
        if(front != rear)
            front = rear = -1;
        else
            front = front + 1;
        return ele;
    }
}
```

// queue using array.

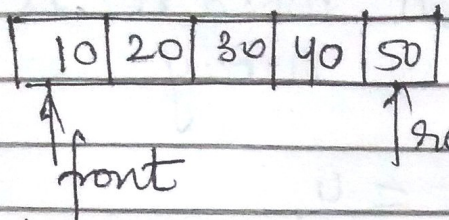
```
typedef struct
{
    int a[SIZE];
    int rear, front;
}Queue;

int main()
{
  Queue q1;
  q1.front = q1.rear = -1;
  enqueue(&q1, ele);
      :
      :
ele = dequeue(&q1);
  //you can process ele
}
void enqueue( Queue *p, int x)
{
  if(p->rear == SIZE-1)
  {
  }
  else
  {   p->rear = p->rear+1;
      p->a[p->rear] = x;
      if( p->front == -1)
          p->front = 0;
}
```

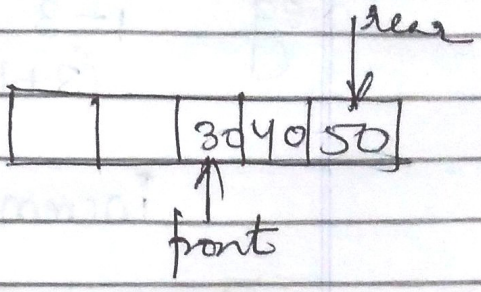


Disadvantage of linear queue

→ Consider the case



Now delete 10 & 20, so

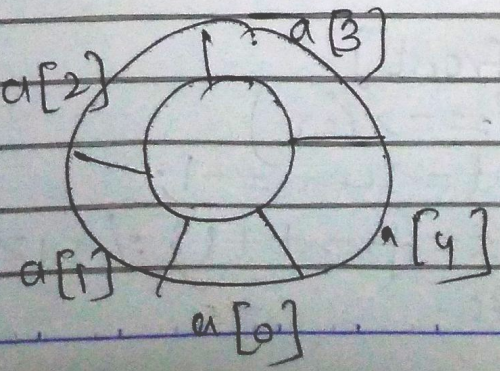


If u want to insert more elements it will check whether rear == SIZE-1, which is true in this case, it will say 'Q is full' even though we have empty space (empty positions in the beginning)

Inorder to remove this disadvantage, we can modify dequeue fenction, in such a way that after deletion, rest of the elements of the queue are shifted by one position to left. But again, this is inefficient since, for one deletion, there are several shifting operations.

Second option: Circular queue

↓ In this queue, the first position & last position of array are assumed to be logically connected to each other. i.e first element a[0] follows last element



The queue would be reported full when all the slots in array are occupied.

In circular queue, if you want to increment 'i' in circular fashion, the expression is

$i = (i+1) \% n$ where n is size of array

eg $i = 3$

$(3+1) \% 5 = 4$.

increment $i \rightarrow (4+1) \% 5 = 0$ //

```
void enqueue(ele)
{
   if((rear+1) % SIZE == front)
   {
      q.full
   }
   else
   {  rear = (rear+1) % SIZE;
      a[rear] = ele;
      if(front == -1)
         front = 0;
   }
}
datatype dequeue()
 of ele
{     if(front == -1)
      {  q.empty
      }
      else
      {  ele = a[front];
         if(front == rear)
            front = rear = -1;
         else
            front = (front+1) % SIZE;
         return ele;
      }
}
```