QUEUE

↳ FIFO

(First In First Out)

Queue ADT

Data : (1) Space for storing elements
(2) Front : for deletion
(3) Rear : for insertion

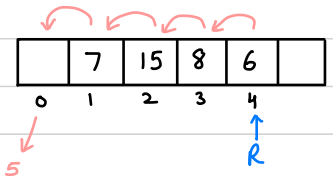
Operations : (1) enqueue (x)
(2) dequeue ()
(3) isEmpty()
(4) isfull()
(5) first()
(6) last()

(1) Array
(2) Linked List

QUEUE USING ARRAY

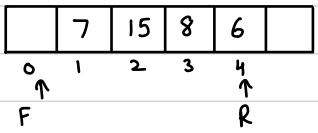
1. Queue using single pointer
2. Queue using front and rear.
3. Drawbacks of queue using array.

1. Queue using single pointer



Insert – O(1)
Delete – O(n)

2. Queue using front and rear



initially front = rear = -1

Insertion : increment rear and insert O(1)
Deletion : increment front and delete O(1)

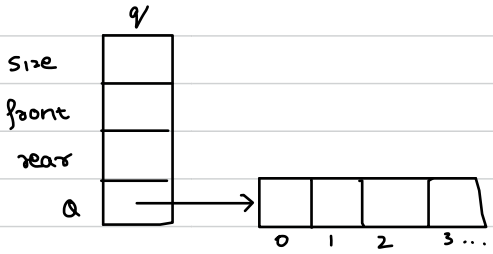
Empty : if (front == rear)
Full : if (rear = Size-1)

PROGRAM FOR QUEUE USING ARRAY

```
Struct Queue
{
    int size;
    int front;
    int rear;
    int * Q;
}

int main ()
{
    Struct Queue q;
    printf(" Enter Size : ");
    scanf (" %d ", &q.size);
    q.Q = new int[q size];
    q.front = q.rear = -1;
}

void enqueue (Queue *q , int x)
{
    if ( q->rear == q->size-1)
        printf (" Queue is full ");
    else
    {
        q->rear++;
        q->Q[q->rear] = x;
    }
}
```

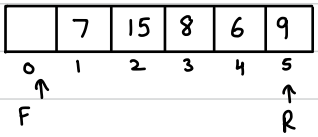



```
Void dequeue ( Queue *q)
{
    int x = -1;

    if (q->front == q->rear)
        printf (" Queue is empty ");
    else
    {
        q->front++;
        x = q->Q[q->front];
    }
    return x;
}
```

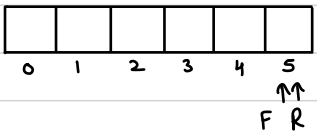
DRAWBACK OF QUEUE USING ARRAY

1. We cannot utilize space of deleted element.



2. Every location can be used only once

3. A situation where queue is empty and full



USING SPACE AGAIN SOLUTION

(1) Resetting Pointers

Whenever Front and Rear are pointing at same place, initialize them as -1

(2) Circular Queue

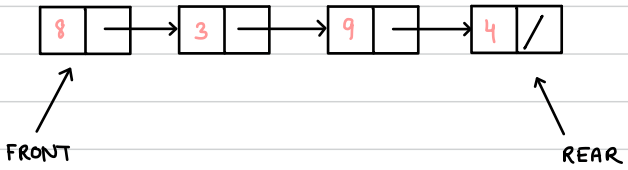
In circular queue, front and rear initialize with array's first position.

```
void enqueue( struct Queue *q, int x)
{
    if((q->Rear+1)%q->size == q->front)        To check if rear's next is front
            printf("Queue is Full");
    else
    {
            q->Rear = (q->rear+1)%q->size;
            q->Q[q->rear] = x;
    }
}
```

rear =	(rear+1) % size
0	(0+1) % 7 = 1
1	(1+1) % 7 = 2
2	(2+1) % 7 = 3
3	(3+1) % 7 = 4
4	(4+1) % 7 = 5
5	(5+1) % 7 = 6
6	(6+1) % 7 = 0

```
void dequeue ( struct Queue *q)
{
    int x=-1
    if( q->front == q->rear)
            printf("Queue is Empty");
    else
    {
            q->front = (q->front+1) % q->size;
            x = q->Q[q->front];
    }
    return x;
}
```

QUEUE USING LINKED LIST



Empty : if (front == NULL)

Full : Node *t = new Node;
if (t == NULL)
// No more nodes can be created
i.e heap is full

```
void enqueue (int x)    O(1)
{
      Node *t = new Node;
      if (t == NULL)
            printf(" Queue is Full");
      else
      {
            t -> data = x;
            t -> next = NULL;
            if (front == NULL)
                  front = rear = t;
            else
            {
                  rear -> next = t
                  rear = t;
            }
      }
}

int   dequeue()    O(1)
{
      int x = -1;
      Node *p;
      if (front == NULL)
            printf(" Queue is Empty");
      else
      {
            p = front;
            front = front -> next;
            x = p -> data;
            free (p);
      }
      return x;
}
```

DEQueue

↳ Double Ended Queue

QUEUE

	Insert	Delete
front	×	✓
rear	✓	×

DEQUEUE

	Insert	Delete
front	✓	✓
rear	✓	✓

INPUT RESTRICTED DEQUEUE

	Insert	Delete
front	×	✓
rear	✓	✓

OUTPUT RESTRICTED DEQUEUE

	Insert	Delete
front	✓	✓
rear	✓	×

PRIORITY QUEUES

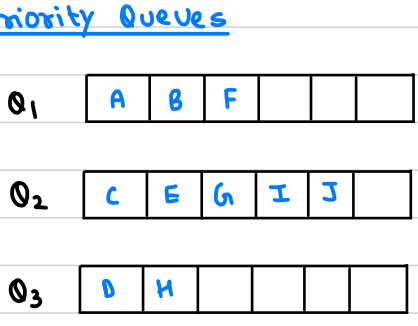
1. Limited Set of priorities
2. Element Priority

1. Limited Set of priorities

Priorities = 3

Element	A	B	C	D	E	F	G	H	I	J
Priority	1	1	2	3	2	1	2	3	2	2

Priority Queues



INSERTION

DELETION

For deletion, elements of Q_1 will be deleted first, then Q_2 and then Q_3

Elements will be deleted in FIFO

2. Element Priority

Elements → 6, 8, 3, 10, 15, 2, 9, 17, 5, 8

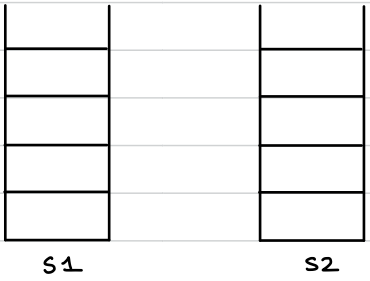
- Element is itself a priority
- Smaller number higher priority

1. Insert in same order O(1)
   Delete max priority by searching it O(n)

2. Insert in increasing order of priority O(n)
   Delete the last element of array O(1)

QUEUE using 2 STACKS

assume stacks to be implemented using linked list




```
enqueue ( int x)
{
      push ( &S1, x);
}

int  dequeue()
{
      int  x = -1;
      if ( isEmpty (S2))
      {
            if ( isEmpty S1)
            {
                  printf (" Queue Empty ");
                  return  x;
            }
            else
            {
                  while ( ! isEmpty (S1))
                        push (&S2, pop (&S1));
            }
      }
     return  pop ( &S2);
}
```