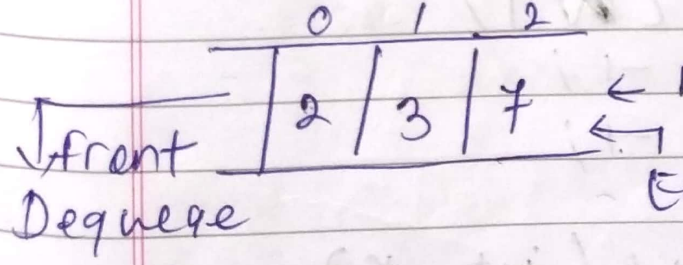
Queues:

- Fundamentals of queue.
- Representation & Implementation of queue using Array.
- Circular Queue: representation & Implementation
- Application of Queue:
  - Josephus problem,
  - Job scheduling
  - Queue simulation
  - Categorizing data.
- Double ended queue.
- priority queue.
- Multiple stack & multiple Queues

⇒ Queue is a linear DS.
ADT (Abstract Data Type)
(FIFO / LILO)

Rule: Insertion (enqueue) → rear /tail
Deletion (dequeue) → front/head

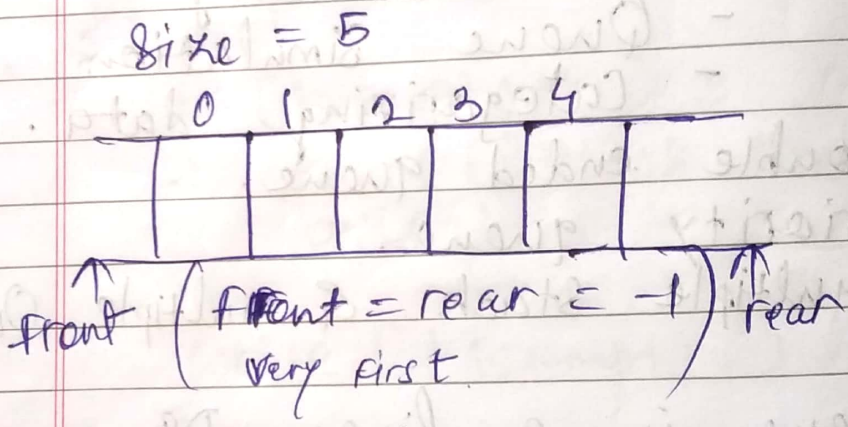


- Insertion can be performed from one end = rear
- Deletion can be performed from another end = front

Time complexity

⇒ operation on queue: O(1)

- Enqueue (2)
- dequeue ( )
- front( ) / peek( )
- isfull ( )
- isEmpty ( ) / underflow overflow



⇒ Implemantation of Queue using Ar

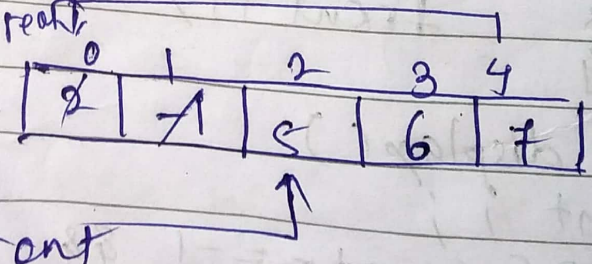
```
#define N 5;
int queue[N];
int front = -1;
int rear = -1;

void enqueue (int x)
{
    if (rear == N-1)
    { printf("overflow");
    }
```

```
else if ( front == -1 && rear == -1)
{
    front = rear = 0;
    queue[rear] = x;
}
else
{   rear++;
    queue[rear] = x;
}

void dequeue ()
{
   if ( front == -1 && rear == -1)
   {
       printf(" underflow");
   }
   else if( front == rear )
   {
      front = rear = -1 ;
   }
   else
   {   front++;
   }
void display():
{ int i;
   if( front == -1 && rear == -1)
   {
      printf(" Queue is Empty");
   }
```

```
        else
        {
            for(i=front ; i<rear+1 ;i++
            {
                printf("%d", queue[i]);
            }
    }
    void peek()
    {
        if(front==-1 && rear==-1)
        {
            printf("Queue is Empty");
        }
        else
        {
            printf("%d", queue[front]);
        }
    }
```

⇒ Circular Queue Implementation using Array.

Example:

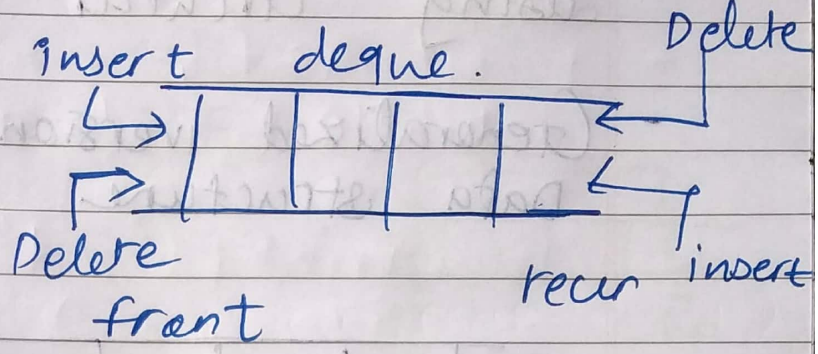


((rear+1) % N) == front ⇒ full

```
#define N 50
int queue [N];
int front = -1;
int rear = -1;

void enqueue (int x)
{
    if( front == -1 && rear == -1)
    {
        front = rear = 0;
        queue [rear] = x;
    }
    else if ((rear + 1) % N == front)
    {
        printf ("overflow / full");
    }
    else
    {
        rear = (rear + 1) % N;
        queue [rear] = x;
    }
}
void dequeue ()
{
    if ( front == -1 && rear == -1)
    {
        printf (" Queue is Empty");
    }
```

```
else if(front == rear)
{
    front = rear = -1;
}
else
{
    front = (front+1) % N;
}
}
void display()
{
  if(front == -1 && rear == -1)
  {
    printf("Queue is empty");
  }
  else
  {  printf("Queue is :")
     while (i != rear)
     {
        printf("%d", queue[i]);
        i = (i+1) % N;
     }
```

* Double ended Queue:



- It follows properties of both
  Stack - LIFO
  Queue - FIFO

Types:

1) Input restricted → input from only one end

2) Output restricted → Delete from only one end.

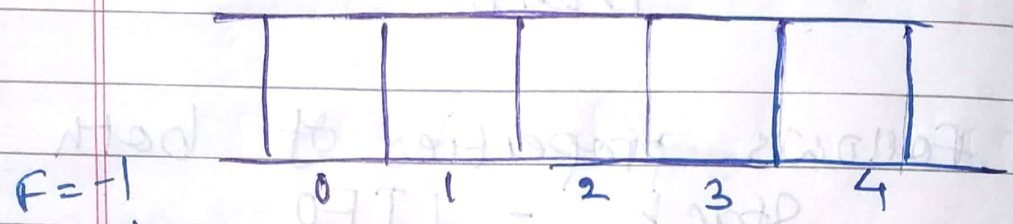
Operations:

- Insert at front
- delete from front
- insert at rear
- delete from rear.

- get front - isfull()
- get rear. - isempty().

* Implementation of Deque using circular Array:

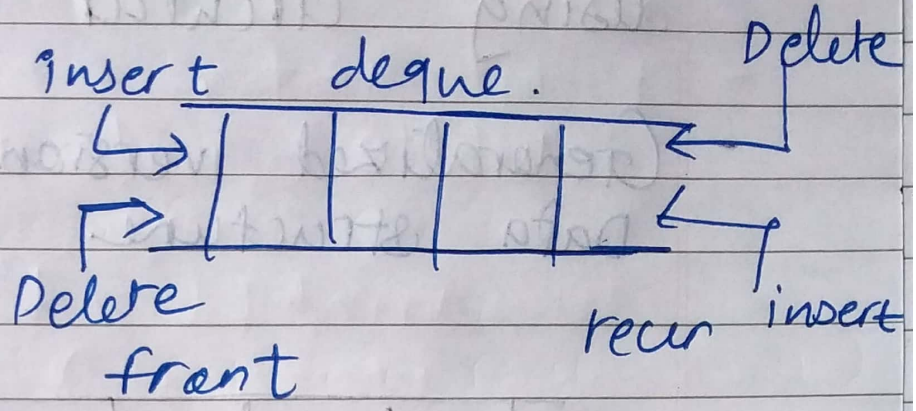
- Generalized version of Queue Data structure.




```
enqueuefront (2);
enqueuefront (5);
enqueue rear(-1);
enqueue rear(0);
enqueue front(7);
enqueue front(4);
display();
```

```
# define N 5
int deque[N];
int f=-1, R=-1;

void enqueuefront(int x)
{
  if((f==0 && R==N-1) ||
          (front == rear+1))
  {
      (queue is Full);
  }
```

* Double ended Queue:



- It follows properties of both
  Stack - LIFO
  queue - FIFO

Types:

1) Input restricted → input from only one end

2) Output restricted → Delete from only one end.

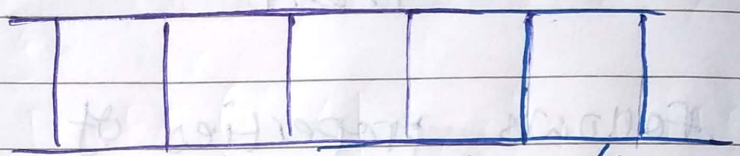
Operations:

- insert at front
- delete from front
- insert at rear
- delete from rear.

- get front - isfull()
- get rear. - isempty().

* Implementation of Deque using circular Array:

- Generalized version of Queue Data structure.



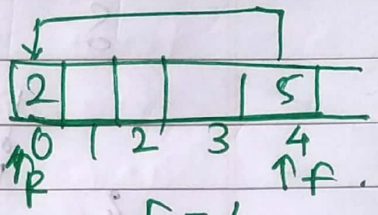
F = -1
R = -1

```
enqueuefront(2);
enqueuefront(5);
enqueuerear(-1);
enqueuerear(0);
enqueuefront(7);
enqueuefront(4);
display();
```

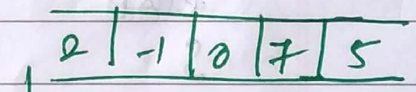
```
#define N 5
int deque[N];
int f=-1, R=-1;

void enqueuefront(int x)
{
  if((f==0 && R==N-1) ||
      (front == rear+1))
  {
      (queue is Full);
```

front-- | insert rear++
rear++ | Delete rear--

```
else if (f==-1 && R==-1)
{
    F=R=0
    deque[F]=x;
}
else if (f==0)
{
    f=N-1;
    deque[F]=x;
}
else
{
    f--;
    deque[F]=x;
}
}
```



f=4
R=0.




```
void enqueue rear(int x)
{
    if((f==0 && R==N-1) ||
       (front==rear+1))
    {
        (queue is full);
    }
    else if(f==-1 && R==-1)
    {
        f==R=0;
        deque[R]=x;
    }
    else if(R==N-1)
    {
        R=0;
        deque[R]=x;
    }
    else
    {
        R++;
        deque[R]=x;
    }
}
```

```
void display()
{
    int i=f;
    while (i != rear)
    {
        printf("%d", deque[i]);
        i = (i+1) % N;
    }
    printf("%d", deque[rear]);
}
void getfront()
{
        full.
    else
    {
        printf("%d", deque[front]);
    }
}
void get rear()
{
        empty
    else
    {
        printf("%d", deque[rear]);
    }
}
```

```
dequeue front();
dequeue rear();
dequeue front();
```

```
void dequefront()
{
   if(f==-1 && r==-1)
   {
      empty
   }
   else if(f==r)
   {
      f=r=-1;
   }
   else if(f==N-1)
   {
      f=0;
   }
   else
   {
      f++;
   }
}
void dequerear()
{  if(f==-1 && r==-1)
   {  empty }
   else if(f==r)
   {  f=r=-1;
   }
   else if(r==0)
   {
      r=N-1;
   }
   else
   {
      r--;
}  }
```

* priority Queue:

- Each element is inserted or deleted on the basis of their priority.
- Higher priority > lower priority
- Same priority [FCFS Basis]
  ↓
  First come first serve

⇒ Types of priority Queue:

Ascending order priority Queue [Lower priority no. to high priority]

1
2
3
4
5

Descending order priority queue [higher priority no. to higher priority]

5
4
3
2
1

5	2	1	4	3	← priority
5	7	16	19	32	← elements
0	1	2	3	4	← index.

→ Implementation of priority queue using Array :

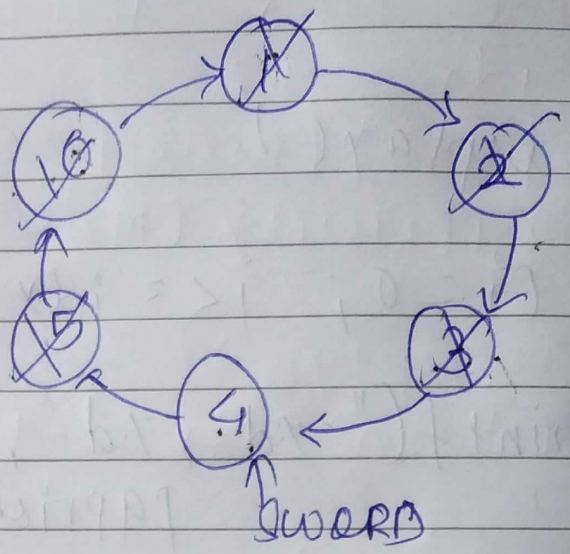
```
#define max 100
int idx = -1;
int pqval[max];
int pqpriority[max];

int isEmpty()
    return idx == -1;

int isfull()
    return idx == max-1;

void enqueue(int data, int priority)
{
    if(!isfull())
    {
        idx++;
        pqval[idx] = data;
    }   pqpriority[idx] = priority;
}

int peek()
{
    int maxpriority = INT_MIN;
    int indexpos = -1;
```

```
for(int i=0; i<=idx; i++)
{
    if(maxpriority == pqpriority[i]
        && indexpos > -1
        && pqval[indexpos] < pqval[
    {
        maxpriority = pqpriority[i];
        indexpos = i;
    }
    else if(Maxpriority < pqpriority[i
    {
        maxpriority = pqpriority[i];
        indexpos = i;
    }
}
return indexpos;
}

void dequeu()
{
    if(!isempty()
    {
        int indexpos = peek()

        for(int i = indexpos; i<idx;
                                i++)
        {
            pqval[i] = pqval[i+1];
            pqpriority[i] = pqpriority[i+1
        }
```

```
        idx--;
    }
}

void display()
{
    for(i=0; i<=idx; i++)
    {
        printf("%d, %d", pqval[i],
                         pqpriority[i]);
}
int main()
{
    enqueue(5,1);
            (10,3);
            (15,4);
            (20,5);
            (500,2);
    printf("Before Dequeue: ");
    display();

    dequeue();
    dequeue();

    printf("After Dequeue: ");
    display();

    return 0;
}
```

* Application of Queue

1)* Josephus problem.



Example: $w(n,k) \Rightarrow (w(n-1),k)+k)\%$

$w(1,k) \rightarrow 0$.

$N = 7 \quad K = 3.$

$w(7,3) = (w(6,3)+3) \% 7$

$= (((w(5,3)+3) \% 6)+3) \% 7$

$= ((((w(1,3)+3) \% 2)+3) \% 3)+3 \% 4))+3$

$= (3) \% 2$

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

$= (1+3) \% 4$

$= (0+3) \% 5$

$= (3+3) \% 6$

$= (0+3) \% 7$

$= \boxed{3}$

Algorithm for Josephus problem

```
int josephus(int n, int k)
{
    if (n == 1)
    {
        return 0;
    }
    else {
        int queue[50];
        for(int i=0; i<n; ++i)
            queue[i] = i;
        }

    int front = 0;
    // removing k-1 people from the front & move to the end.
    for(int i=0; i< k-1; ++i)
    {                          } no. of iterations till k-1
        front = (front + 1) % n;
    }

    int survivorIndex = josephus(n-1, k);
    return (queue[front + survivorIndex] % n);
}
}
```

```
int main()
{
pf("n = ")
scanf("%d", &k)

int survivorIndex = josephus(n, k);

}
```

Note:

If n = 1

No person is left.

~~that~~ person is the survivor.

no elimination needed.

$$(survivor + k) \% i$$