C-Programming and DS

Today Class Topics

Queue

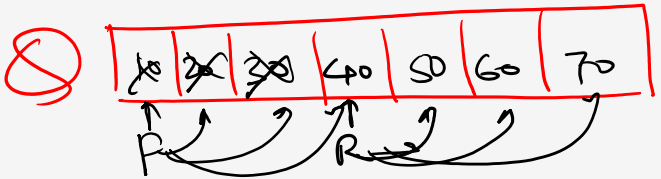
-Simple Linear queue
-Circular Queue
-Double ended queue
-Array implementation of queue
-linked list implementation of queue
-Stack implementation by queue
-Queue implementation by stack

Queue:- is a data structure used to store the data elements with the restriction of FIFO or LILO

Two pointer variables are used to implement Queue

Front :- used for deleting an element

Rear :- used for inserting an element



Applications of Queue:-

1) Ticket reservation ✓
2) Printer Spooling ✓
3) Job scheduling ✓
4) Finding minimum Spanning Tree (Prim's) and Shortest path (Dijkstra's)

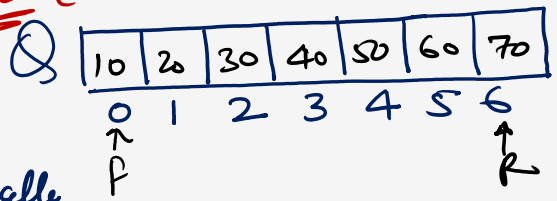
Implementing Simple Linear Queue :-

Array
Linked list

1) Using Array :

```
#define N 7
int Q[N];
```

Standard



Initially

F=0
R=-1

R=-1

Enqueue :- R = -1, 0, 1, 2, 3, 4, 5

```
R++;
Q[R] = x;
```

Dequeue :

```
y = Q[F];
F++;
```

overflow of R == N-1

Underflow Condition of R < F

Single element of R == F

Initially

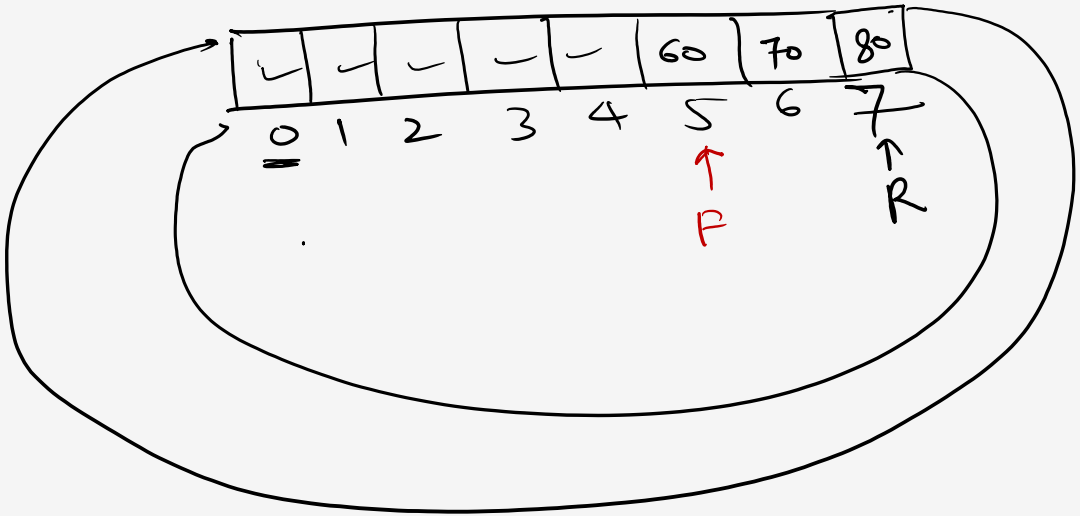
F=-1
R=-1

```
Algo: Enqueue(Q, N, R, x)
{
  if(R == N-1)
  {
    printf("Queue is full");
    return;
  }
  else
  {
    R = R+1
    Q[R] = x;
  }
}
```

TC = O(1)

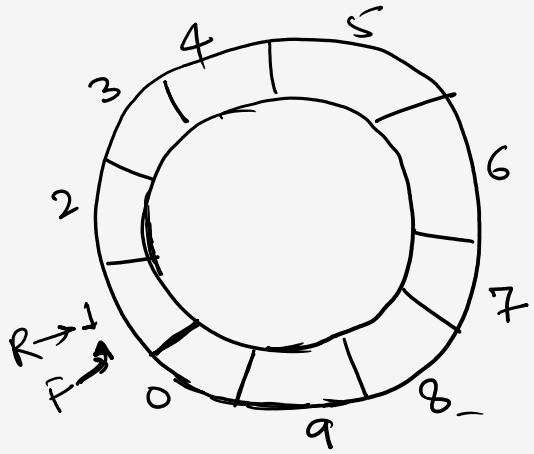
```
Algo: Dequeue(Q, F, R)
{
  int y;
  if(R < F)
  {
    printf("Queue is empty");
    return -1
  }
  if(R == F) // Single element
  {
    y = Q[F];
    Set F = 0; R = -1; //Initial Condition
  }
  else
  {
    y = Q[F++];
  }
  return y;
}
```

TC = O(1)


60
70
80
0 1 2 3 4 5 6 7
F
R

Circular Queue:-

```
#define N 10
int Q[N];
```



Standard Implementation ✓ ✓ ✓

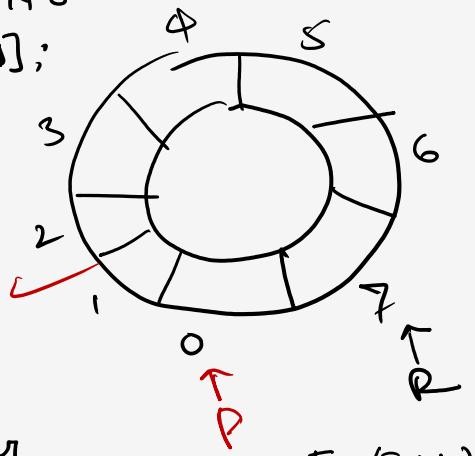
Initially
F = 0
R = 0

Overflow:
F == (R+1) % N

Underflow: F == R

Increment:
R = (R+1) % N , F = (F+1) % N

```
#define N 8
int Q[N];
```



Initially
F = -1
R = -1

F = (F+1) % N
R = (R+1) % N

Overflow:
F == (R+1) % N

Underflow: F == -1, or R == -1

Single element: F == R

Standard Queue:

```
Algo: Enqueue(Q, N, F, R, x)
{
    if(F == (R+1)%N)
    {
        pf("overflow");
        return;
    }
    else
    {
        R = (R+1)%N
        Q[R] = x;
    }
}
```

TC = O(1)

```
Algo: Dequeue(Q, N, F, R)
{
    int y;
    if(R == F)
    {
        pf("Underflow");
        return -1;
    }
    else
    {
        y = Q[F];
        F = (F+1)%N;
    }
    return y;
}
```

TC = O(1)

Suppose a circular queue of capacity (n – 1) elements is implemented with an array of n elements. Assume that the insertion and deletion operation are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0. The conditions to detect queue full and queue empty are

- [x] Full: (REAR+1) mod n == FRONT, empty: REAR == FRONT
- [ ] Full: (REAR+1) mod n == FRONT, empty: (FRONT+1) mod n == REAR
- [ ] Full: REAR == FRONT, empty: (REAR+1) mod n == FRONT
- [ ] Full: (FRONT+1) mod n == REAR, empty: REAR == FRONT

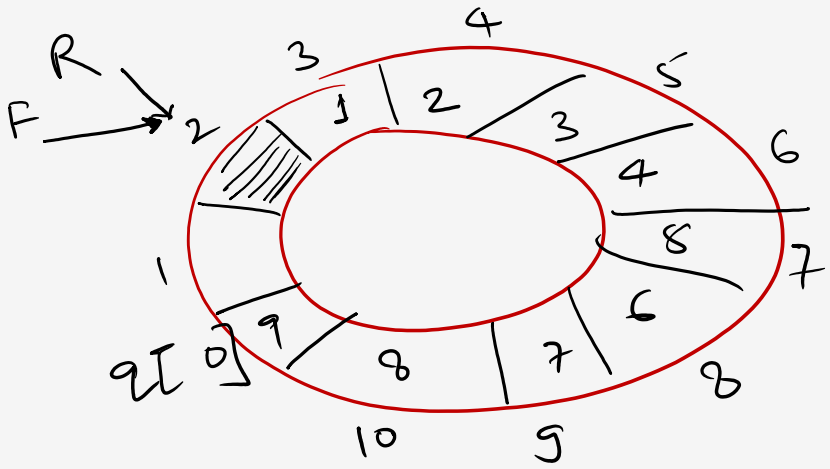
Question:- Consider a Standard Circular Queue (Q) implementation where size is 11 and the elements of the Queue are Q[0], Q[1], Q[2] - - - Q[10] The front and rear Pointers are initialized to Point Q[2]. In which Position will the 9th element be added

a) Q[0]

b) Q[1]

c) Q[9]

d) Q[10]



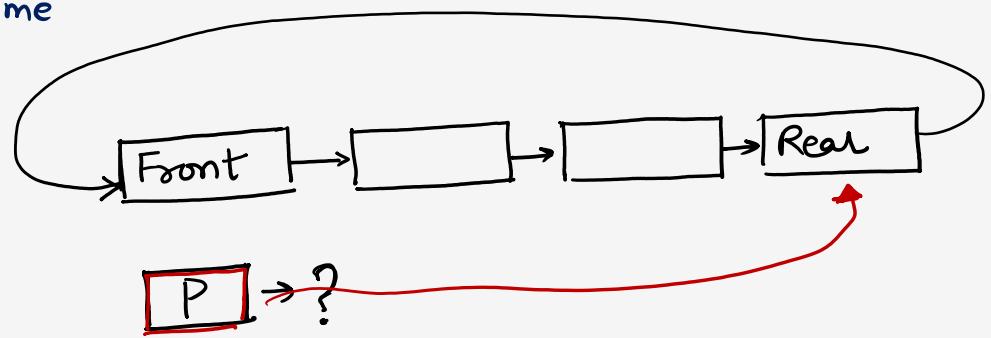
Question:- The circular linked list is used to represent a Queue. A single variable P is used to Accessed the queue, to which node should P point such that both the operation enqueue and dequeue can be Performed in Constant time

a) Rear

b) front

c) Not Possible with single Pointer

d) Node not to front

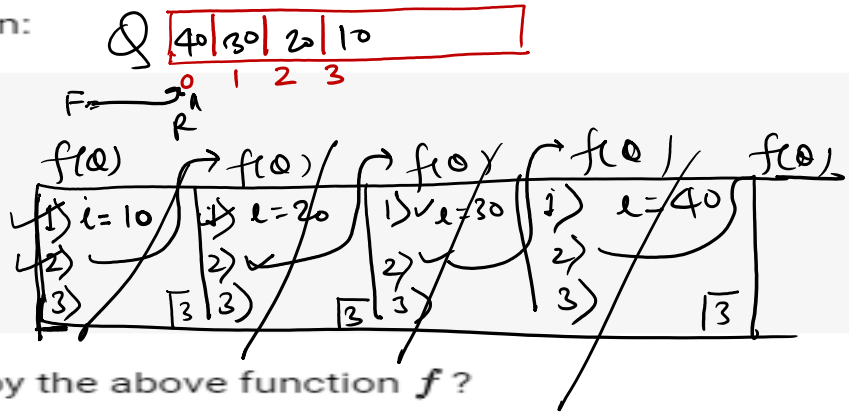


Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are:

i. $isEmpty(Q)$ — returns true if the queue is empty, false otherwise.
ii. $delete(Q)$ — deletes the element at the front of the queue and returns its value.
iii. $insert(Q, i)$ — inserts the integer i at the rear of the queue.

Consider the following function:

```
void f (queue Q) {
int i ;
if (!isEmpty(Q)) {
  i = delete(Q);
  f(Q);
  insert(Q, i);
  }
}
```



What operation is performed by the above function f ?

A. Leaves the queue Q unchanged
B. Reverses the order of the elements in the queue Q
C. Deletes the element at the front of the queue Q and inserts it at the rear keeping the other elements in the same order
D. Empties the queue Q

Question:- Consider the following Program which uses 3-data structure, Queue(Q_1), Queue(Q_2) and Stack(St). The Program takes Q_1 values as input and performs some operations

Q_1 = 16, 13, -21, 8, 6, 0, 19, 21, -11, 8, 36, 0, 18, 13

```
xyz(Queue Q1)
{ int S=0;
  while(!isEmpty(Q1))
    { x = dequeue(Q1);
      if(x==0)
      { S+=pop(St);
        enqueue(Q2,S);
        S=0;
      }
      push(St,x);
    }
}
```

i) What is the Sum of all value in Stack After Completion of the above Program 84 ?

ii) What is the max stack size is used 12 ?

iii) What is the Sum of all values of Q_2 42 ?

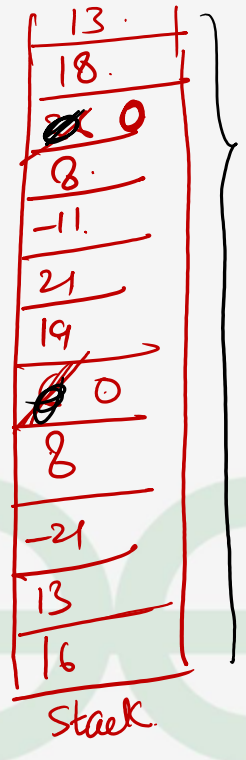
iv) What is the max Q_2 size used 2 ?

$S=S+pop(St) \Rightarrow 0+36=36$

$S=0+6=6$

Q_2 | 6 | 36 |

1 2



Question: The Program takes Q_1 values as input and Perform some operations

$Q_1 = 3, 31, 19, -11, 6, 0, 13, 21, -15, 18, 7, 0, 18, 9$

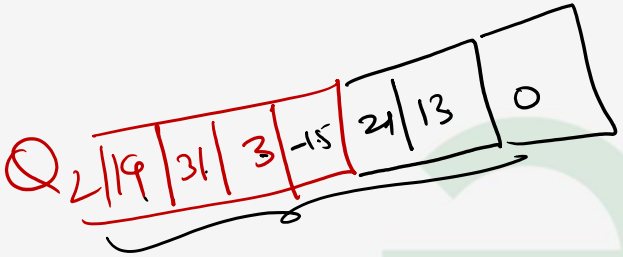
```
xyz(Queue Q1)
{While(!isEmpty(Q1))
   { x = dequeue(Q1);
     if(x==0)
     { pop(St);
       pop(St);
       While(!isEmpty(St)){
         enqueue(Q2, pop(St));}
     }
       push(St,x);
   }
}
```

i) What is the Sum of all values in STACK After Completion of the above Program (27)?

ii) What is the max stacksize used (6)?

iii) What is Sum of all values of Q_2 (72)?

iv) What is max Q_2 size used (7)?



Question:- The Program takes Q_1 values as input and Perform some operations

Q_1: 27, 14, 16, -18, 12, 0, 31, -19, 6, 11, 14, 0, 13, 18

```
xyz (Queue Q1)
{ Static int S;
 While(!isEmpty(Q1))
  { x= dequeue(Q1);
   if(x==0)
    {while(!isEmpty(st)){
        S+=pop(st); }
        S+=S; //double
      enqueue(Q2,S);
    }
    push(st,x);
  }
}
```

1) What is Sum of all values in Stack, after Completion of above Program (31) ?

2) What is max stacksize used (6) ?

3) What is Sum of all values of Q_2 (392) ?

4) What is max Q_2 size used (2) ?

Break: 10 mins.

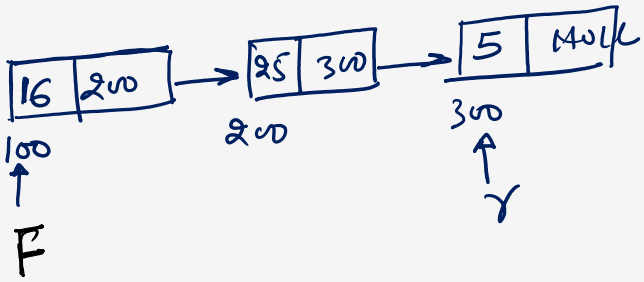
Linked List Implementation of Simple Linear Queue:-

Initially F = R = NULL

Enqueue:

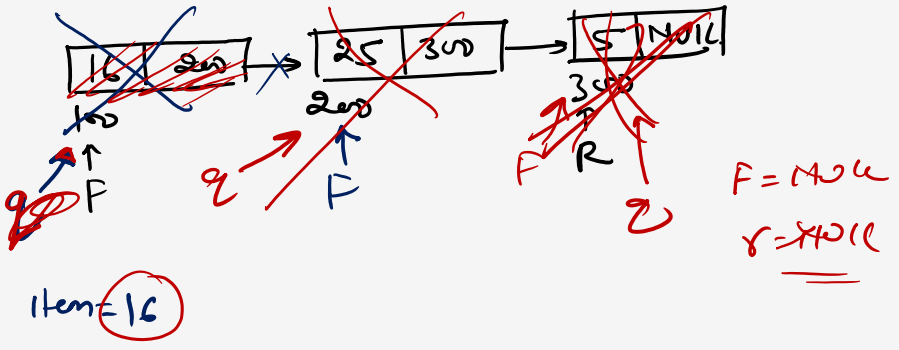
```
void Enqueue(node *f, node *r, int item)
{ node *q;
  q = (node *) malloc(sizeof(node));
  q->data = item;
  q->link = NULL
  if(f==NULL)
     f=q
  else r->link = q
  r=q;
}
```

TC = O(1)



Dequeue :-

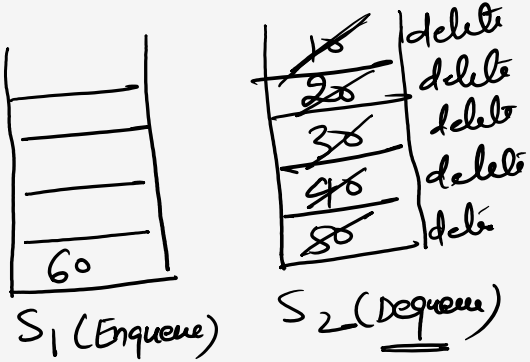
```
int Dequeue (node *f, node *r)
{ node *q;
  int item;
  if(f==NULL)
     Pf("Queue is Empty");
  else { q=f;
         item = q->data;
         f= q->link;
         free(q);
         q=NULL;
       if(f==NULL)
            r=NULL;
       return(item);
  }
}
```



Implementing Queue using Stacks:-

2 stacks required

Q | 10 | 20 | 30 | 40 | 50 | 60 |

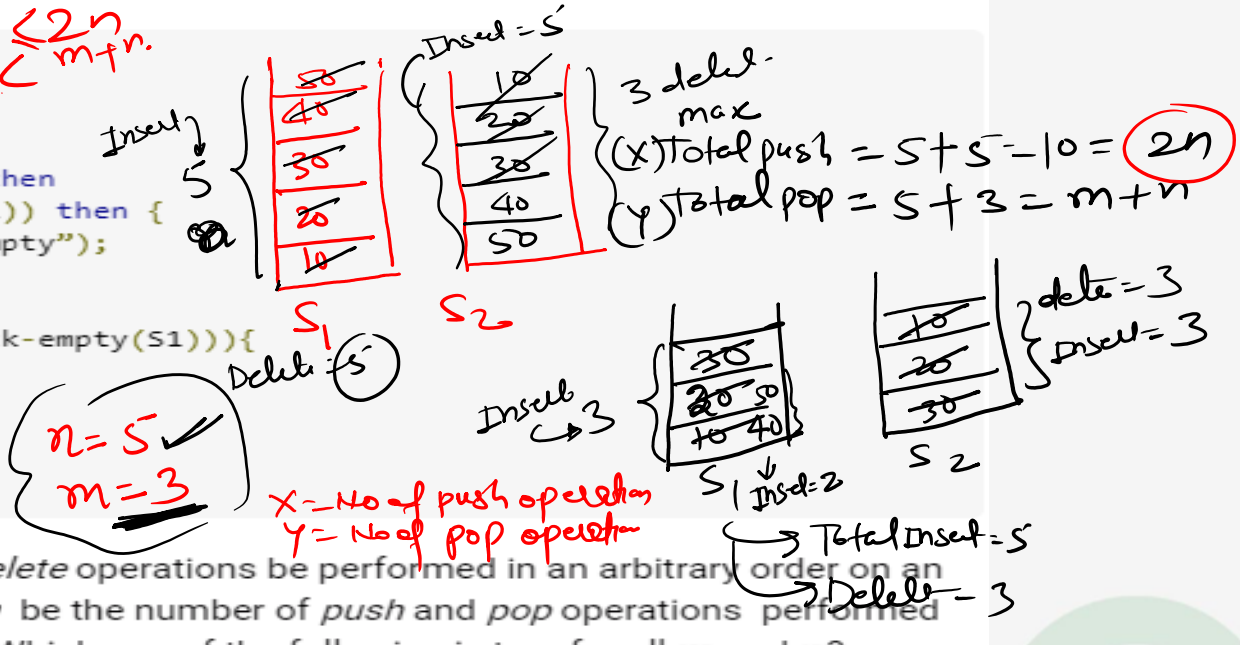



```
Enqueue (S1, x)
{
    push(S1, x);
}
```

```
int Dequeue(S1, S2)
{
    if (S2 is not empty)
        return pop(S2);
    if (S1 is empty)
    {
        printf("Underflow");
        return -1;
    }
    else
    {
        while (S1 is not empty)
        {
            x = pop(S1);
            push(S2, x);
        }
        return pop(S2);
    }
}
```

An implementation of a queue Q, using two stacks $S1$ and $S2$, is given below:

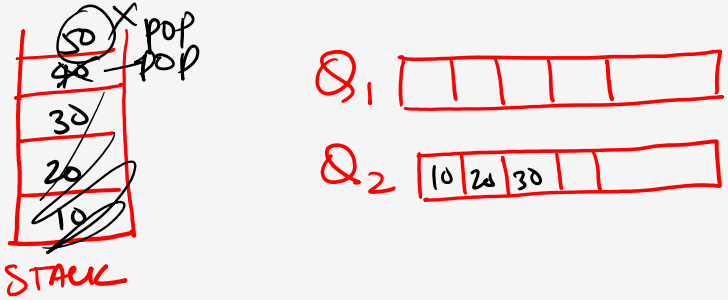
```
void insert (Q, x) {
    push (S1, x);
}
void delete (Q) {
    if (stack-empty(S2)) then
        if (stack-empty(S1)) then {
            print("Q is empty");
            return;
        }
        else while (!(stack-empty(S1))){
            x=pop(S1);
            push(S2,x);
        }
    x=pop(S2);
}
```



Let n *insert* and $m(\leq n)$ *delete* operations be performed in an arbitrary order on an empty queue Q. Let x and y be the number of *push* and *pop* operations performed respectively in the process. Which one of the following is true for all m and n?

A. $n+m \leq x \leq 2n$ and $2m \leq y \leq n+m$
B. $n+m \leq x < 2n$ and $2m \leq y \leq 2n$
C. $2m \leq x < 2n$ and $2m \leq y \leq n+m$
D. $2m \leq x < 2n$ and $2m \leq y \leq 2n$

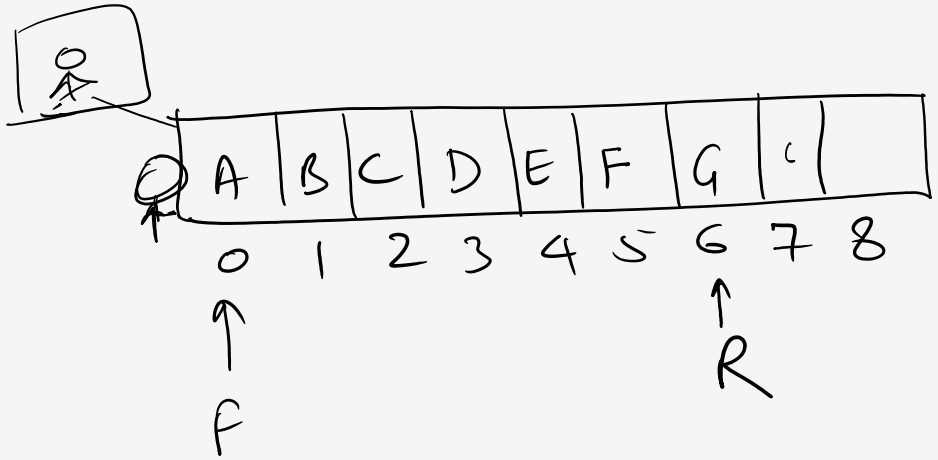
Implementing Stack using Queue's :- 2 Queue



Algo:

```
push(Q1, Q2, x)
{ if(Q1 is empty)
      enqueue(Q2, x);
  else
      enqueue(Q1, x);
}
```

```
int pop(Q1, Q2)
{ if(Q1 is empty)
  { if(Q2 is empty)
    { pf("stack is empty");
      return -1;
    }
    else
    { while(Q2 does not contain one element)
      { x = dequeue(Q2);
        enqueue(Q1, x);
      }
      return dequeue(Q2);
    }
  }
  else { while(Q1 does not contain one element)
         { x = dequeue(Q1);
           enqueue(Q2, x);
         }
         return dequeue(Q1);
  }
```

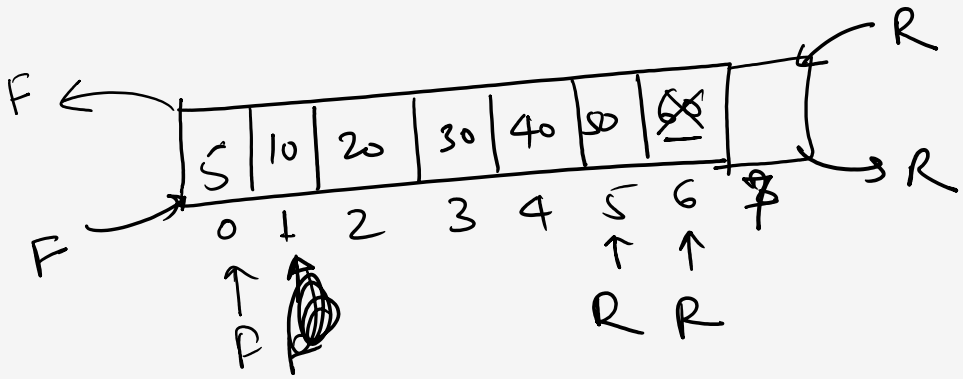

A
B
C
D
E
F
G
0
1
2
3
4
5
6
7
8
F
R

Double ended Queue:- FIFO & LIFO

↳ Two types

1) Input restricted Dqueue
   1) Insertion at rear
   2) Deletion at rear
   3) Deletion at front

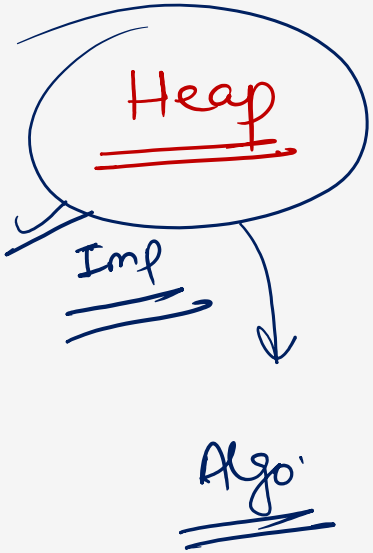
2) output restricted Dqueue:
   1) Insertion at rear
   2) Insertion at front
   3) Deletion at front



Pointer	Insertion	Deletion
F	F=F-1	F=F+1
R	R=R+1	R=R-1

Priority Queue :-

elements 70, 13, 80, 7, 90, 16, 72



1) Ascending Priority Queue (delete min)

deletion order : 7, 13, 16, 70, 72, 80, 90

2) Descending Priority Queue (delete max) :

deletion order : 90, 80, 72, 70, 16, 13, 7.

Thank You !