Queue → FIFO

Array Implementation

front

rear

enqueue

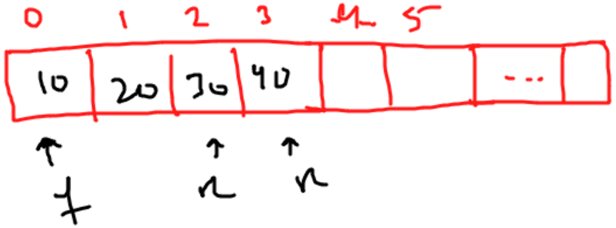
Insert → rear end

Delete → front end

dequeue

deque

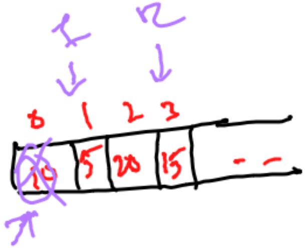
```
int Queue[], size, f, r;

f = r = -1;

void enqueue (int x)
{
    if( r == size-1)
    {
        s.o.p("Queue overflow");
        return;
    }
    if( r == -1)
        f = r = 0;
    else
        r++;
    Q[r] = x;
}
```



f r
0 2 3

```
r++;
Queue[r] = x;
```

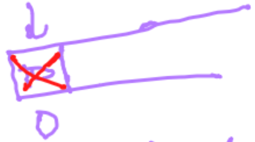
f r
-1 -1
0



$\frac{f}{\cancel{0},\ 1}$ $\frac{r}{3}$

s.o.p(queue[f]);

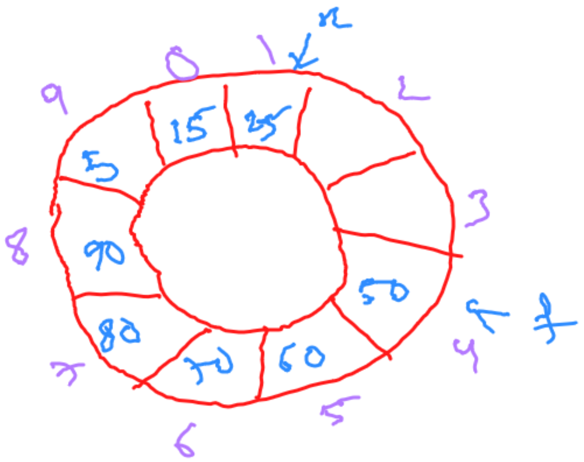
f++;



$\frac{f}{\cancel{0}}$ $\frac{r}{\cancel{0}}$

-1 -1

Circular Queue



Cons of Simple Queue :-

solution

Memory utilization is not efficient

(Efficient memory utilization

Circular Queue

overflow condition:

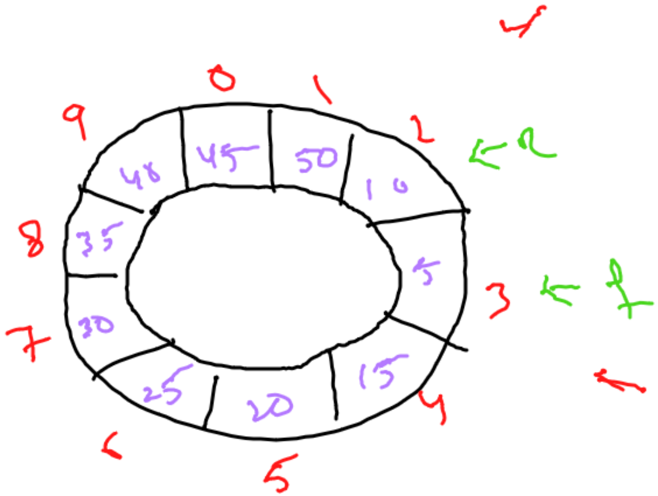
$(f == r+1)$ ←

or $(f == 0$ && $r == size-1)$

$\boxed{\text{if}(\ f == (r+1) \% \ size\ )}$

→ $f{+}{+} \Rightarrow f = (f+1) \% \ size$

→ $r{+}{+} \Rightarrow r = (r+1) \% \ size$



$(9+1) \% \ 10$

$= 0$

$(2+1) \% \ 10 = 3$

Deque → Double Ended Queue

↳ Insert & delete can be done on both end.

Types → (1) Input Restricted Queue ← Insert at one end, delete on both ends

(2) Output Restricted Queue ← Delete on one end, Insert at both ends.

deleteFront()	insertFront()
deleteRear()	insertRear()
insert()	delete()

Linked List

head · tail

20 → 25 → 10 → 15 → 30 → 40

Node

Node head;

Node tail;

Node	data	next

data ⇒ data of node

next → refer to the next node

```
class Node
{   int data;
    Node next;
}
```

	data	next
k →	10	null

k.data = 10

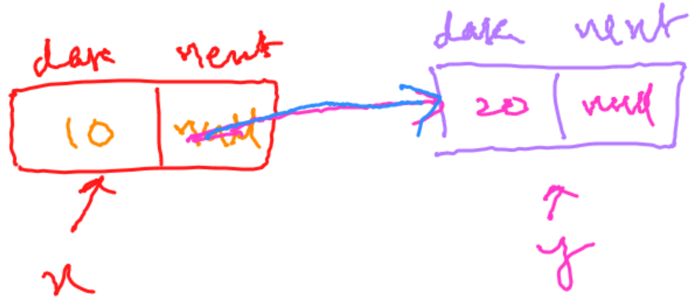
k.next = null

Test ob;

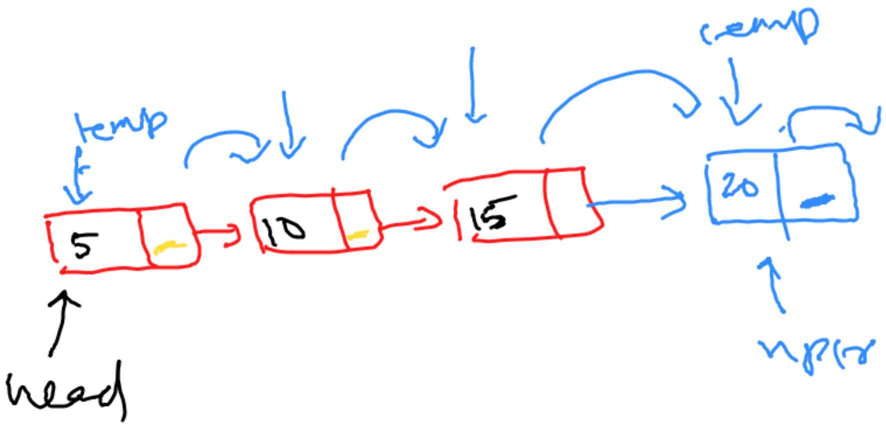
```
class Node
{ int data;
  Node next;
}
```

```
Node x = new Node();
x.data = 10
x.next = null
Node y = new Node();
y.data = 20;
y.next = null

x.next = y;
```

```
Node *x;
```



head = next.



temp.next = nptr

5 10 15 20