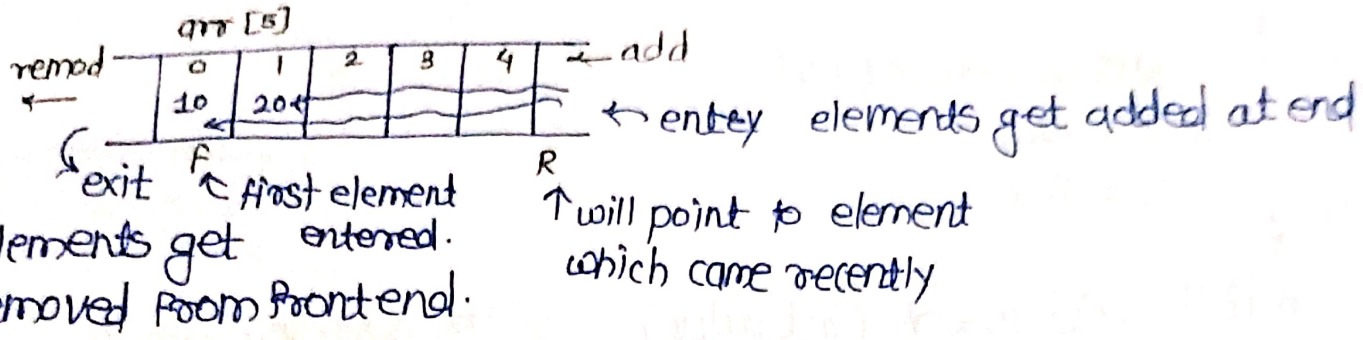
Student stack

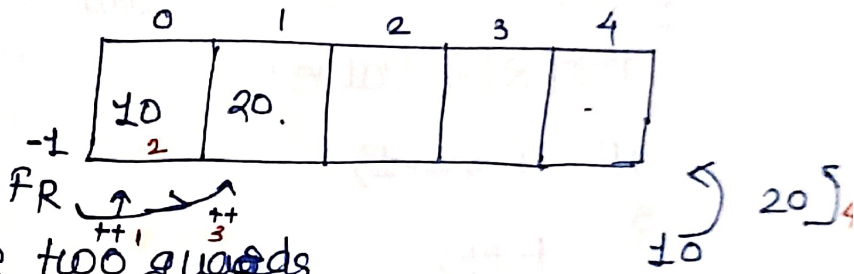
take class of student as it is for stack element
logic of stack will not change only argument / attributes of stack will change.

Queue



code Logic

public static void Main
{
}

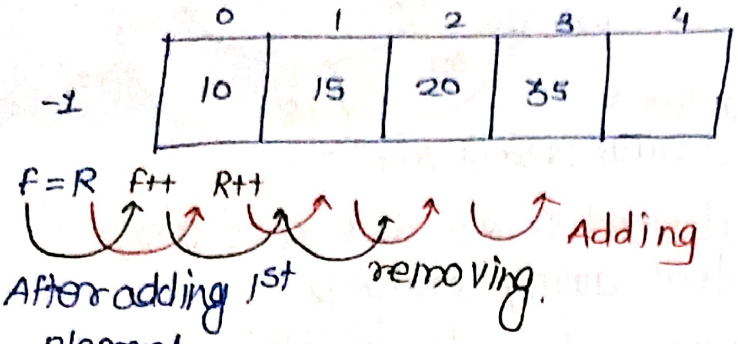


- First of all, F & R are two guards we take pointing to -1.
- wanna put element, (10)
  increment R,
  tell element, enter queue & place where you meet R.
- wannal put another (20)
  stop stop — ++R first
  then add 20 at R.

```
Public class Queue
{
    int [] arr;
    int F;
    int R;

  public Queue ()
 {
    arr = new int [5];
    f = -1;
    R = -1;
}
public void insert (int value)   // To add element in
 { If (R < arr.length-1) //Full cond^n   que Queue.
  { R++                                  push operation.
    arr[R] = value;
    if ( f == -1)
    {
      f++;
    }
  }
}

public ~~void~~ int remove ()   // To remove element from
{ ~~if ( f == -1)~~ if(f != -1){   Queue
  int value = 0                    pop operation.
  value = arr[f];
  ~~f++~~ if (f != R)                Main {
        f++;                          Queue q1 = new Queue ();
      else                            q1. insert(20);
        f = R = -1;                   q1.
  ~~return value;~~
  * else
        SOP - ("Queue is empty");

to string
    string ste = " ";
  if (F == -1) ste = "que is empty"
  else
  { for (int i=F; i<=R; i++)
    { ste += arr[i] + ",";
```

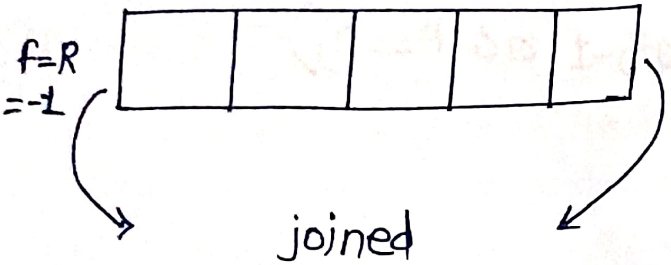
1st April (day-8)

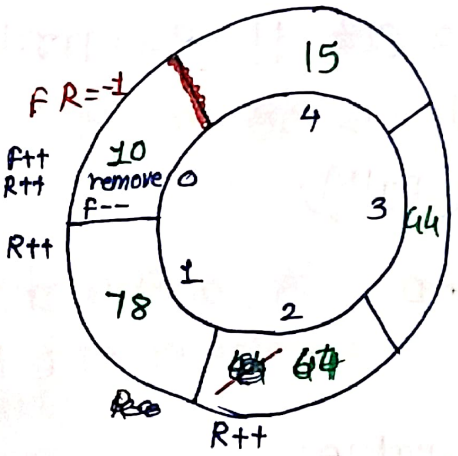
Disadvantage of Queue:- We will not be able to use free space after removal of element until all element get removed.



After removing elements, front gets shifted and space before f will not be able to use further.

To overcome this advantage : Circular Queue came into picture





operations to perform:

insert (10) — f & R gets ++, add 10

insert (78) — R++, add 78

insert (67) — R++, add 67

insert (44) — R++, add 44

remove() — ~~remove 44~~, R—, remove 10, f++

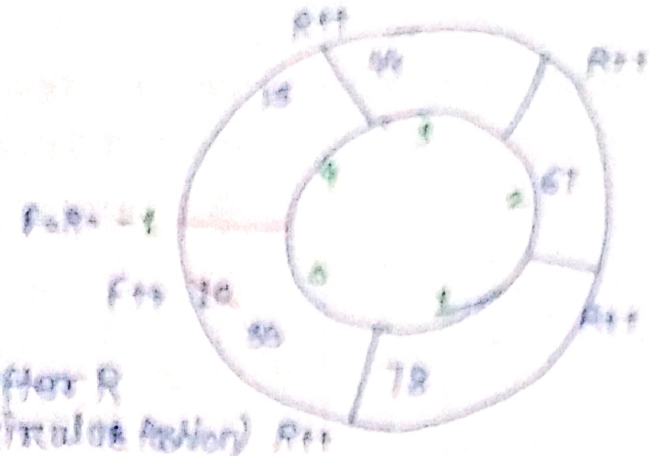
insert (15) - R++, add 15

insert (35) - R++, add 35 (As there is space after R)

```
insert ()
{
    if(R != arr.length)
        R++
    else
        R=0
}
```

R will be either at length or at zero. for insert operation.

Array gets fulled when
there will be R & F Rotated
by F



Full :

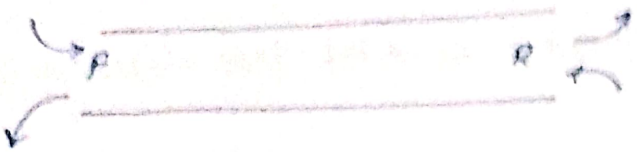
- R + 1 == F // there is F after R (in circular fashion)
- R == last && F == 0
  arr.length-1 // F is at start & R is at end of array [Ideal array straight]

code is same as of Queue only exit conditions & full queue insert, remove conditions will change.

```
Public void insert (int value)
{
  if ( R+1==F || (R==arr.length-1 && F==0))
  // full condition for queue
  {
    Sysout (Full);
  }
  else  // now it is confirmed that there is space to insert available
        so increment Rear & put value over R.
  {
    R++ ;
    arr[R]=value;
    if ( F==-1)  // if it is the first value to insert in que, increase
      F++;       F as well who is at -1.
  }

public int remove()
{   int value=0;   //declaration
if (F==-1)
  { que empty
  }
else // now it is confirmed que is not empty so remove first value.
{  int value = arr[F];   if( F!=R)   else
                         {           {
                           F++;        F=R=-1;
                         }           }
```

day-4 2nd April (online lecture)

DQueue : double ended Queue

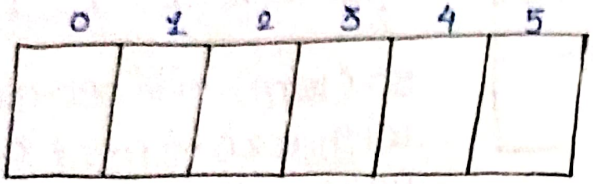


In double ended Queue, we can insert and remove element from both the ends.

insert fn → from f / from R

remove fn → from f / from R.

arr



-1

f=R

insert:

R++

f--

remove:

R--

F++

contents :- 10 34 126
45 26
90

Full f==0
R== length-1

While inserting :- from front

i) Que may be full // can't insert

ii) Que is empty // add at 1st position

Empty f=R=-1.

iii) f is not at '0' index so decrement and add

iv) R is ~~at~~ not at "arr.length -1" so ~~add at last~~ (circular que insert)
↑ handled in first

So now, to at at front,

shift all values 1 space later & add at first.