Multiple Stacks and Queues

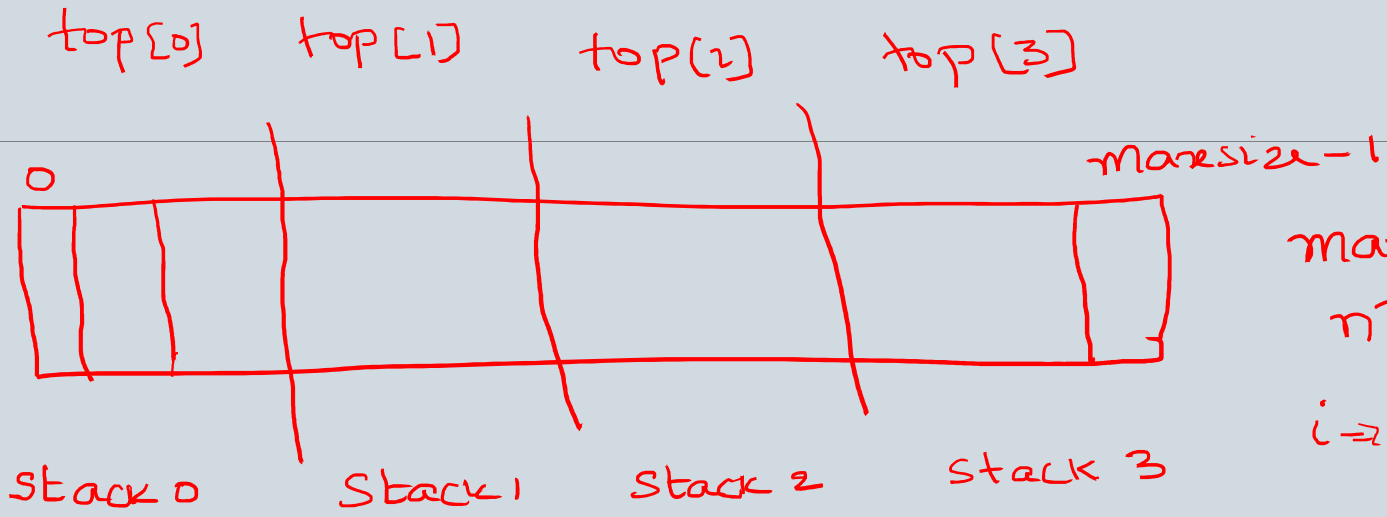
BY

DR. RASHMI NAVEEN RAJ, I&CT DEPT.,

RASHMI.NAVEEN@MANIPAL.EDU

LECTURE- 10, OCT 30, 2021

Multiple stacks



maxsize

n → no of stacks

n = 4 (0 to 3

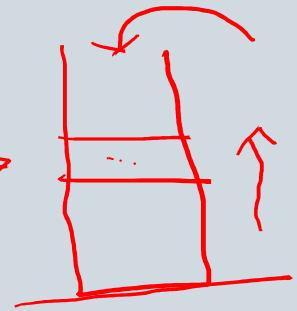
i ⇒ ith stack (0 to 3)

int top[10];

n

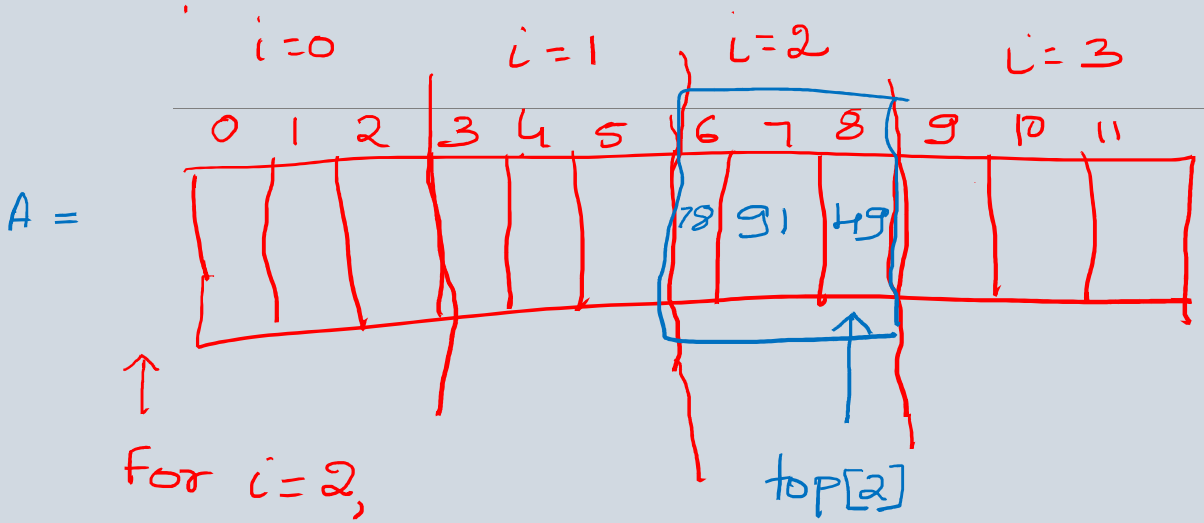
top[i] ⇒ index of top element of the ith stack

top →



maxsize of each stack = maxsize / n

Multiple stacks



maxsize = 12
n = 4

ith stack size $= \frac{12}{4} = 3$

For i=2,

top[2] = 8

if (top[i] == $\frac{\text{maxsize}}{n} \times (i+1)) - 1$)

ith stack is full

if (top[2] == 8) stack 2 is full

i=1 top[1] == 5 1 is full

i=0 top[0] == 2

i=3 top[3] == 11 same as maxsize − 1

0
1
0
1
2
3

$12/4 * 1 = 3 - 1 = 2$

$3 * 2 - 1 = 5$

$3 * 3 - 1 = 8$

$3 * 4 - 1 = 11$

Multiple stacks

Emptycase

$top[i] == \frac{maxsize}{n} * i - 1$

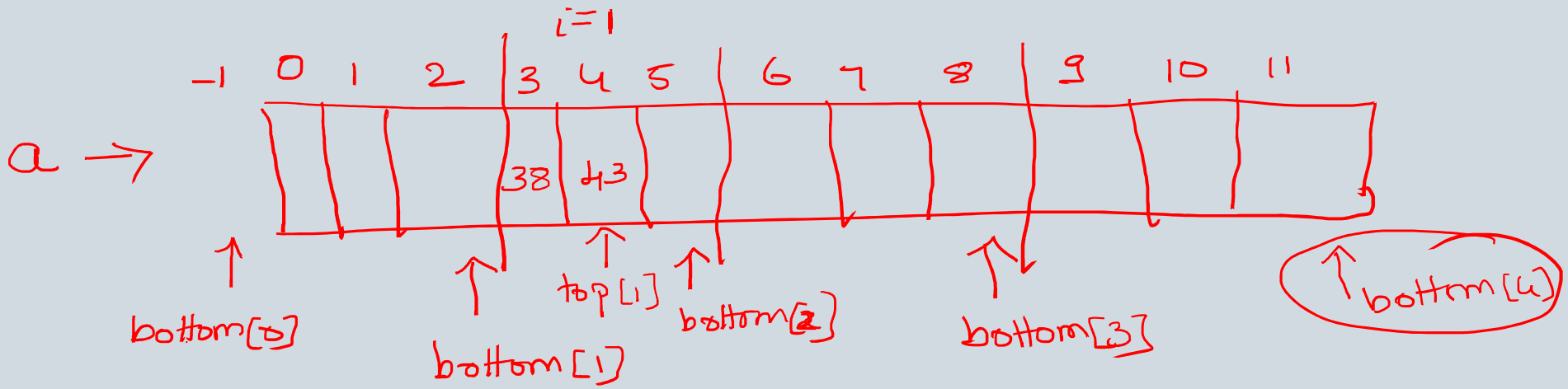
i

0 $top[0] == -1$ empty

1 $top[1] = 3*1 - 1 = 2$
$top[1] == 2$ } stack 1 is empty

$top[2] == 5$

$top[3] == 8$



$top[i] = bottom[i] = \frac{maxsize}{n} * i - 1$ // initialize

maxsize = 12
n = 4

Empty: if (top[i] == bottom[i]), stack is empty

int bottom[10];

Full: if((top[i] == bottom[i+1]) || (top[i] == maxsize-1))

i = 0, 1, 2 stack is full

push(int i, int ele)
{ if ith stack is not full, then a[++top[i]] = ele;

Pop

```
return(a[top[i]--]);
```

Multiple stacks

```
#include<iostream>
using namespace std;
#define max_size 20
class stack
{
   int top[10];
   int a[50];
   int boundary[10];
   public:
      stack(int);     // no. of stacks
      void push(int ,int);
      void pop(int);
      void display(int);
};
```

```
stack::stack(int n)
{
    for(int i=0;i<n;i++)
    boundary[i]=top[i]=(max_size/n)*i-1;
}
```

```
void stack::push(int i,int x)
{
        if((top[i]==boundary[i+1])||(top[i]==(max_size-1)))
        cout<<"Stack is full \n";
        else
        a[++top[i]]=x;
}
```

$i = 0, 1 \ldots n-2$

last stack ie $i = n-1$

```
void  stack::pop(int i)
{
   if(top[i]==boundary[i])
      cout<<"stack is empty\n";
   else
         cout<<"deleted element is "<<(a[top[i]--]);}
```

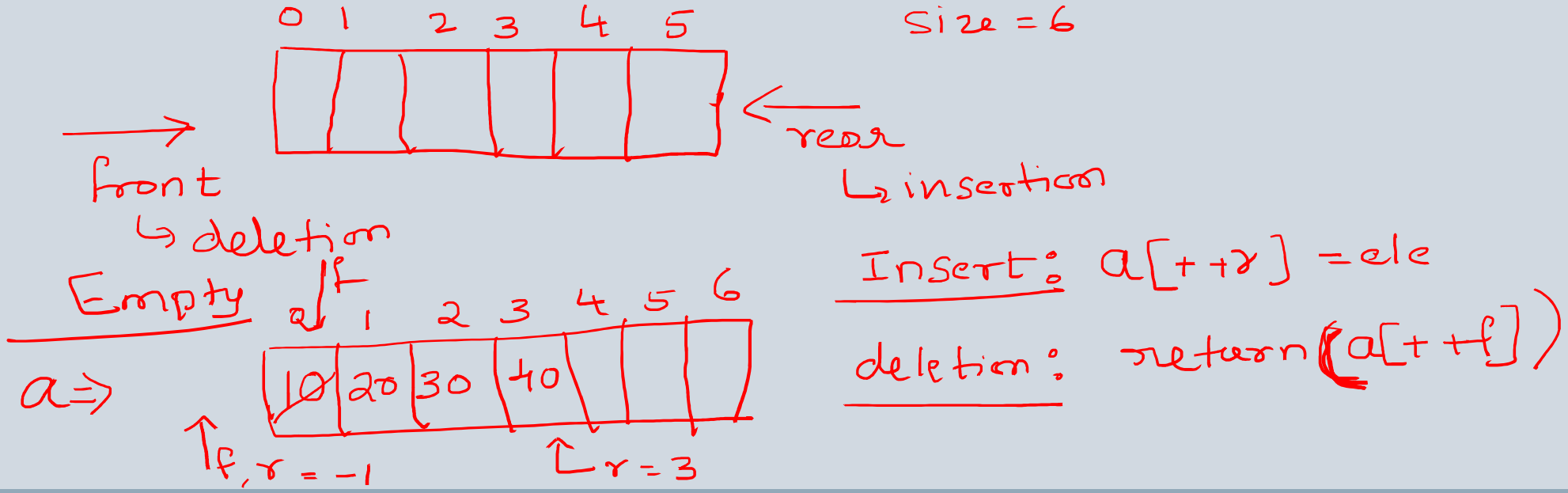
```
void stack::display(int i)
{
        if(top[i]==boundary[i])
        cout<<"stack is empty\n";
        else
        for(int j=top[i];j>boundary[i];j--)
        cout<<"\nThe elements of stack are "<<"\n"<<a[j];
}
```

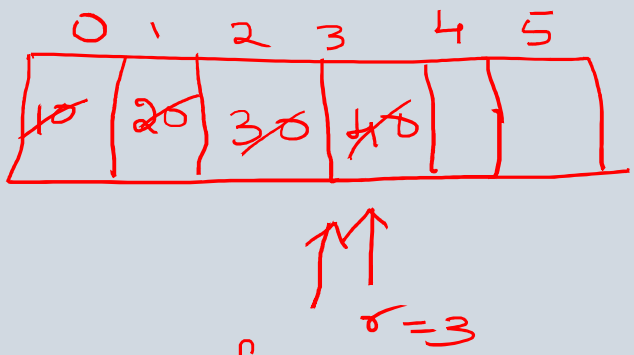
```
int main()
{
Write a menu driven program
return 0;
}
```

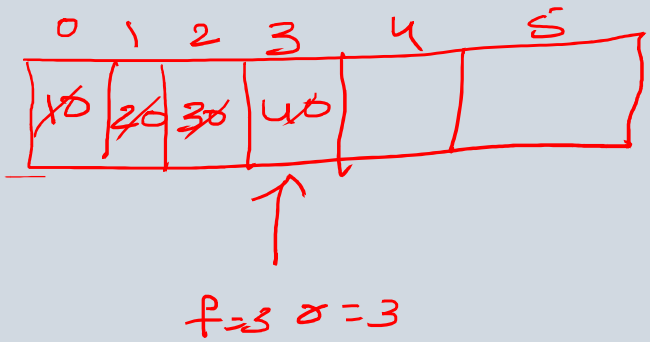
Linear Queue

- Queue is an ordered list in which insertions and deletions happen at different ends (First in First Out)

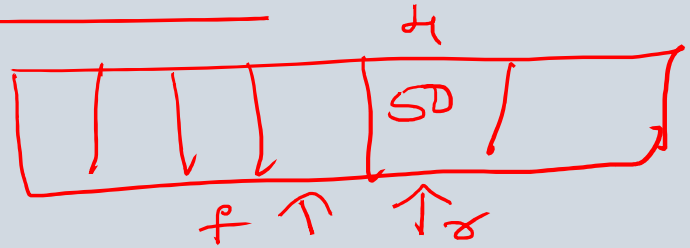
- The elements are added at the rear end and deleted at the front end.





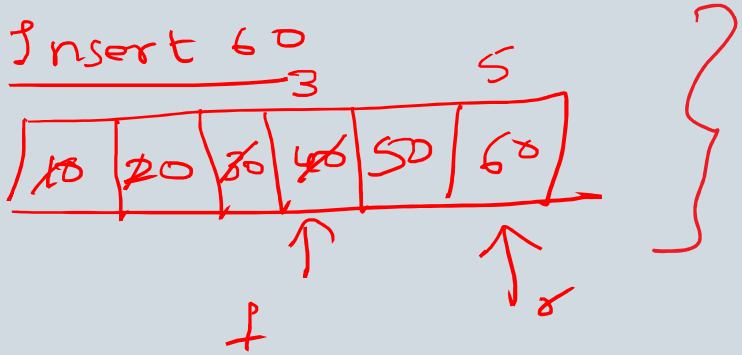


Insert 50

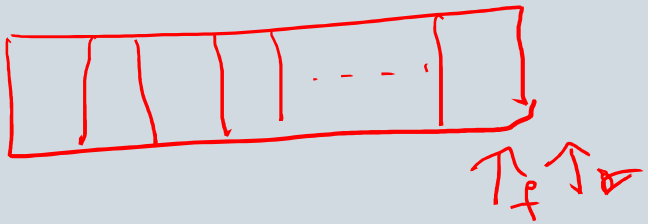


Empty ⇒ if (f = = r)

Insert 60



Full ⇒ if (r = = maxsize − 1)

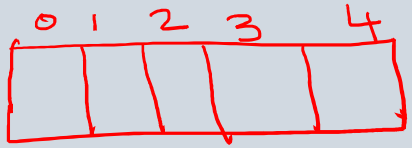


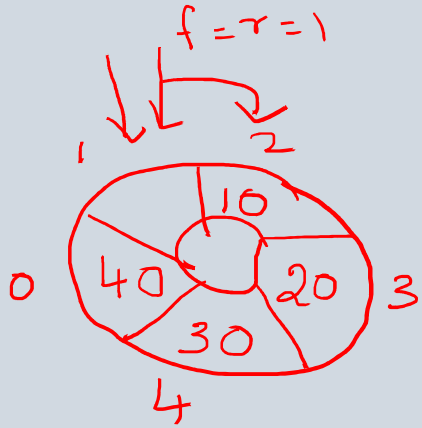
```
class  queue{
            int front, rear;
            int a[10];
            int size;
public:
            queue(int n)
            {  front=-1;rear=-1; size=n}
            void insert(int);
            void remove();
            void display();
};

void  queue:: insert(int ele)
            {
            if (rear==size-1)
                  cout<<"queue is full";
            else
                  a[++rear]=ele;
            }
```

```
void  queue:: remove()
            {
            if (rear==front)
                  cout<<"queue is empty";
            else
                  return(a[++front]);
            }

void display()
{
  if (rear==front)
                  cout<<"queue is empty";
  else
            for(int i=rear; i>front;i--)
            cout<<a[i]<<" ";

}
```

// i = front+1
i <= rear
i++

Circular Queue





① Insert 10, r = 2, f = 1

② Insert 20 r = 3, f = 1

③ Insert 30 r = 4 f = 1

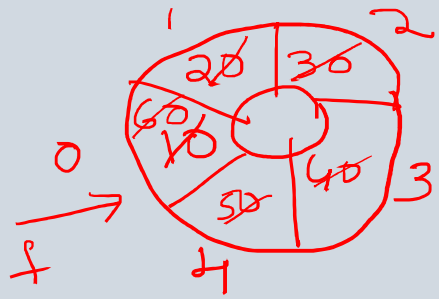
④ Insert 40 (4+1) % 5 = 0

Insertion

r = (r + 1) % maxsize

a[r] = ele

```
if(r == maxsize-1)
        r = 0;
else
        r = r+1;
```



① Delete 10, $f = 0$ $r = 4$

② Insert 60, $r = 0$ $f = 0$ ⟸ Full

③ Delete all, $r = 0$ $f = 0$ ⟸ Empty

Remove :

$f = (f + 1) \% \text{maxsize}$

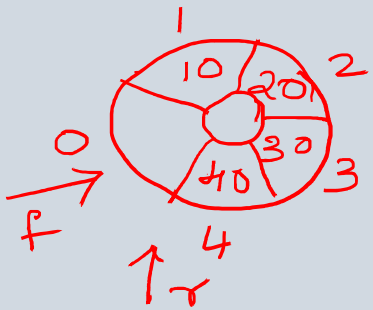
return $a[f]$

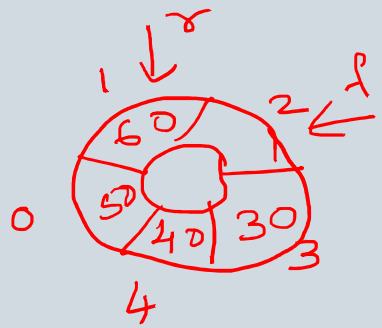
---

if maxsize $= n$, then store $(n-1)$ elements ✓

Initialize $f = 0$, $r = 0$

Insert 10, 20, 30, 40





Remove	10	$f = 1$	$r = 4$
Remove	20	$f = 2$	$r = 4$
Insert	50	$f = 2$	$r = 0$
Insert	60	$f = 2$	$r = 1$

if $((r+1) \% maxsize == f)$, circular Q is full

Remove 30, 40, 50, 60: $f = 1$, $r = 1$

if $(f == r)$, circular Q is empty.

```
class CQ{
        int rear, front;
        int a[20];
        int maxsize;

public:  CQ(int n)
        { maxsize = n; }
        void insert(int ele);
        void remove();
        void display();
}

void CQ::insert(int ele)
{   if((rear+1)%maxsize == front)
            cout << "CQ is full\n";
    else
        { rear = (rear+1)%maxsize;
          a[rear] = ele;
        }
}
void CQ::remove()
{ if(front == rear)
        cout << "CQ is empty\n";
  else
      { front = (front+1)%maxsize;
        cout << "deleted element is:" <<
                a[front];
      }
}
```

```
void CQ::display()
{
    if(front==rear)
        cout<<"CQ is empty\n";

    else
    {

    }

}
```

$\rightarrow$ $i = rear;\ i\%=f;$