(10 20)

Delete

(10)

Delete

( )

40

30

20

10

Last In First Out (LIFO)

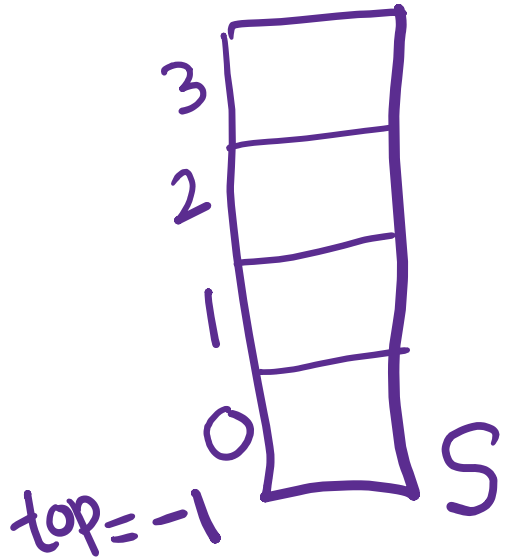
Insert a new element :— push

Delete :— pop

Implementation of stack using array:

- Elements will be kept in an array
- we need one variable to keep track of the 'top' of the stack

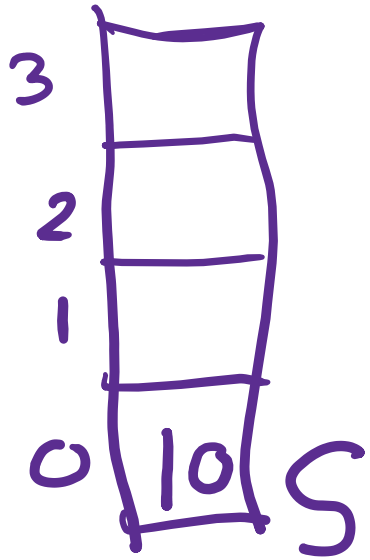
✓Push 10



top= top+1

top = 0

S[top] = 10



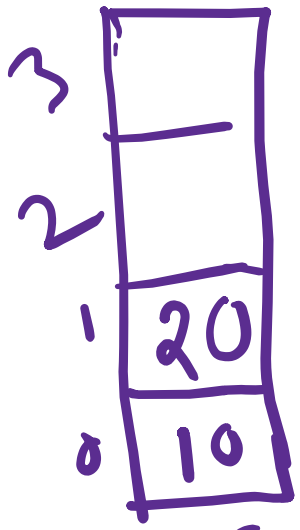
top=0

push 20

top = top + 1

top = 1

S[top] = 20



top = 1 S

#define MAXSIZE 4

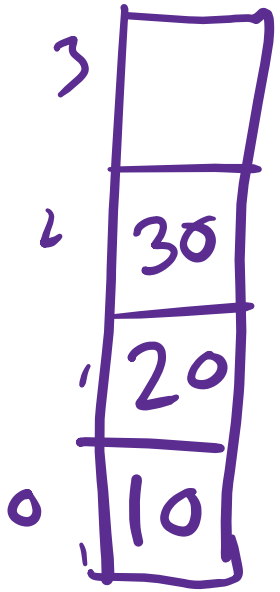
int S[MAXSIZE]

push 30

top = top + 1

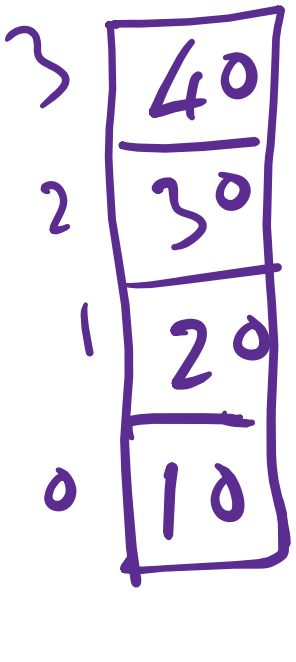
top = 2

S[top] = 30



top = 2 S

push 40



top = 3

S

push 50

stack is full

Linear list

Stack

10
20
30
40

( )

(10)
↑top

(10 20)
↑top

(10 20 30)
↑top

(10 20 30 40)
↑top

Delete

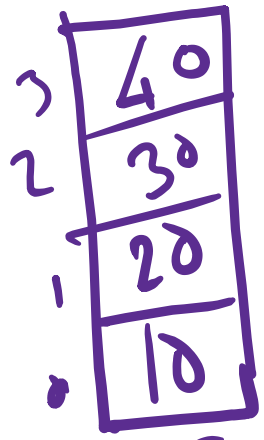
Element deleted is 40

(10 20 30)

Delete

(10 20)

Stack Overflow

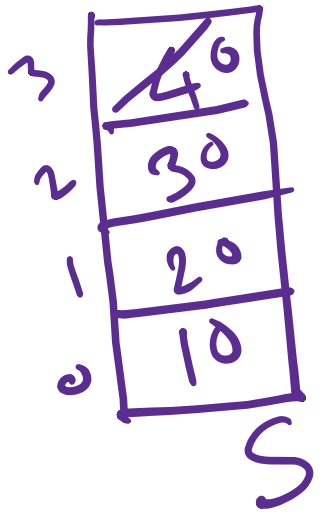


top=3

Pop

top = top - 1

top = 2

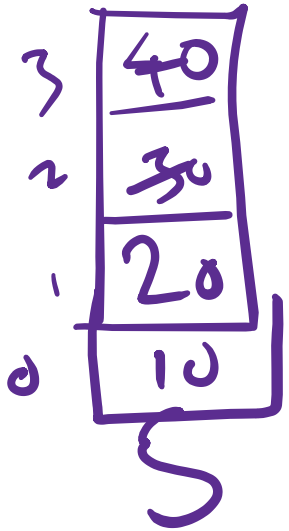


top=2

Pop

top = top - 1

top = 1

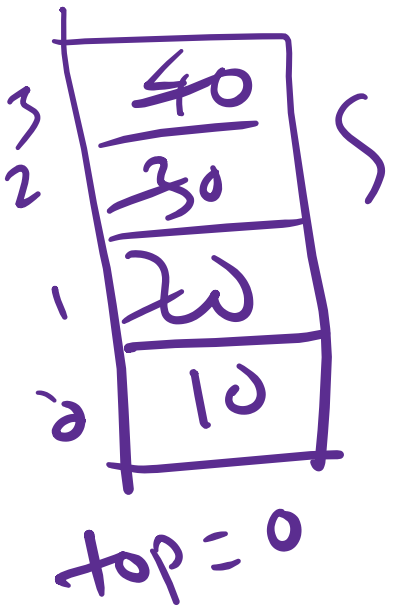


top=1

Pop

top = top - 1

top = 0



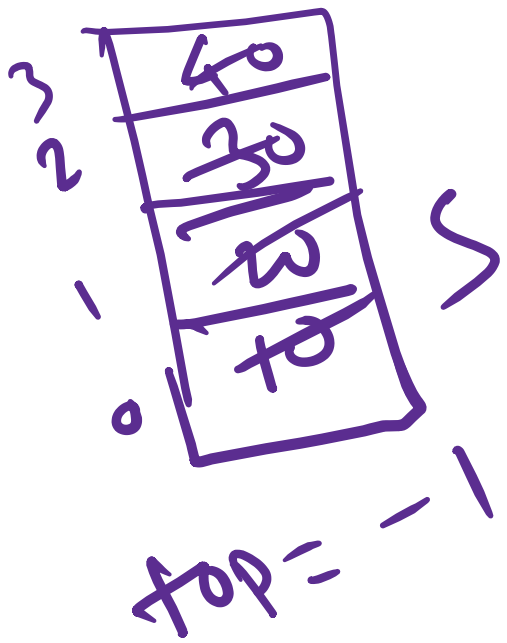
top=0

Stack from: 0 to top

Pop

top = top - 1

top = -1



Stack is empty

Pop

Stack underflow

If (top == -1)

then 'Underflow'

```
procedure push (S, MAXSIZE, top, element)

1. If (top == MAXSIZE - 1) Then
2.     Write('Stack Overflow')
3.     Return
4. Endif
5. top = top + 1
6. S[top] = element
7. Return
```

Constant time

O(1)

procedure pop(S, top) O(1)

```
1. If (top == -1) Then
2.     Write('Stack underflow')
3.     Return
4. EndIf
5. Write('Deleted Element:', S[top])
6. top = top - 1
7. Return
```

```
procedure display(S, top)                    { }
1. If (top == -1) Then                       N
2.     Write('Stack underflow')              ---
3.     Return                                N elements
4. EndIf                                     0 to N-1
5. Repeat step 6 for I=>1, 2, .., top        O(N)
6.     Write("The Element:", S[I])           N/2
7. Return
```

5 I = top

6. Repeat Step 7 and 8 while I >= 0

7. Write (S[I])

8. I = I - 1

9.

s : array

function: elements of s

where to declare s?

int s[20];

push()

pop()

display()

- globally ? — outside all functions
- — inside main: local to main
  - pass as arguments

```
t- main( ) {
    int s[20];

    push(s);

}
void push(        )
```

The C Programming Language

— Kernighan & Ritchie

```
#define <stdio.h>
#define maxsize 20

int s[maxsize], top;

int main(){
top=-1;
 push();
 push();
 pop();
 push();
 display()
   return 0; }
```

```
void push(){
 int element;
 print('Element:');
 scanf("%d",&elem
 if (top==maxsize-
   {
    printf('StackOve
    return;
   }
 s[++top]= element
 return; }
```

```
void pop() {
  if (top == -1)
      {
       printf("Stack underflow");
       return;
      }
  printf("Popped Element is: %d\n", s[top--]);

   return;
}
```

```
void display() {
    int i;
    for (i=0; i<=top; i++)
        printf("Element is: %d\n", s[i]);
    return;
}
```

```
for (i = top; i>=0; i--)
    printf. .....
```

push(s&top, element);

```
int main( ) {
int s[maxsize];
int top, element;
.
.
.
push (s, &top, element);
```

array is implicitly passed by reference

Queue: A queue is a linear list where all the insertions are done at one end of the list (known as rear) and all the deletions are done at the other end of the list (known as front). A queue exhibits FIFO property.

Insert : 10 20 30

( )

(10)

(10 20)

(10 20 30)

Delete: Deleted element is 10

(20 30)

Delete: (20 30)

Deleted element is 20

(30)

FIFO

Delete: Deleted element is 30

( )

Insertion Order: 10 20 30

Deletion " : ✓10 ✓20 ✓30

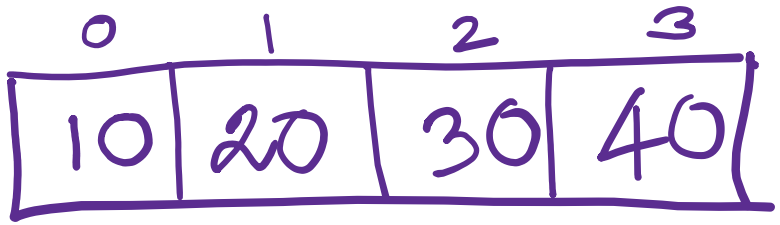
Implementation of Queue using Array

We use an array to keep the elements of the queue

We need two more variables to keep track of the 'rear' end and 'front' end.

We call these: REAR and FRONT

Q



Rear = 3, Front = 0

1) Insert 10 :

Rear = Rear + 1

Q[Rear] = 10

Front = Front + 1

2) Insert 20:

Rear = Rear + 1

Q[Rear] = 20;

Insert 30:

3) Rear = Rear + 1

Q[Rear] = 30

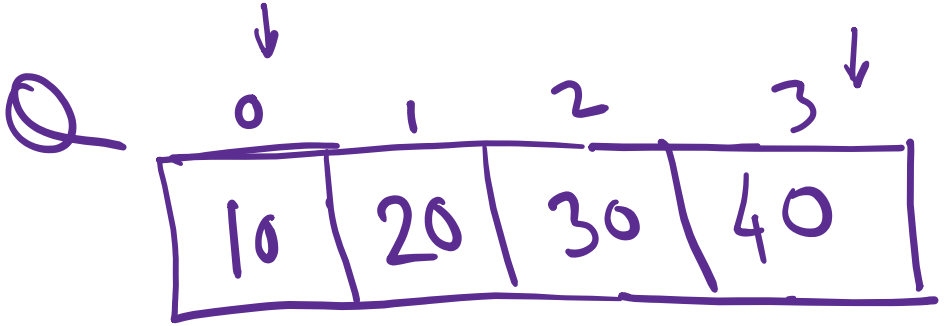
4) Insert 40:

Rear = Rear + 1

Q[Rear] = 40

5) Insert 50

Queue Overflow or Queue Full Cond^{n}.



Rear = 3, Front = 0

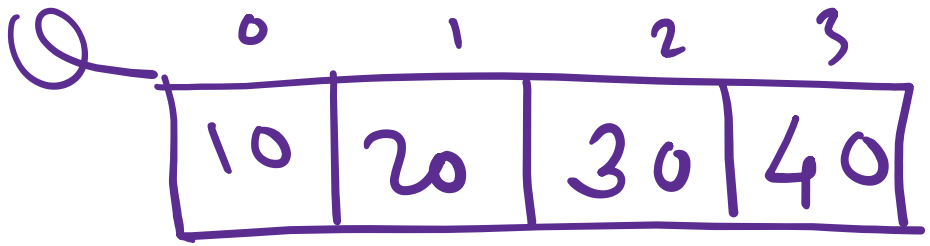
```
# define maxsize 4

int Q[maxsize];
```

If (Rear == maxsize - 1)

Deletions: 1) Delete

Front = Front + 1



Rear = 3, Front = 3

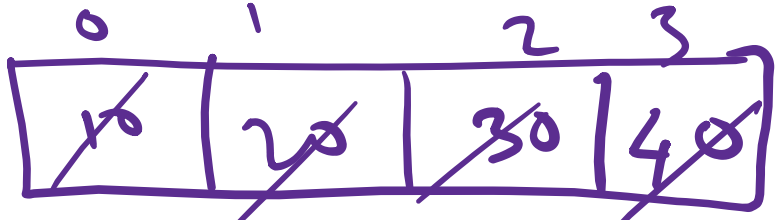
Queue is always from index 'Front' to index 'Rear'

2) Delete: Front = Front + 1

20 is deleted

3) Delete: Front = Front + 1

30 is deleted

Q 

Rear = 3, Front = 3

40 is deleted

Rear = -1 Front = -1

Deletion of only element in queue

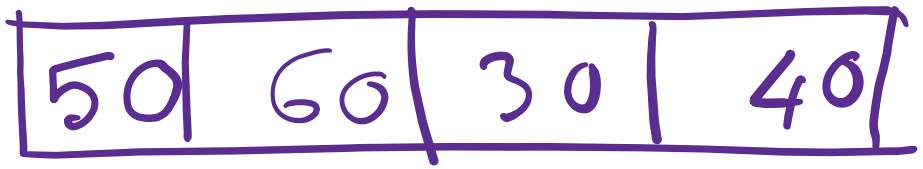
5) Delete x : Queue Underflow or Queue Empty Conditia

Circular Queue:

maxsize = 4

Initially,
Rear = -1
Front = -1

Q



Rear = 1, Front = 2

Delete: Q[Front] or Q[0]

Delete Q[1]

Insert 50

Insert 60

Circularly Updated

modulus (%) operator

$$Rear = (Rear + 1) \bmod maxsize$$

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

$$= 4 \% 4$$

$$= 0$$

$$Rear = (Rear + 1) \% maxsize$$

$$= (0 + 1) \% 4$$

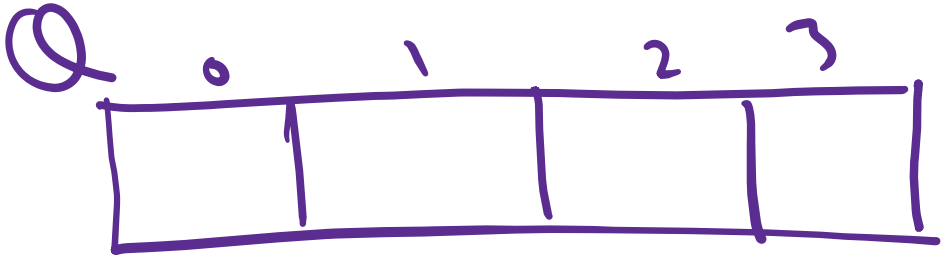
$$= 1 \% 4$$

$$= 1$$

Circular Queue Insertion and Deletion

Insert 10, 20, 30, 40

1)



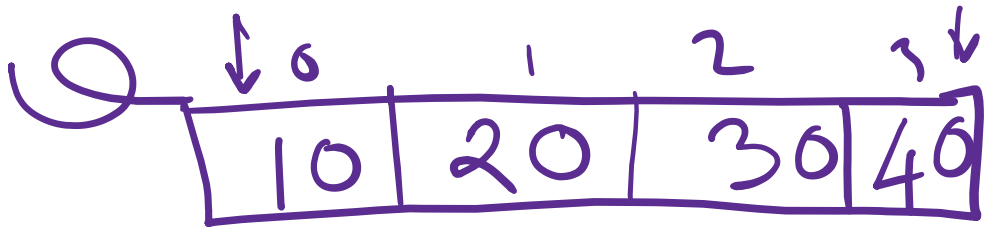
Rear = -1, Front = -1

2) Insert 10

Rear = (Rear + 1) % 4

= (-1 + 1) % 4

= 0



Rear = 3, Front = 0

3) Insert 20

Rear = (0+1) % 4

= 1

4) Insert 30

Rear = (1+1) % 4

= 2

5) Insert 40

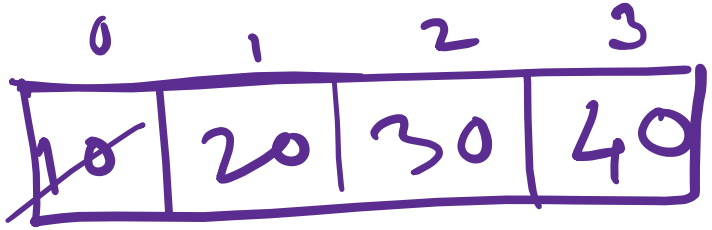
Rear = (2+1) % 4

= 3

If (Front == 0)

&& (Rear = maxsize

6) Delete :

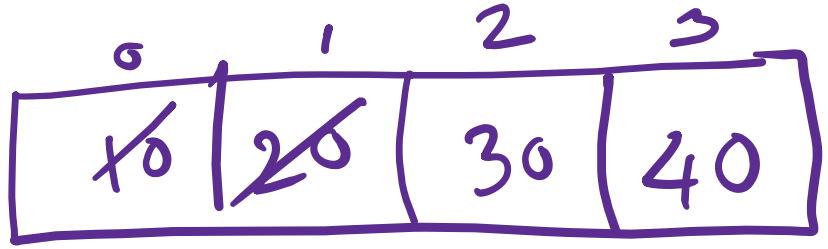


Rear=3, Front=1

$$Front = (Front + 1) \% maxSize$$
$$= (0 + 1) \% 4$$
$$= 1 \% 4$$
$$= 1$$

7) Delete :

$$Front = (1 + 1) \% 4$$
$$= 2$$



Rear=3, Frnt=2

7) Insert 50

Q

0	1	2	3
50	~~20~~	30	40

Rear = 0, Front = 2

Rear = (3 + 1) % 4

= 0

Q[Rear] = 50

8) Insert 60

Q

0	1	2	3
~~50~~	~~60~~	~~30~~	~~40~~

Rear = -1, Front = -1

Rear = (0 + 1) % 4

= 1

Q[Rear] = 60

Delete

Delete

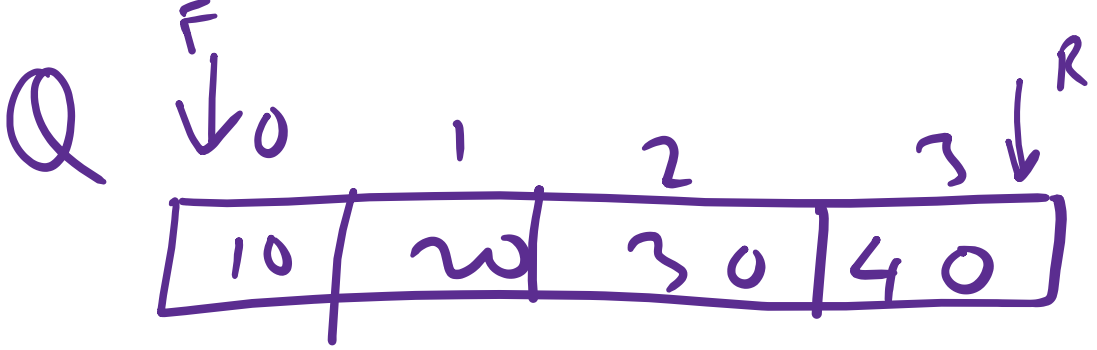
Delete

Delete

Front = -1

Rear = -1

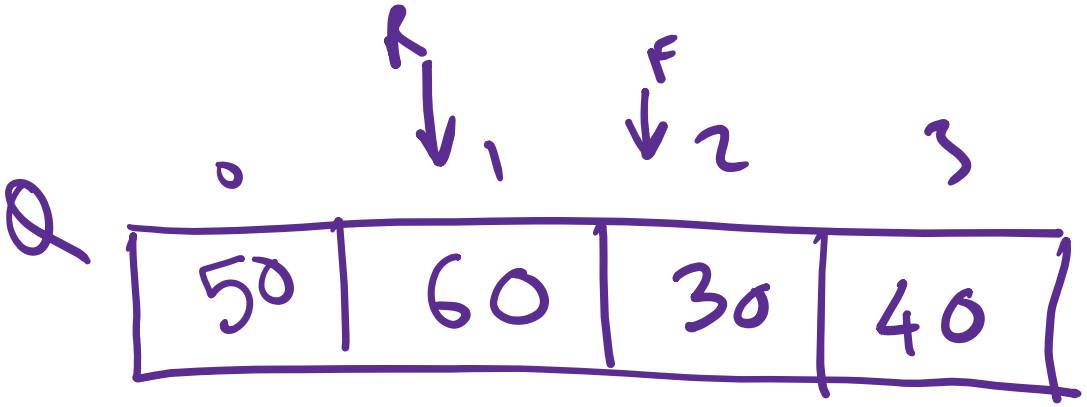
Circular Queue



Rear = 3, Front = 0

(4 % 4)

((Rear + 1) % maxsize) == Front



Rear = 1, Front = 2

0 == 0 True

Normal Queue: Insert: enqueue
Delete: dequeue

procedure Enqueue(Q, Rear, Front, maxsize)

1. If (Real == maxsize - 1) Then
2. Write('Queue Overflow')
3. Return
4. EndIf
5. Rear = Real + 1

```
6.  Q[Rear] = element
7.  If (Front == -1) Then
8.      Front = 0
9.  EndIf
10. Return
```

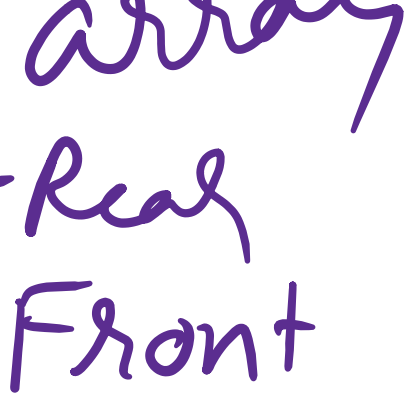
```
Procedure Dequeue(Q, Rear, Front)
1. If (Front == -1) then
2.     Write('Queue Underflow')
3.     Return
4  Endif
5. Write('Element to be deleted:', Q[Front])
6. If  (Front == Rear) then
7.     Front = Rear = -1
```

8. Else

9. Front = Front + 1

10 EndIf

11. Return

C Implementation:- Global 

- One function for Enqueue
- One " " Dequeue
- One " " Display

- Declare the array, rear, front locally within main

— Pass the array, rear, front as arguments.

&rear
&front