CodeCraft Academy
9359164822

M T W T F S S
Page No.:
Date:
YOUVA

2. Queues

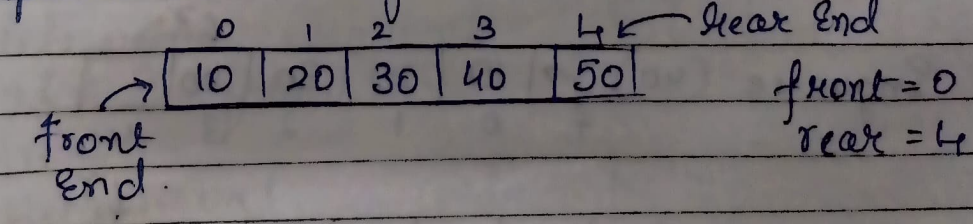
The Concept:

1. People standing outside the ticketing window of a cinema hall.
The first person in the line will get the ticket first & thus will be the first one to move out of it.

2. People moving on an escalator. The people who got on the escalator first will be the first one to step out of it.

- ✓ Stores data in an ordered manner.
- ✓ It is a FIFO Data Structure.
  → First In First Out.
- ✓ Elements added from rear end.
- ✓ Elements are removed from front end.

ARRAY Implementation of Linear Queue.



In Stack we use one variable top.
where as In Queue we use
front Variable for front end.
rear Variable for rear end.

CodeCraft Academy
9359164822

M T W T F S S
Page No.:
Date:
YOUVA

Operations On Queues

1. Insertion / Enqueue.
2. Deletion / Dequeue.
3. traversing / Display.
4. peek.

Note. For Queues. Array should be used in horizontal Manner.

1. Insertion / Enqueue.

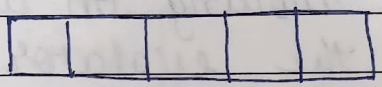
→ Adding new element to the queue.

① **Initial Scenario.**

MAX = 5

front = ~~Empty~~ rear = -1

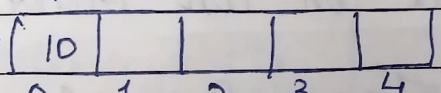
(f) (r)



r++;
q[r] = val;

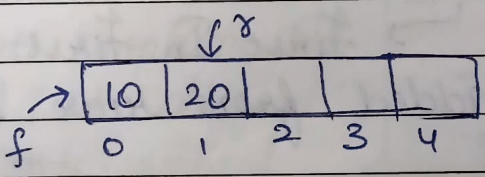
f = r = -1 ← Queue is Empty.

② Enqueue (10)



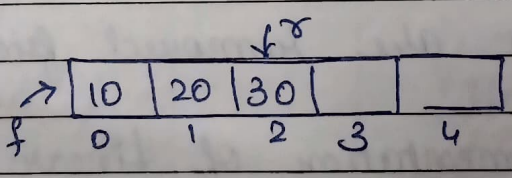
r++;
q[r] = 10

③ Enqueue (20)



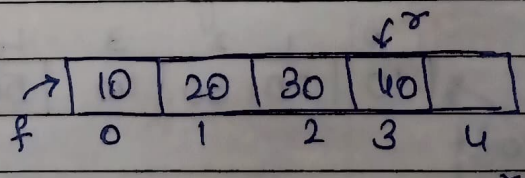
r++
q[r] = 20

④ Enqueue (30)



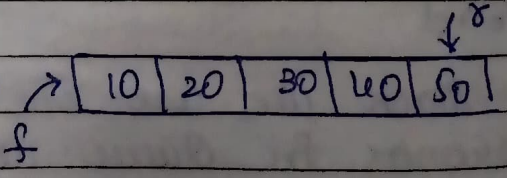
r++
q[r] = 30

⑤ Enqueue (40)



r++
q[r] = 40

⑥ Enqueue (50)



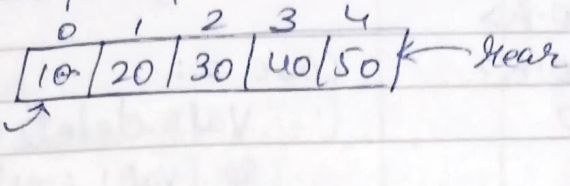
r++
q[r] = 50.

⑦ Enqueue (60) ⟶ Overflow Condition

∴ rear == MAX-1 is Overflow Condition.

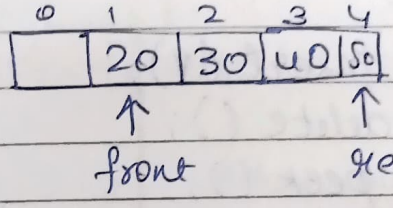
CodeCraft Academy
9359164822

3) Deletion / Dequeue.

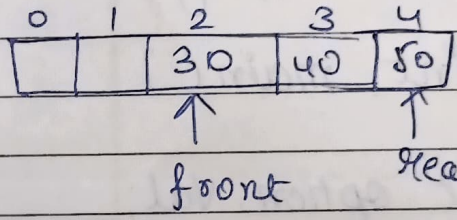


① Queue Empty Condition 1. if (front == -1)

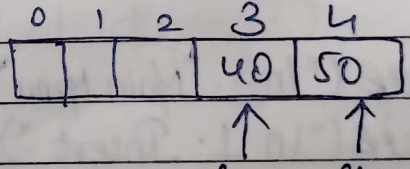
② delete one element.
Val = q[front]
= 10.
front++
Return val.



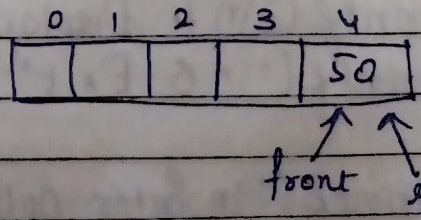
③ delete one element.
val = q[front] = 20.
front++
Return val



④ delete one element
Val = q[front] = 30
front++;
Return Val;

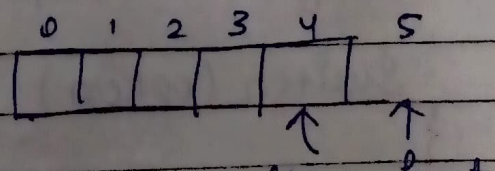


⑤ delete one element
Val = q[front] = 40
front++;
Return Val;



Condition for 1 ele.
if (front == Rear)

⑥ delete one element
Val = q[front] = 50
front++;
Return Val;



front > Rear
2nd Condition for queue to be Empty.

Codecraft Academy
9359164822

Q: WAP to implement a Linear Queue:

```
#Include <stdio.h>
#Include <conio.h>
# define MAX 10

int queue [MAX];
int front = rear = -1;

Void Insert (#);
int delete ();
int peek ();
Void display ();

int main ()
{
int option, Val;
do
{
printf ("In --- Main Menu ---- ");
printf ("In 1. Insert ");
printf ("In 2. Delete ");
printf ("In 3. Peek ");
printf ("In 4. Display ");
printf ("In 5. Exit");

printf ("In Enter option:");
Scanf ("%d", & option);

Switch (option)
{
case 1:
        Insert ();
        break;
case 2:
        Val = delete ()
        if (val != -1)
        pf ("In del. ele = %d", Val);
        break;

case 3:
        Val = peek ();
        if (val != -1)
        pf ("In 1st ele in queue %d",
                                Val);
        break;

Case 4:
        display ();
        break;
} // switch ends

} while (option != 5);
return 0;
}
```

CodeCraft Academy
9359164822

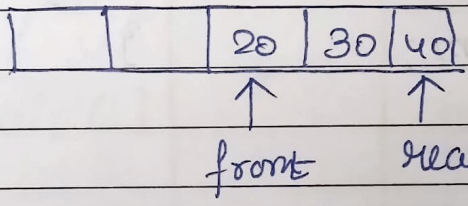
```
Void insert()
{
  int num;
printf("\n Enter the no:");
Scanf("%d", &num);

 if (rear==(MAX-1))
 printf("\n Overflow");
 else if ( front==-1 && rear==-1)
          {frbnt = rear=0;}
        else
          {rear++;}
    queue[rear] = num;
}

int delete()
{
  int val;
if (front==-1 || front > rear)
  {
   printf("\n Underflow");
     return -1;
  }
else
    {
      val = queue[front];
      front++;
     if (front > rear)
       front = rear=-1;
       return val;
     }
}

int peek()
{
if (front==-1 ||
          front>rear)
  {
   printf("\n Q is Empty");
    return -1;
  }
 else
   {
   Return queue[front];
   }
}

Void display()
{
  int i;
 printf("\n");
if(front==-1 || front>rear)
printf("\n Q is Empty");
else
{
for(i=front; i<=rear; i++)
 printf("\t%d", queue[i]);
}
}
```

Application of Linear Queue

✓ Linear queues are used in applications where data elements need to be processed in the order in which they are received.

✓

Limitation of Linear Queue



Though the queue is empty still we cannot insert an element in the queue.

Here it will show Overflow Condition
rear = MAX-1 as true.

Hence Circular Queue.

* The queue data structure can be classified into the following types:

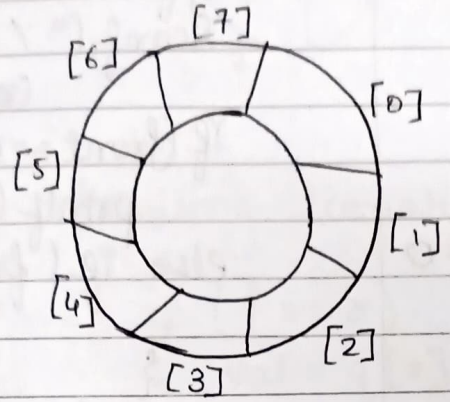
1. Circular Queue
2. Deque
3. Priority Queue
4. Multiple Queue.

CodeCraft Academy
9359164822

* Circular Queues.

- The first index comes right after the last index.

1) Initially

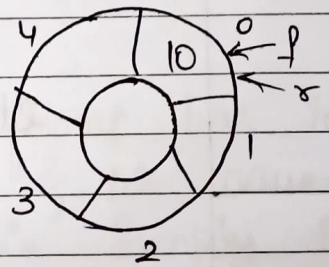


MAX = 8
front = -1
rear = -1
q[MAX].

2) Insertion

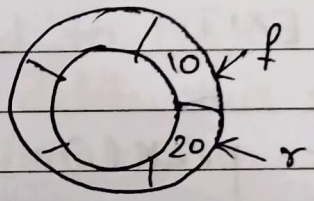
2.1) Insert 10 [1st ele]

```
if (r == -1 && f == -1)
    f = r = 0.
q[r] = val;
```



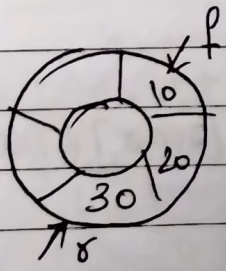
2.2) Insert 20

```
r++
q[r] = 20
```



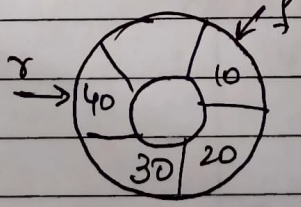
2.3 Insert 30.

```
r++;
q[r] = 30
```



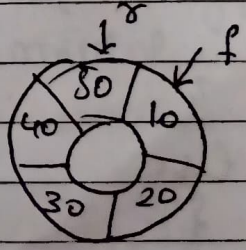
2.4) Insert 40

```
r++;
q[r] = 40;
```



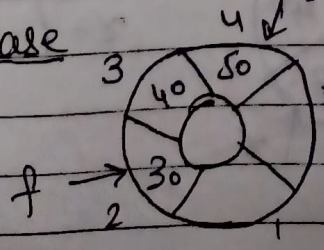
2.5) Insert 50.

```
r++;
q[r] = 50
```



∴ Full condition / overflow condition :- rear = MAX-1 and front = 0

Special case



Here
r = MAX-1
but f != 0.
in that case just do r=0

∴ r=0
q[r] = val.

Codecraft Academy
9359164822

M T W T F S S
Page No.:
Date:
YOUVA

Algorithm

1. If F == 0 & R = MAX-1
   write "Overflow"
   Goto step 4

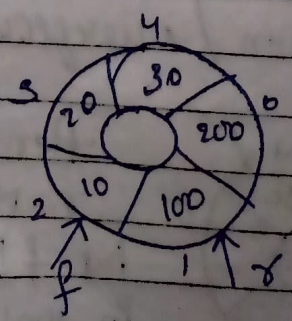
2. ~~Step~~ If F = -1 & R = -1
   F = R = 0
   ELSE If R = MAX-1 & F != 0
   R = 0
   ELSE
   R = R+1

3. Q[R] = VAL
4. EXIT

```
int peek()
{
if (front == -1 && rear == -1)
  {
Pf("In Queue is Empty");
   return -1;
  }
else
  {
   return queue [front];
  }
}
```

Function

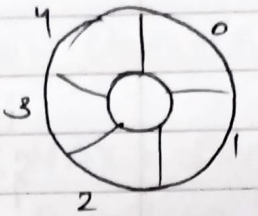
```
Void insert()
{ int num
 pf("Enter ele to Insert");
 Scanf("%d", &num);
          (rear = front-1) ;-;
if (front == 0 && rear == MAX-1)
    printf("In Overflow");
else if (front == -1 &&
                rear == -1)
  {
    front = rear = 0;
    queue [rear] = num;
  }
else if (rear = MAX-1 &&
                front != 0)
  {
    rear = 0;
    queue[rear] = num;
  }
else
  {
    rear ++;
    queue [rear] = num;
  }
}
```



2nd overflow condition.
r = f - 1

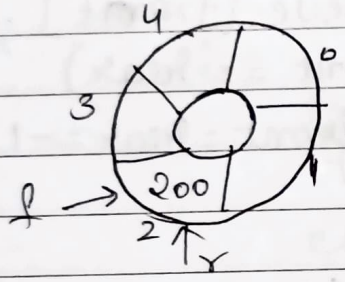
2] Deletion Operation On Circular Queue.

1) Initial MAX=5
f=-1;
r=-1;
q[MAX]



∴ Underflow Condition / Queue Empty
f = r = -1.

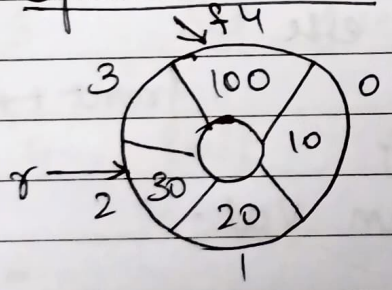
2) Consider the following scenario :- Only 1 ele in Queue



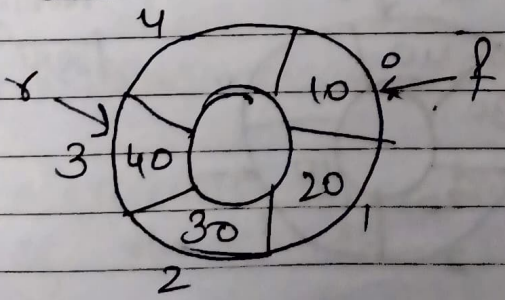

```
if (f == r)
{ val = q[r];
  f = r = -1.
}
```

If there is only 1 ele in Queue, store the value in a variable set f = r = -1

3) Special case :- front has reached to MAX-1 & we have to delete this ele.
∴ Simply set front = 0.



4) Normal deletion :-



delete element 10
∴ val = q[f];
f++

Codecraft Academy
9359164822

M T W T F S S
Page No.
Date
YOUVA

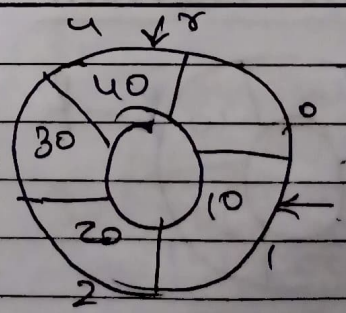
ALGORITHM

1. If FRONT = -1
   Write "Underflow"
   Goto step 4
2. val = Queue[FRONT]
3. If FRONT = REAR
   FRONT = REAR = -1
   Else
   If FRONT = MAX-1
   SET FRONT = 0
   ELSE
   SET FRONT++;
4. EXIT

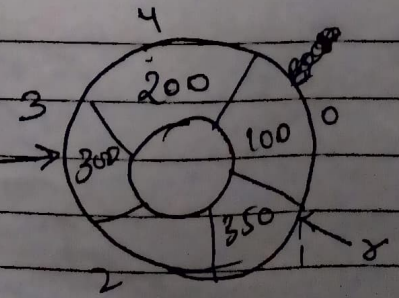
```
Void display()
{
  int i;
printf("\n");
if(front == -1 && rear == -1)
    pf("\n Queue is Empty");
else
 {   if (front < rear)
    {
     for(i=front; i<=rear; i++)
         printf("\t%d", queue[i]);
else
 {
 for(i=front; i<=MAX; i++)
     pf("\t%d", queue[i]);
for(i=0; i<=rear; i++)
     pf("\t%d", queue[i]);
 }
```

```
int delete()
{
 int val;
if (front == -1 && rear == -1)
{
printf("\n Underflow");
 return -1;
}
val = queue[front];
if (front == rear)
      front = rear = -1;
else
 {
    if (front == MAX-1)
          front = 0;
    else
          front++;
 }
return val;
}
```

1) Normal case front < rear



2) front > rear



Codecraft Academy
9359164822

Circular Queue Updated

1) Insertion / Enqueue Operation

1° if (rear + 1) % MAX = FRONT
write "Overflow"

2° if front = -1 & Rear = -1 // 1st ele.
Set front = rear = 0
else
if rear = MAX-1 & front != 0
rear = 0;
else
rear = (rear + 1) % MAX

3. Set queue [rear] = val

4. Exit.

2°

* DEQUE

- ✓ pronounced as 'deck'
- ✓ elements are inserted or deleted from either of the end.
- ✓ No element can be added or deleted from the middle.
- ✓ In computer's memory, a deque is implemented using either a circular array or a circular doubly linked list.

✓ They are classified into two types.

1. Input Restricted Deque.
2. Output Restricted Deque.

✓ INPUT Restricted Deque.

→ Operations.

1) Insertion of an element at the Rear end.
2) Deletion of an ele. from both ends.

✓ Output Restricted Deque.

→ Operations.

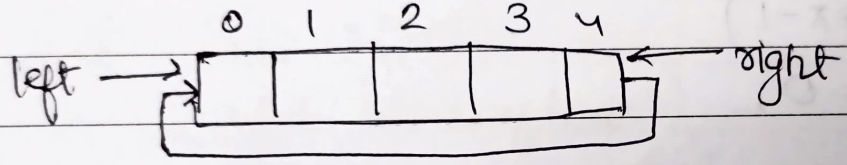
1) Insertion of an element from both ends
2) Deletion of an element from front end.

* Operations Associated with Deque.

a) empty() :- i.e. Underflow / Empty queue condn.
b) full() :- i.e. Overflow condition.
c) enqueueR() :- Add item to the Rear end.
d) enqueueF() :- Add item to the Front end.
e) dequeueR() :- Delete item from the Rear End
f) dequeueF() :- Delete item from the front End.

M T W T F S S
Page No.
Date
YOUVA

Input Restricted Deque



Imagine the queue as above.

Operations.

1) insert_right
2) delete_left
3) delete_right

Output Restricted Deque.

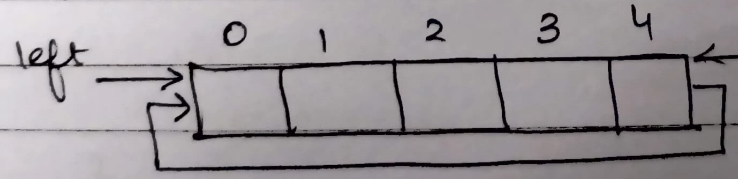


Image the queue as above

Operations:

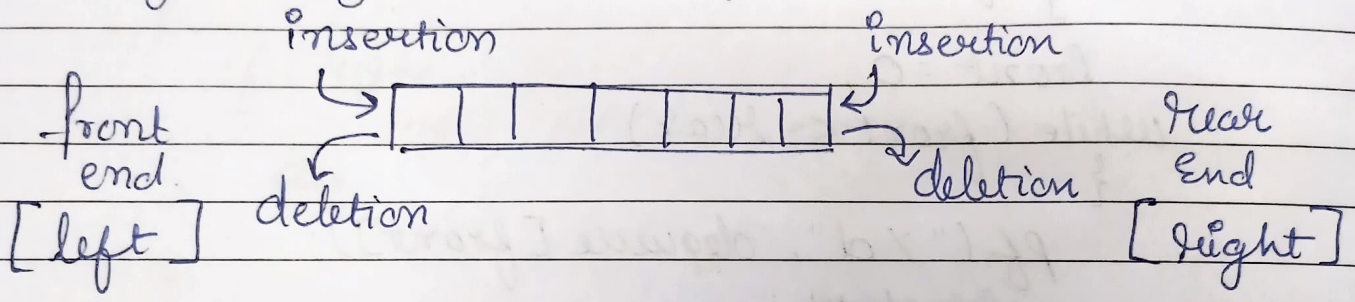
1) insert_right
2) insert_left
3) delete_left.

CodeCraft Academy
9359164822

M T W T F S S
Page No.:
Date:
YOUVA

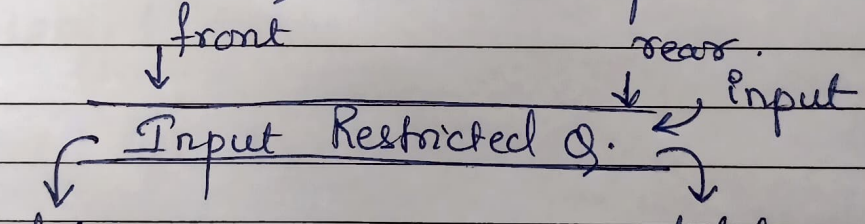
Double Ended Queue [deck].

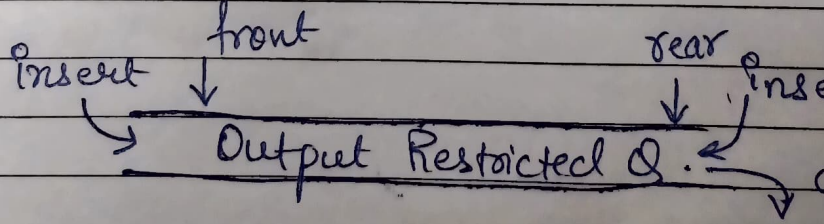
- ✓ Linear data structure.
- ✓ Insertion & deletion operations are performed from both ends.
- ✓ Generalized version of the queue.



Types of deque

1) Input Restricted Queue 2) Output Restricted Queue:





Operations to be performed on deque

1. Insertion at the front.
2. Insertion at the rear. (Same as linear queue)
3. Deletion at front (" " ")
4. Deletion at rear &

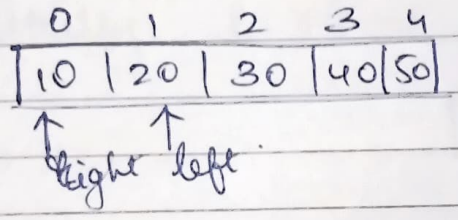
CodeCraft Academy
9359164822

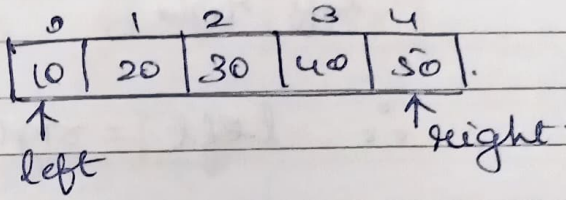
M T W T F S S
Page No.
Date
YOUVA

* Insertion at the front end.

Case 1: Full Queue

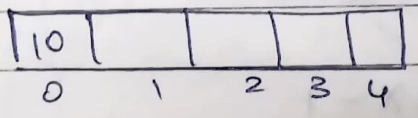
left = right + 1 left = 0 & right = MAX-1.





Case 2: Queue is Empty.

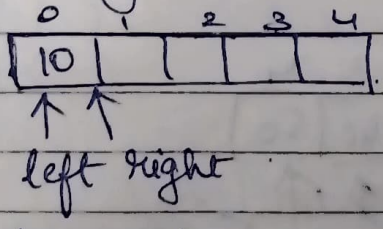
left = right = -1 ← Empty Queue

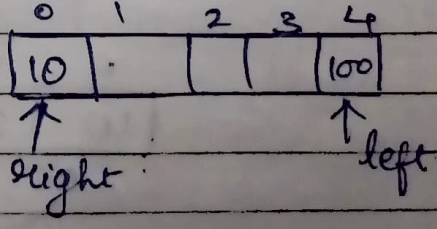


∴ left = right = 0

q[left] = val (say 10)

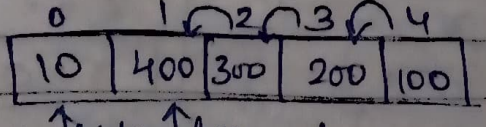
Case 3: Only 1 element.





∴ left = MAX-1 ∴ left = 4.

Case 4: Normal Insertion



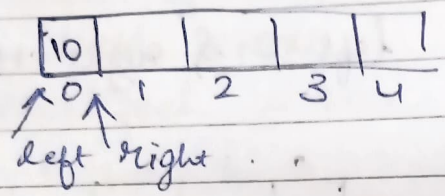
left --	left --	left --
q[left] = 200	q[left] = 300	q[left] = 400

Codecraft Academy
9359/64822

M T W T F S S
Page No.:
Date:
YOUVA

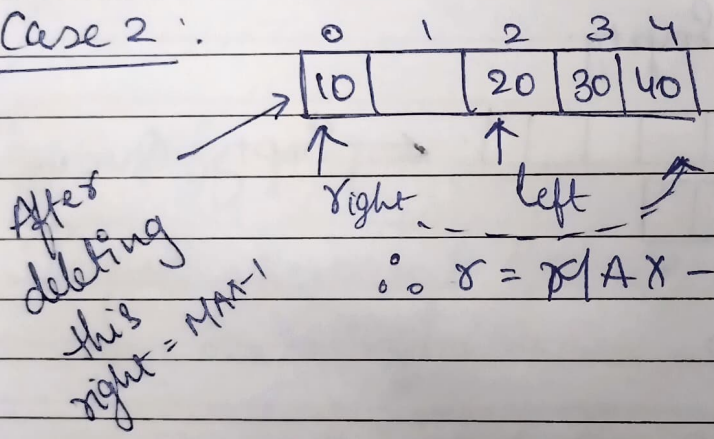
Deletion from Right End

Case 1: only 1 element.

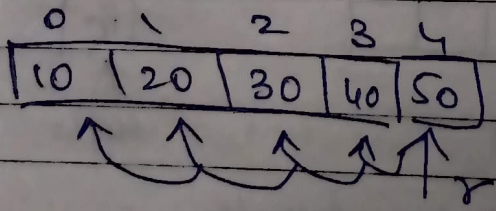


∴ left = right = -1;

Case 2:



Case 3: Normal deletion from rear/ end.
right.



r = 4.

delete(50)	delete(40)	delete(30)	delete(20)	delete(10)
r--	r--	r--	r--	r--
r = 3	r = 2	r = 1	r = 0	r = -1

Codecraft Academy
9359164822

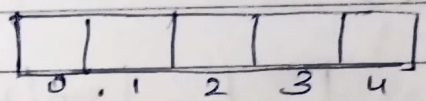
Input Restricted Queue.

Operations which can be Performed.

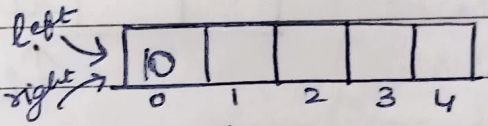
1) insertR() 2) deletel() 3) deleteR()

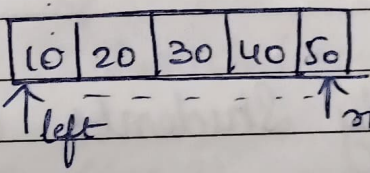
Operations are called in the following manner.

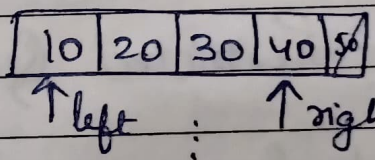
Initially l = r = -1 Max = 5 deque[MAX].

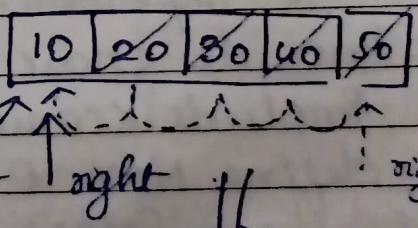


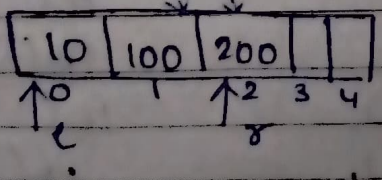
1) insertR(10)
insertR(20)
insertR(30)
insertR(40) insert(50)
deleteR()
deleteR()
deleteR()
deleteR()
insertR(100)
insert(200)
insert(300)
deletel()
deletel()

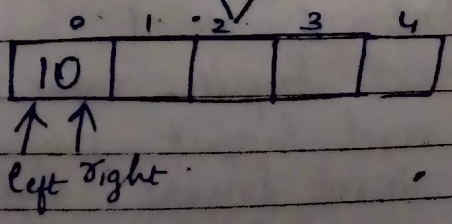


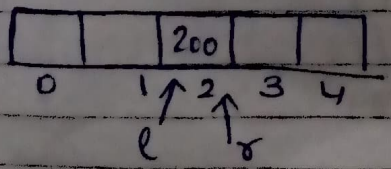




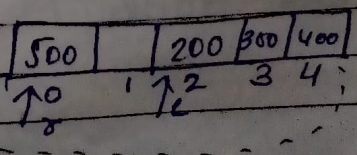








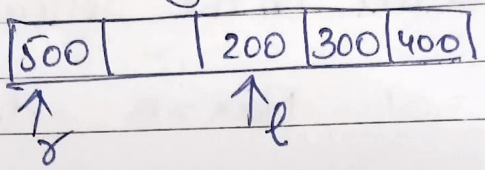
Now Perform 3 times insertR ---> 500 200 300 400



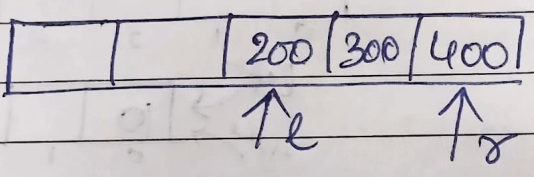
Page No:
Date:
YOUVA

Now, if we perform deleteR()
r which is pointing to 0th index will be pointing to MAX-1 next.

Initially



after deleteR()



Similarly Students can perform operations on Output Restricted Queue.

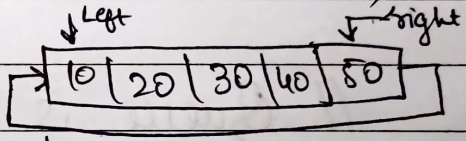
WAP to implement Input & Output Restricted deques.

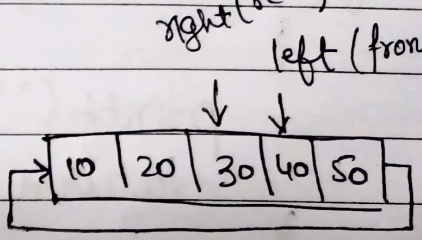
```
#Include <stdio.h>
#Include <conio.h>
#define MAX 10.
int deque [MAX];
int f=r=-1;
void Input_deque();
void Output-deque();
void insert_right();
void insert-left();
void delete_right();
void delete-left();
void display();
int main()
{ int option;
  clrscr();
Pf("... MAIN MENU...");
Pf(" -- 1. Input Restricted");
Pf("... 2. Output Restricted");
 Pf(" Enter option");
sf("%d", & option");
Switch (option)
 {
case 1: input-deque();
        break;
case 2: Output-deque();
        break;
}//switch
}// main.
```

```
Void Input-deque()
{
 int option;
do
{ pf("-- Menu --- ");
    ("-- 1. Insert Right");
    (" 2. Delete left");
    (" 3. Delete Right");
    (" 4. Display");
    (" 5. Quit");
    (" Enter option");
sf("%d", &option);
Switch (option)
 {
  case 1:
       insert-right();
       break;
  case 2:
       delete-left();
       break;
  case 3:
       delete-right();
       break;
  case 4:
       display()
       break;
   }
 } while (option!=5);
}
```

```
Void output-deque()
{
 int option;
 do
 {
  pf("---MENU---");
    ("-- 1. Insert Right");
    ("--- 2. Insert Left");
    ("--- 3. Delete left");
    ("---4. Display");
    ("--- 5. Quit");
    ("--- Enter option");
 sf("%d", &option)
  Switch (option)
  {
  case 1:
       insert-right();
       break;
  case 2:
       insert-left();
       break;
  case 3:
       delete-left();
       break;
  case 4:
       display();
       break;
  }
 }while (option!=5);
}
```

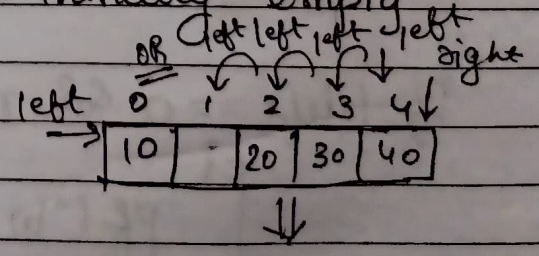
```
Void insert_right()
{
    int val;
    Pf("Enter value to be added");
    Sf("%d", &val);
```






```
    if (left == 0 && right == MAX-1) || (left == right+1))
    {
        Pf("In overflow");
        return;
    }
    if (left == -1) // Queue is initially empty
    {
        left = right = 0;
    }
```

```
else
{
    if (right == MAX-1)
        right = 0;
    else
        right ++;
}

deque[right] = val;
}

Void insert_left()
{
  Int val;
printf("\n Enter the Value to be added");
 scanf("%d", &val);

if( left == 0 && right == MAX-1) || (left == right+1))
  {
     printf("\n overflow");
      return;
  }
if ( left == -1)    // Queue is initially empty
  {
     left = right = 0;
  }
else
  {
     if (left == 0)   // only 1 element
          left = MAX-1;
     else
          left = left-1;
  }
deque [left] = val;
}
```



```
Void delete_left()
{
    if (left == -1)
      { Pf("Underflow");
        return;
      }
printf("The deleted ele is: %d", deque[left]);
if (left == right) // Queue has only one element
    {
        left = -1;
        right = -1;
    }
else
    {
        if (left == MAX-1)
                left = 0;
        else
                left = left+1;
    }
}

Void delete_right()
{
    if (left == -1)
      {
        Pf("In Underflow");
        return;
      }
printf("In The ele deleted is: %d", deque[right]);
}
```

```
if (left == right) // deque has only one ele
{
    left = -1;
    right = -1;
}
else
{
    if (right == 0)
        right = MAX-1;
    else
        right = right -1;
}
}
```

Display

```
Void display ()
{
int front = left, rear = right;
if (front == -1)
{
    printf("\n Queue is Empty");
    return;
}
printf("\n The element of the queue are:");
    if (front <= rear)
    {
        while (front <= rear)
        {
            printf("%d", deque[front]);
            front ++;
        }
    }
```

```
else
  {
    while (front<=MAX-1)
      {
        printf("%d", deque[front]);
        front++;
      }
    front=0;
    while (front<=rear)
      {
        pf("%d", dequeue[front]);
        front++;
      }
  }
printf("\n");
}
```

CodeCraft Academy
9359164822

M T W T F S S
Page No:
Date:
YOUVA

Priority Queue

- ✓ Arranges elements based on their priority values
- ✓ Elements with higher priority values are typically removed or retrieved before elements with lower priority values.
- ✓ Each element has a priority value associated with it. When we add an item, it is inserted in a position based on its priority value.
- ✓ General rules of processing.
  - An element with higher priority is processed before an element with a lower priority.
  - Two elements with the same priority are processed on a First-come-first-served (FCFS) basis.
- ✓ Types of Priority Queue:- Ascending & Descending.
- ✓ Application of Priority Queue. priority Queue
  - CPU Scheduling
  - Data Compression in Huffman code.
  - Event driven simulation such as Customer Waiting queue.
  - Finding kth largest / smallest ele.
  - All queue applications where priority is involved.
  - Graph algo. like Prim's Minimum Spanning tree, Dijkstra's shortest path algo.

Advantages: 1) Helps access the elements faster way.
2) Dynamic in Nature.
3) Efficient.
4) Included in real-time systems.

Disadvantages:- 1) High Complexity
2) High consumption of memory.

Questions

1. Define Linear Queue? Give its applications.
2. WAP to perform basic operations on Linear Queue using Arrays. OR. Array Implementation of Linear Que.
3. Give limitation of Linear Queue.
4. WAP to calculate no. of items in a queue.
5. What is a priority queue? Give its applications
6. Explain the concept of Circular Queue. How is it better than a Linear Queue?
7. WAP to Implement Circular Queue using Arrays.
8. Give applications of Circular Queue.
9. WAP to Implement Input & Output restricted deque.