ॐ

Unit-2

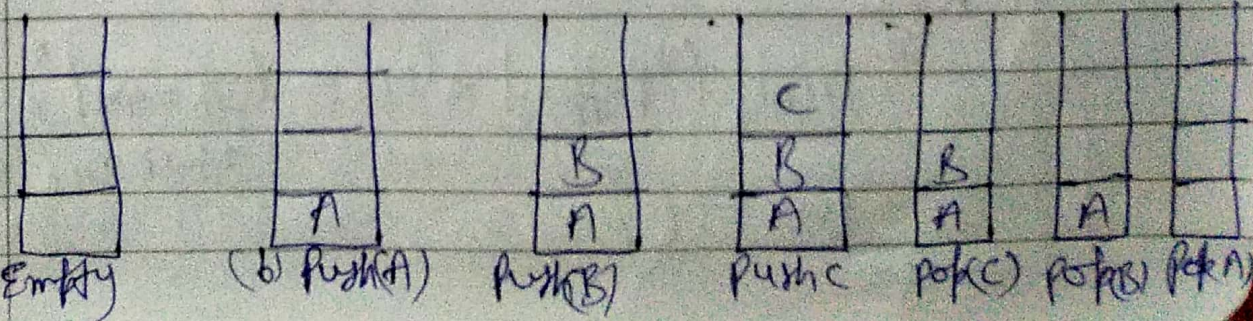
Stack :- Stack is defined as a list of elements in which we can insert or delete the element only at the top of the stack. Suppose we can see that we can take CD from the CD stand only from top. Stack is called a last in first out (LIFO) collection of data. Stack is useful to show the history of web browser, match the Parentheses in the calculations, and also apply for a program which use function in side of functions etc.

Array implementation of Stack :-

Operations on Stack :- we can perform two basic operation on stack

(1) Push ( ) function —
(2) Pop ( ) "

(1) Push function- Push function is used to insert the elements in the stack. But before insert the element we have to check that stack is overflow or not.



```
push ()
{
    int pushed_item;
    if (top == MAX-1) //overflow Condition
      printf("stack overflow");
    else
    {
      printf("Enter the item to be pushed");
      scanf("%d", &pushed_item);
      top = top+1;
      stack_arr[top]
      Stackarr[top] = pushed_item;
    }
```

Pop operation on Stack

Pop operation on stack is used to delete the elements from the stack. for pop operation on stack first we check the condition of underflow then pop the element.

```
pop ()
{
    if (top == -1) // underflow Condition
      printf("Stack is underflow\n");
    else
    {
      printf("Popped element is = %d",
                                  Stackarr[top]);
      top = top-1;
    }
```

Program of Stack using array

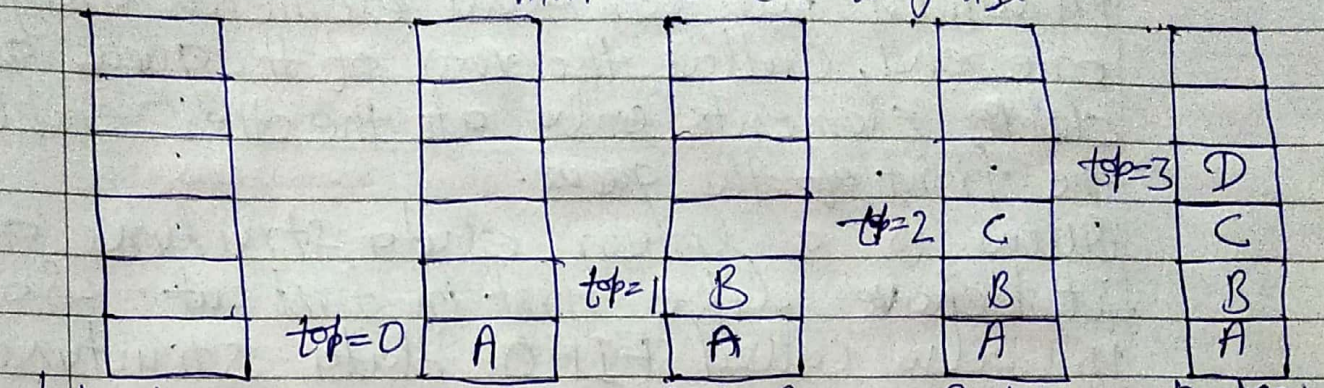
```
#include <stdio.h>
#include <conio.h>
#define MAX 5
int top = -1;  void push(); void pop();
int stack[MAX]; void display();
void main()
{
   int choice
   while(1)
   {
    printf("Enter your choice \n");
    printf(" 1. Push \n");
    printf(" 2. pop \n");
    printf(" 3. Display \n");
    printf(" 4. Quit \n");
    printf(" Enter Your choice \n");
    scanf("%d", &choice);
switch(choice)
   {
      case 1:
         Push();
         break;
      Case 2:
         pop()
         break;
      case 3:
         display()
         break;
      case 4:
         exit(1);
      default:
      printf("\n Enter Correct
                  Choice");
   } // End of switch.
} // End of main.
```

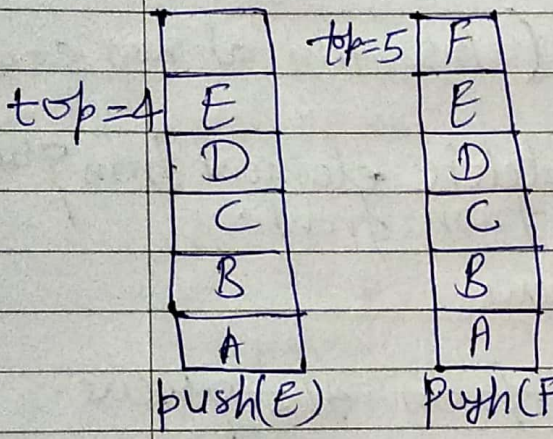
```
void push()
{
    int pushed item;
    if(top == MAX-1)
      printf(" stack overflow \n");
    else
    { printf(" Enter value of pushed item \n");
      scanf(" %d ", &pushed item);
      top = top+1;
       stackarr[top] = pushed item;
    }
}

void pop()
{
    if (top == -1)
      printf (" stack is underflow ");
    else
    {
      printf(" popped item is = %d ", stackarr[top]);
      top = top-1;
    }
}

void display()
{ int i;
  if(top == -1)
  printf(" \n stack is empty ");
  else
  { printf(" stack elements : \n ");
    for(i=top ; i>=0 ; i--)
     printf(" %d \n ", stackarr[i] );
  }
}
```

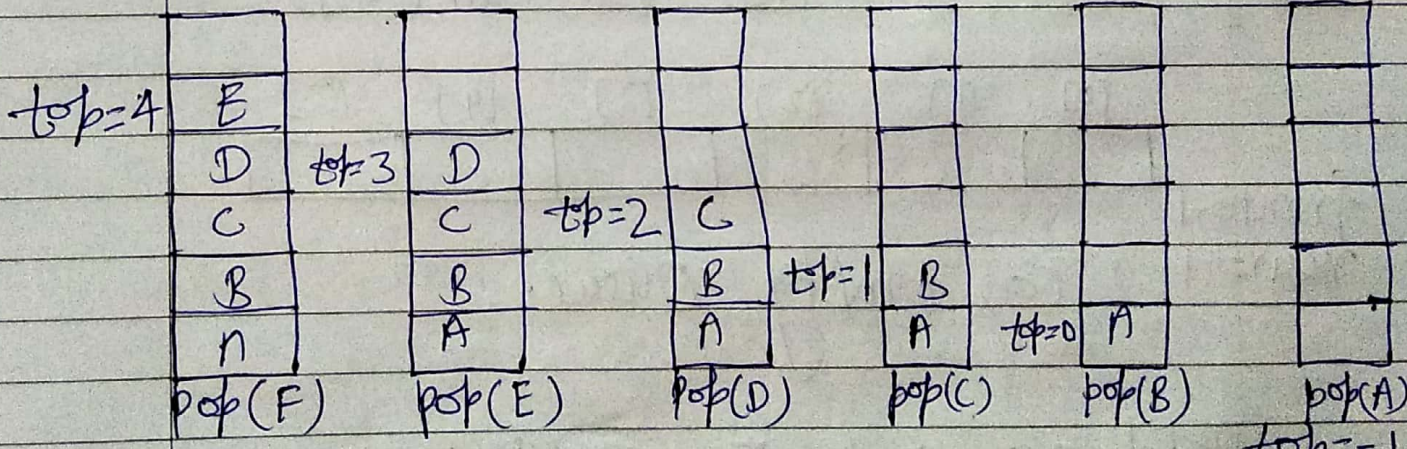
Ex- of stack - Stack size is 6 ie 0 to 5.
MAX = 6 ie array size.





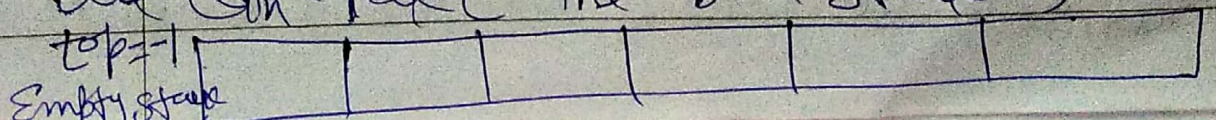
Now stack is full means overflow. Now we can not add element in the stack because top==MAX-1.

So the overflow condition is
if (top==MAX-1)
printf("stack is overflow");



Now stack is underflow because value of top==-1.
So underflow condition is if(top==-1)
printf("stack underflow");

We can take the above stack as



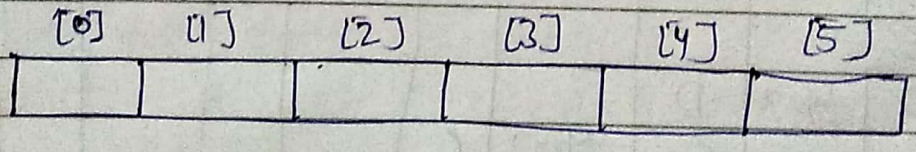
Queue or (simple Queue)

Queue is an ordered list of elements in which we can add elements only at one end, called the rear of the queue and delete elements only at the other end called the front of the queue.
Queue is a linear data structure and it behave like first in first out. So queue is also called FIFO data structure.

Operation on queue

1. Insertion in queue (insertion at rear end)
2. Deletion element (delete element from queue from front.
3. Traversing the queue.
4. Search an element from the queue.

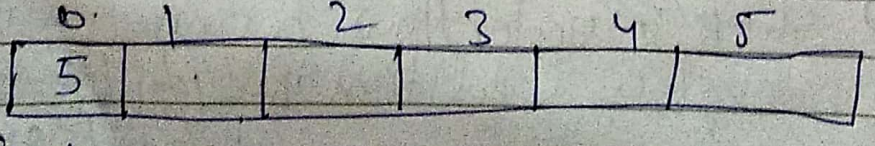
Ex of queue } int queue[6];



front=-1
rear=-1

(a) Empty Queue.

Add element 5 in queue.

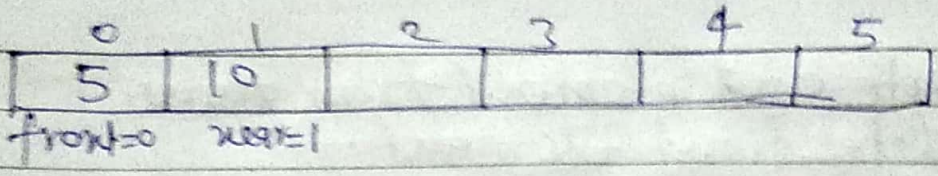


front=0
rear=0

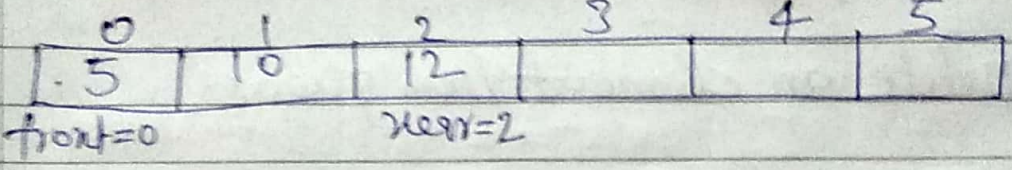
(b) queue with one element.

Add 10 in queue.

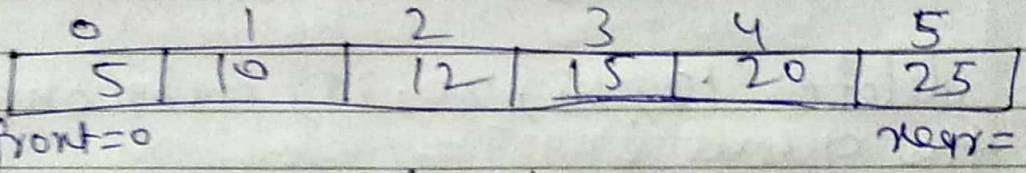
(c)



(d) Add 12 in queue.



(E) Add 15, 20, 25 in queue one by one.

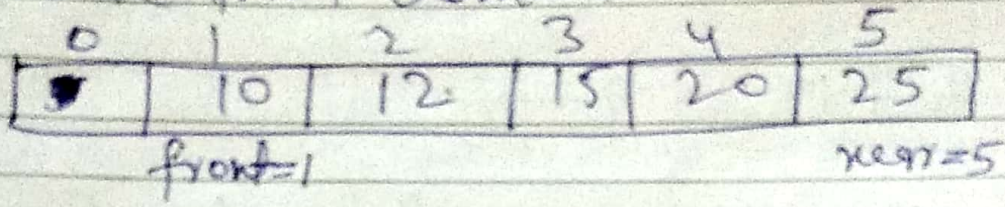


Now the value of rear is equal to MAX-1. the value of max is 6 (ie array size.
Now if we want to add one more element in queue then it is not possible because queue is ~~empty~~ overflow. (full)
So the Condition of overflow of simple linear queue is

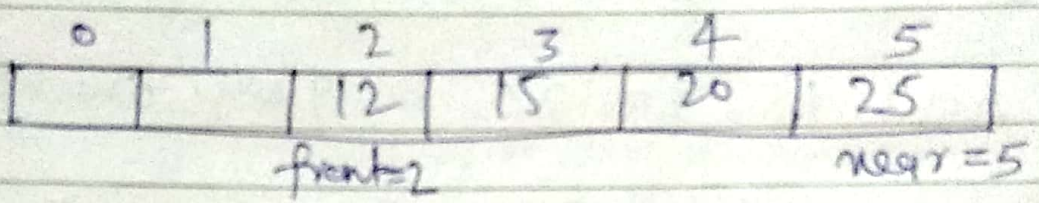
```
if (rear == MAX-1)
    printf("Queue is overflow");
```

In the above example we learn that we can insert element only at one end that is called rear of queue. the value of rear and front in empty Queue is -1. If queue have one element then value of ~~front~~ front and rear=0.

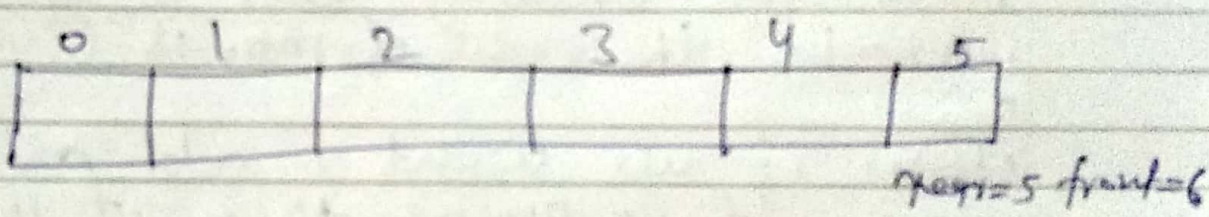
(F) Delete and element from queue.
delete front element.



(g) Delete an element from queue.



(h) delete 12, 15, 20, 25, one by one then the queue is



Now if we want to delete element from queue then it is not possible because queue is underflow.
So the underflow condition is

```
if ( front == -1 || front > rear )
    printf("Queue is Underflow");
```

Ex- from F to h show that how we can delete element from queue. The deletion always perform from the front of queue.

⇒ Array implementation of queue Program.

```
#include <stdio.h>
#include <conio.h>
#define MAX 6
int queuearr[MAX];
int front = rear = -1;
void insert();
void delete();
void display();
void main()
{ int choice
  while(1)
  {
     printf("1. Insert\n 2. Delete\n,
            3. Display\n, 4. Exit\n");
  printf("Enter your choice\n");
  scanf("%d", &choice);
  switch(choice)
  { case 1;
        insert();
        break;
    case 2;
        delete();
        break;
    case 3:
        display()
        break;
    case 4;
        exit(1);
  default;
     printf("Enter correct choice\n");
}}} //switch, while and main close.
```

```
void insert ()
{
    int added_item;
    if (rear == MAX-1)
      printf (" Queue is overflow \n");
    else
    {  if (front == -1) //if queue is empty.
          front = 0;
       printf ("Input the element for adding \n");
       scanf ("%d", &added_item);
       rear = rear + 1;
       queuearr[rear] = added_item;
    }
}

void delete ()
{
    if (front == -1 || front > rear)
    {  printf(" Queue is underflow \n");
       return;
    }
    else
    {
    printf(" Element deleted from queue is
                          %d \n", queuearr[front]);
    front = front + 1;
    }
}
```

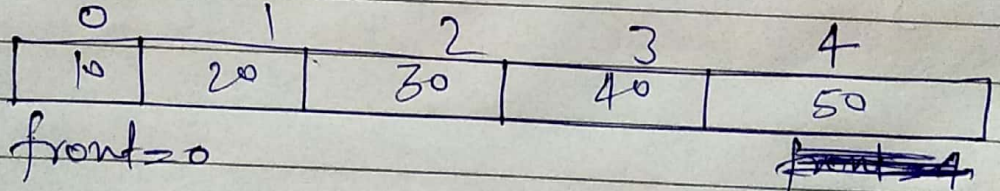
```
void display()
{
    int i;
    if (front == -1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is \n");
        for(i = front; i <= rear; i++)
            printf("%d \t", queue_arr[i]);
        printf("\n");
    }
}
```

(Circular queue)

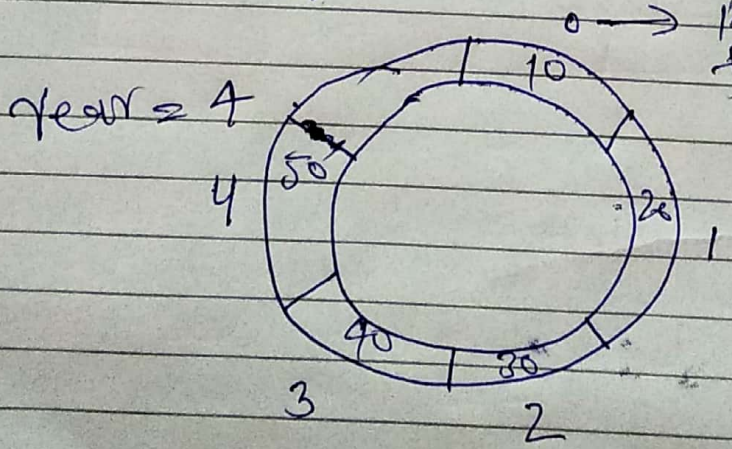
Circular queue is a linear data structure in which the operations are performed based on FIFO (first in first out) principle and the last position (element) is connected back to the first position to make a circle.

Ex Suppose we have five elements in queue. So the value of MAX = 5
int queue[5];

We can show in both form like.

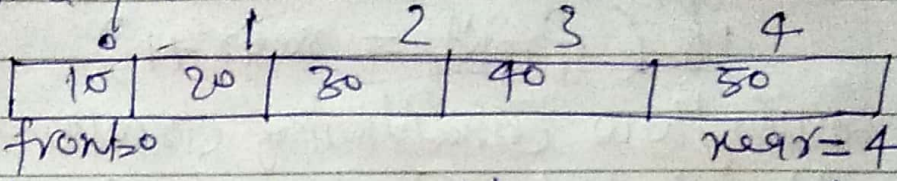


or



In Circular queue if value of front or rear goes to MAX-1 then value of front or rear set as by zero.
But Remember one thing if front=0 and rear == MAX-1 then we can not assign zero to rear.

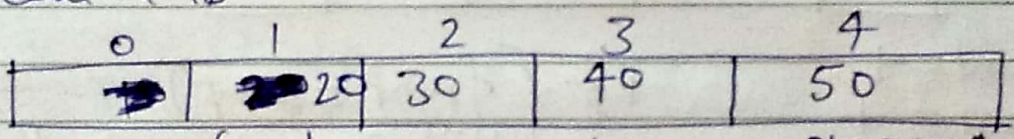
Ex1 Initial queue:



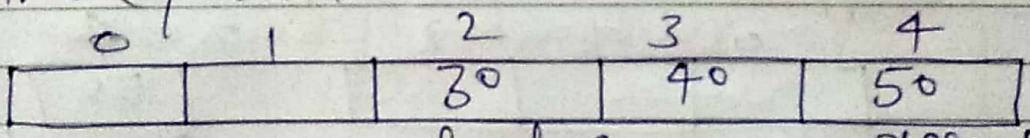
Now if we want to add one more element in queue then it show queue is overflow so one Condition of overflow is

(front == 0 && rear == MAX-1)

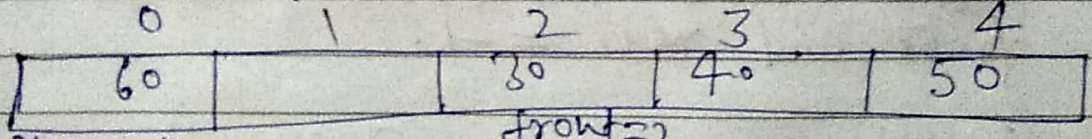
Ex-2 Delete front element from queue then queue is



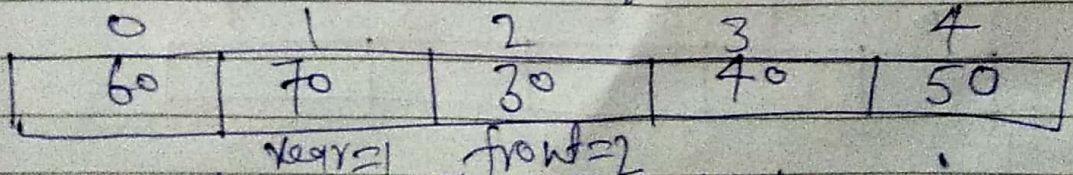
Now delete next element from queue. then queue is



Ex-3 Now add 60 in queue. So for insert 60 in queue first we set rear = 0 then insert element.



Now add 70 in queue.



So second overflow condition is -

Second overflow Condition is –

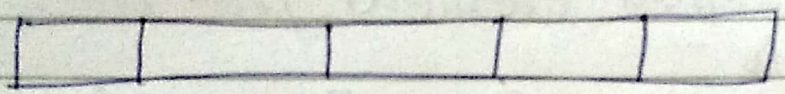
if (front = rear + 1)

overflow Condition of Circular queue.

So the total Combined Condition is

```
if((front==0 && rear==MAX-1) || (front=rear+1))
    printf("queue is overflow");
```

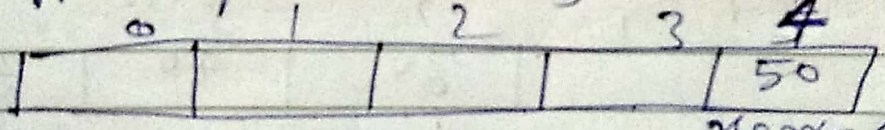
Now example for underflow of Circular queue.



front = -1
rear = -1

```
if (front == -1)
    printf("Queue is underflow");
```

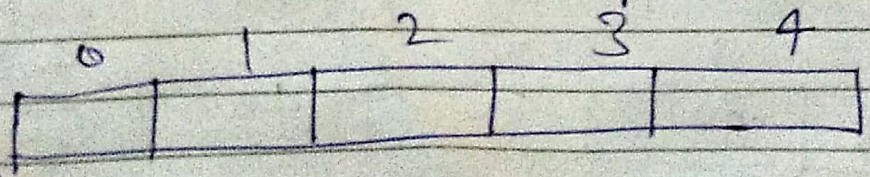
Ex. Suppose queue has only one element.



rear = 4
front = 4

delete front element from queue. Here front represent last element of queue so after delete this element we set value of front and rear = -1.

So the resultant queue is



front = -1
rear = -1

// Program for circular queue using array

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 5
int cqueuearr[MAX];
int front = -1;
int rear = -1;
void insert();
void delete();
void display();
void main()
{ int choice;
  while(1)
  {
    printf("1. Insert \n"; 2. Delete \n, 3. Display \n,
                    4. Quit \n");
    printf("Enter your choice: ");
    scanf("%d", choice);
    switch(choice)
    {
      case 1;
          insert();
          break;
      case 2;
          delete();
          break;
      case 3;
          display()
          break;
      Case 4;
          exit(1);
    default;
          printf("Enter correct choice \n");
}}//end main()
```

```
    }//End of switch
  } end of while
getch();
} // end of main.

void insert()
{ int item;
  if((front==0&&rear==MAX-1)||(front==rear+1))
  { printf("Queue is overflow\n");
    return;
  }
  if(front==-1) // if queue is empty
  { front=0;
    rear=0;
  }
  else
    if(rear==MAX-1) //rear is at last position of queue.
      rear=0
    else
      rear = rear+1;
  printf("Enter value of item for insertion\n");
  scanf("%d",&item);
  queuearr[rear] = item;
} // end of insertion function.

void delete()
{ if(front==-1)
  [ { printf("Queue is underflow\n");
      return;
  ]
```

```
printf(" Element deleted from queue is : %d \n",
                                    cqueue_arr[front] );
if (front == rear ) //queue has only one element.
  {  front=-1;
     rear = -1;
  }
  else
     if (front == MAX-1)
        front = 0;
     else
        front = front+1;
} // End of delete function.

void display()
  {
    int frontpos, rearpos;
    frontpos= front; rearpos = rear;
    if (front==-1)
      { printf("Queue is empty \n");
        return;
      }
    printf(" Queue elements are \n");
    if (frontpos <= rearpos )
       while ( frontpos <=rearpos )
        {
          printf(" %d \t", cqueue_arr[frontpos]);
          frontpos++;
        }
```

```
else
{   while ( frontpos <= MAX-1 )
    {
        printf("%d \t", cqueuearr[frontpos]);
        frontpos++;
    }
    frontpos = 0;
    while( frontpos <= rearpos )
    {
        printf("%d \t", cqueuearr[frontpos]);
        frontpos++;
    }
}
    printf("\n");
} // end of display().
```