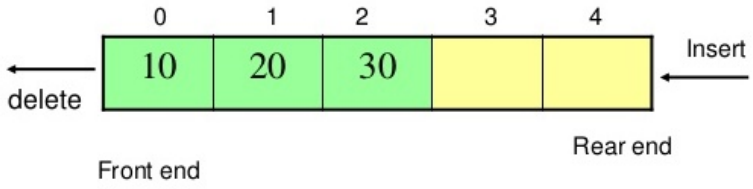
Circular Queue

Linear Queue:

- Linear queue follows “First In First Out” rule.
- We insert elements from Rear and delete from Front.
- In linear queue, “Front” is fixed location.
- When we delete an element from the queue, shifting of elements takes much time.

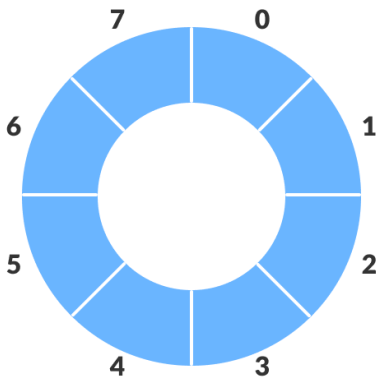


Circular Queue:

- Solution to Linear Queue data structure is Circular queue.
- In circular queue, we are representing the Linear Queue in circle form.
- We can move front location also after deletion of element from the queue.
- We can move both front and rear locations while inserting and deleting the elements.
- As front value is not fixed, front and rear values starts with -1

Declaration:

- We declare array variable with fixed size.
- Front and Rear variables are used to process the location data.
- Front and Rear variables not pointing to any location initially.
- Front location is not fixed in Circular Queue.
    - int cqueue[8];
    - int front=-1;
    - int rear=-1;



Operations: We can perform following operations on Circular Queue.

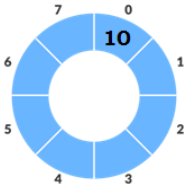
1. Insertion
2. Deletion

3. Display

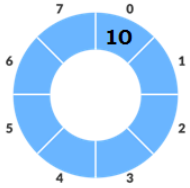
Insertion:

- The following diagrams describe the queue initially.
- Inserting elements from ‘rear’.
- First insertion changes both the values of front and rear variables.
- Continues insertion of elements results “Queue is full”.

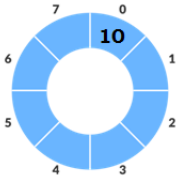

```
front = -1
rear = -1

if(front == -1)
    empty....
```



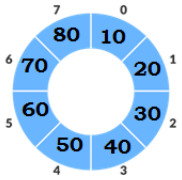

```
1st insertion:
    if(rear == -1)
    {
        f=0;
        r=0;
        cq[r] = ele;
        inserted...
    }
```



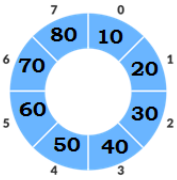

```
Queue has only 1 element :
    if(front == rear)
    {
        deleted : cq[f];
        f = r = -1 ;
    }
```




```
1st insertion:
    if(rear == -1)
    {
        f=0;
        r=0;
        cq[r] = ele;
        inserted...
    }
```

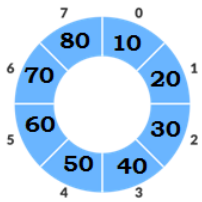



```
From 2nd insertion :

    ++rear ;
    cq[r] = ele ;
    inserted...
```

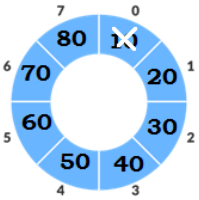



```
Queue is Full :
    if (f == 0 && r==size-1)
        Full.....
```

- Deleting elements using ‘front’ variable.
- After deletion of element, front value increase by 1.

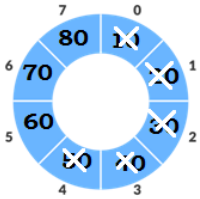


f=0 , r=7



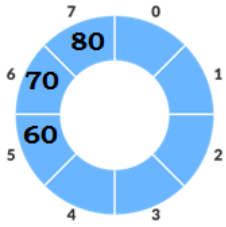
Deletion :
Deleted : cq[f];
++front

Keep on deleting



Now
f=5
r=7

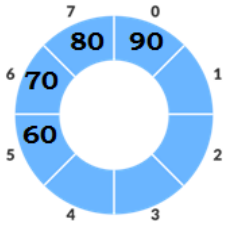
- When we try to insert the element and the rear value reaches size-1, again rear value starts from 0.



f = 5
r = 7

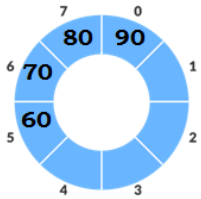
inserting

```
if(r==size-1)
{
    r=0;
    cq[r]=ele;
    inserted...
}
```



f = 5
r = 0

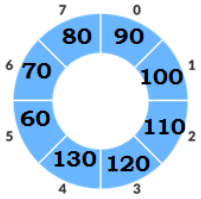
- Keep on inserting elements reaches “Queue is full” condition.
- Two conditions gives “Queue is full” situation.



f = 5
r = 0

Keep on inserting.....

++rear
cq[r] = ele;
inserted....



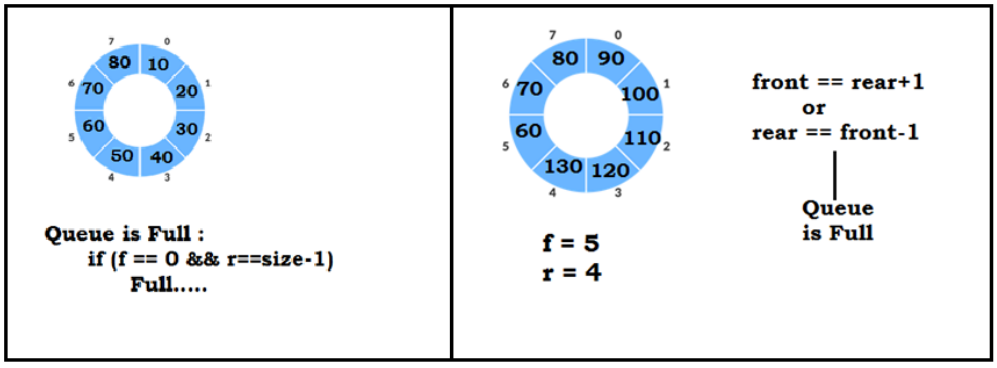
f = 5
r = 4

front == rear+1
or
rear == front-1

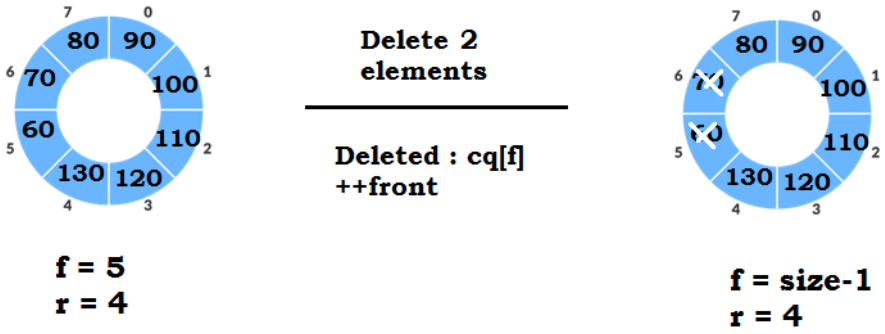
Queue
is Full

In circular queue, we need to check 2 situations for “Queue is Full” condition.

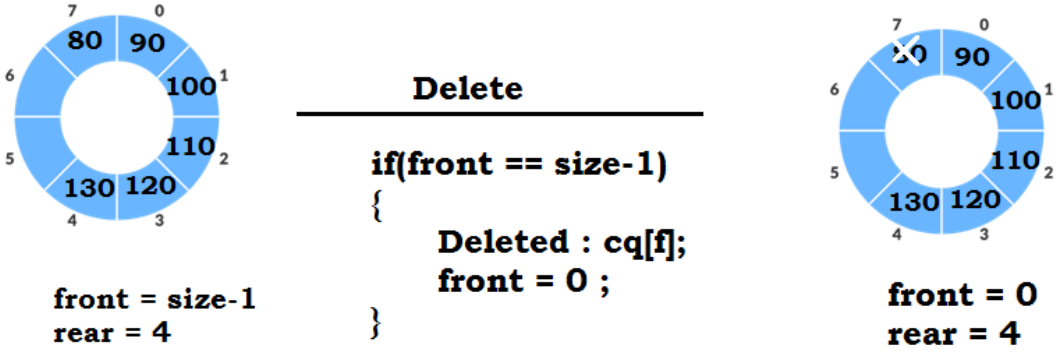
2 cases of "Queue is Full"



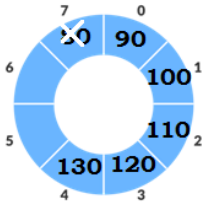
Deleting elements to make the front pointing to last location of Queue:



- While deleting, if it reaches the location “size-1”, it starts with 0 again in the next cycle.



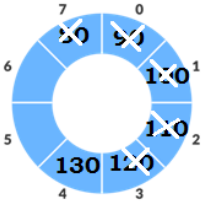
- When front and rear values are equal means Queue has only 1 element.
- When we remove the last element from the queue, then front and read values become -1.



front = 0
rear = 4

Delete 4 elements

Deleted : cq[f];
++front ;



front = 4
rear = 4

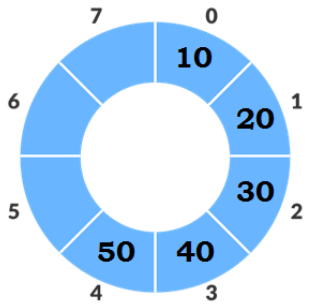
if(front==rear)
{
Queue has only 1 element.
}

```
void enque(int ele)
{
	if(isOverflow())
	{
		printf("Queue is Overflow \n");
	}
	else
	{
		if(rear==-1)
		{
			front=0;
			rear=0;
		}
		else if(rear==size-1)
		{
			rear=0;
		}
		else
		{
			++rear;
		}
		cqueue[rear] = ele;
		printf("Inserted...\n");
	}
}

int isOverflow()
{
	if((front==0 && rear==size-1) || (front==rear+1))
	{
		return 1;
	}
```

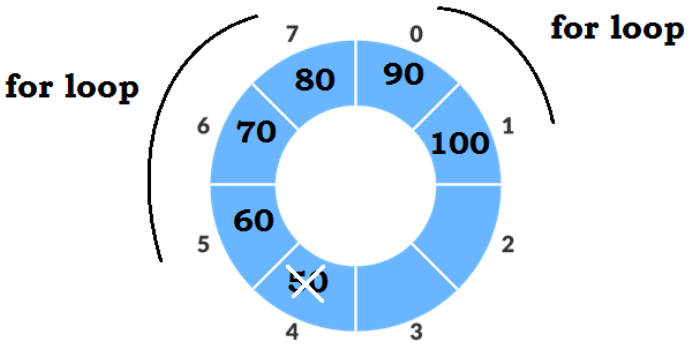
```
        else
        {
                return 0;
        }
}

void deque()
{
        if(isUnderflow())
        {
                printf("Queue is Underflow \n");
        }
        else
        {
                printf("Deleted : %d\n",cqueue[front]);
                if(front==rear)
                {
                        front=-1;
                        rear=-1;
                }
                else if(front==size-1)
                {
                        front=0;
                }
                else
                {
                        ++front;
                }
        }
}

int isUnderflow()
{
        if(front==-1 && rear==-1)
        {
                return 1;
        }
        else
        {
                return 0;
        }
}
```

Display elements:

Display():



Front = 0
Rear = 4



Front = 5
Rear = 1

Code program of CQUEUE:

```
#include<stdio.h>
#define SIZE 5
int cqueue[SIZE];
int front=-1, rear=-1;
void enqueue(int);
int dequeue(void);
void display(void);
int underflow(void);
int overflow(void);
void main()
{
        int ch,item;
        printf("***Circular Queue Operations***\n");
        while(1)
        {
                printf("1. Insert \n");
                printf("2. Delete \n");
                printf("3. Display \n");
                printf("4. Exit \n");
                printf("Enter your choice : ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1  :       if(!overflow())
                                        {
                                                printf("Enter item to insert : ");
                                                scanf("%d",&item);
                                                enqueue(item);
                                        }
                                        else
                                        {
                                                printf("Queue is Full\n\n");
                                        }
                                        break;

                        case 2  :       if(!underflow())
                                        {
                                                int ele = dequeue();
                                                printf("Deleted : %d\n\n",ele);
                                        }
                                        else
                                        {
                                                printf("Queue is Empty\n\n");
                                        }
                                        break;
```

```
                        case 3    :     if(!underflow())
                                        {
                                                printf("The queue is : \n");
                                                display();
                                        }
                                        else
                                        {
                                                printf("No elements to display \n");
                                        }
                                        break;

                        case 4    :     exit(1);
                        default   :     printf("Your choice is wrong \n\n");
                }
        }
}


int underflow()
{
        if((front==-1)&&(rear==-1))
        {
                return 1;
        }
        else
        {
                return 0;
        }
}


int overflow()
{
        if(((front==0)&&(rear==SIZE-1))||(front==rear+1))
        {
                return 1;
        }
        else
        {
                return 0;
        }
}

void enqueue(int item)
{
```

```
	if((front==-1)&&(rear==-1))
	{
		front=0;
		rear=0;
	}
	else if(rear==SIZE-1)
	{
		rear=0;
	}
	else
	{
		rear++;
	}
	cqueue[rear]=item;
	printf("Inserted : %d\n\n",item);
}

int dequeue()
{
	int item ;
	item = cqueue[front];
	if(front==rear)
	{
		front=-1;
		rear=-1;
	}
	else if(front==SIZE-1)
	{
		front=0;
	}
	else
	{
		front=front+1;
	}
	return item ;
}

void display()
{
	int i;
	if(front<=rear)
	{
		for(i=front ; i<=rear ; i++)
		{
			printf("Element %d : %d \n", i+1, cqueue[i]);
		}
```

```
        }
        else
        {
                for(i=front ; i<=SIZE-1; i++)
                {
                        printf("Element %d : %d \n",i+1,cqueue[i]);
                }
                for(i=0 ; i<=rear ; i++)
                {
                        printf("Element %d : %d \n",i+1,cqueue[i]);
                }
        }
}
```

Note:

- The circular queue works according to the below two conditions.
- The SIZE indicates the maximum number of items the queue can consist of.
    1. rear = (rear +1) % SIZE;
    2. front = (front + 1) % SIZE;
- And the implementation as follows :

```
#include <stdio.h>
#define SIZE 5
int cqueue[SIZE];
int front=-1, rear=-1;
int isFull(void);
int isEmpty(void);
void enQueue(int);
int deQueue(void);
void display(void);
int main()
{
        deQueue();

        enQueue(10);
        enQueue(20);
        enQueue(30);
        enQueue(40);
        enQueue(50);
        display();

        deQueue();
        deQueue();
        enQueue(60);
        display();
```

```
	enQueue(70);
	display();
	return 0;
}
int isFull()
{
	if((front == rear + 1) || (front == 0 && rear == SIZE-1))
		return 1;
	else
		return 0;
}
int isEmpty()
{
	if(front == -1)
		return 1;
	else
		return 0;
}
void enQueue(int data)
{
	if(isFull())
		printf("Queue is Overflow\n");
	else
	{
		if(front == -1) front = 0;
		rear = (rear + 1) % SIZE;
		cqueue[rear] = data;
		printf("Inserted : %d \n", data);
	}
}
int deQueue()
{
	int data;
	if(isEmpty())
	{
		printf("Queue is Underflow\n");
		return(-1);
	}
	else
	{
		data = cqueue[front];
		if (front == rear)
		{
			front = -1;
			rear = -1;
		}
```

```
                else
                {
                        front=(front + 1)%SIZE;
                }
                printf("Deleted data : %d \n", data);
                return(data);
        }
}
void display()
{
        int i;
        if(isEmpty())
        {
                printf("Empty Queue\n");
        }
        else
        {
                printf("Elements are : ");
                for( i = front; i!=rear; i=(i+1)%SIZE)
                {
                        printf("%d ",cqueue[i]);
                }
                printf("%d \n",cqueue[i]);
         }
}
```