Q 9. Write a program of queue using linked list.

```
#include <stdio.h>
#include <malloc.h>
struct node
{     int info;
      struct node * link;
} *front = NULL, *rear = NULL;

main()
{   int choice;
    while (choice != 4)
    {
        printf ("\n1. Insert");
        printf ("\n2. Delete ");
        printf ("\n3. Display");
        printf ("\n4. Quit ");
        printf ("\n Enter your choice :");
        scanf ("%d", &choice);
        switch (choice)
        { case 1 : insert();
                   break;
          case 2 : del();
                   break;
          case 3 : display();
                   break;
          case 4 : exit (1);
          default : printf ("Wrong choice\n");
        }
    }
}
```

```
insert()
{
    struct node *temp;
    int added_item;
    temp = (structnode *) malloc (sizeof (struct node));
    printf ("Input the element for adding in queue :");
    scanf ("%d", &added_item);
    temp -> info = added_item;
    temp -> link = NULL;
    if (front == NULL)
        front = temp;
    else
        rear -> link = temp;
    rear = temp;
}
del()
{
    struct node *temp;
    if (front == NULL)
        printf ("\n Queue Underflow");
    else
    {   temp = front;
        printf ("Deleted element is %d \n", temp -> info);
        front = front -> link;
        free (temp);
    }
}
```

```
display()
{
    struct node *ptr;
    ptr = front;
    if (front == NULL)
        printf("Queue is empty\n");
    else
    {   printf("Queue elements :\n");
        while (ptr != NULL)
            {  printf("%d", ptr -> info);
               ptr = ptr -> link;
            }
        printf("\n");
    }
}
```

Circular Queue : A queue is called circular when the last element comes just before the first element

Here after n-1 th element , 0th element occurs.
Similarly, we take assumption that after last element of queue, the first element will occur.
To overcome a particular type of problem in queue, we use the concept of circular queue
The problem in queue arises when rear is at the last position of array and front is not at the 0th position and we cannot add any element in queue because rear is at the n-1 th position

queue

[0]	[1]	[2]	[3]	[4]	[5]	[6]
		5	10	15	20	25

front = 2 rear = 6

So to overcome this type of situation, we use the circular queue

1) queue

[0]	[1]	[2]	[3]	[4]	[5]	[6]
30		5	10	15	20	25

rear = 0, front = 2

2) queue

[0]	[1]	[2]	[3]	[4]	[5]	[6]
30	35		10	15	20	25

rear = 1 front = 3 .

Add Operation in Circular Queue.

First we have to check that rear is at the (n-1)th position. If rear = N-1 then we set the value of rear to 0 and add the element at the 0th position of the array. Otherwise element will be added same as in simple queue.

Here first we are checking for overflow condition.

```
if((front == 0 && rear == MAX-1) || (front == rear+1))
  {  printf("\nQueue is Overflow");
     return;
  }
```

If queue is initially empty then we set the value 0 to front and rear and then add the element in queue otherwise we increase the value of rear only and then element will be added in queue.

```
if (front == -1)
   {  front = 0;
      rear = 0;
   }
else
if (rear == MAXSIZE-1)
        rear = 0;
else
        rear = rear+1;
cqueue[rear] = added_item;
```

DATE

Delete Operation in Circular queue.

First we have to check that front is at the N-1 th position. If front = N-1 then delete the element from queue and set the value of front to 0 as -

```
if (front == MAXSIZE - 1)
        front = 0;
```

If there is only one element in queue then we delete the element from queue and set the value of front and rear to -1 as

```
if (front == rear)
  {  front = -1;
     rear = -1;
  }
```

Q 10 Write a program of circular queue using array.

```
#include <stdio.h>
#define MAX 5
int cqueue [MAX];
int front = -1;
int rear = -1;
main()
{
    int choice;
    while (choice != 4)
        {
            printf ("1. Insert\n");
            printf ("2. Delete \n");
            printf ("3. Display\n");
            printf ("4. Quit\n");
            printf (" Enter your choice:");
            scanf ("%d", & choice);
            switch (choice)
            {
              case 1:  insert ();
                       break;
              case 2:  del ();
                       break;
              case 3:  display();
                       break;
              case 4:  exit (1);
              default: printf ("wrong choice\n");
            }
        }
}
```

DATE

```
insert()
{
    int added_item;
    if (front ==0 && rear ==MAX-1) || (front == rear+1))
    {
        printf ("Queue overflow\n");
        return;
    }
    if (front == -1)
    {
        front = 0;
        rear = 0;
    }
    else
        if (rear == MAX-1)
            rear = 0;
        else
            rear = rear+1;
    printf ("Input the element for insertion in queue:");
    scanf ("%d", &added_item);
    equeue [rear] = added_item;
}
del()
{
    if (front == -1)
    {
        printf ("Queue underflow\n");
        return;
    }
```

Natural

```
        printf("Element deleted from queue is: %d\n",
                        cqueue[front]);
        if (front == rear)
        {
            front = -1;
            rear = -1;
        }
        else
            if (front == MAX-1)
                front = 0;
            else
                front = front + 1;
    }
display()
{   int front_pos = front;
    int rear_pos = rear;
    if (front == -1)
    {   printf("Queue is empty\n");
        return;
    }
    printf("Queue elements:\n");
    if (front_pos <= rear_pos)
        while (front_pos <= rear_pos)
        {
            printf("%d", cqueue[front_pos]);
            front_pos++;
        }
```

```
else
{
    while ( front_pos <= MAX-1)
    {
        printf ("%d", equeue [front_pos]);
        front_pos++;
    }
    front_pos = 0;
    while (front_pos <= rear_pos)
    {
        printf ("%d", equeue[front_pos]);
        front_pos++;
    }
}
printf("\n");
}
```