Algorithm For enqueue() In Circular Queue
=================================

1. Check for overflow

2. If Queue is full then print the message QUEUE OVERFLOW and return.

3. If Queue is not full ,then ADJUST REAR

4. Insert the element in the QUEUE at the position pointed by REAR

5. If it is FIRST INSERTION then set FRONT to 0

6. Finish and return.

Algorithm For dequeue() In Circular Queue
=================================

1. Check for underlow

2. If Queue is empty then print the message QUEUE UNDERFLOW and return.

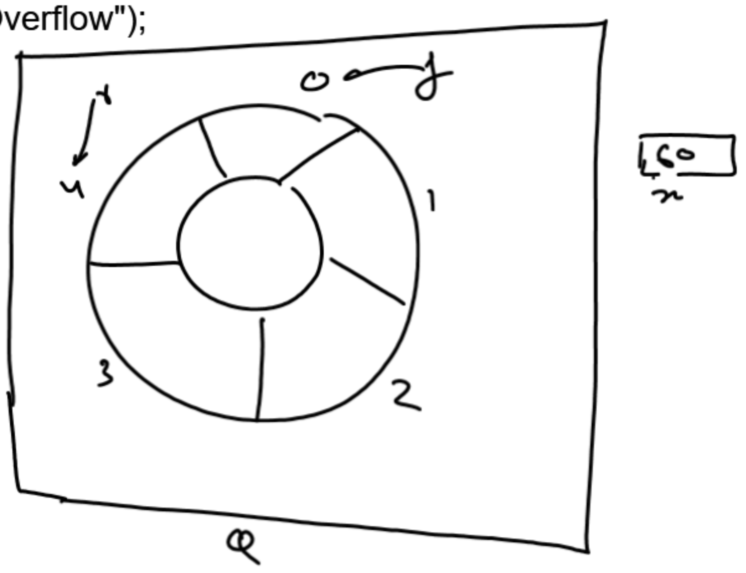
3. If Queue is not empty ,then delete the element pointed by FRONT.
4 ADJUST FRONT and if required then REAR also.

5. Return the deleted ele.
6. Finish

Implementing Circular Queue In C

```c
#include <stdio.h>
struct CQueue
{
   int arr[5];
   int front,rear;
};
void enqueue(struct CQueue *,int);
int dequeue(struct CQueue *);
int main()
{
   struct CQueue Q;
   int choice,x;
   Q.front=Q.rear=-1;
  do
  {
    printf("Select an operation:");
    printf("\n1. Enqueue\n2.Dequeue\n3.Quit");
    printf("\nEnter your choice:");
    scanf("%d",&choice);
switch(choice)
{
  case 1:
            printf("Enter ele:");
            scanf("%d",&x);
            enqueue(&Q,x);
            break;
 case 2:
            x=dequeue(&Q);
            if(x!=0)
               printf("\nDequeued %d",x);
           break;
case 3:
           printf("Thank you!");
           break;
default:
           printf("Wrong choice! Try
again");
     }
}while(choice!=3);
return 0;
}
```

```c
void enqueue(struct CQueue *p,int x)
{
  if((p->rear==4 && p->front==0)||(p->rear+1==p->front))
  {
       printf("Queue Overflow");
       return;
  }
  if(p->rear==4)
      p->rear=0;
  else
       p->rear++;
 p->arr[p->rear]=x;
 if(p->front==-1)
       p->front=0;
}
```

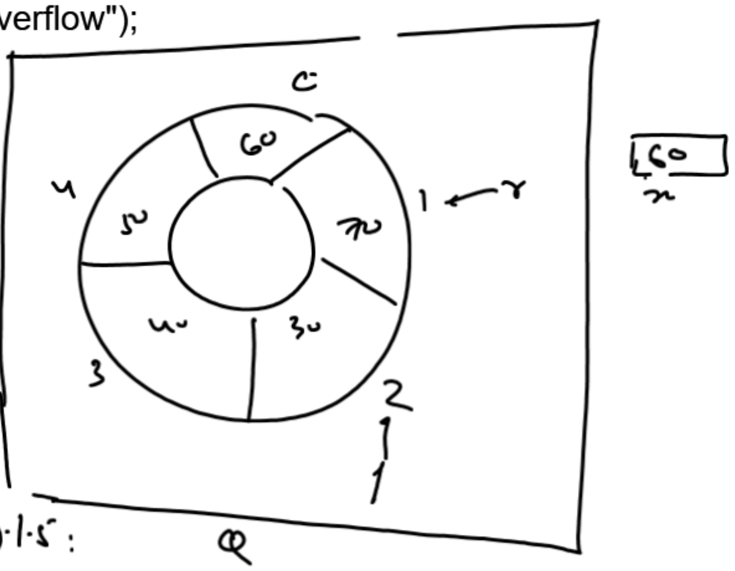


```
int dequeue(struct CQueue *p)
{
   int x;
   if(p->front==-1)
   {
          printf("Queue Underflow");
          return 0;
   }
   x=p->arr[p->front];
   if(p->front==p->rear)
          p->front=p->rear=-1;
  else if(p->front==4)
          p->front=0;
  else
         p->front++;
return x;

}
```

~~if (p->front == 4)~~
~~p->front = 0;~~
~~else if (p->front == p->rear)~~
~~p->front = p->rear = -1;~~
~~else~~
~~p->front++;~~

```
void enqueue(struct CQueue *p,int x)
{
   if((p->rear+1)%5==p->front)
   {
         printf("Queue Overflow");
         return;
   }
   if(p->rear==4)
        p->rear=0;
   else
         p->rear++;

  p->arr[p->rear]=x;
  if(p->front==-1)
         p->front=0;
}
```

p->rear = (p->rear+1)%5;
p->arr[p->rear]=x;



c
60
70
30
40
50
4
1
r
2
3
Q
60
x