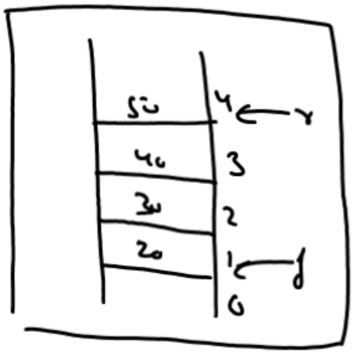
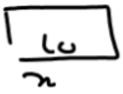
Writing The function enqueue()

```
=========================
void enqueue(struct Queue *p,int x)
{
  if(p->rear==4)
   {
       printf("Queue overflow");
       return;
   }
   p->rear++;
   p->arr[p->rear]=x;
   printf("\nEnqueued %d",x);
}
```

① In

② Dele





Writing The function dequeue()

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

Circular Queue

Overflow

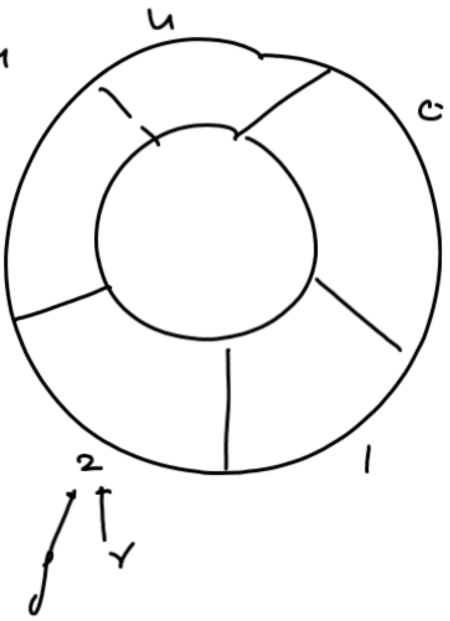
① f=0 && r=4

OR

~~② f=1 && r=0~~

~~③ f=2 && r=1~~

② r+1 == f



① Check for overflow

└→ a. $f == 0$ && $r == 4$

OR

└→ b. $r + 1 == f$

② Adjust rear

③ Insert

④ Reset front

① Check for underflow

└→ a if $f == -1$

② Delete

③ Adjust front and if reqd then rear

④ Rele —

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 at the position pointed by REAR.

5. If it is first insertion then set FRONT to 0.

6. Finish and return.

Assignment:
===========

Write the algorithm for dequeu() in CIRCULAR QUEUE.