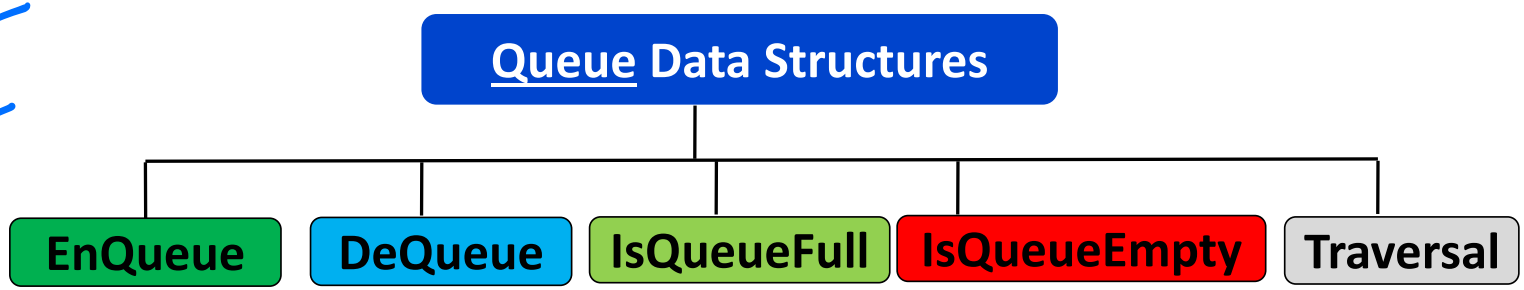
CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

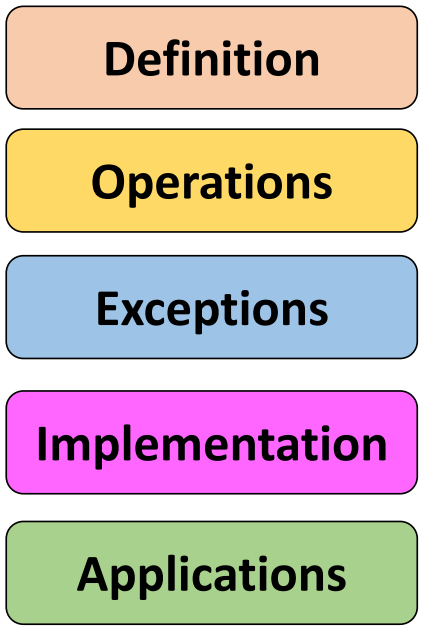
Recap

- Queue (FIFO) Implementation
  - EnQueue ()
  - DeQueue ()
  - IsQueueFull ()
  - IsQueueEmpty ()
  - PrintQueue ()
- Limitations:



Outline

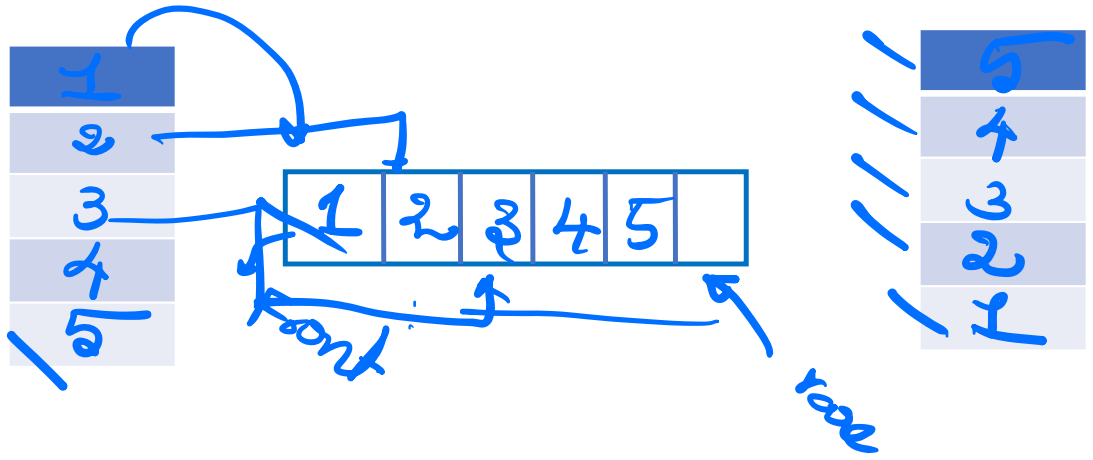
- Exercise on Stack and Queue
- Limitations of Queue data structures
- Circular Queue data structures
- Circular Queue operations
- Circular Queue exceptions
- Circular Queue implementation
- Circular Queue applications



Exercise: Stack and Queue (1)

Given a 5 element stack S (from top to bottom: 1,2,3,4,5), and an empty queue Q, remove the elements one-by-one from S and insert them into Q, then remove them one-by-one from Q and re-insert them into S. S now looks like (from top to bottom)?

(a) 1 , 2, 3, 4, 5
(b) 5, 1, 2, 3, 4
(c) 5, 4, 3, 2, 1
(d) 1, 3, 5, 2, 4



Exercise: Queue (2)

Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are:

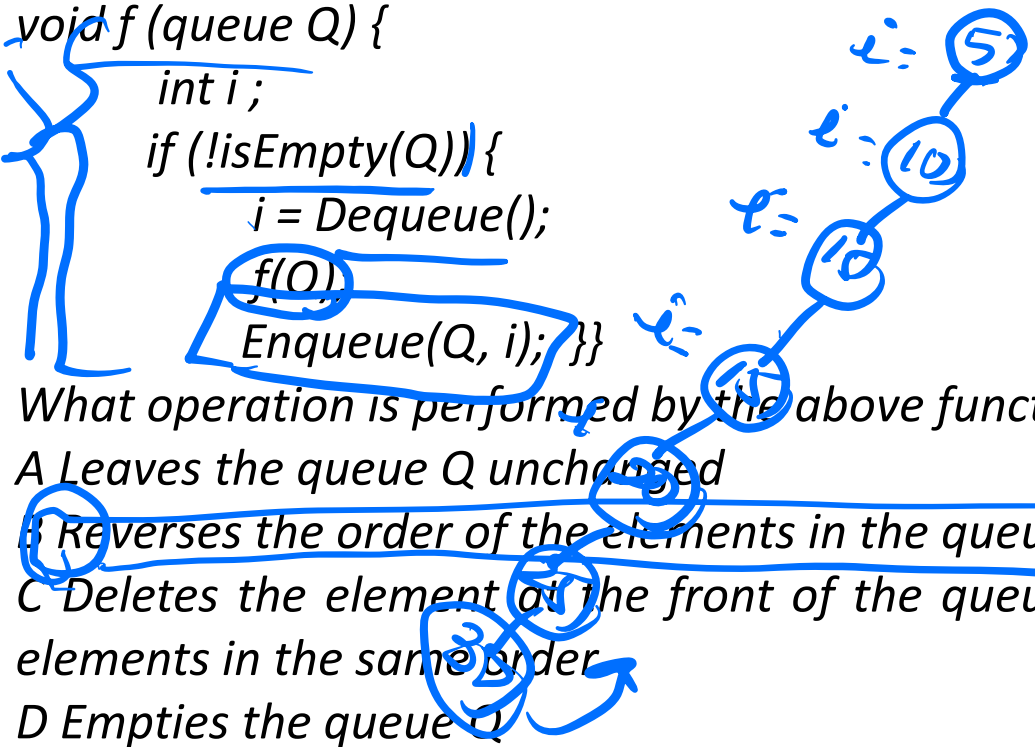
i. isEmpty (Q) — returns true if the queue is empty, false otherwise

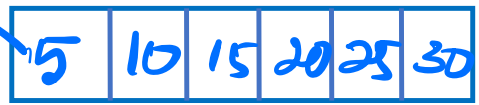
ii. Dequeue () — deletes the element at the front of the queue and returns its value.

iii. Enqueue (Q, i) — inserts the integer i at the rear of the queue.

Consider the following function:

```
void f (queue Q) {
    int i ;
    if (!isEmpty(Q)) {
        i = Dequeue();
        f(Q);
        Enqueue(Q, i); }}
```





What operation is performed by the above function f ?

A Leaves the queue Q unchanged

B Reverses the order of the elements in the queue Q

C Deletes the element at the front of the queue Q and inserts it at the rear keeping the other elements in the same order

D Empties the queue Q

Exercise: Stack and Queue (3)

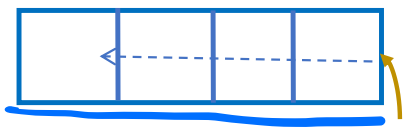
Which of the following is/are not a valid queue application(s)?

(a) When printing jobs are submitted to printer
(b) Lines at tickets counters
(c) Evaluating a mathematical expression
(d) Calls to large companies, when all lines are busy

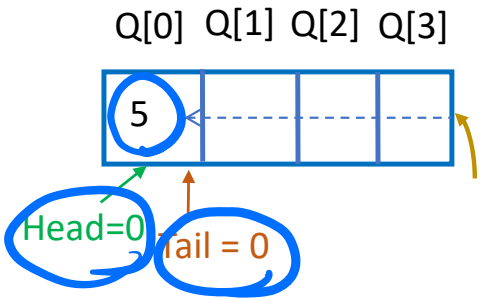
Queue: Limitations (1)

Real

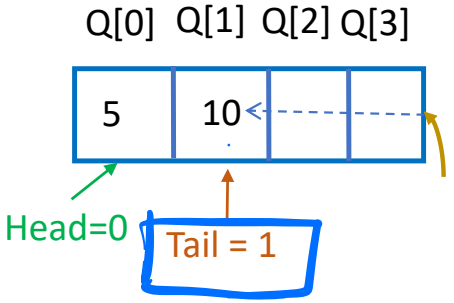
Head = Tail = -1



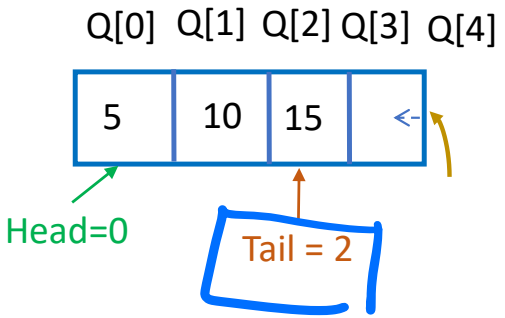
Enqueue(5)



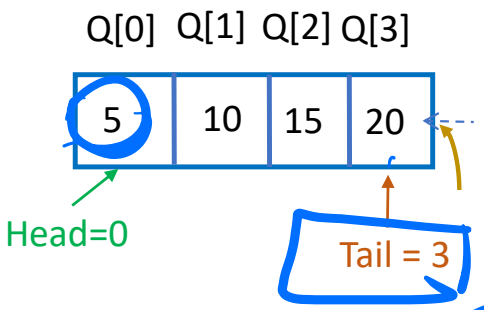
Enqueue(10)



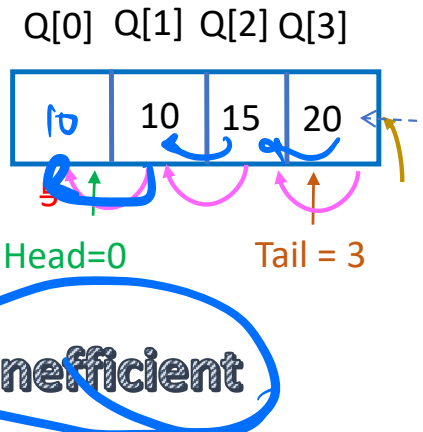
Enqueue(15)



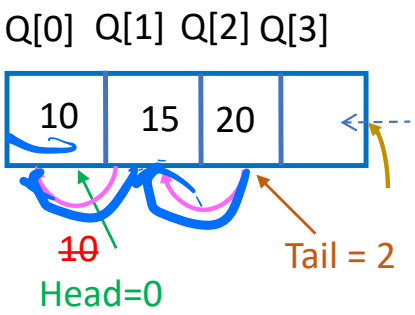
Enqueue(20)

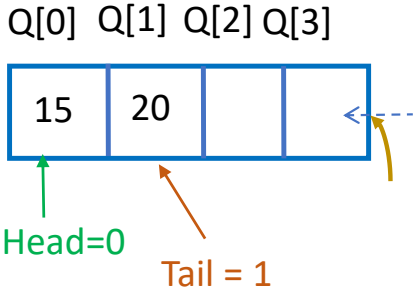


Dequeue()

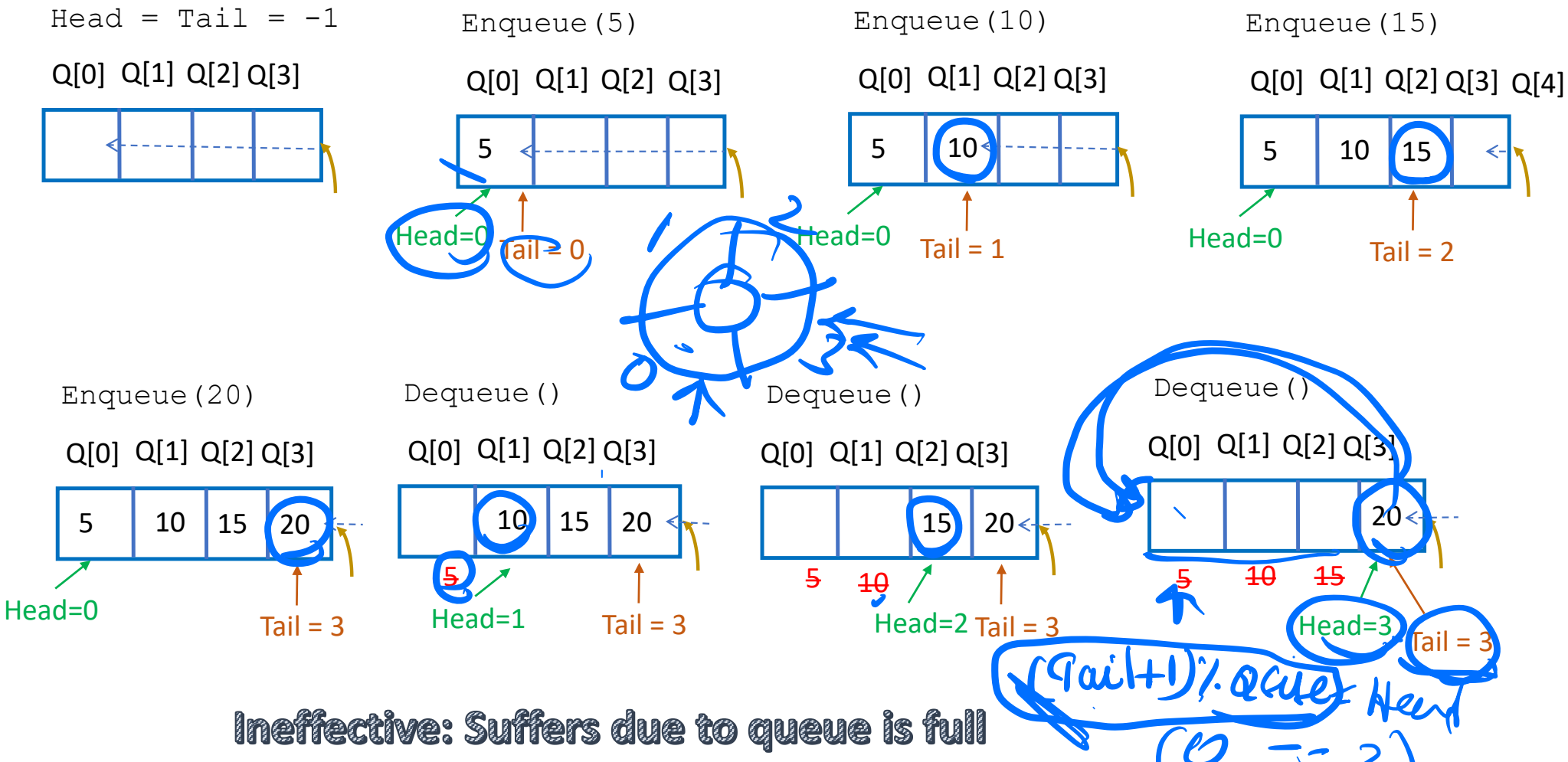


Dequeue()





Queue: Limitations (2)

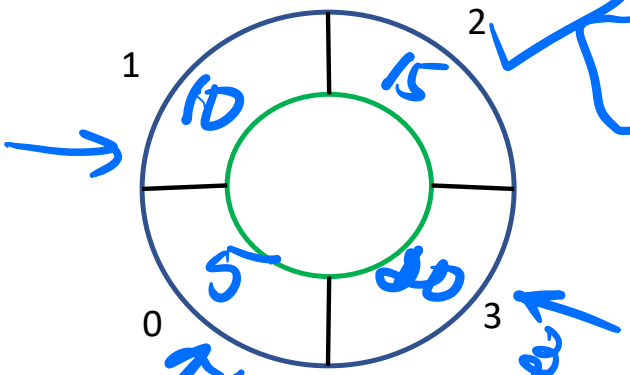


Ineffective: Suffers due to queue is full

Circular Queue: EnQueue Example

```
Head = Tail = -1
Qsize = 4
```

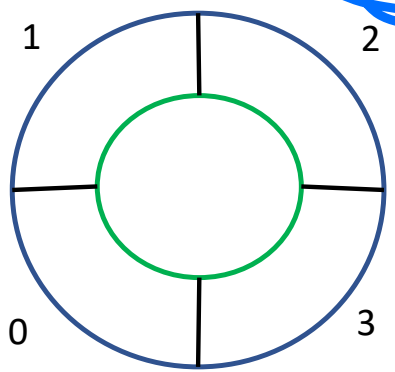
Case 1:
```
Tail = Tail + 1;
Tail = Tail % Qsize;
Queue[Tail] = Val;
```



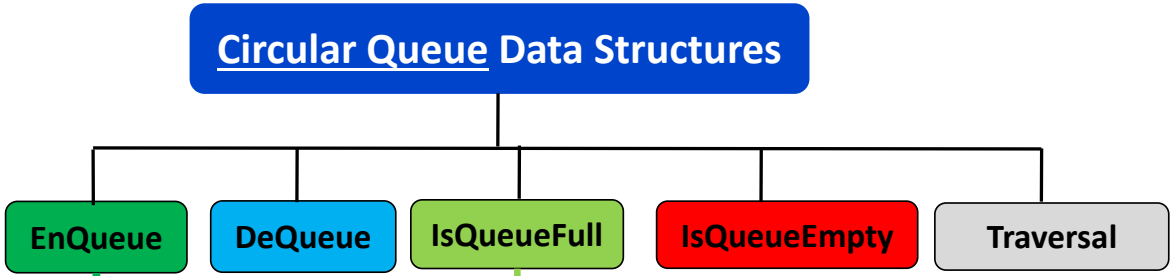

```
Head = (Tail+1)%Qsize
```

```
if (Head == -1)
    Head = Head + 1;
```

Case 2:
```
if ((Tail +1)% Qsize == Head)
    Overflow;
```



Implementation: EnQueue



Head = Tail = -1

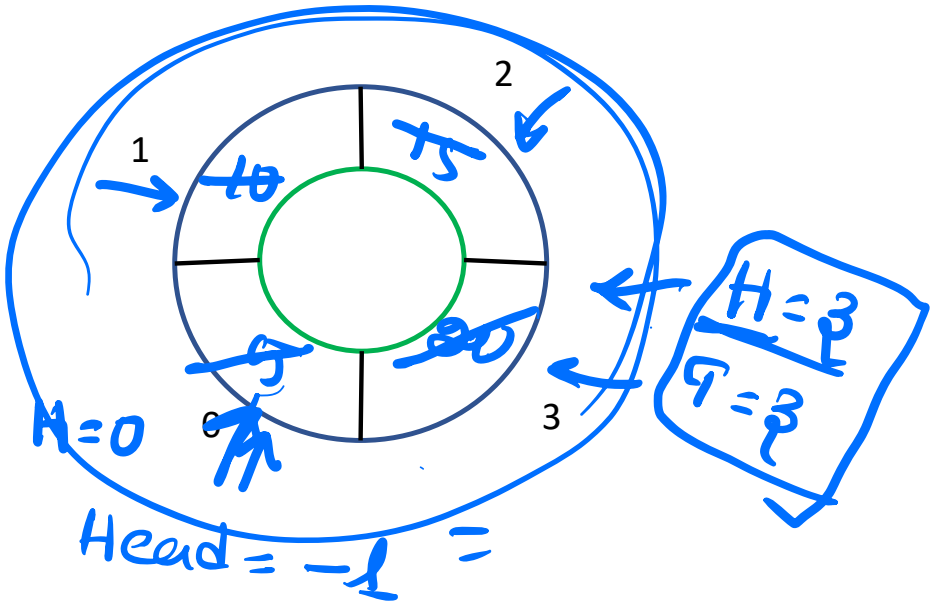
```
void Enqueue(){
  int Element;
  if(!IsQFull()){
   printf("Enter the element to be Enqueued...\n");
   scanf("%d", &Element);
   Tail++;
   Tail = Tail%QSize;
   Queue[Tail] = Element;
   if(Head == -1)
      Head++;
  } else
    printf("Element cannot be inserted as Queue is full\n");}
```

```
int IsQFull(){
   if((Tail+1)%QSize == Head)
      return 1;
   else return 0;
}
```

linux/ring_buffer.c at master · torvalds/linux · GitHub

Circular Queue: DeQueue Example

```
Head = Tail = -1
Qsize = 4
```



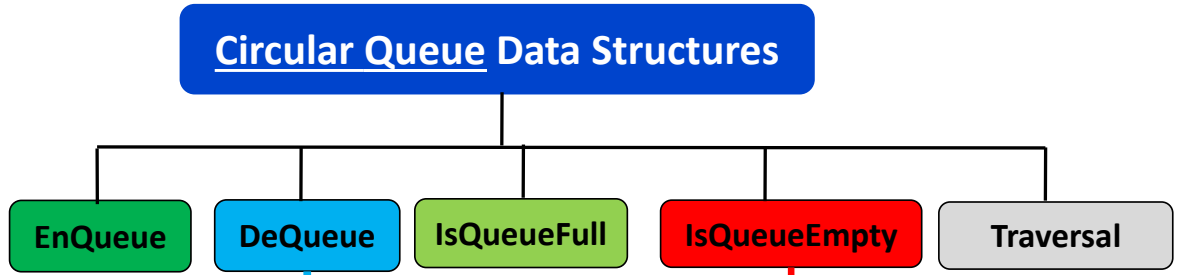
Case 1: *if (Head == -1)*
Queue empty;

Case 2 *if (Tail == Head)*

Head = -1
Tail = -1

Case 3: *if (Tail != Head)*

Implementation: Circular DeQueue



Head = Tail = -1

```
void Dequeue(){
 if(!IsQEmpty()){
    if(Head == Tail){
     /*That means only one element is present*/
      printf("%d is deleted from the queue\n", Queue[Head]);
      Queue[Head] = -1;
      Head = Tail = -1;
    } else{
      printf("%d is deleted from the queue\n", Queue[Head]);
      Queue[Head] = -1;
      Head++;
      Head = Head%QSize;
    }}
 else{printf("Element cannot be deleted as Queue is already empty \n")}}
```

```
int IsQueueEmpty(){
   if(Head == -1){
     return 1;
   } else return 0; }
```

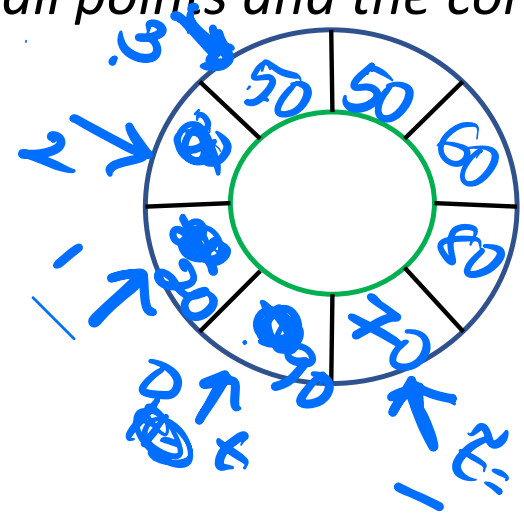
Exercise: Circular Queue (1)

A circular queue consists of size 8, and it contains: 10, 20, 30, 50, 50, 60, 80, 70 [Note: the index starts from 0].

Following operations are performed on the circular queue: Dequeue (), Dequeue (), Enqueue (90), Dequeue (), Enqueue (20)

What are the values at Head and Tail points and the corresponding index values?

(a) Head -> 30 & 2; Tail -> 70 & 7
(b) Head -> 50 & 3; Tail -> 70 & 7
(c) Head -> 50 & 3; Tail -> 90 & 1
(d) Head -> 50 & 3; Tail-> 20 & 1

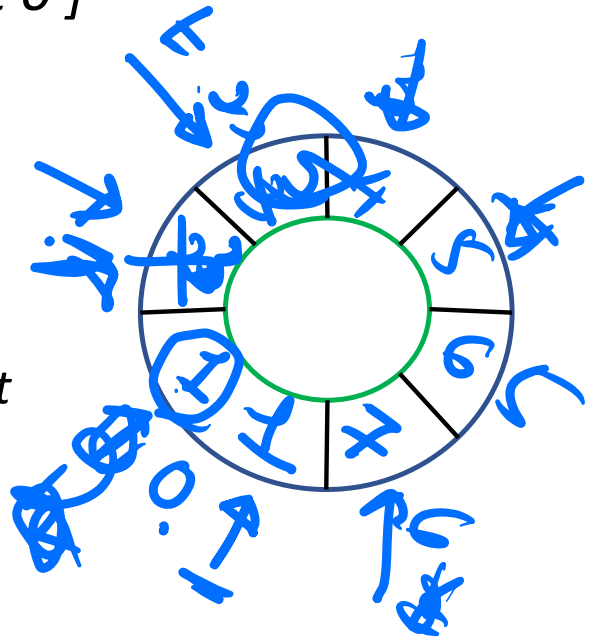


Exercise: Circular Queue (2)

Given a circular queue with size 7. What is the final value at index 2, after the following code is executed:

[Note: the circular queue array index starts at 0]

```
for (int k = 1; k <= 7; k++){
EnQueue(k);
}
for (int k = 1; k <= 4; k++){
  int delete; //to store the dequeued/deleted element
  delete=DeQueue();
  EnQueue(delete);
  Dequeue();
}
```



(a) 4 (b) 3 (c) 5 (d) 7

Exercise: Circular Queue (3)

Suppose a circular queue of capacity (n – 1) elements is implemented with an array of *n elements*. Assume that the *insertion* and *deletion* operation are carried out using *REAR and FRONT* as array index variables, respectively.

Initially, *REAR = FRONT = 0*. The conditions to *detect queue full* and *queue empty* are:

(A) Full: (REAR+1) mod n == FRONT, empty: REAR == FRONT

(B) Full: (REAR+1) mod n == FRONT, empty: (FRONT+1) mod n == REAR

(C) Full: REAR == FRONT, empty: (REAR+1) mod n == FRONT

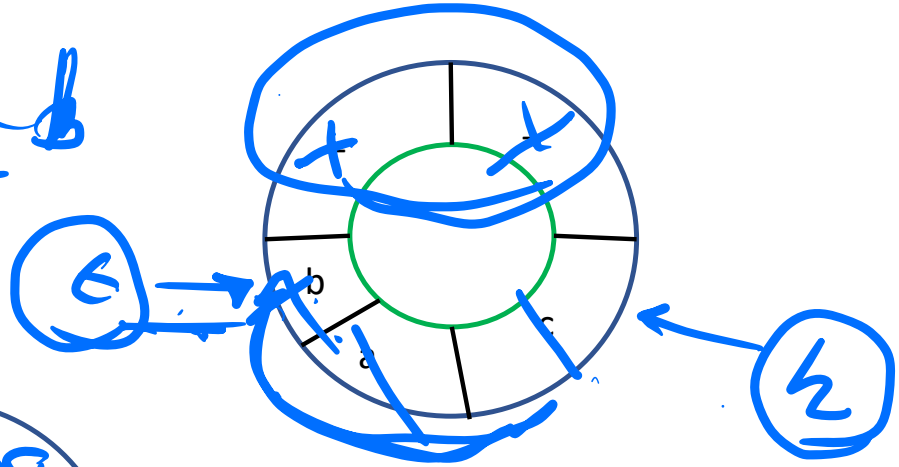
(D) Full: (FRONT+1) mod n == REAR, empty: REAR == FRONT

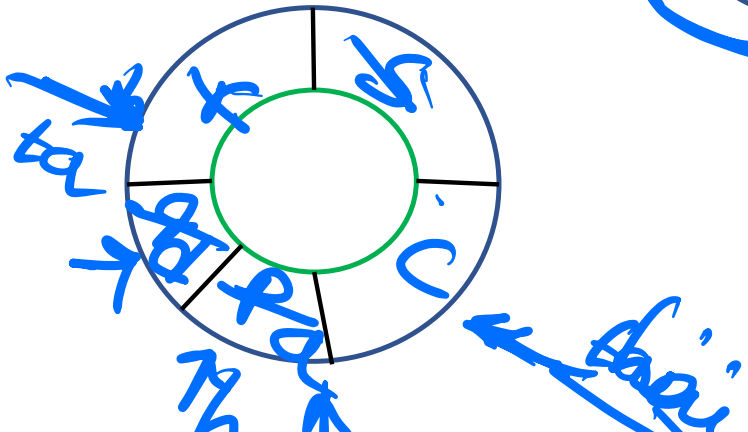
Exercise: Circular Queue (4)

What is status of states of queue contents after the following sequence of steps

Enqueue x
Dequeue
Enqueue y
Dequeue
Dequeue

a) x, y, _____, _____, _____
b) x, ____, y, _____, _____
c) _____, _____, x, y, _____
d) _____, x, y, _____, _____





thank you!

email:
k.kondepu@iitdh.ac.in

NEXT Class: 26/04/2022