# Queue Basics

Course on C-Programming & Data Structures: GATE - 2024 & 2025

**Vishvadeep Gothi** • Lesson 32 • Jan 7, 2023

# Data Structure:
# Queue 1

By: Vishvadeep Gothi

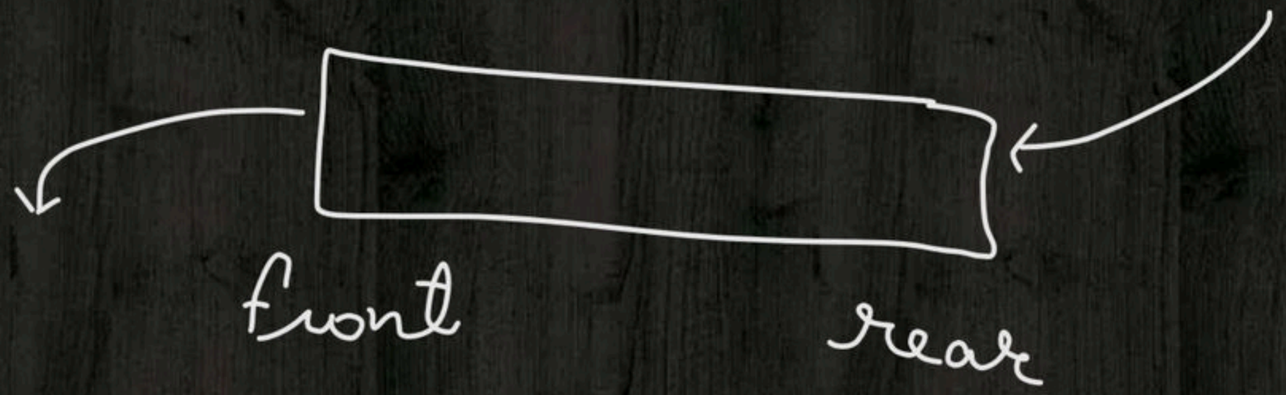
*Hello!*

# I am Vishvadeep Gothi

I am here because I love to teach

# Queue / FIFO list

It is a linear D.S., in which insertion is done from one end (rear end) and deletion is done from other end (front end).

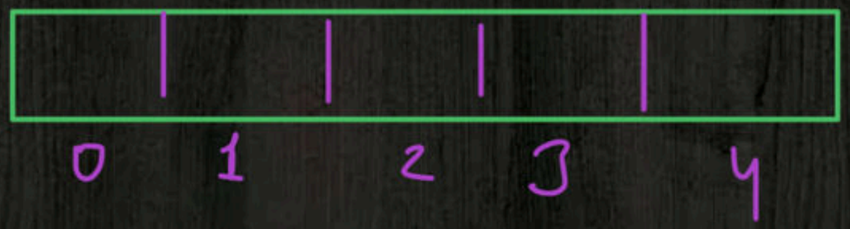


Insertion in queue => EnQueue

Deletn —— 11 —— => DeQueue

# Implementation of Queue: Using Array

n = 5 size array ⇒ Queue[5]




for insert ⇒ rear index changes

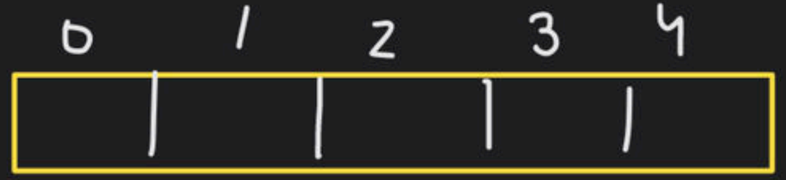
for delet ⇒ front —"— ———

2 index variables:-

Front ⇒ stores index of front element of Queue

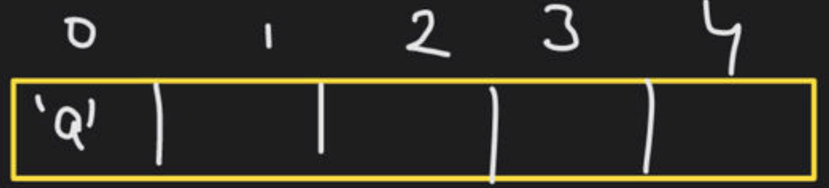
Rear ⇒ —"— ——— rear —"— ———

Front = rear = −1 (when Queue is empty)




$F = R = -1$

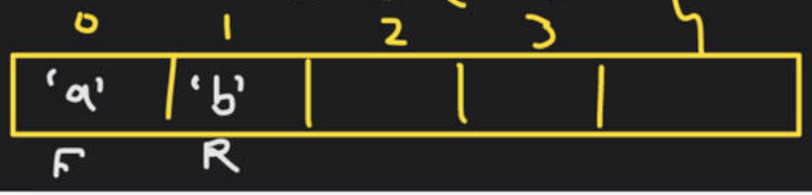
(1) EnQueue ('a')




$F = 0$

$R = 0$

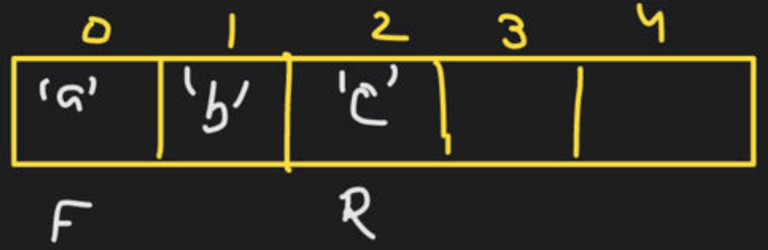
(2) EnQueue ('b')




$F = 0$

$R = 1$

(3) EnQueue ('c')

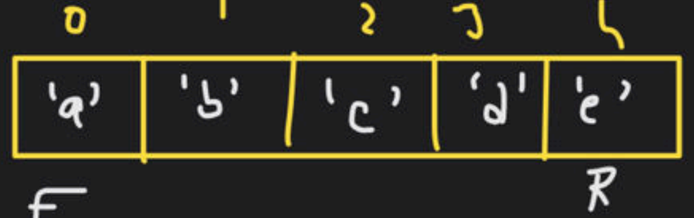



$F = 0$

$R = 2$

(4) EnQueue ('d')

(5) enQueue ('e')




$F = 0$

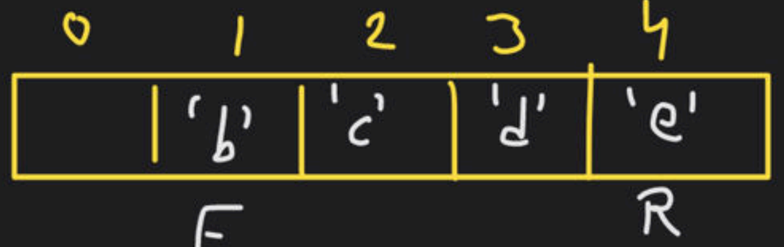
$R = 4$

⑥ EnQueue ('g')

↓↓

overflow condition ⇒ if (F == 0 and R == N-1)

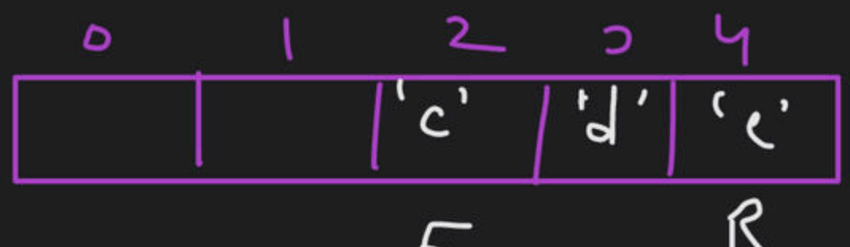
⑦ DeQueue()




F = 1

R = 4

⑧ DeQueue () :-




F = 2

R = 4

③ EnQueue('h') :-

## Linear Queue

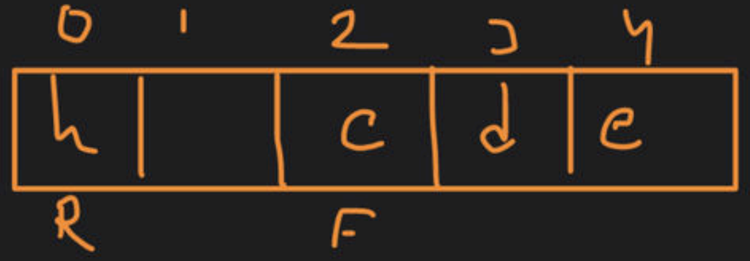
Insertion can be done on next index of 'Rear' linearly

⇓

Overflow

(x'ist better space utilizatⁿ in Linear Queue)

## Circular Queue

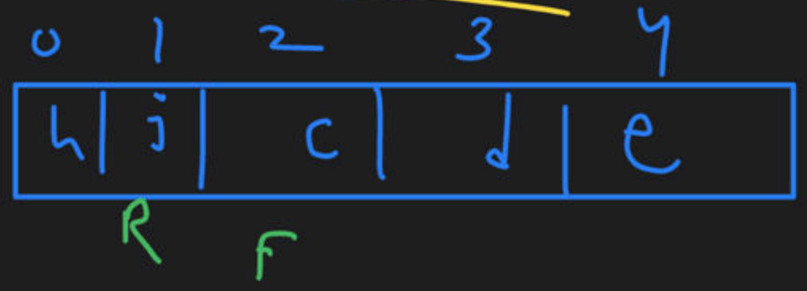



F = 2

R = 0

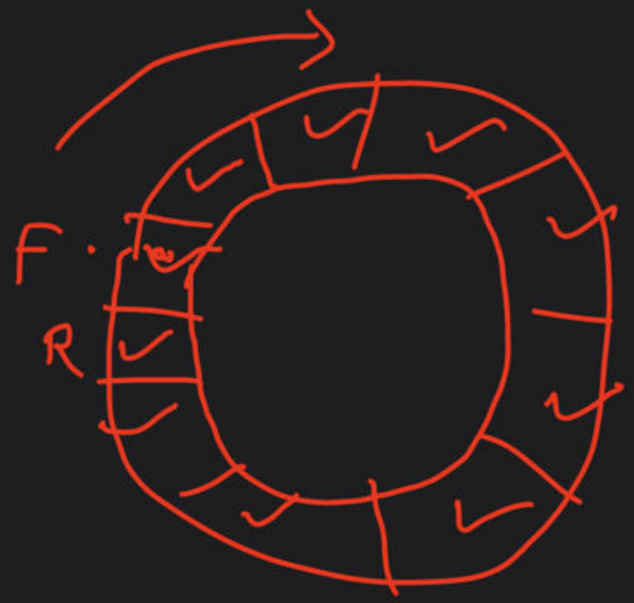
⇓

(10) EnQueue ('j') :-




F = 2

R = 1



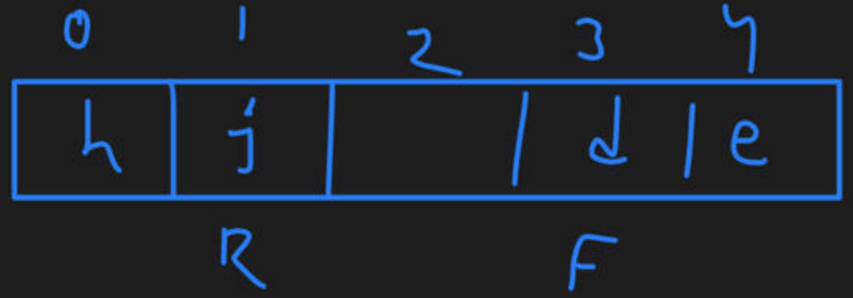

(11) EnQueue ('k') :-

⇓

Overflow

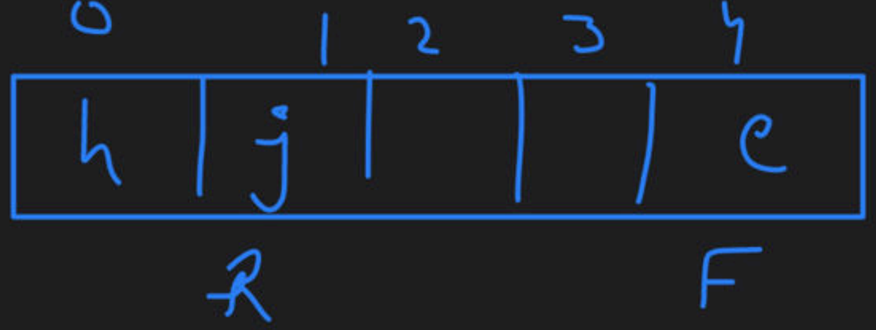
Front == (Rear + 1) % n

(12) Dequeue ():-



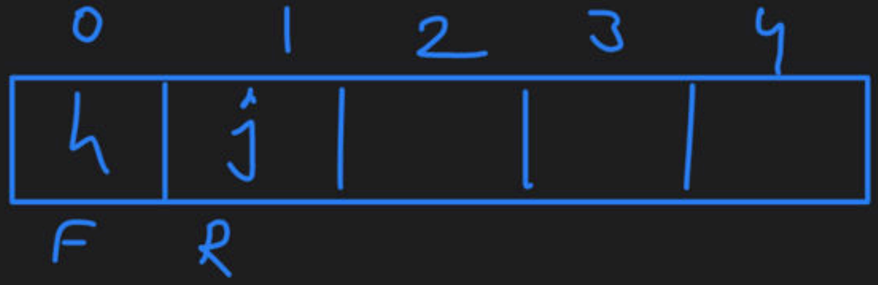

R = 1
F = 3

(13) Dequeue ():-



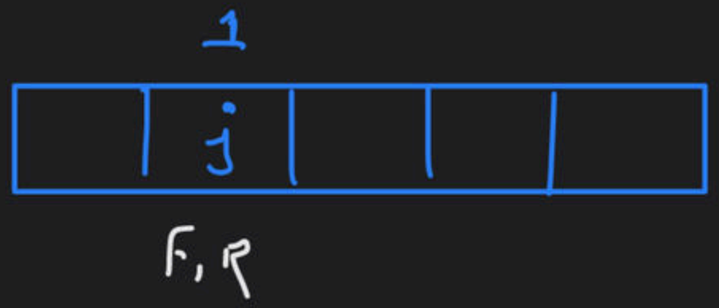

F = 4, R = 1

(14) Dequeue:-




F = 0
R = 1

(15) Dequeue:-




F = 1
R = 1

(16) Dequeue:-

F = -1
R = -1

```
Enqueue(item, queue[], n, Front, Rear)
{
1.  if (Front == (Rear + 1) % n)
        printf("overflow")
        return
2.  if (Front == Rear == -1)
        Front = Rear = 0
    else
        Rear = (Rear + 1) % n
3.  Queue[Rear] = item
}
```

R.T. complexity = $\Theta(1)$

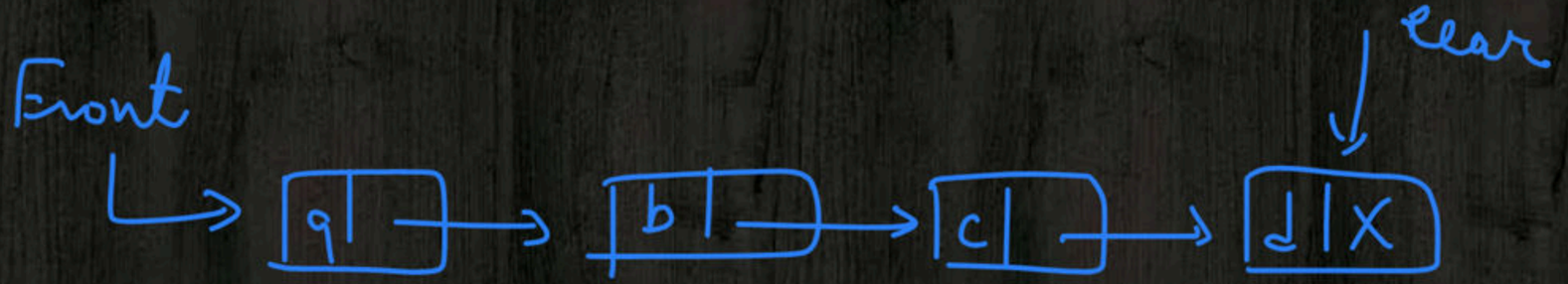
Space => $\mathcal{O}(1)$

```
Dequeue (Queue[], front, rear, n)
{
1.   if (front == rear == -1)
         print underflow
         return
2.   item = queue [front]
3.   if (front == rear)
         Front = Rear = -1
     else
         front = (front + 1) % n
}
```

R T. Complexity = $\theta(1)$

Space —"— = $\theta(1)$

# Implementation of Queue: Using Linked List




EnQueue() => Insert at the end $\Rightarrow \theta(1)$

DeQueue() => Delete from front $\Rightarrow \theta(1)$

# Question 1

- What is the content of queue after following operations on an empty queue?

1. Enqueue(a)
2. Enqueue(b)
3. Dequeue()
4. Enqueue(d)
5. Enqueue(e)
6. Dequeue()

# Question 2

What is the content of queue after following operations on an empty queue?

4 insertions

2 deletions

(A) First inserted 2 elements

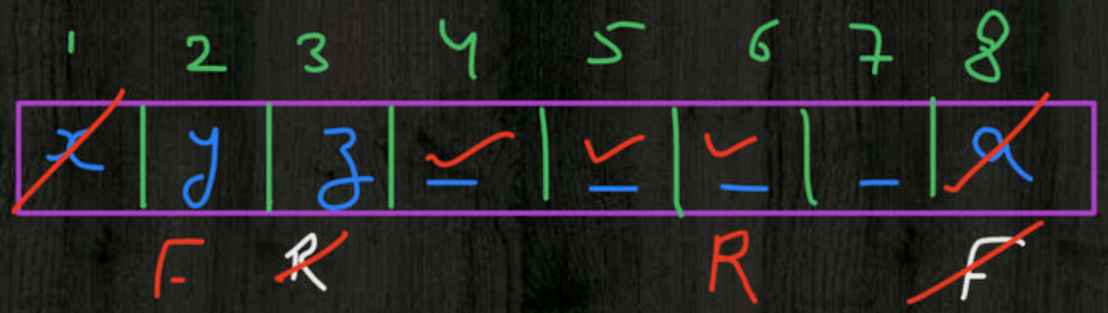
(B) Last inserted 2 elements

(C) Random 2 elements

(D) None

# Question 3

A circular queue is implemented using an array A[1:8]. The array contains values [x, y, z, _, _, _, _, a]. In the queue 3 enqueue and 2 dequeue operations are performed in random arbitrary order. What is the value of front and rear indexes?




F = 2
R = 6

Ans

F = 8

R = 3

# Other Functions on Queue

QueueFront()

QueueRear()

IsEmptyQueue()

# Question 4

Consider a linear queue Q. What would be the content of queue Q after the following code is executed?

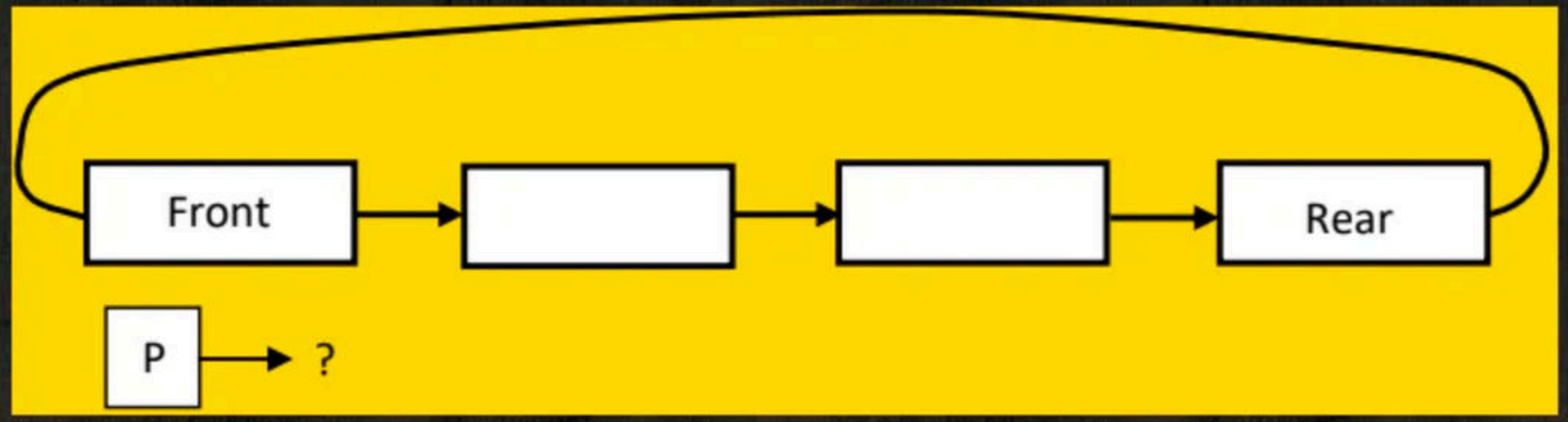
File = 4,7,5,9,0,4,7,8,3,5,0,7,0,8,9,0,5

```
loop(till end of file)
X= read number
if(x ! = 0)
        enqueue(Q, X)
else
        a = queueRear(Q)
        enqueue(Q, a)
End Loop
```

# GATE Question

CS- 2004

- A circularly linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue and deQueue can be performed in constant time?




(A) Rear node
(B) Front node
(C) Not possible with a single pointer
(D) Node next to front

# GATE Question

CS- 2018

- A queue is implemented using a non-circular singly linked-list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the number of nodes in the queue. Let *enqueue* be implemented by inserting a new node at the head, and *dequeue* be implemented by deletion of a node from the tail?


Which of the following is the time complexity of the most time-efficient implementation of *enqueue* and *dequeue*, respectively, for this data structure?

(A) $\theta(1), \theta(1)$

(B) $\theta(1), \theta(n)$

(C) $\theta(n), \theta(1)$

(D) $\theta(n), \theta(n)$

# Double Ended Queue

# Priority Queue

# Happy Learning