



SNS COLLEGE OF TECHNOLOGY

Coimbatore-35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai



**COURSE NAME : COURSE NAME : 23ITT101&PROGRAMMING IN C AND DATA
STRUCTURES**

I YEAR/ II SEMESTER

UNIT – IV STACK AND QUEUE

Topic: Queue ADT



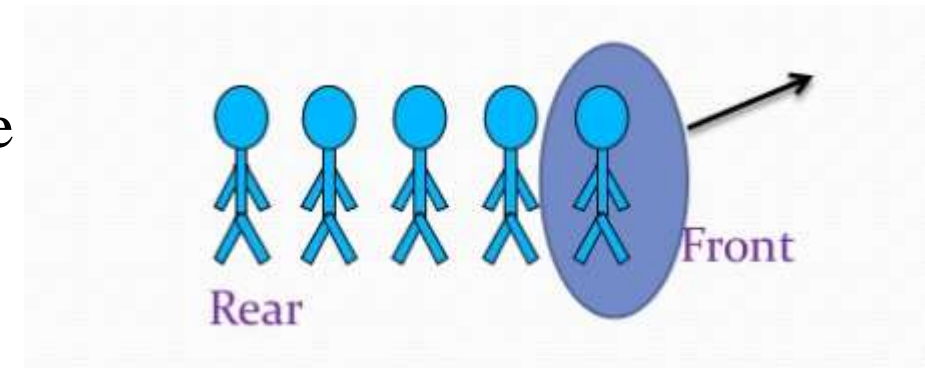
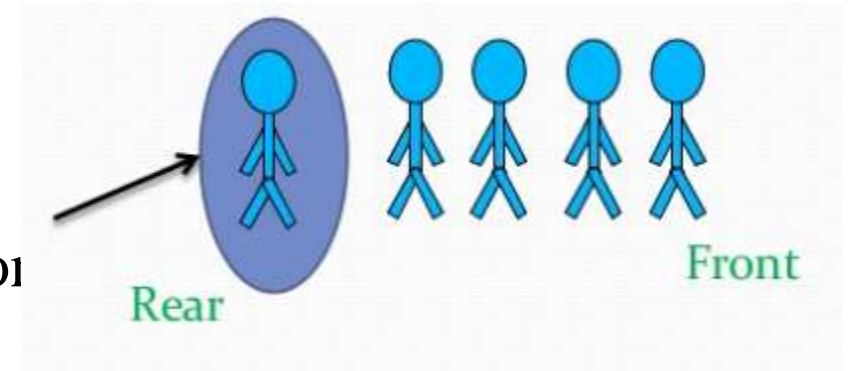
Guess





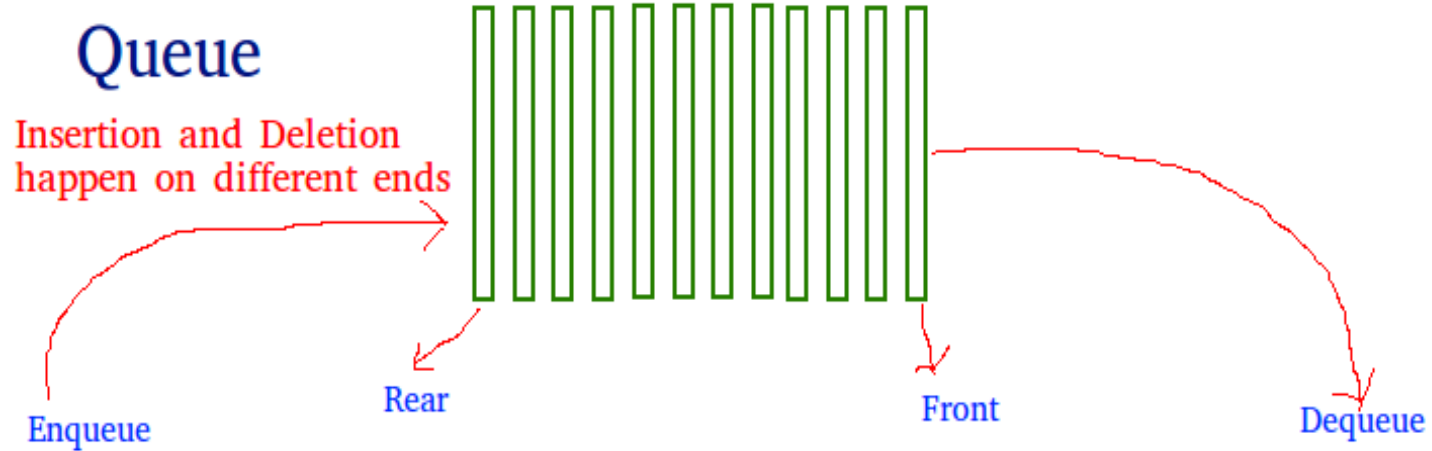
Queue -Introduction

- It is a Linear Data Structure
- Elements are in Sequential manner
- Follows FIFO-First in First out Mechanism
- Example : Standing on a line for Ticket reservation
- Two Operations
 - ENQUEUE-Inserting an element into the Queue
 - DEQUEUE-Deleting an element from the Queue
- Two Conditions
 - Overflow-Insertion into Queue which is full
 - Underflow-Deletion from Empty Queue





Introduction – Queue



First in, first out





Queue -Representation

* TWO ENDS

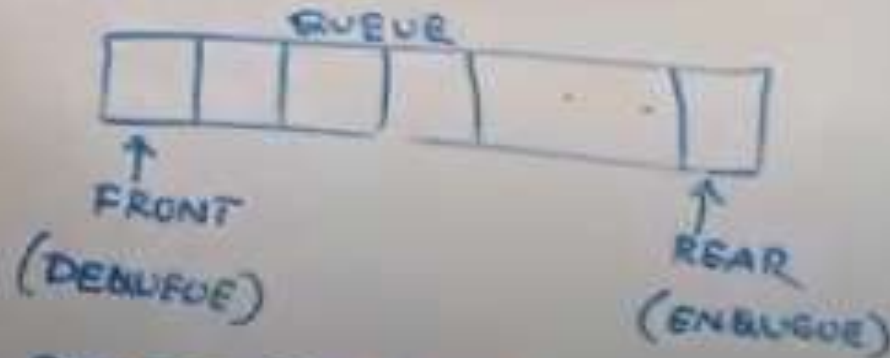
→ FRONT - Points to Starting Element

→ REAR - points to Last Element

* ENQUEUE will be done at REAR

* DEQUEUE will be done at FRONT

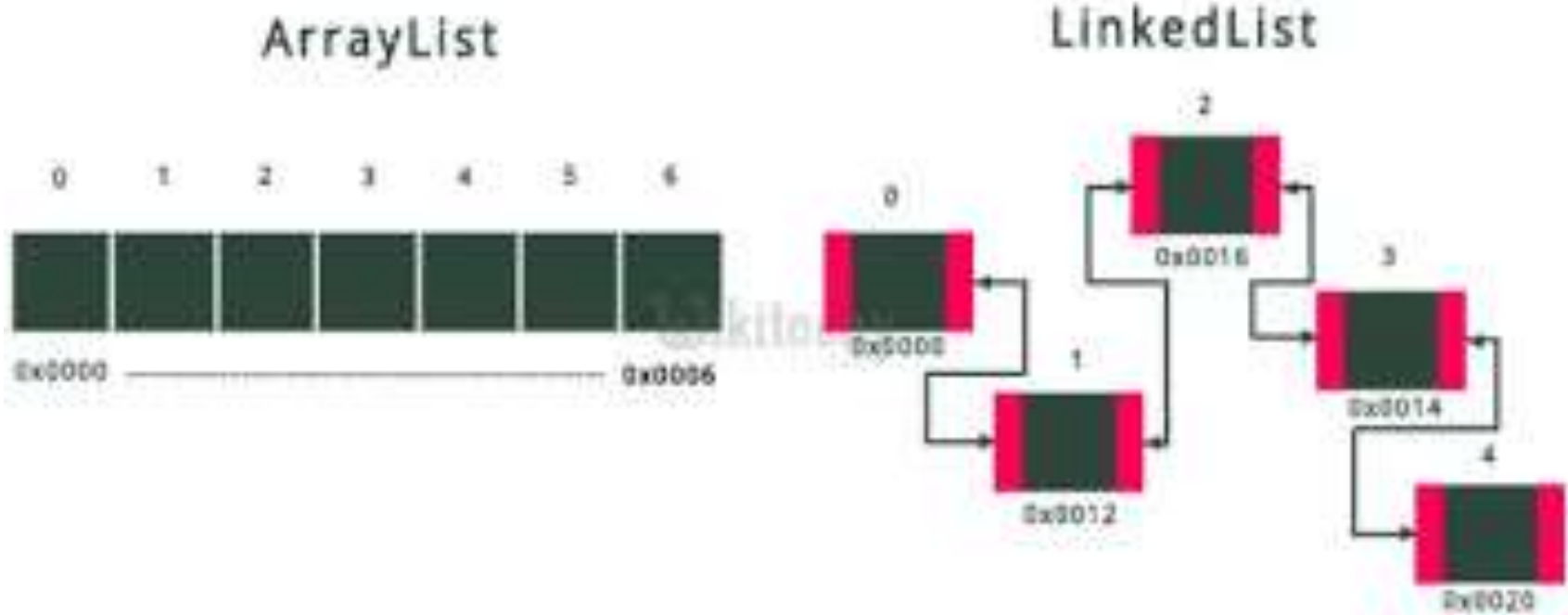
* Representation



* Initially $FRONT = REAR = -1$



Queue - Way of Implementation





Implementation Queue Using Array-Insertion

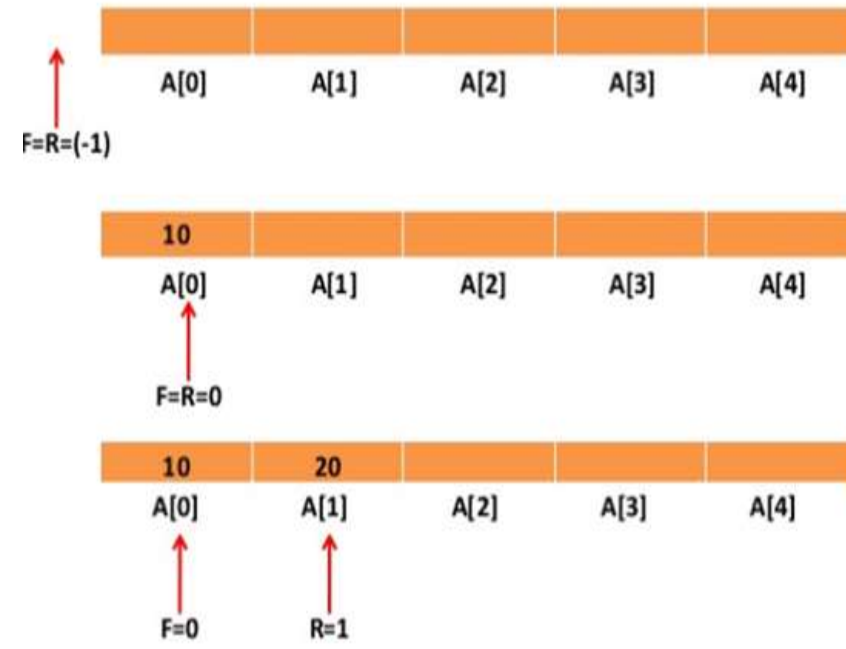
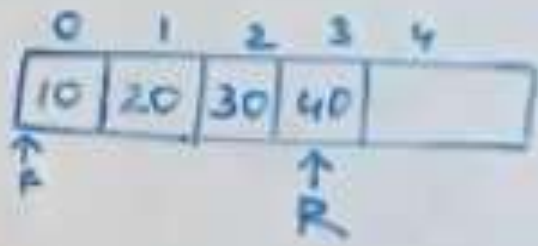


```

SIZE = 10 | QUEUE [ ] ;
FRONT = REAR = -1

ENQUEUE ( )
{
    if ( REAR == SIZE - 1 )
        printf ( " OVER FLOW " );
    else
    {
        scanf ( " %d ", &ele ); // - 10, 20, 30, 40
        if ( FRONT == - 1 )
            FRONT = 0;
        REAR++;
        QUEUE [ REAR ] = ele;
    }
}

```





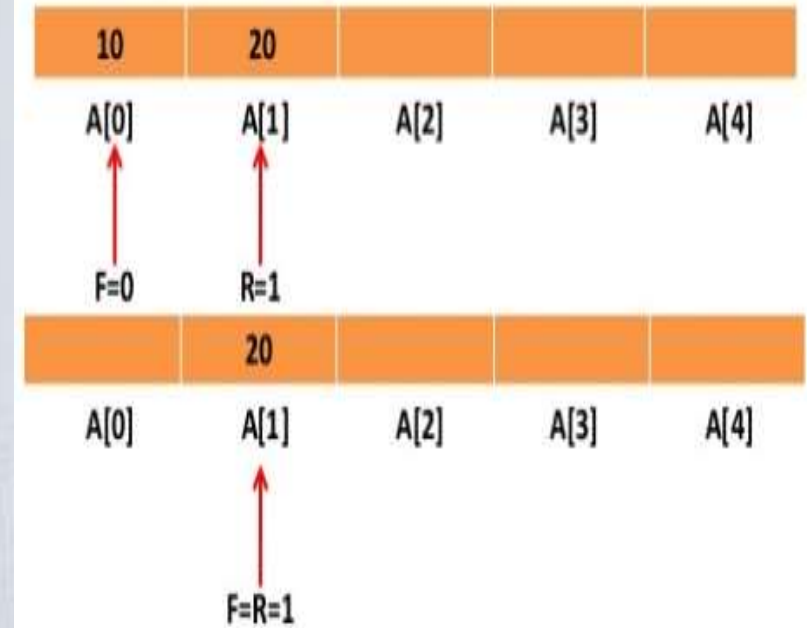
Implementation Queue Using Array-Deletion



```
SIZE = 10 | QUEUE [ ] ;
FRONT = REAR = -1

DEQUEUE ( )
{
    if ( REAR == -1 || FRONT > REAR )
        printf ( " UNDERFLOW " );
    else
    {
        ele = QUEUE [ FRONT ]
        printf ( " %d is deleted ", ele );
        FRONT ++ ;
    }
}
```

0 1 2 3 4
10 20 30 40
↑ R ↑ R
4 > 3
10x
20x
30x
40x





Implementation Queue Using Array-Display



```
SIZE = 10 | QUEUE [ ] ;
FRONT = REAR = -1

DISPLAY ( )
{
    if ( REAR == -1 || FRONT > REAR )
        PRINT ( " EMPTY QUEUE " );
    else
    {
        for ( i = FRONT ; i <= REAR ; i++ )
        {
            PRINT ( " %d " , QUEUE [ i ] );
        }
    }
}
```

Diagram illustrating the queue implementation using an array:

0	1	2	3	4
10	20	30	40	

Front (F) points to index 0, and Rear (R) points to index 3.

Verification of element insertion:

- $i = 0$ $0 <= 3$ ✓ $Q[0] = 10$
- $i = 1$ $1 <= 3$ ✓ $Q[1] = 20$
- $i = 2$ $2 <= 3$ ✓ $Q[2] = 30$
- $i = 3$ $3 <= 3$ ✓ $Q[3] = 40$
- $i = 4$ $4 <= 3$ ✗



Implementation Queue Using Linked List- Insertion

```
ENQUEUE ( )  
{  
  new = (struct *node) malloc ( sizeof (struct node) )  
  scanf ( "%d", &ele );  
  new -> data = ele;  
  new -> next = NULL;  
  if ( rear == NULL )  
  {  
    front = new;  
    rear = new;  
  }  
  else  
  {  
    rear -> next = new;  
    rear = new;  
  }  
}
```

struct node
{
 int data;
 struct node *next;
} *new, *rear, *front,
*temp;

rear = NULL
front = NULL

Diagram illustrating the insertion of a new node into a queue implemented as a linked list. The queue initially contains nodes with data 10 and 20. A new node with data 30 is being inserted. The front pointer points to the node with data 10, and the rear pointer points to the node with data 30. The next pointer of the node with data 20 is NULL.



Implementation Queue Using Linked List- Deletion

The image shows handwritten code and a diagram illustrating the deletion of an element from a queue implemented as a linked list. The diagram shows three nodes: Node 1 (data: 10, next: NULL) with 'rear' and 'temp' pointers; Node 2 (data: 20, next: 3000) with 'front' pointer; and Node 3 (data: 20, next: NULL) with 'temp' pointer. An arrow points from Node 2 to Node 3. The code defines a 'DEQUEUE()' function that checks if the queue is empty. If not, it updates 'temp' to 'front', moves 'front' to 'front->next', sets 'temp->next' to NULL, and frees 'temp'. A 'struct node' definition is also shown with fields 'int data;', 'struct node *next;', and pointers 'new, rear, front, temp;'. Finally, 'rear = NULL' and 'front = NULL' are set.

```
DEQUEUE()
{
    if (front == NULL)
        printf("QUEUE IS EMPTY");
    else
    {
        temp = front;
        front = front->next;
        temp->next = NULL;
        free(temp);
    }
}

struct node
{
    int data;
    struct node *next;
} *new, *rear, *front, *temp;

rear = NULL;
front = NULL;
```



Implementation Queue Using Linked List- Display



Diagram illustrating a queue implemented using a linked list. The queue contains three nodes: Node 1 (data: 10, next: 2000), Node 2 (data: 20, next: 3000), and Node 3 (data: 30, next: NULL). The front pointer points to Node 1, and the rear pointer points to Node 3.

```
graph LR
    Node1["10 | 2000"] --> Node2["20 | 3000"]
    Node2 --> Node3["30 | NULL"]
    front --> Node1
    rear --> Node3
```

Struct node

```
struct node
{
    int data;
    struct node *next;
} *new, *rear, *front, *temp;
```

front = 1000
temp = 1000 temp != NULL ✓

temp = 2000 temp != NULL ✓

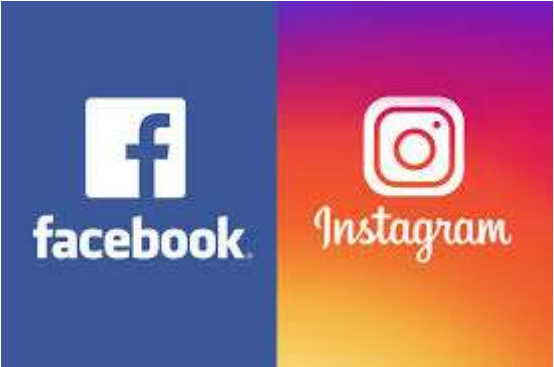
temp = 3000 temp != NULL ✓

temp = NULL temp != NULL ✓

```
DISPLAY()
{
    temp = front;
    if (front == NULL)
        printf("QUEUE IS EMPTY");
    else
    {
        while (temp != NULL)
        {
            printf("%d", temp->data);
            temp = temp->next;
        }
    }
}
```



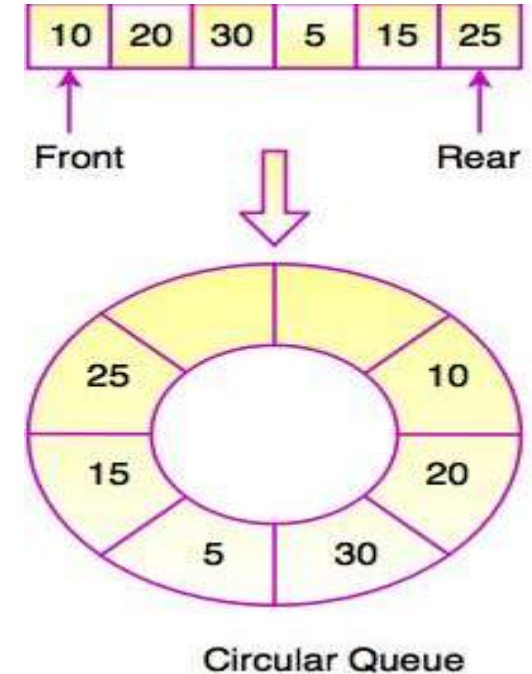
Applications





Circular Queue Implementation

- In a circular queue, all nodes are treated as circular. Last node is connected back to the first node.
- Circular queue is also called as **Ring Buffer**.
- It is an abstract data type.
- Circular queue contains a collection of data which allows insertion of data at the end of the queue and deletion of data at the beginning of the queue.
- One of the benefits of the **circular queue** is that **we** can make use of the spaces in front of the **queue**. In a normal **queue**, once the **queue** becomes full, **we** cannot insert the next element even if there is a space in front of the **queue**. But using the **circular queue**, **we** can use the space to store new values.

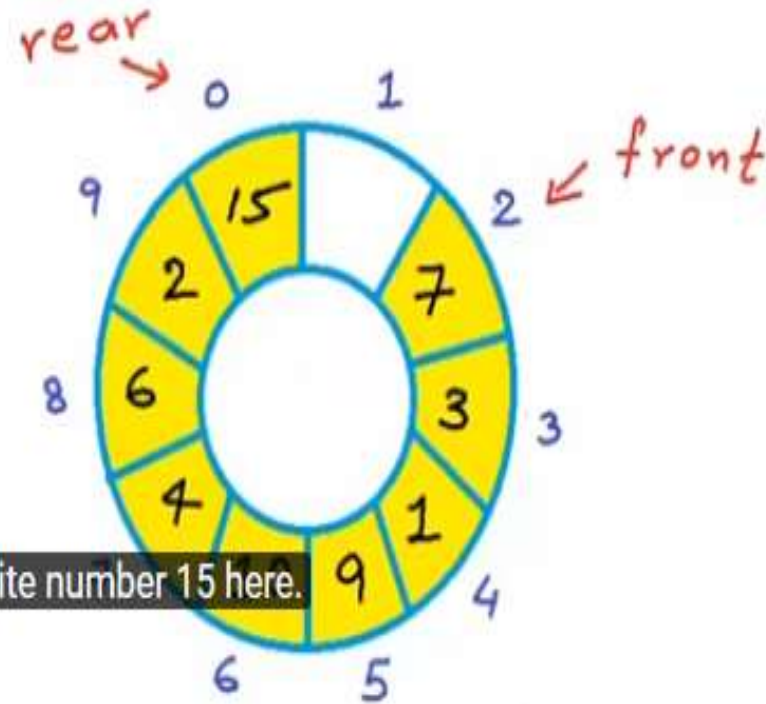




Circular Queue Implementation

Enqueue(x)

```
{ if (rear+1)%N == front
  return
else if isEmpty()
{ front ← rear ← 0
}
else
{ rear ← (rear+1)%N
}
A[rear] ← x
```



$(9+1) \% 10 = 0$ Enqueue(15)



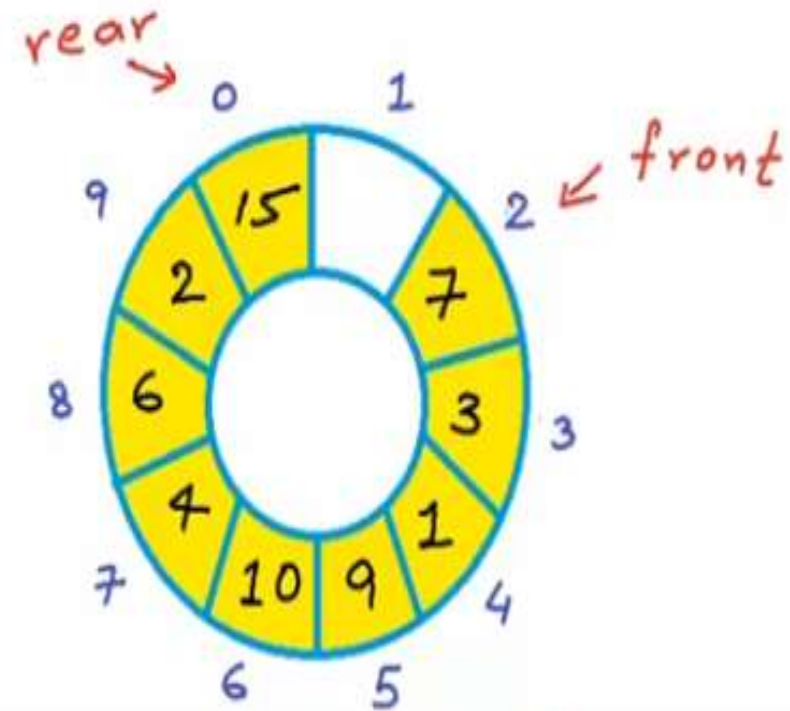


Circular Queue Implementation

Dequeue()

```
{  
  if IsEmpty()  
    return  
  else if front == rear  
    front ← rear ← -1  
  else  
    front ← (front + 1) % N  
}
```

$(2+1) \% \dots$



Enqueue(15)

Dequeue()





References

1. M. A. Weiss, “Data Structures and Algorithm Analysis in C”, Pearson Education, 2nd Edition, 2002.
2. A. V. Aho, J. E. Hopcroft and J. D. Ullman, “Data Structures and Algorithms”, Pearson Education, 2nd Edition, 2007
3. Ashok Kamthane, " Data Structures Using C ", Pearson Education, 2nd Edition, 2012.
4. Sahni Horowitz, “Fundamentals of Data Structures in C”Universities Press; Second edition 2008



Thank You