



Course Title – Data Structure Using C

Course Code - 313301

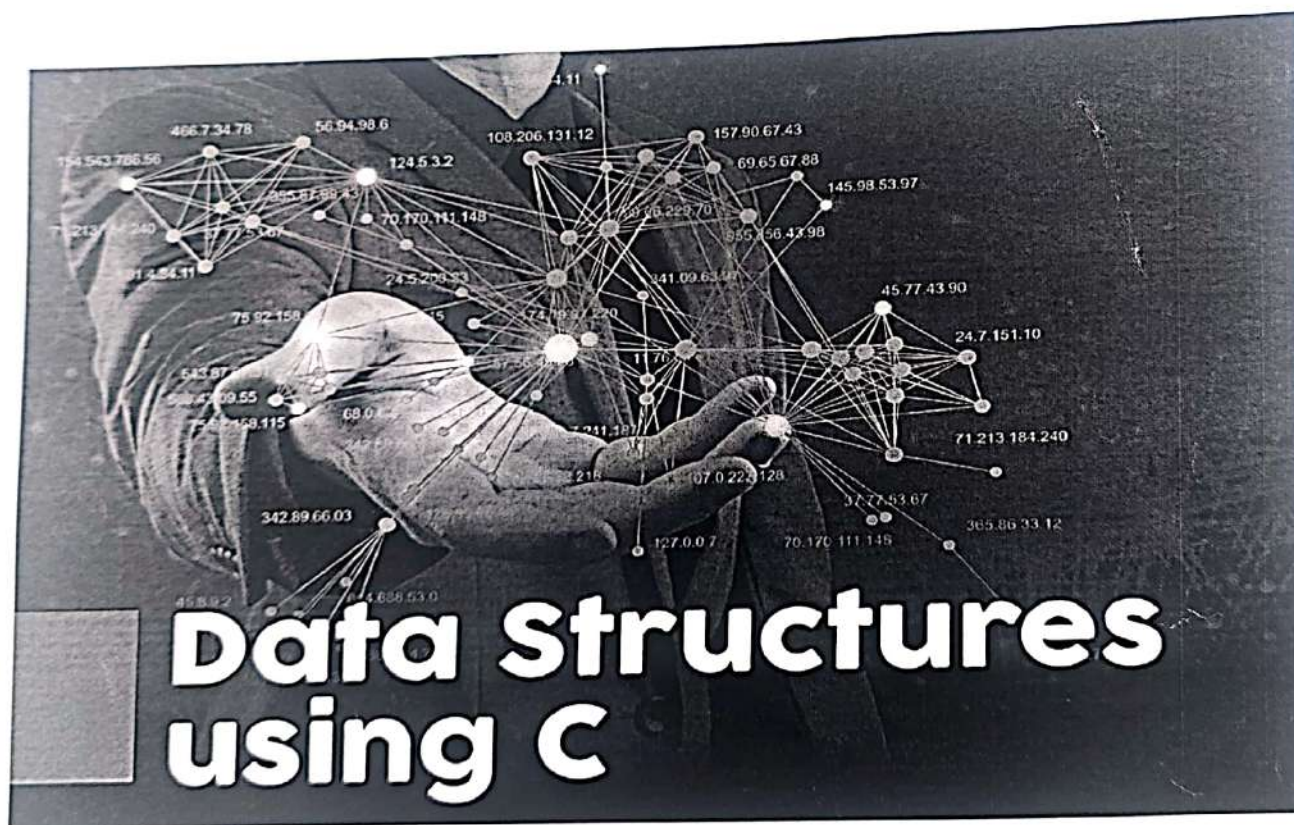
Programme Name - Computer Engineering

Semester - Third

### Unit - V Queue

Total Marks : 10

Prepared By: Mr. Milind R. Birhade



# UNIT - IV

## QUEUE

10-marks

### 5.1 Introduction to Queue:

Queue as an ADT

Queue representation in memory using Array and representation using linked list

### 5.2 Types of Queue:

Linear Queue

Circular Queue

concept of Priority queue

Double Ended Queue

### 5.3 Queue Operations:

Insert

Delete

Queue Operation Conditions:

Queue Full

Queue Empty

### 5.4 Applications of Queue

## 5- Queue

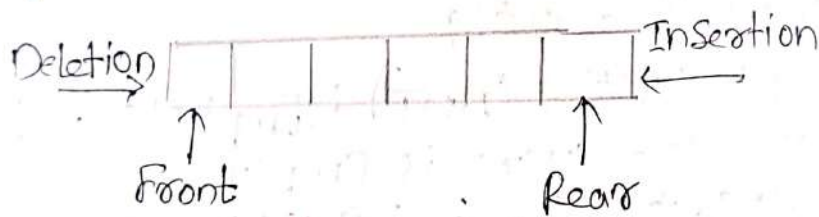
### Introduction:

It is a special kind of list where items are inserted at one end (the rear) and deleted from the other end (front).

Queue is a FIFO (first in first out) list.

The Queue is a linear data structure where operations of insertion and deletions are performed at separate ends that are known as front and rear.

We come across the term queue in our day to day life. For example, in a network where many computers share only few printers, the print jobs may accumulate in a print queue.



### Applications of Queue:

- Scheduling of processes (Round Robin algorithm)
- Spooling (to maintain a queue of jobs to be printed)
- A queue of client processes waiting to receive the service from the server process.
- Various application software using non-linear data structure tree or graph requires a queue for breadth first traversal.

Queue as an ADT: An Abstract Data type (ADT) defines the operations performed on a data structure without specifying how they are implemented.

Data Structure Declaration:

```
typedef struct Q
{
    int R, F;
    int data[Max];
} Q;
```

R & F stores the indices of the rear & the front element respectively. The array data[] is used to store the queue elements. Max is predefined constant.

Prototype of queue functions

- 1] void initialize(Q\*);
- 2] int empty(Q\*); (ADT) isEmpty() - Check whether the queue is Empty
- 3] int full(Q\*); (ADT) isFull() - Check whether the queue is Full (for array elements)
- 4] void enqueue(Q\*, int); ADT Insert (Enqueue) add an Element at Rear
- 5] int dequeue(Q\*); ADT Delete (Dequeue) Remove an element from the front
- 6] void print(Q\*);
- 7] peek(front) - view the first element.

During initialization of queue

# define max 5

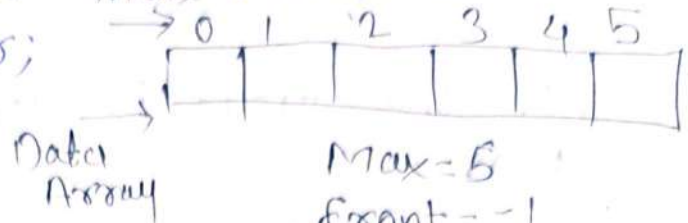
typedef struct queue

{

int data[max]; index front & Rear = -1

int front, rear;

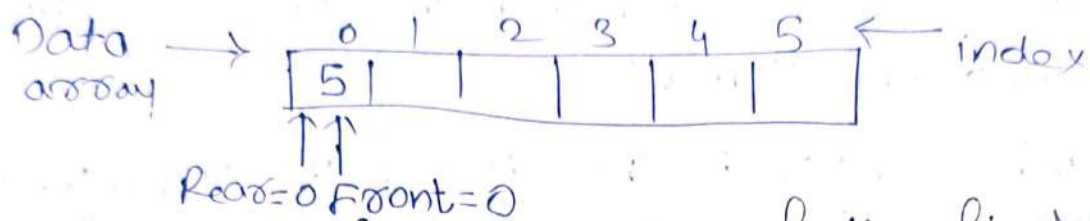
} queue;



front = -1

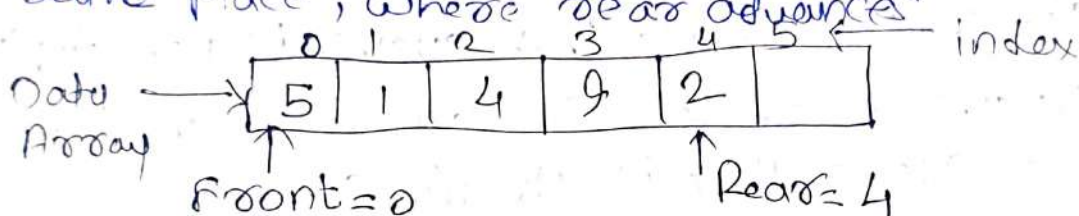
Rear = -1

An Empty queue after initialization

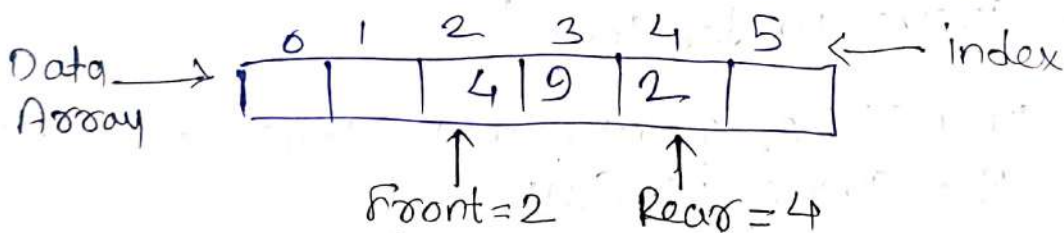


A queue. After insertion of the first element '5'

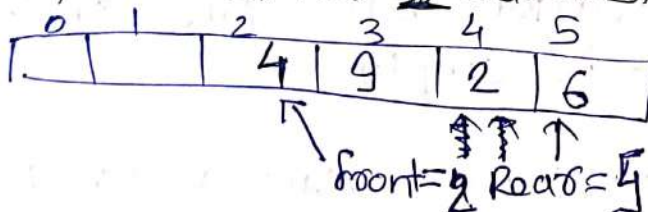
On subsequent insertions, front remains at the same place, where rear advances



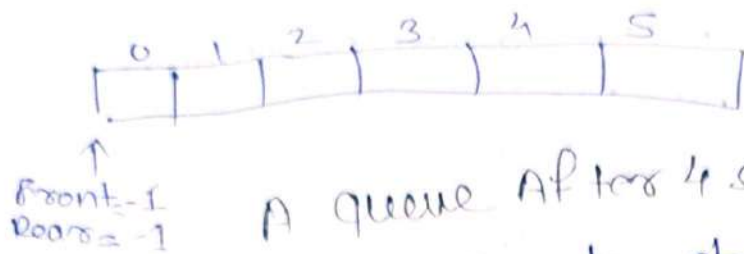
A queue after insertion of 5 elements '2'.



A queue after 2 successive deletion



A queue after 1 successive insertion



A queue After 4 successive deletion

Following points can be observed

1] If the queue is empty then  $\text{Front} = -1$  &  $\text{Rear} = -1$

2] If the queue is full then  $\text{Rear} = \text{Max} - 1$

3] If  $\text{Rear} = \text{Front}$  then the queue contains just one element

4] where Max is the size of the array used for storing of queue Elements.

Queue overflow is caused by insertion in a queue that is already full point no(2)

5] If  $\text{Rear} > \text{Front}$  then queue is non-Empty  
Queue overflow is caused by insertion in a queue that is already full.

Deletion from an empty queue will cause a queue underflow

Enqueue() operation will cause an overflow if the queue is full.

Dequeue() operation will cause an underflow if the queue is empty

we, already discuss in Queue as an AOT

## Stack

1. Stack is a LIFO structure Last In First out
2. In stack all insertions & deletions are permitted only at one end of the list
3. Stack is always implemented using a linear array
4. Stack is widely used in recursion and processing of expressions.

## Queue

- Queue is a FIFO Structure First In First Out
- In queue items are inserted at one end (the rear) and deleted from the other end (the front).
- Queue should preferably be implemented using circular array
- Queue is more useful in many of the real life applications.

## Program for insert and delete Operation on queue

```
#include <stdio.h>
#define MAX 5
int queue[MAX];
int front = -1;
int rear = -1;

void insert()
{
    int item;
    if (rear == MAX - 1)
    {
        printf("Queue Overflow\n");
        return;
    }
    printf("Enter Element: ");
    scanf("%d", &item);
    if (front == -1)
        front = 0;
    rear++;
    queue[rear] = item;
    printf("Element Inserted\n");
}

void delete()
{
    if (front == -1 || front > rear)
    {
        printf("Queue Underflow\n");
        return;
    }
    printf("Deleted Element = %d\n", queue[front]);
    front++;
    if (front > rear)
    {
        front = -1;
        rear = -1;
    }
}

void display()
{
    int i;
    if (front == -1)
    {
        printf("Queue is Empty\n");
        return;
    }
    printf("Queue Elements:");
    for (i = front; i <= rear; i++)
    {
        printf("%d", queue[i]);
        printf("\n");
    }
}

int main()
{
    int choice;
    do
    {
        printf("\n1. Insert");
        printf("\n2. Delete");
        printf("\n3. Display");
        printf("\n4. Exit");
    }
}
```

```

printf("\nEnter choice: ");
scanf("%d", &choice);
switch (choice)
{
    case 1: insert();
            break;
    case 2: delete();
            break;
    case 3: display();
            break;
    case 4: printf("Program Ended\n");
            break;
    default: printf("Invalid choice\n");
}
while (choice != 4);
return 0;
}

```

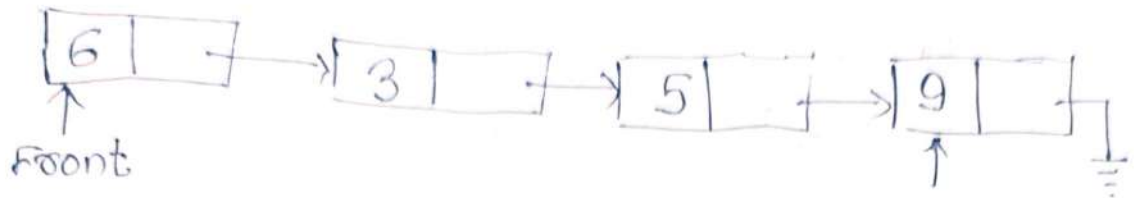
## Operations on queue

- 1] initialize(): Initializes a queue by setting the value of rear and front to -1.
- 2] enqueue(): Inserts an element at the rear end of the queue.
- 3] dequeue(): Deletes the front element and returns the same.
- 4] empty(): It returns true(1) if the queue is empty and returns false(0) if the queue is not empty.
- 5] full(): It returns true(1) if the queue is full and returns false(0) if the queue is not full.
- 6] print(): Printing of queue elements.

## Linked Representation of a queue

A queue can be represented by a linked structure of elements. Each element stored in a node.

Memory area for a node can be acquired during runtime through the C function "malloc".



### Linked representation of a queue.

A queue can be declared as follows

```
typedef struct node
{
    int data;
    struct node *next;
} node;
```

```
typedef struct Q
{
    node *R;
    node *F;
} Q;
```

Structure node, has two fields:

1] data  $\rightarrow$  stores Queue Elements

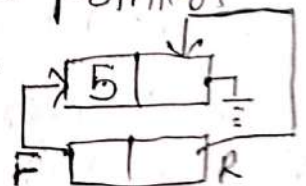
2] next  $\rightarrow$  stores the address of the next node holding the next element of the queue.

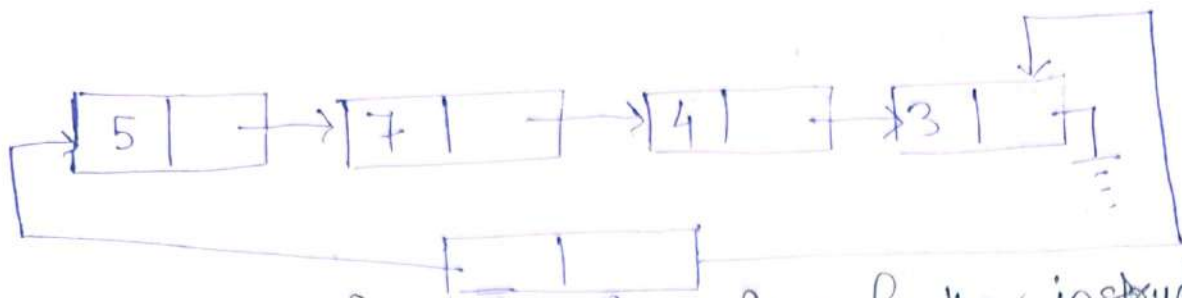
Structure queue [Q] has two fields.

R  $\rightarrow$  address of the rear node.

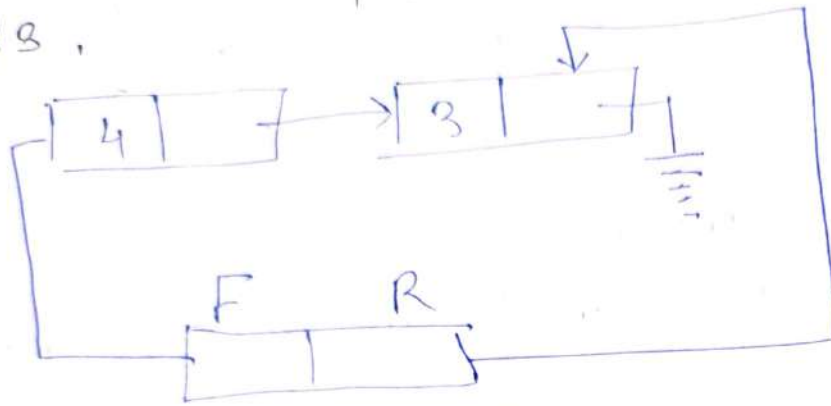
F  $\rightarrow$  address of the front node.

when a queue is initialized both its pointers "R" & "F" should be set to NULL.





Status of the queue after further instructions of 7, 4 & 3.



Status of the queue after two successive deletions.

## Difference between Array & Linked list Representation

| Array Representation                                                                        | Linked List Representation                                                               |
|---------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| 1. Fixed size (size is defined in advance using <code>#define MAX</code> )                  | Dynamic size (memory is allocated as needed).                                            |
| 2. uses contiguous memory locations.                                                        | uses non-contiguous memory locations.                                                    |
| 3. Easy & Simple to implement                                                               | slightly more complex because pointers are used.                                         |
| faster access using array index                                                             | Elements are accessed sequentially through links.                                        |
| memory wastage may occur in a linear queue after deletion<br>ex. <code>int queue[5];</code> | No memory wastage due to dynamic memory allocation<br>ex. Nodes connected using pointers |

# TYPES of Queue

- 1] Linear Queue.
- 2] Circular Queue.
- 3] Priority Queue.
- 4] Dequeue

**Linear Queue:** It can be implemented by using either arrays or linked list

A Linear queue has two basic operation insert and delete.

The insert operation adds an element to the end of the queue, and in the queue delete operation removes an element from the front or the start of the queue.

## Algorithm for Insert

```
Step 1: IF REAR = MAX - 1
    Write OVERFLOW
    GOTO Step 4
    [End of IF]

Step 2: IF FRONT = -1 and REAR = -1
    Set FRONT = REAR
ELSE
    SET REAR = REAR + 1
    [End of IF]

Step 3: SET QUEUE[REAR] = NUM

Step 4: EXIT
```

## Algorithm for Delete

```
Step 1: IF FRONT = -1 OR FRONT > REAR
    Write UNDERFLOW
ELSE
    SET VAL = QUEUE[FRONT]
    Set FRONT = FRONT + 1
    [END OF IF]

Step 2: EXIT
```

## 2. Circular Queue

A Circular queue is a queue whose start and end locations are logically connected with each other. That means, the start location comes after the end location.

If we continue to add elements in circular queue till its end location, then after the end location has been filled, the next element will be added at the beginning of the queue.

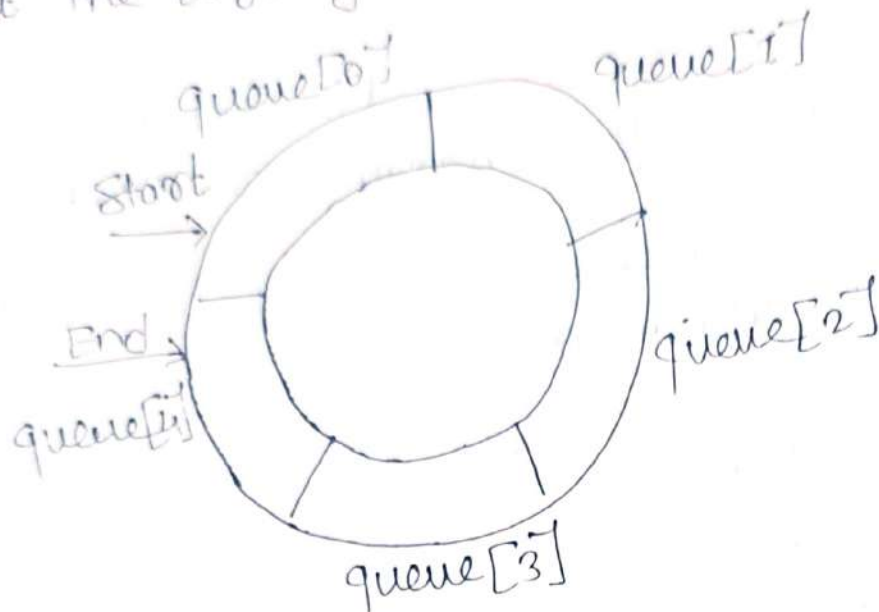


fig: circular queue

Circular queues remove one of the main drawbacks of array implemented queues in which a lot of memory space is wasted due to inefficient utilization.

**Insert operation:** 1] checking whether the queue is already full.

2] updating the rear pointer

a] If the queue is empty, set front and rear to point to the first location in the queue.

b] If rear is pointing at the last location of the queue, set rear to point to the first location in the queue.

c] If none of the situations exist, simply increment the rear pointer by 1

5] Inserting the new element at the rear location

### Algorithm

- Step 1: Start
- Step 2: if (front = 0 and rear = Max-1) goto or front = rear  
goto step 3 else goto step 4
- Step 3: Display message, "queue is full" and goto step 10
- Step 4: If front = Null goto step 5 else goto step 6
- Step 5: Set front = rear = 0
- Step 6: If rear = Max-1 goto step 7 else goto step 8
- Step 7: Set rear = 0
- Step 8: Set rear = rear + 1
- Step 9: Set queue[rear] = element
- Step 10: Stop

### Delete Operation

- 1] checking whether the queue is already empty
- 2] Retrieving the element at the front of the queue
- 3] Updating the front pointer
  - a] If the queue has only one element left, set front and rear to point to Null.
  - b] If front is pointing at the last location of the queue, set front to point to the first location in the queue.
  - c] If none of the above situations exist, simply increment the front pointer by 1
- 4] Returning the element retrieved from the front location.

### Algorithm

- Step 1: Start
- Step 2: if front = Null goto step 3 else goto step 4
- Step 3: Display message, "Queue is empty" and goto step 13
- Step 4: Set i = queue[front]
- Step 5: if front = rear goto step 6 else goto step 6
- Step 6: Set front = rear = Null

Step 7: Return the deleted element  $i$  and goto step 13  
 Step 8: if  $front = Max - 1$  goto step 9 else goto step 11  
 Step 9: set  $front = 0$   
 Step 10: Return the deleted element  $i$  and goto step 13  
 Step 11: Set  $front = front + 1$   
 Step 12: Return the deleted element  $i$   
 Step 13: stop.

## Concept of Dequeues : DOUBLE-ENDED QUEUE

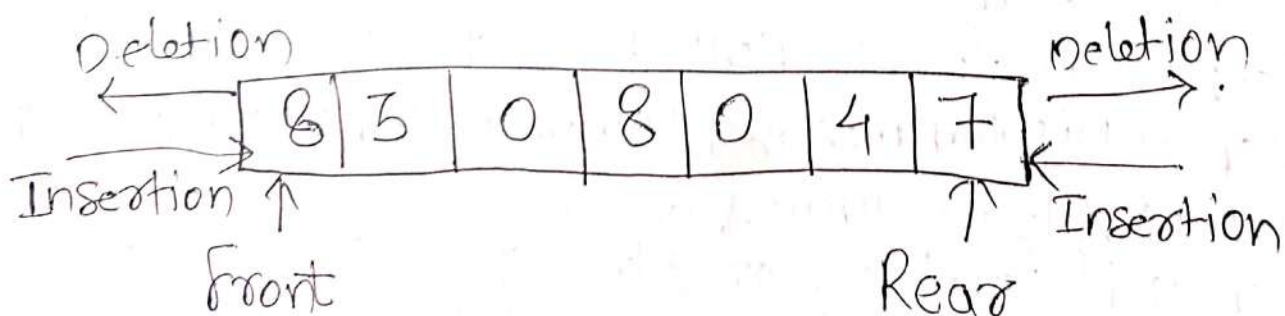
The word dequeue is short form of double ended queue. It is general representation of both stack and queue and can be used as stack and queue.

In a dequeue is a short form of double ended queue, insertion as well deletion can be carried out either at the rear end or the front end.

Based on the type of restrictions imposed on insertion and deletion of elements, a double-ended queue is categorized into two types.

1] Input-restricted deque: It allows deletion from both the ends but restricts the insertion at only one end.

2] Output-restricted deque: It allows insertion at both the ends but restricts the deletion at only one end.



# Applications

- 1] CPU Scheduling: The operating system uses a queue to schedule process waiting for CPU execution (e.g, first come first served)
- 2] Printer Spooling: Print jobs are stored in a queue and printed in the order they are received.
- 3] Breadth-First Search (BFS): A queue is used in graph and tree traversal to visit nodes level by level.
- 4] Waiting Line Management: Queues are used in banks, hospitals, ticket booking system and customer service centers to serve customers in the order of arrival (FIFO).
- 5] Network packet scheduling in routers
- 6] Call Center management
- 7] Josephus Problem
- 8] Job Scheduling
- 9] Queue Simulation

Bidul

Shub  
10/7/21

## Operations on Dequeue

- 1] `empty()`: whether the queue is empty
- 2] `Full()`: whether the queue is full
- 3] `initialize()`: Make the queue empty
- 4] `enqueueR()`: add item at the rear end of the queue
- 5] `enqueueF()`: add item at the front end of the queue
- 6] `dequeueR()`: Delete item from the rear end of the queue
- 7] `dequeueF()`: Delete item from the front end of the queue

**Priority Queue:** A priority queue is a special type of queue where every element is given a specific priority. Instead of just waiting in a standard line, the order in which elements are processed or removed depends entirely on their priority level.

Elements are removed from a priority queue based on following two simple rules.

**Rule 1:** (By Priority): Elements with the highest priority are always processed and removed first, ahead of any elements with lower priority.

**Rule 2:** (By Arrival): If two elements have the exact same priority, they are processed based on who arrived first (First come first serve)  
FIFO - first in first out.

## Question Bank

DSU 313301 Unit-5 Queue 10-M

1] Write a 'c' program to implement a queue with insert and delete operation

[S-26] [W-15] [S-17] [W-25] 6M

2] write a 'c' program to insert into and delete the element from linear queue.

[W-24] [S-25] 6M.

3] Show the effect of insert and delete operations on to the linear queue of size 10. The linear queue sequentially contain 11, 22, 33, 44 and 55 where 11 is at front of queue.

Show diagrammatically the effect of

(i) Insert (60).

(ii) Insert (70).

(iii) Delete.

(iv) Insert (80). [S-25]

4] List types of Queue data structure [S-25] 2M

5] List any four applications of queue [W-25] 2M

6] With a neat sketch explain working of Priority queue. [W-25] 4M.

7] Show the effect of Insert and Delete operations on linear queue of size 10. the linear queue sequentially contains 10, 20, 30, 40 and 50 where 10 is at front of queue. Show diagrammatically the effect of

(i) Insert (30) (ii) DELETE (iii) INSERT (45)  
(iv) INSERT (60).

8] Differentiate between stack and queue.  
[W-24] 2M

9] Explain types of queue and implement a program for circular queue [W-24] 4M

10] Describe queue implementation using linked list [W-24] 6M.

11] Define circular queue with example.  
[S-26] 2M

12] Compare linear queue and circular queue  
[S-26] 4M

13] Explain application of queue with example.  
[S-26] 4M

14] Describe how queue works as an abstract data type. [W-12]

15] Define Queue. Enlist queue operation & Explain it [S-15, 19].

15] Define following terms:

(i) queue as an abstract data type

(ii) Queue Overflow (Queue Full)

(iii) Queue Underflow (Queue Empty)

(iv) Front

(v) REAR

(vi) Priority Queue.

[6-M]



