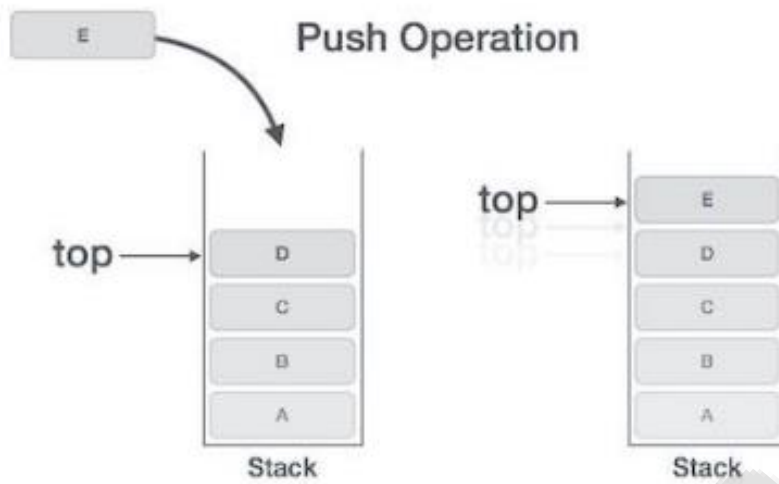


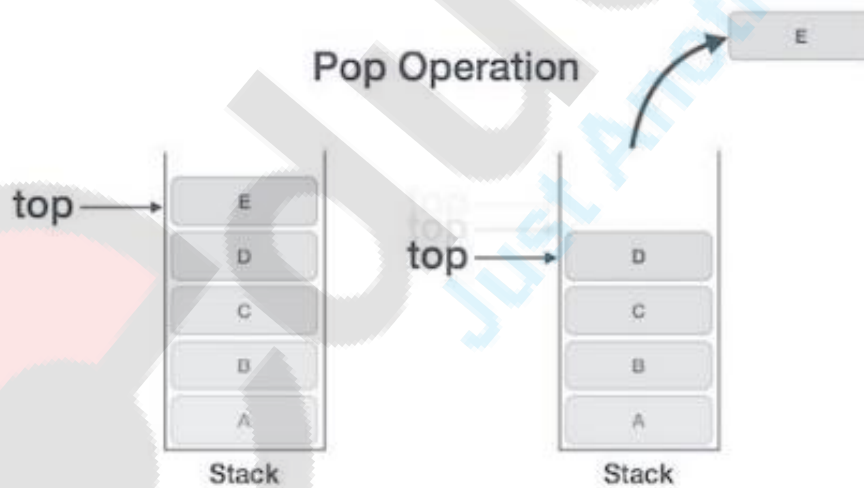
Q.1	Explain the term stack. Explain its Operations: Push and Pop.
Ans.	<p><b><u>Stack:</u></b></p> <p>Stack is an abstract data type with a bounded(predefined) capacity. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack, the only element that can be removed is the element that was at the top of the stack, just like a pile of objects.</p> <p><b>Basic features of Stack</b></p> <ol style="list-style-type: none"><li>1. Stack is an ordered list of similar data type.</li><li>2. Stack is a LIFO structure. (Last in First out).</li><li>3. push() function is used to insert new elements into the Stack and pop() is used to delete an element from the stack. Both insertion and deletion are allowed at only one end of Stack called Top.</li><li>4. Stack is said to be in Overflow state when it is completely full and is said to be in Underflow state if it is completely empty.</li></ol> <p><b>A stack is an abstract data type (ADT) that supports the following operations:</b></p> <p>push(e): Insert element e at the top of the stack. pop(): Remove the top element from the stack; an error occurs if the stack is empty. top(): Return a reference to the top element on the stack, without removing it; an error occurs if the stack is empty. Additionally, let us also define the following supporting functions: size(): Return the number of elements in the stack. empty(): Return true if the stack is empty and false otherwise.</p> <p><b><u>Push Operation:</u></b></p> <p>The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps –</p> <p>Step 1 – Checks if the stack is full.</p> <p>Step 2 – If the stack is full, produces an error and exit.</p> <p>Step 3 – If the stack is not full, increments top to point next empty space.</p> <p>Step 4 – Adds data element to the stack location, where top is pointing.</p> <p>Step 5 – Returns success.</p>



### Pop Operation

Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and deallocates memory space.

**A Pop operation may involve the following steps –**



Q2. What is Queue ? Explain its operations: Enqueue and Dequeue.

Ans.

**Queue:**

Queue is also an abstract data type or a linear data structure, in which the first element is inserted from one end called REAR(also called tail), and the deletion of existing element takes place from the other end called as FRONT(also called head). This makes queue as FIFO data structure, which means that element inserted first will also be removed first.

The process to add an element into queue is called Enqueue and the process of removal of an element from queue is called Dequeue.

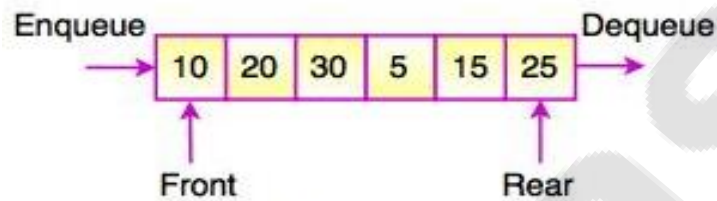


Fig. Queue

**Basic features of Queue**

1. Like Stack, Queue is also an ordered list of elements of similar data types.
2. Queue is a FIFO( First in First Out ) structure.
3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element.
4. peek( ) function is oftenly used to return the value of first element without dequeuing it

**Basic Operations**

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues –

enqueue() – add (store) an item to the queue.

dequeue() – remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are –

peek() – Gets the element at the front of the queue without removing it.

isfull() – Checks if the queue is full.

isempty() – Checks if the queue is empty.

**EnQueue:**

Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue –

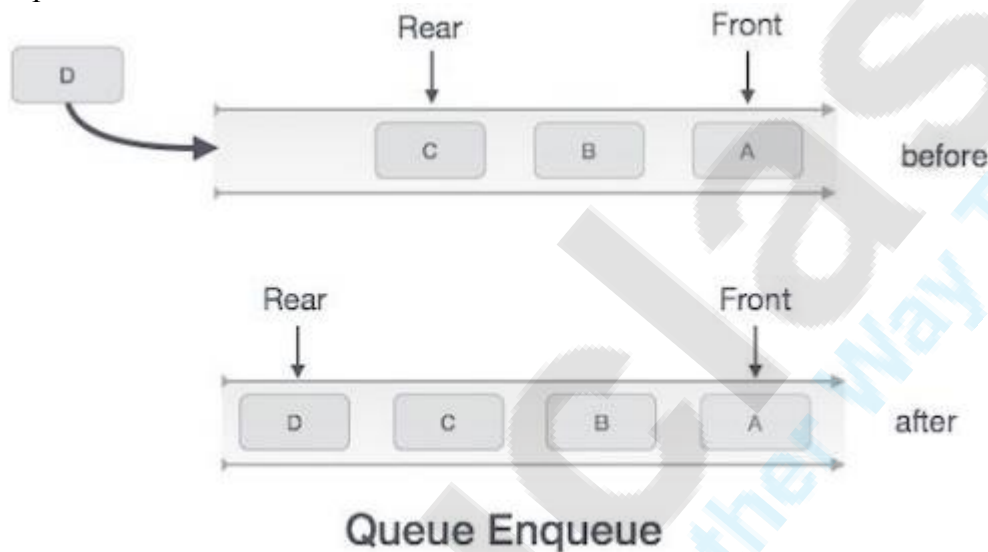
Step 1 – Check if the queue is full.

Step 2 – If the queue is full, produce overflow error and exit.

Step 3 – If the queue is not full, increment rear pointer to point the next empty space.

Step 4 – Add data element to the queue location, where the rear is pointing.

Step 5 – return success.



### **Dequeue:**

Accessing data from the queue is a process of two tasks – access the data where front is pointing and remove the data after access. The following steps are taken to perform dequeue operation –

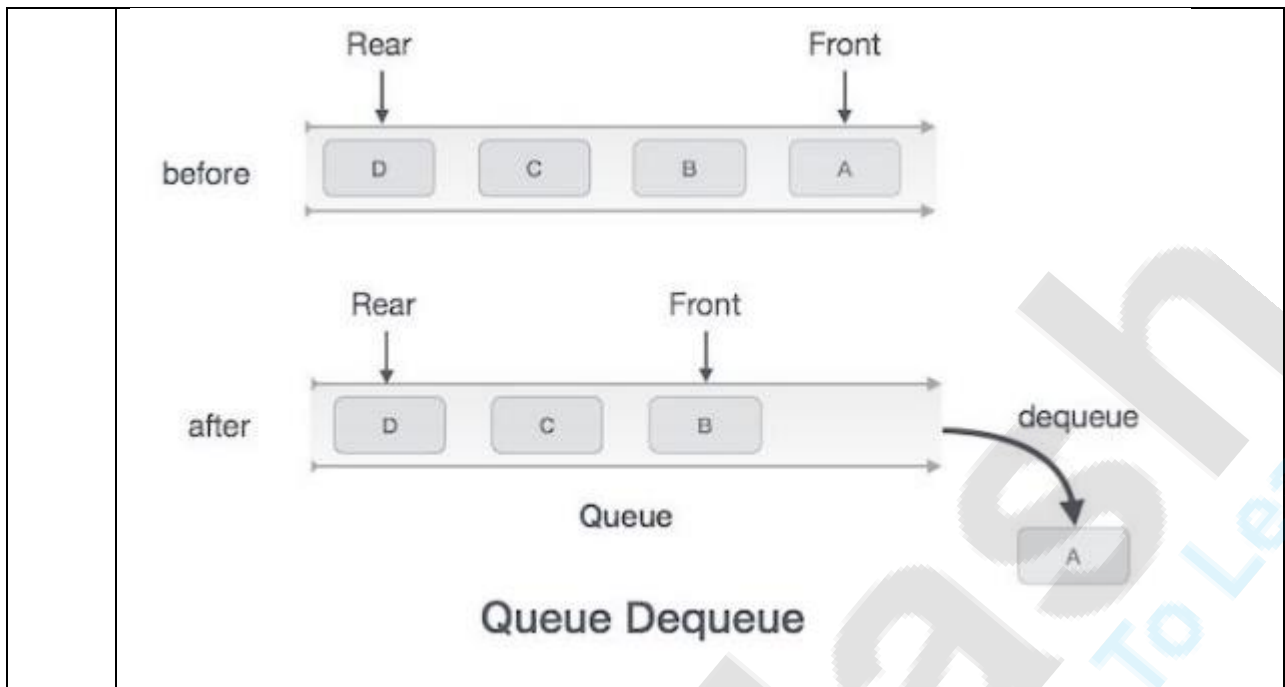
Step 1 – Check if the queue is empty.

Step 2 – If the queue is empty, produce underflow error and exit.

Step 3 – If the queue is not empty, access the data where front is pointing.

Step 4 – Increment front pointer to point to the next available data element.

Step 5 – Return success.



Q.3 Explain the different types of queue.

Ans.

**Queue:**

Queue is also an abstract data type or a linear data structure, in which the first element is inserted from one end called REAR(also called tail), and the deletion of existing element takes place from the other end called as FRONT(also called head). This makes queue as FIFO data structure, which means that element inserted first will also be removed first.

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

Queue can be of four types:

1. Simple Queue
2. Circular Queue
3. Priority Queue
4. Dequeue (Double Ended queue)

1. Simple queue:

Simple queue defines the simple operation of queue in which insertion occurs at the rear of the list and deletion occurs at the front of the list.

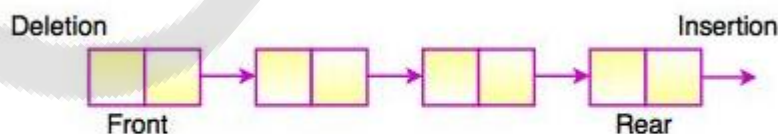


Fig. Simple Queue

2. Circular Queue:

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.

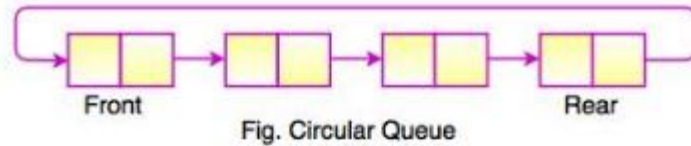


Fig. Circular Queue

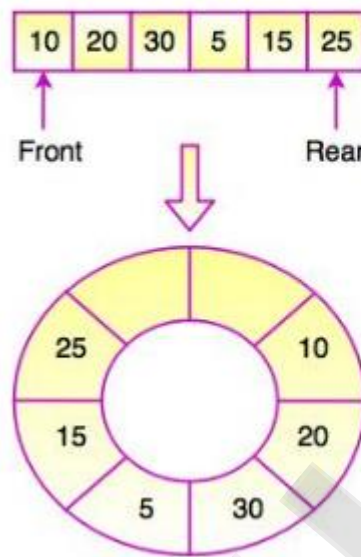
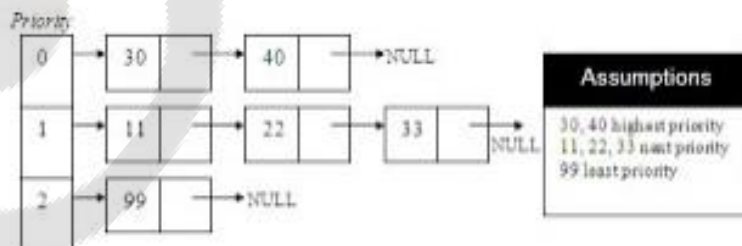


Fig. Circular Queue

The above figure shows the structure of circular queue. It stores an element in a circular way and performs the operations according to its FIFO structure.

3. Priority queue:

A priority queue is a queue that contains items that have some preset priority. When an element has to be removed from a priority queue, the item with the highest priority is removed first



## 4. Dequeue (Double Ended queue):

In Double Ended Queue, insert and delete operation can be occur at both ends that is front and rear of the queue

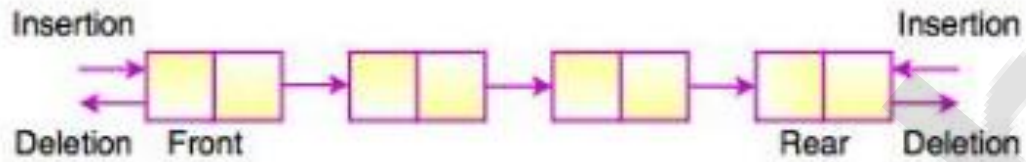


Fig. Double Ended Queue (Deque)

Q.5 Implementation of queue using Array

Ans. **Queue Implementation**

Array is the easiest way to implement a queue. Queue can be also implemented using Linked List or Stack.

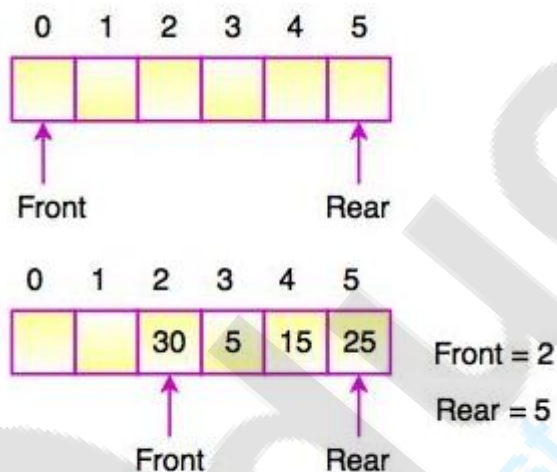


Fig. Implementation of Queue using Array

In the above diagram, Front and Rear of the queue point at the first index of the array. (Array index starts from 0).

While adding an element into the queue, the Rear keeps on moving ahead and always points to the position where the next element will be inserted. Front remains at the first index.

#### Queue data structure using array can be implemented as follows...

Before we implement actual operations, first follow the below steps to create an empty queue.

Step 1: Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 2: Declare all the user defined functions which are used in queue implementation.

Step 3: Create a one dimensional array with above defined SIZE (int queue[SIZE])

Step 4: Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)

Step 5: Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue

### **Queue operation using array :**

#### **enqueue(value) - Inserting value into the queue**

In a queue data structure, enqueue() is a function used to insert a new element into the queue. In a queue, the new element is always inserted at rear position. The enqueue() function takes one integer value as parameter and inserts that value into the queue. We can use the following steps to insert an element into the queue...

Step 1: Check whether queue is FULL. (rear == SIZE-1)

Step 2: If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.

Step 3: If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value.

#### **dequeue() - Deleting a value from the Queue**

In a queue data structure, dequeue() is a function used to delete an element from the queue. In a queue, the element is always deleted from front position. The dequeue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...

Step 1: Check whether queue is EMPTY. (front == rear)

Step 2: If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.

Step 3: If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).  
display() - Displays the elements of a Queue

#### **We can use the following steps to display the elements of a queue...**

Step 1: Check whether queue is EMPTY. (front == rear)

Step 2: If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.

Step 3: If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'.

Step 3: Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value is equal to rear (i <= rear)



Q. 6 Explain Linked list and its Operations.

Ans

**Linked List:**

Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts.

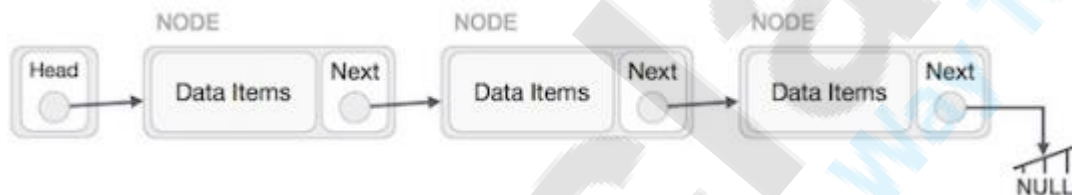
Each node consists of its own data and the address of the next node and forms a chain.

Linked Lists are used to create trees and graphs.



**Linked List Representation**

Linked list can be visualized as a chain of nodes, where every node points to the next node.



As per the above illustration, following are the important points to be considered.

- Linked List contains a link element called first.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

**Basic Operations**

Following are the basic operations supported by a list.

Insertion – Adds an element at the beginning of the list.

Deletion – Deletes an element at the beginning of the list.

Display – Displays the complete list.

Search – Searches an element using the given key.


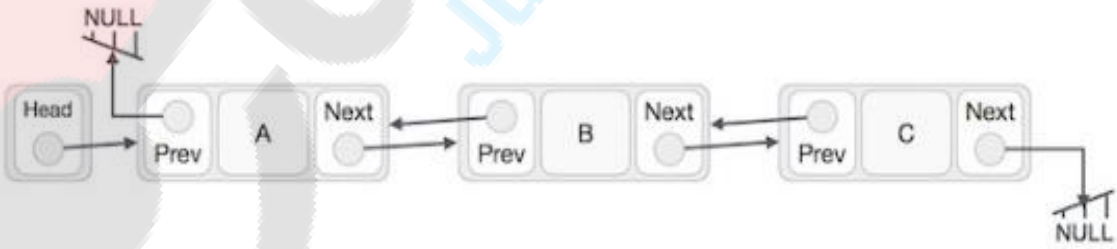
Delete – Deletes an element using the given key

**Advantages of Linked Lists**

They are a dynamic in nature which allocates the memory when required.

Insertion and deletion operations can be easily implemented.

Stacks and queues can be easily executed.

	<p>Linked List reduces the access time.</p> <p><b>Disadvantages of Linked Lists</b></p> <p>The memory is wasted as pointers require extra memory for storage.          No element can be accessed randomly; it has to access each node sequentially.          Reverse Traversing is difficult in linked list.</p>
Q.7	Explain types of :inked List
Ans.	<p><b>Types of Linked List</b></p> <p>Following are the various types of linked list.</p> <ol style="list-style-type: none"> <li>1.Simple Linked List – Item navigation is forward only.</li> <li>2.Doubly Linked List – Items can be navigated forward and backward.</li> <li>3.Circular Linked List – Last item contains link of the first element as next and the first element has a link to the last element as previous</li> </ol> <p><b>1. Singly Linked List :</b> Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.</p>  <p><b>2. Doubly Linked List:</b></p> <p>In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.</p>  <p>As per the above illustration, following are the important points to be considered.</p> <ul style="list-style-type: none"> <li>• Doubly Linked List contains a link element called first and last.</li> <li>• Each link carries a data field(s) and two link fields called next and prev.</li> <li>• Each link is linked with its next link using its next link.</li> </ul>

- Each link is linked with its previous link using its previous link.
- The last link carries a link as null to mark the end of the list.

### **Basic Operations**

Following are the basic operations supported by a list.

Insertion – Adds an element at the beginning of the list.

Deletion – Deletes an element at the beginning of the list.

Insert Last – Adds an element at the end of the list.

Delete Last – Deletes an element from the end of the list.

Insert After – Adds an element after an item of the list.

Delete – Deletes an element from the list using the key.

Display forward – Displays the complete list in a forward manner.

Display backward – Displays the complete list in a backward manner.

### **3. Circular Linked List:**

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

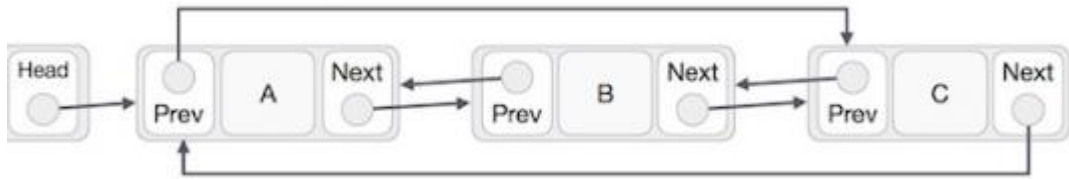
#### **A. Singly Linked List as Circular**

In singly linked list, the next pointer of the last node points to the first node.



#### **B. Doubly Linked List as Circular**

In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.



As per the above illustration, following are the important points to be considered.

The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.

The first link's previous points to the last of the list in case of doubly linked list.

### Basic Operations

Following are the important operations supported by a circular list.

insert – Inserts an element at the start of the list.

delete – Deletes an element from the start of the list.

display – Displays the list.