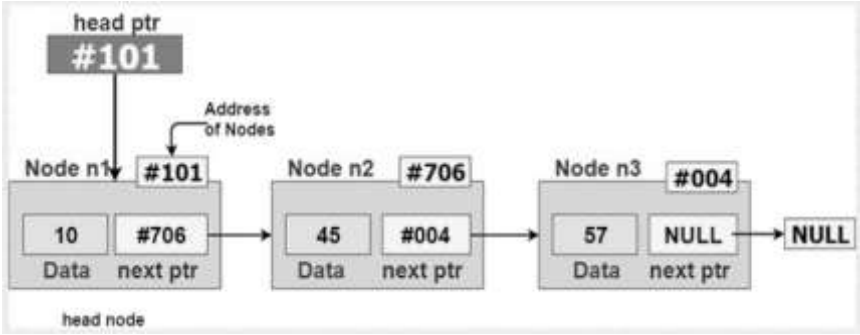
UNIT – III

1.1 INTRODUCTION TO LINKED LIST:

- A linked list is a linear data structure, in which the element are not stored at contiguous memory locations.
- The elements in a linked list are linked using pointers.
- A linked list is a collection of "nodes" connected together via links.
- These nodes consist of the data to be stored and a pointer to the address of the next node within the linked list.
- In the case of arrays, the size is limited to the definition, but in linked lists, there is no defined size.
- Any amount of data can be stored in it and can be deleted from it.

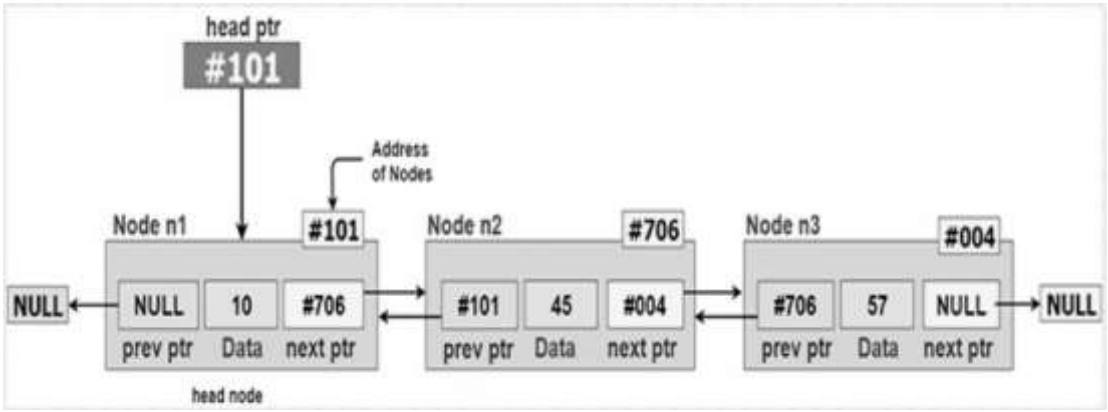
Singly Linked List:

The nodes only point to the address of the next node in the list.



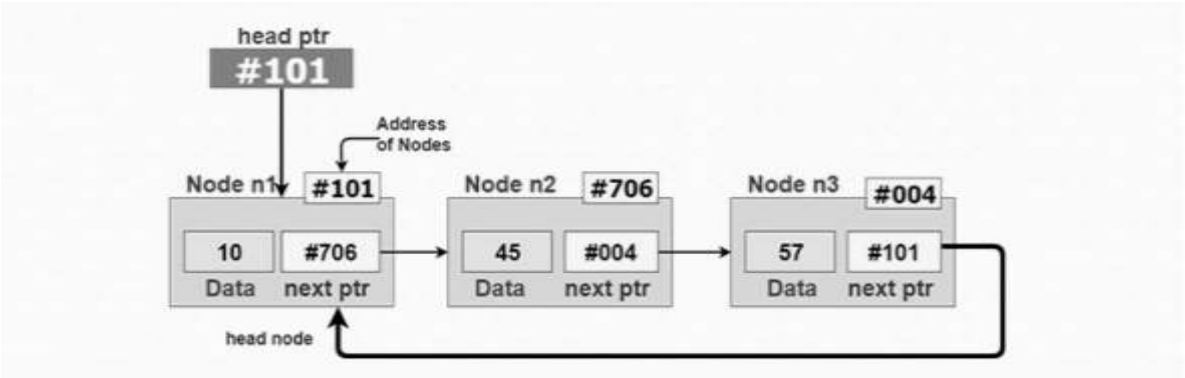
Doubly Linked List:

The nodes point to the address of both previous and next nodes.



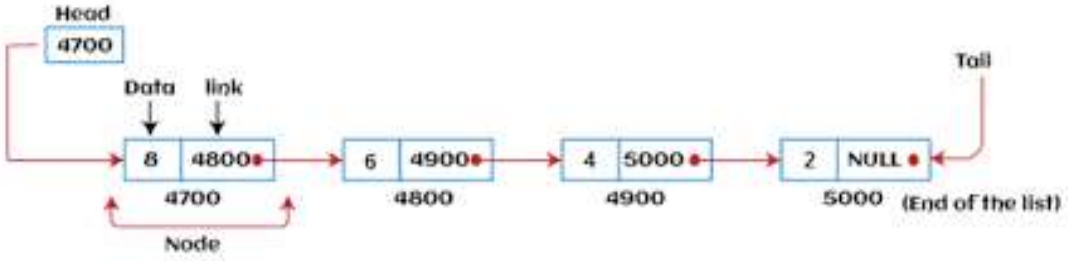
Circular Linked List:

The last node in the list will point to the first node in the list.
It can either be singly linked or doubly linked.



1.2 REPRESENTATION OF LINKED LIST IN MEMORY

(1) Linked lists can be represented in memory by using two arrays respectively known as INFO and LINK, such that INFO[K] and LINK[K] contains information of element and next node address respectively.
(2) The list also requires a variable 'Name' or 'Start', which contains address of first node. Pointer field of last node denoted by NULL which indicates the end of list. e.g., Consider a linked list given below:
(3) The linked list can be represented in memory as –



		INFO	LINK
START→	4700	8	4800
	4800	6	4900
	4900	4	5000
	5000	2	NULL

Above figure shows linked list. It indicates that the node of a list need not occupy adjacent elements in the array INFO and LINK.

1.3 OPERATIONS ON LINKED LIST:

a) Creating a node
b) Insertion
c) Deletion
d) Search
e) Traversal

a) Creating a node:

- It is a declaration of a node that consists of the first variable as data and the next as a pointer, which will keep the address of the next node.

- Here you need to use the malloc function to allocate memory for the nodes dynamically.

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

b) Insertion:

In a single linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the single linked list...

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether list is Empty (head == NULL)
- Step 3 - If it is Empty then, set newNode→next = NULL and head = newNode.
- Step 4 - If it is Not Empty then, set newNode→next = head and head = newNode.

Inserting At End of the list

We can use the following steps to insert a new node at end of the single linked list...

- Step 1 - Create a newNode with given value
- Step 2 - Check whether list is Empty (head == NULL).
- Step 3 - If it is Empty then, set head = newNode.
  newNode → next as NULL.
- Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.
- Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL).
- Step 6 - Set temp → next = newNode.
  newNode → next as NULL.

Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the single linked list...

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether list is Empty (head == NULL)
- Step 3 - If it is Empty then, set newNode → next = NULL and head = newNode.
- Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.
- Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).
- Step 6 - Every time check whether temp is reached to last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.
- Step 7 - Finally, Set 'newNode → next = temp → next' and 'temp → next = newNode'

c) Deletion

In a single linked list, the deletion operation can be performed in three ways. They are as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the single linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
- Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
- Step 4 - Check whether list is having only one node (temp → next == NULL)
- Step 5 - If it is TRUE then set head = NULL and delete temp (Setting Empty list conditions)
- Step 6 - If it is FALSE then set head = temp → next, and delete temp.

Deleting from End of the list

We can use the following steps to delete a node from end of the single linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
- Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.
- Step 4 - Check whether list has only one Node (temp1 → next == NULL)
- Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate the function. (Setting Empty list condition)
- Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until it reaches to the last node in the list. (until temp1 → next == NULL)
- Step 7 - Finally, Set temp2 → next = NULL and delete temp1.

Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the single linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
- Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.
- Step 4 - Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node.
- Step 5 - If it is reached to the last node then display 'Given node not found in the list! Deletion not possible!!!'. And terminate the function.
- Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
- Step 7 - If list has only one node and that is the node to be deleted, then set head = NULL and delete temp1 (free(temp1)).
- Step 8 - If list contains multiple nodes, then check whether temp1 is the first node in the list (temp1 == head).
- Step 9 - If temp1 is the first node then move the head to the next node (head = head → next) and delete temp1.

- Step 10 - If temp1 is not first node then check whether it is last node in the list (temp1 → next == NULL).
- Step 11 - If temp1 is last node then set temp2 → next = NULL and delete temp1 (free(temp1)).
- Step 12 - If temp1 is not first node and not last node then set temp2 → next = temp1 → next and delete temp1 (free(temp1)).

d) Displaying a Linked List

We can use the following steps to display the elements of a single linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!!' and terminate the function.
- Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
- Step 4 - Keep displaying temp → data with an arrow (--->) until temp reaches to the last node
- Step 5 - Finally display temp → data with arrow pointing to NULL (temp → data ---> NULL).

1.4 OPERATIONS ON DOUBLY LINKED LIST:

a) Insertion

In a double linked list, the insertion operation can be performed in three ways as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the double linked list...

- Step 1 - Create a newNode with given value and newNode → previous as NULL.
- Step 2 - Check whether list is Empty (head == NULL)
- Step 3 - If it is Empty then, assign NULL to newNode → next and newNode to head.
- Step 4 - If it is not Empty then, assign head to newNode → next and newNode to head.

Inserting At End of the list

We can use the following steps to insert a new node at end of the double linked list...

- Step 1 - Create a newNode with given value and newNode → next as NULL.
- Step 2 - Check whether list is Empty (head == NULL)
- Step 3 - If it is Empty, then assign NULL to newNode → previous and newNode to head.
- Step 4 - If it is not Empty, then, define a node pointer temp and initialize with head.
- Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL).
- Step 6 - Assign newNode to temp → next and temp to newNode → previous.

Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the double linked list...

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether list is Empty (head == NULL)
- Step 3 - If it is Empty then, assign NULL to both newNode → previous & newNode → next and set newNode to head.
- Step 4 - If it is not Empty then, define two node pointers temp1 & temp2 and initialize temp1 with head.
- Step 5 - Keep moving the temp1 to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).

- Step 6 - Every time check whether temp1 is reached to the last node. If it is reached to the last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp1 to next node.
- Step 7 - Assign temp1 → next to temp2, newNode to temp1 → next, temp1 to newNode → previous, temp2 to newNode → next and newNode to temp2 → previous.

b) Deletion

In a double linked list, the deletion operation can be performed in three ways as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the double linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
- Step 3 - If it is not Empty then, define a Node pointer 'temp' and initialize with head.
- Step 4 - Check whether list is having only one node (temp → previous is equal to temp → next)
- Step 5 - If it is TRUE, then set head to NULL and delete temp (Setting Empty list conditions)
- Step 6 - If it is FALSE, then assign temp → next to head, NULL to head → previous and delete temp.

Deleting from End of the list

We can use the following steps to delete a node from end of the double linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty, then display 'List is Empty!!! Deletion is not possible' and terminate the function.
- Step 3 - If it is not Empty then, define a Node pointer 'temp' and initialize with head.
- Step 4 - Check whether list has only one Node (temp → previous and temp → next both are NULL)
- Step 5 - If it is TRUE, then assign NULL to head and delete temp. And terminate from the function. (Setting Empty list condition)
- Step 6 - If it is FALSE, then keep moving temp until it reaches to the last node in the list. (until temp → next is equal to NULL)
- Step 7 - Assign NULL to temp → previous → next and delete temp.

Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the double linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
- Step 3 - If it is not Empty, then define a Node pointer 'temp' and initialize with head.
- Step 4 - Keep moving the temp until it reaches to the exact node to be deleted or to the last node.
- Step 5 - If it is reached to the last node, then display 'Given node not found in the list! Deletion not possible!!!' and terminate the fuction.
- Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not

- Step 7 - If list has only one node and that is the node which is to be deleted then set head to NULL and delete temp (free(temp)).
- Step 8 - If list contains multiple nodes, then check whether temp is the first node in the list (temp == head).
- Step 9 - If temp is the first node, then move the head to the next node (head = head → next), set head of previous to NULL (head → previous = NULL) and delete temp.
- Step 10 - If temp is not the first node, then check whether it is the last node in the list (temp → next == NULL).
- Step 11 - If temp is the last node then set temp of previous of next to NULL (temp → previous → next = NULL) and delete temp (free(temp)).
- Step 12 - If temp is not the first node and not the last node, then set temp of previous of next to temp of next (temp → previous → next = temp → next), temp of next of previous to temp of previous (temp → next → previous = temp → previous) and delete temp (free(temp)).

Displaying a Double Linked List

We can use the following steps to display the elements of a double linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty, then display 'List is Empty!!!' and terminate the function.
- Step 3 - If it is not Empty, then define a Node pointer 'temp' and initialize with head.
- Step 4 - Display 'NULL <--- '.
- Step 5 - Keep displaying temp → data with an arrow (<===>) until temp reaches to the last node
- Step 6 - Finally, display temp → data with arrow pointing to NULL (temp → data ---> NULL).

1.5 OPERATIONS ON CIRCULAR LINKED LIST:

a) Insertion

In a circular linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the circular linked list...

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether list is Empty (head == NULL)
- Step 3 - If it is Empty then, set head = newNode and newNode→next = head .
- Step 4 - If it is Not Empty then, define a Node pointer 'temp' and initialize with 'head'.
- Step 5 - Keep moving the 'temp' to its next node until it reaches to the last node (until 'temp → next == head').
- Step 6 - Set 'newNode → next =head', 'head = newNode' and 'temp → next = head'.

Inserting At End of the list

We can use the following steps to insert a new node at end of the circular linked list...

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether list is Empty (head == NULL).

- Step 3 - If it is Empty then, set head = newNode and newNode → next = head.
- Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.
- Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next == head).
- Step 6 - Set temp → next = newNode and newNode → next = head.

Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the circular linked list...

- Step 1 - Create a newNode with given value.
- Step 2 - Check whether list is Empty (head == NULL)
- Step 3 - If it is Empty then, set head = newNode and newNode → next = head.
- Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.
- Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).
- Step 6 - Every time check whether temp is reached to the last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.
- Step 7 - If temp is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).
- Step 8 - If temp is last node then set temp → next = newNode and newNode → next = head.
- Step 8 - If temp is not last node then set newNode → next = temp → next and temp → next = newNode.

b) Deletion

In a circular linked list, the deletion operation can be performed in three ways those are as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the circular linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
- Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize both 'temp1' and 'temp2' with head.
- Step 4 - Check whether list is having only one node (temp1 → next == head)
- Step 5 - If it is TRUE then set head = NULL and delete temp1 (Setting Empty list conditions)
- Step 6 - If it is FALSE move the temp1 until it reaches to the last node. (until temp1 → next == head)
- Step 7 - Then set head = temp2 → next, temp1 → next = head and delete temp2.

Deleting from End of the list

We can use the following steps to delete a node from end of the circular linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

- Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.
- Step 4 - Check whether list has only one Node (temp1 → next == head)
- Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate from the function. (Setting Empty list condition)
- Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until temp1 reaches to the last node in the list. (until temp1 → next == head)
- Step 7 - Set temp2 → next = head and delete temp1.

Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the circular linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
- Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.
- Step 4 - Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node.
- Step 5 - If it is reached to the last node then display 'Given node not found in the list! Deletion not possible!!!'. And terminate the function.
- Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node (temp1 → next == head)
- Step 7 - If list has only one node and that is the node to be deleted then set head = NULL and delete temp1 (free(temp1)).
- Step 8 - If list contains multiple nodes then check whether temp1 is the first node in the list (temp1 == head).
- Step 9 - If temp1 is the first node then set temp2 = head and keep moving temp2 to its next node until temp2 reaches to the last node. Then set head = head → next, temp2 → next = head and delete temp1.
- Step 10 - If temp1 is not first node then check whether it is last node in the list (temp1 → next == head).
- Step 11 - If temp1 is last node then set temp2 → next = head and delete temp1 (free(temp1)).
- Step 12 - If temp1 is not first node and not last node then set temp2 → next = temp1 → next and delete temp1 (free(temp1)).

Displaying a circular Linked List

We can use the following steps to display the elements of a circular linked list...

- Step 1 - Check whether list is Empty (head == NULL)
- Step 2 - If it is Empty, then display 'List is Empty!!!' and terminate the function.
- Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
- Step 4 - Keep displaying temp → data with an arrow (--->) until temp reaches to the last node
- Step 5 - Finally display temp → data with arrow pointing to head → data.