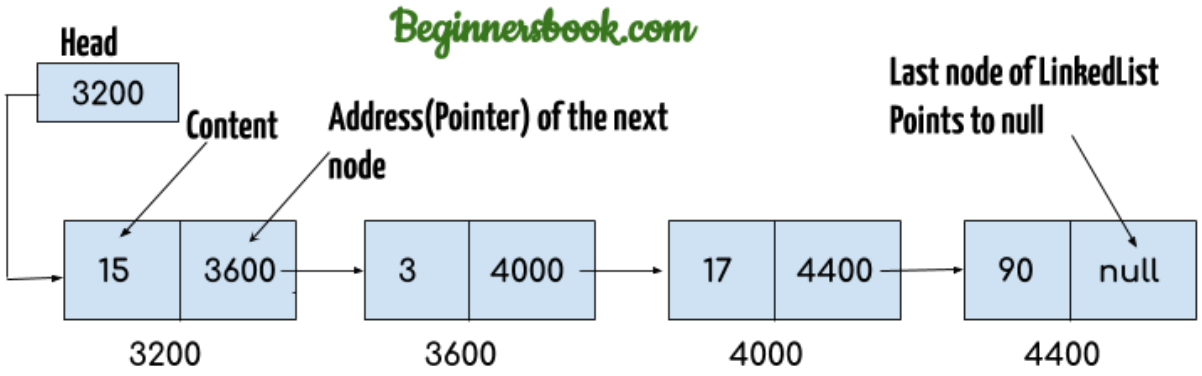
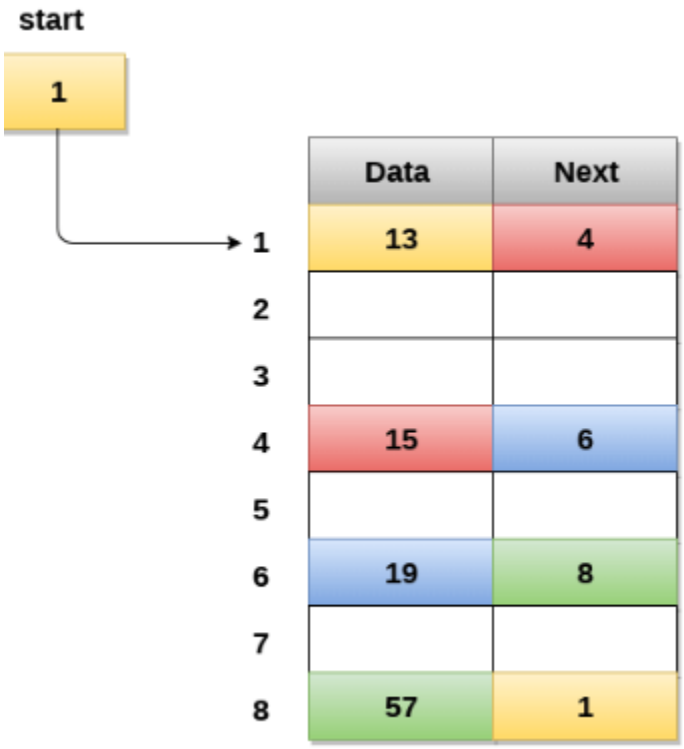
What Is Linked List ?

A Linked List can be defined as a collection of objects called nodes that are randomly stored in the memory.

- A node contains two fields :
- data stored at that particular address
- the pointer which contains the address of the next node in the memory.



A Linked List Does Not Need Contiguous Locations In The Memory Like Array As We See In The Previous Chapter But The Nodes Of A Linked List Can Store In Different Locations And Are Connected With Each Others By The Reference Or The Pointers. Every Node Has Reference To The Address Of The Next Node.



Advantages Of Linked List :

1.Linked List Dynamic Structure Advantages :

1.1 We do not need to know the size of the linked list or the count of the nodes before we declare the linked list **as an array.**

1.2 in The Run Time We Can Extend The Linked List By Adding New Nodes Until We Reach The Max Size Of The Memory.

Unlike Array When We Need To Extend The Size Of The Array To Add New Elements In The Run Time. We Cannot Do That Directly With Array. We Need To Declare A New Array With A Bigger Size And Copy The Elements From The Old Array To The New.

2.Linked List Operations Advantages :

insert or delete any element at any position is very fast.

These two operations take a constant time (O) = 1.
And this is one of the biggest advantages to using the linked list.

3.Linked List Memory Advantages :

no unused elements can appear and waste memory space like arrays.

Example : If we need to store data = { 1 , 2 , 3 }.

In array if we declare the size of array = 5

1	2	3	unused	unused

In the linked list We Used Only Numbers Of Nodes As Required

Current address	data	Next node address
100	1	200
The space between 100 and 200 used for other purposes during the execution of the program		
200	2	300
The space between 200 and 300 used for other purposes during the execution of the program		
300	3	null

Disadvantages of using Linked List :

1.Linked List Operations Disadvantages :

Access Operations Are Very Slow (O) = N.
To Read Or Write At Any Node We May Be Need To Loop Through The Entire Linked List Until We Reach The Required Node.
Unlike The Array Which Can Reach To The Required Element And Access It In One Step Through The Equation Of The Base Address As We See in The Previous Chapter.

2.Linked List Memory Disadvantages

The Linked List Takes Extra Space Over The Array To Store The Pointers Of The Next Node Address.
Arrays store only the data of elements.
but linked list need to store the data and the address of next node.

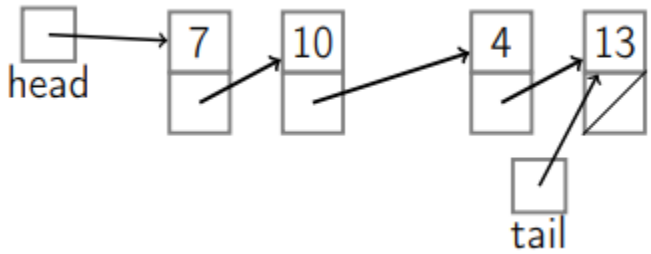
Types Of Linked List :

1- Singly Linked List
2- Doubly Linked List
3- Circular Linked List

Singly Linked List

What Is a Singly Linked List ?

Its One way chain or singly linked list can be traversed only in one direction. In other words, we can say that each node contains only the next pointer, therefore we can not traverse the list in the reverse direction.



Singly Linked List Structure :

1- Header : Used To Point To The First Node In The Linked List
2- Tail : Used To Point To The Last Node In The Linked List
3- Data Nodes : The Other Nodes Between Header And Tail That used to Store Actual Data Of LinkedList and every node consist of two fields :

- Data field : To Store Data
- Next Field : To Store The Address Of The Next Node.

Singly Linked List Structure Code :

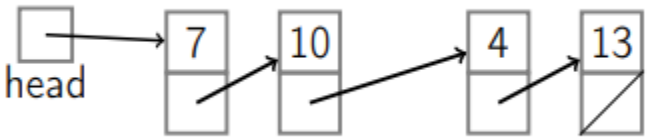
Singly Linked List Class :

```
3 references
public class SinglyLinkedList<T>
{
    9 references
    public Node<T> Header { get; set; }
    6 references
    public Node<T> Tail { get; set; }

    int Count = 0;

    2 references
    public SinglyLinkedList()
    {
        Header = new Node<T>();
        Tail = new Node<T>();
    }

    Operations

}
```

Node Class :

```
39 references
public class Node<T>
{
    2 references
    public T Data { get; set; }
    26 references
    public Node<T> Next { get; set; }
    2 references
    public Node()
    {

    }
    1 reference
    public Node(T data)
    {
        this.Data = data;
    }
}
```

Operations In Singly Linked List



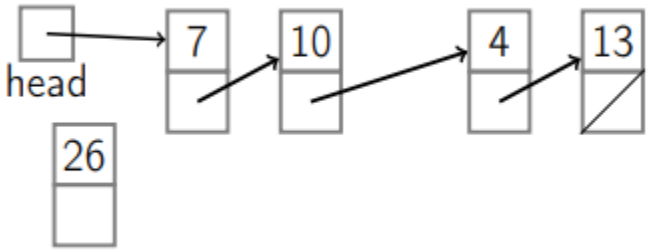
1- Insert Operations

1.1 Insert At First

The new node will be the first node in the linked list.
Called Also in some reference : Insert After Header or Push Front

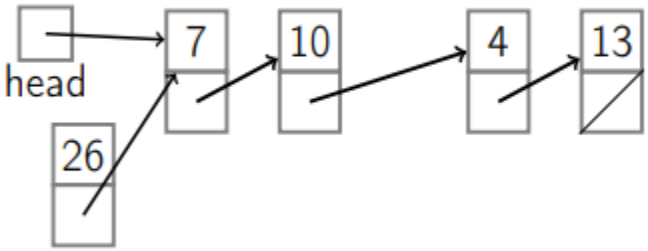
- Steps :

1. Create New Node

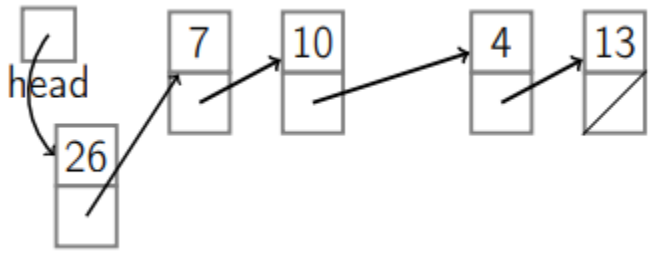


2. If LinkedList Is Not Empty

2.1 The Next Of The New Node Will Point To The Current First Node



2.2 The Next Of Header Will Point To The New Node



3. Else (When LinkedList Is Empty)

3.1 The Next Of Header Will Point To The New Node

3.2 The Next Of Tail Will Point To The New Node

- Code :

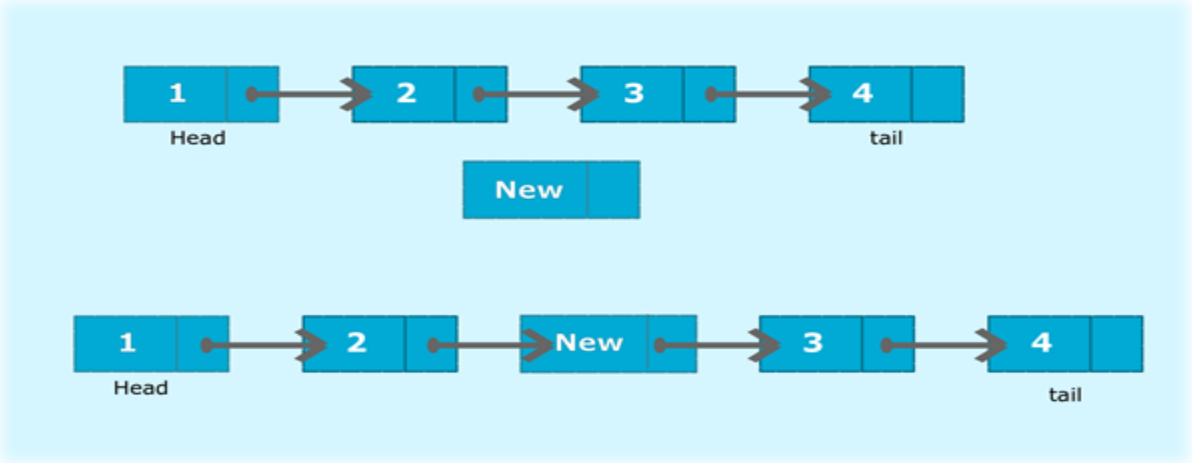
```
4 references
public Node<T> InsertNewNodeAtFirst(T data)
{
    Node<T> NewNode = CreateNewNode(data);

    if (!CheckIfLinkedListIsEmpty())
    {
        Node<T> currentFirstNode = GetFirstNode();
        NewNode.Next = currentFirstNode;
        Header.Next = NewNode;
    }
    else
    {
        Header.Next = NewNode;
        Tail.Next = NewNode;
    }

    this.Count++;
    Console.WriteLine("The New Node Inserted At First With Data : " + data);
    return NewNode;
}
```

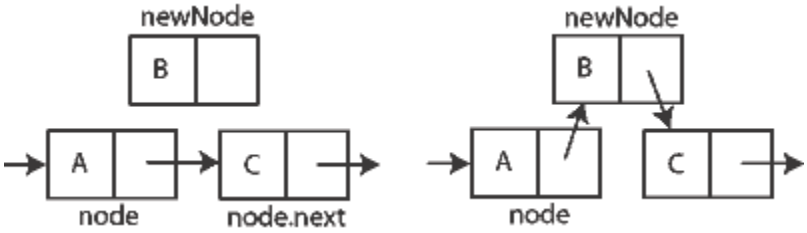
- Time Complexity :

Time Complexity Of Insert At First = O(1)

1.2 Insert After Node

The new node will be added after a specific node.
Ex : insert new node after second node :





- Steps :

1. Create new node
2. The Next Of New Node Point to The Next Of Previous Node
3. The Next Of Previous Node Point to The New Node

- Code :

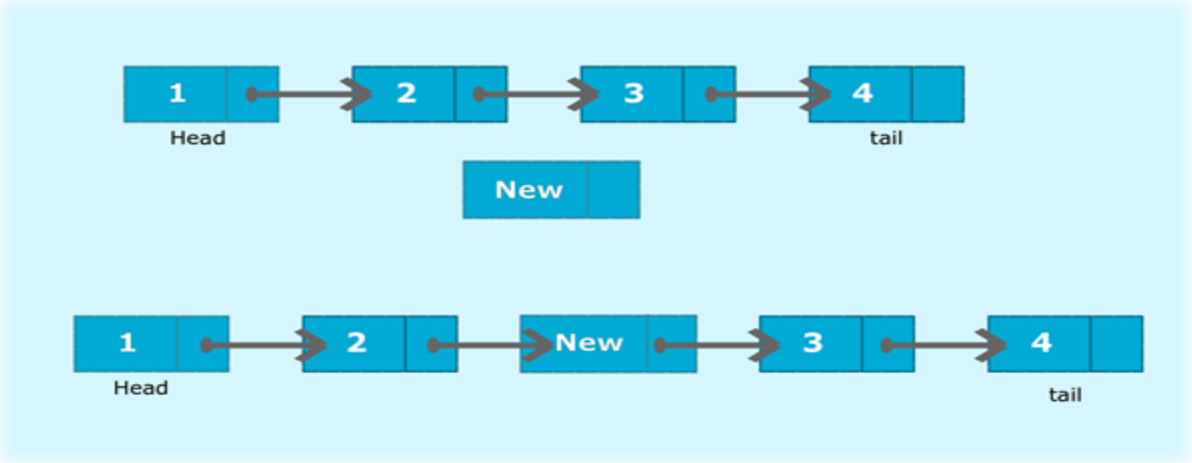
```
1 reference
public Node<T> InsertNewNodeAfter(Node<T> previousNode, T dataOfNewNode)
{
    Node<T> NewNode = CreateNewNode(dataOfNewNode);
    NewNode.Next = previousNode.Next;
    previousNode.Next = NewNode;
    this.Count++;
    Console.WriteLine("The New Node Inserted After Node With Data : " + dataOfNewNode);
    return NewNode;
}
```

- Time Complexity :

  Time Complexity Of Insert After Node = O(1)

1.3 Insert Before Node

The new node will be added before a specific node.
Ex : insert new node before third node :



- Steps :

  Inserting a new node before given node operation is not recommended with singly linked list because it has cost of n. And this Because The Singly Linked List Has Pointer only to the next Node.
  We solve this problem By Using Doubly linked List Which Contain Pointers To Next Node And **Also Previous Node**.
  We Will Explain The Doubly Linked List In Details In Later And We Will See A Lot Of Advantages For Doubly Linked List Over Singly Linked List

- Time Complexity :

  Time Complexity Of Insert Before Node = O(n)

1.4 Insert At Last

The New Node Will Be The Last Node In The Linked List.

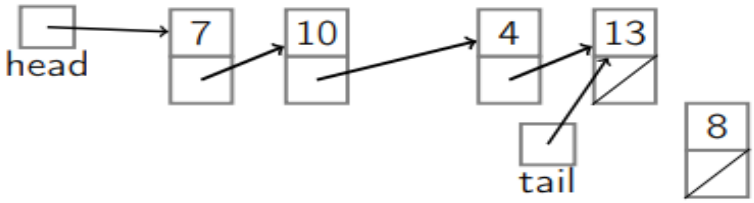
Steps :

1. If Linked List Is Empty :

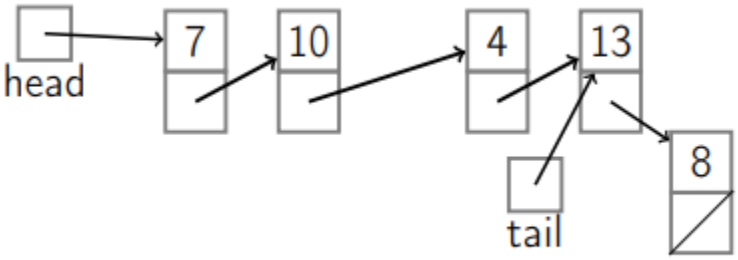
1.1 The New Node Will Insert At First As We See In The Previous Operation

2 Else (Linked List Is Not Empty)

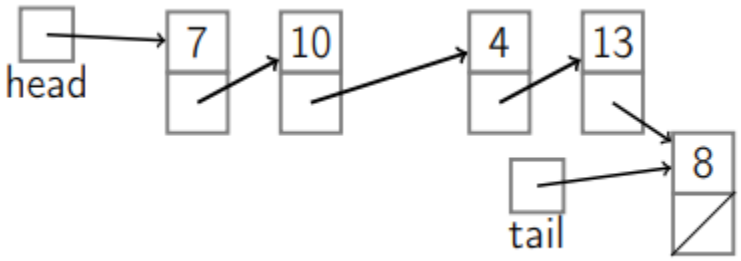
2.1 Create New Node



2.2 The Next Of Current Last Node Will Point To The New Node



2.3 The Tail Will Point To The Last Node



Code :

```
6 references
public Node<T> InsertNewNodeAtLast(T data)
{
    if (CheckIfLinkedListIsEmpty())
    {
        return InsertNewNodeAtFirst(data);
    }
    else
    {
        Node<T> newLastNode = CreateNewNode(data);
        Node<T> currentLastNode = GetLastNode();
        currentLastNode.Next = newLastNode;
        Tail.Next = newLastNode;
        this.Count++;
        Console.WriteLine("The New Node Inserted At Last With Data : " + data);
        return newLastNode;
    }
}
```

Time Complexity

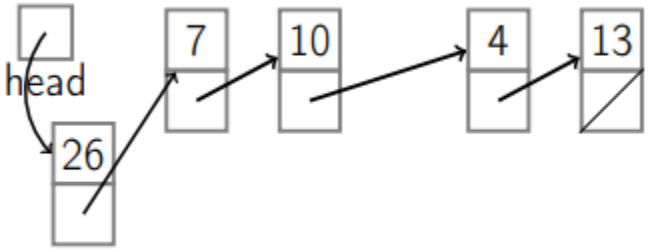
Time Complexity Of Insert At Last = O(1)

Note : We Can Implement Singly LinkedList Without Tail But In This Case The Time Complexity Of Insert At Last = O(n) so we used Tail to prevent this problem

2- Delete Operations

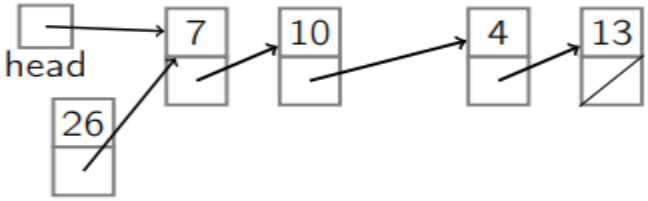
2.1 Delete The Node At The First

The First Node In The Linked List Will be Deleted And The Current Second Node Will Be The First.

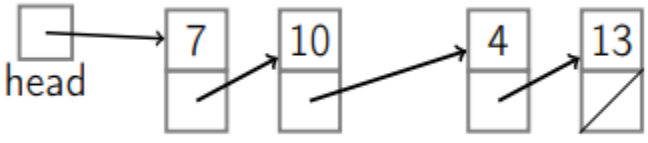


Steps :

1. The Header Will Point To The Next Of Current First Node.



2. The Next Of Current First Node Will Point To Null



Code :

```
2 references
public Node<T> DeleteTheNodeAtFirst()
{
    Node<T> firstNode = GetFirstNode();
    if(firstNode != null)
    {
        Header.Next = firstNode.Next;
        firstNode.Next = null;
        this.Count--;
        Console.WriteLine("The Node At First Is Deleted Successfully... ");
    }

    return firstNode;
}
```

- ## Time Complexity :

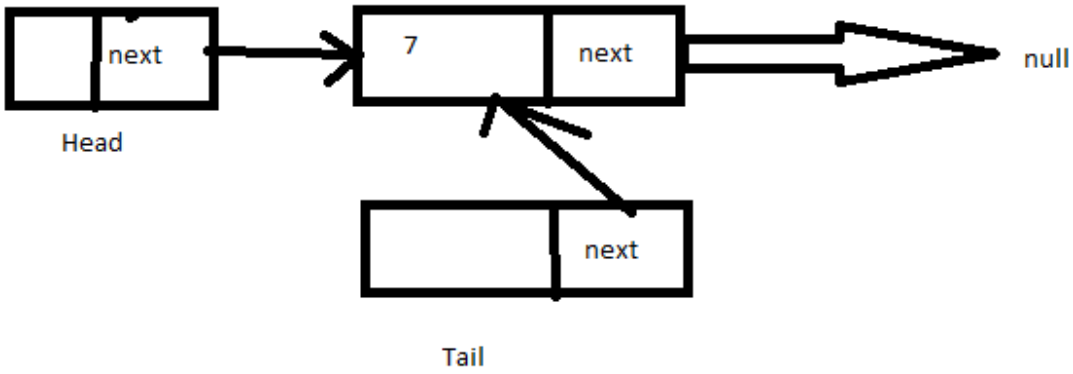
  Time Complexity Of Delete Node At First = O(1)

2.1 Delete The Node At Last

The Last Node Will Be Deleted And The Node Before The Current Last Node Will Be The New Last Node in The Linked List.

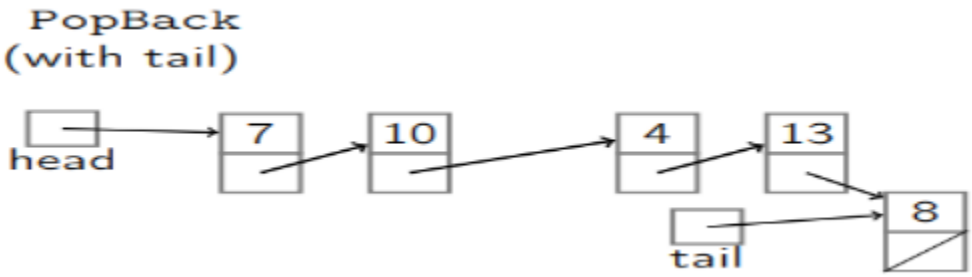
Steps

1.If Linked List Contain Only One Node

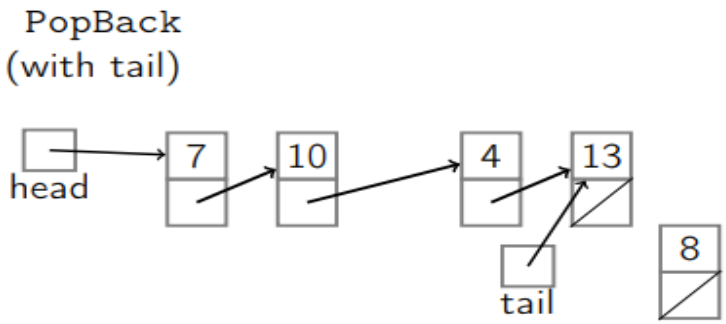


1.1 Delete The Node At First
1.2 Header Point To Null
1.3 Tail Point To Null

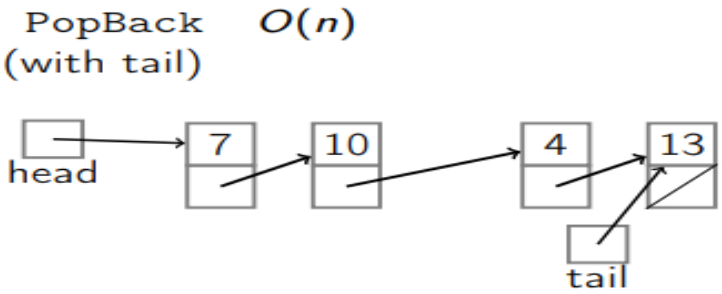
2. Else If Linked List Contain More Than One Node



2.1 The Tail Point To The Node That Was Before The Current Last Node.



2.2 The Next Of The New Last Node Point To Null



Code :

```
1 reference
public Node<T> DeleteTheNodeAtLast()
{
    if (CheckIfLinkedListContainOnlyOneNode())
    {
       var firstNode = DeleteTheNodeAtFirst();
        Header.Next = null;
        Tail.Next = null;
        this.Count--;
        return firstNode;
    }

    Node<T> TheNodeBeforeCurrentLastNode = GetTheNodeBeforeCurrentLastNode();
    if (TheNodeBeforeCurrentLastNode != null)
    {
        Tail.Next = TheNodeBeforeCurrentLastNode;
        TheNodeBeforeCurrentLastNode.Next = null;
        this.Count--;
        Console.WriteLine("The Node At Last Is Deleted Successfully... ");
    }

    return TheNodeBeforeCurrentLastNode;
}
```

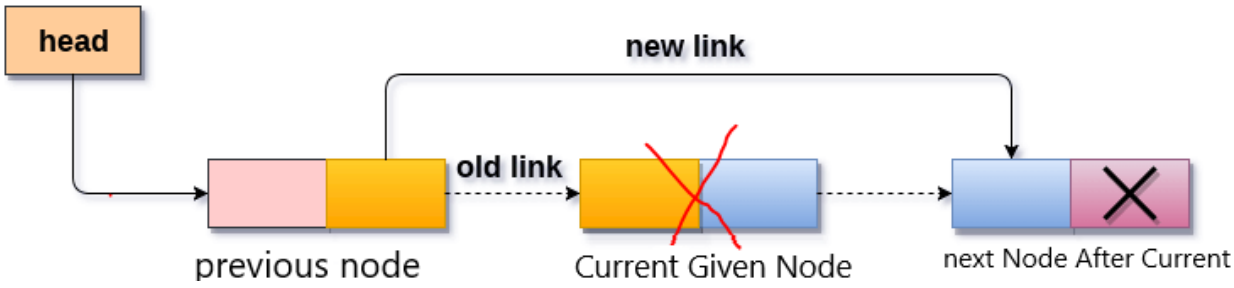
```
1 reference
private Node<T> GetTheNodeBeforeCurrentLastNode()
{
    Node<T> TheNodeBeforeCurrentLastNode = null;
    Node<T> currentNode = GetFirstNode();

    if (currentNode == null || currentNode.Next == null)
        return currentNode;

    while (currentNode.Next != null && currentNode.Next.Next != null)
    {
        currentNode = currentNode.Next;
    }

    TheNodeBeforeCurrentLastNode = currentNode;

    return TheNodeBeforeCurrentLastNode;
}
```

Time Complexity

Time Complexity Of Delete Node At Last = O(n).
In Singly Linked List , The Node Contain Only Pointer To The Next Node. So We Need To Traverse The Whole Linked List To Get The Node Before Current Last Node.We Solve This Problem By Using Doubly Linked List Which Contain Pointer To The Previous Node, and Get The Node Before The Current Last Node By Just One Step.

2.3 Delete Given Node

Delete Current Given Node And Make The Next Of Previous Node Point To The Next Node After The Current Node.



Steps :

1.If Current Node Is First Node
1.1 Delete Node At First
1.2 RETURN

2.If Current Node Is Last Node
2.1 Delete Node At Last
2.2 RETURN

3. If Current Node != First Node OR Current Node != Last Node
3.1 Get The Previous Node Of The Current Node
3.2 The Next Of The Previous Node Point To The Node After The Current Node
3.3 RETURN

Code :

```
6 references
public Node<T> DeleteThisNode(Node<T> currentNode)
{
    if (IsFirstNode(currentNode))
        return DeleteTheNodeAtFirst();

    if (IsLastNode(currentNode))
        return DeleteTheNodeAtLast();

    Node<T> TheNodePrevious = GetThePreviousNode(currentNode);
    Node<T> TheNodeAfter = TheNodePrevious.Next.Next;

    TheNodePrevious.Next = TheNodeAfter;
    currentNode = null;
    this.Count--;
    Console.WriteLine("The Node Is Deleted Successfully... ");
    return currentNode;
}
```

```
1 reference
private Node<T> GetThePreviousNode(Node<T> currentNode)
{
    if (!CheckIfLinkedListContainOnlyOneNode())
    {
        Node<T> ThePreviousNode = GetFirstNode();

        while (ThePreviousNode.Next != currentNode)
        {
            ThePreviousNode = ThePreviousNode.Next;
        }

        return ThePreviousNode;
    }

    return null;
}
```

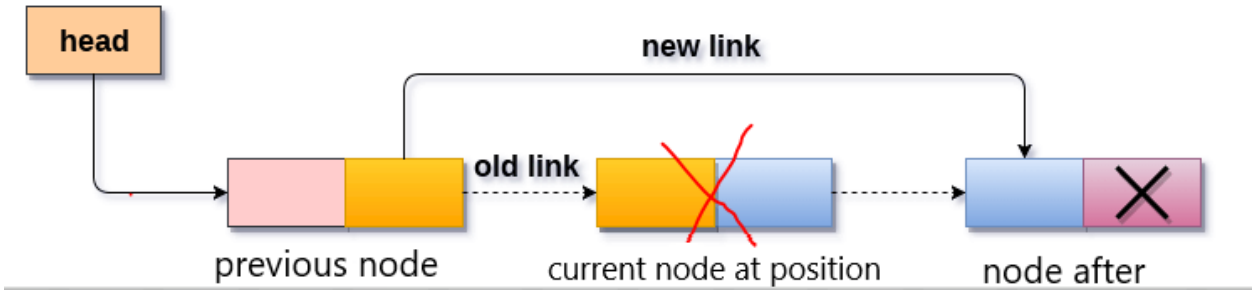
Time Complexity

Time Complexity Of Delete Given Node = O(n).
In Singly Linked List , The Node Contain Only Pointer To The Next Node. So We Need To Traverse The Linked List To Get The Node Before Current Node.We Solve This Problem By Using Doubly Linked List Which Contain Pointer To The Previous Node, and Get The Node Before The Current Node By Just One Step.

2.4 Delete The Node At Specific Position

Steps :

1. If Position == 0
1.1 Delete The Node At First
2. If Position == Linked List Length
2.1 Delete The Node At Last
3.If Position != 0 OR Position != Linked List Length



3.1 Get The Previous Node Of This Position
3.2 The Next Of The Previous Node Point To The Node After This Position

Code :

```
4 references
public Node<T> DeleteTheNodeAtPosition(int position)
{
    if (position == 0)
        return DeleteTheNodeAtFirst();

    if(position == Count - 1)
        return DeleteTheNodeAtLast();

    Node<T> TheNodePreviousThisPosition = GetTheNodeAtPosition(position - 1);
    Node<T> TheCurrentNode = TheNodePreviousThisPosition.Next;
    Node<T> TheNodeAfterThisPosition = TheNodePreviousThisPosition.Next.Next;

    TheNodePreviousThisPosition.Next = TheNodeAfterThisPosition;
    TheCurrentNode = null;
    this.Count--;
    Console.WriteLine("The Node At Position : " + position + " Is Deleted Successfully...
    return TheCurrentNode;
}
```

```
1 reference
public Node<T> GetTheNodeAtPosition(int position)
{
    Node<T> currrentNode = GetFirstNode();

    if (position < this.Count)
    {
        for (int i = 0; i <= position; i++)
        {
            currrentNode = currrentNode.Next;
        }
    }

    if (currrentNode != null)
    {
        Console.WriteLine("The Node At Position : " + position + " Has Dat
    }
    else
    {
        Console.WriteLine("There Is No Node At Position : " + position);
    }

    return currrentNode;
}
```

Time Complexity :

Time Complexity Of Delete Node At Specific Position = O(n).
This Because As We See We Need To Loop The Linked list To Get The Previous Node.