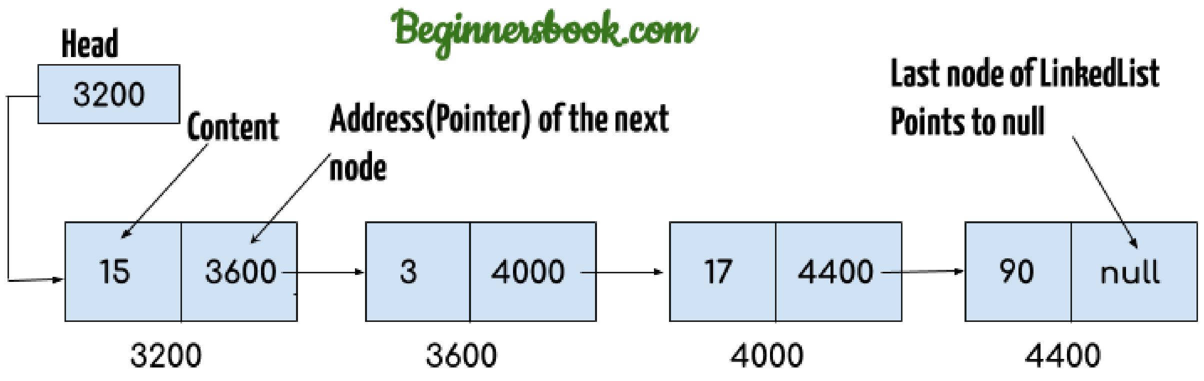
Time Complexity = O(n)

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.



Beginnersbook.com
Head
3200
Content
Address(Pointer) of the next node
Last node of LinkedList Points to null
15
3600
3
4000
17
4400
90
null
3200
3600
4000
4400

Structure Of Linked List Vs Structure Of Array In The Memory

Array Stored As One Block Or Contiguous Locations

Address	Data
Beginning Of Array	
0x01	23
0x02	5
0x03	47
End Of Array	

Linked List Stored in Different Or Random Locations And The Items Connected With Each Other By Next Pointers.

Address	Data	Next
0x01	23	0x04
0x02		
0x03		
0x04	5	0x06
0x05		
0x06	47	null
0x07		

Advantages Of Linked List :

1.Linked List Dynamic Structure Advantages :

1.1 The Size Of Linked List is not Required When Linked List is Created At Compile Time.

So Linked List Can Be Useful Over Array When The Size Of Elements Is Unknown And Depend On the inputs of user or data On the Run Time.

1.2 The Size Of Linked List Is Dynamic At Run Time.

The Size Of Array Is Fixed , So When An Array is Created , The Size Of Array Can Not Be Changed (Extended Or Shrinked).
Example : If You Declare Array With Size Of 200 Elements
- You Can not Change This Size At Run Time To 300 If You Need To Add New Elements.
- You Can not Change This Size At Run Time To 100 If You Need To Delete Unused Elements To Save Memory Space.

With Linked List in The Run Time :

- The Size Of Linked List Can Be Extended By Adding New Nodes Until We Reach The Max Allowed Size on The Memory.
- The Size Of Linked List Can Be Shrinked By Deleting Unwanted Nodes So we can save memory space.

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 of using the linked list.
We will Explain All Operations In Details In The Next Sections.

3.Linked List Memory Advantages :

no unused elements can appear and waste memory space like arrays.
Example : If we need now to store data = { 1 , 3 , 5 } and maybe in the future we will add { 2 , 4 }.
In array if we declare the size of array = 5

1	3	5	unused	unused

linked list Used Only Numbers Of Nodes As Required and When New Data Coming The Pointer Of Last Node Will Reference To It.

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	5	300
The space between 200 and 300 used for other purposes during the execution of the program		
300	3	null

Disadvantages of Linked List :

1.Linked List Operations Disadvantages :

1.1 Access Operations Are Very Slow [(O) = N] When Compared With Arrays Which Have [(O) = 1].

linked list dose not have indexes Like Array so Random access is not allowed. We have to access elements sequentially starting from the first node 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 .

1.2 Search Operation is Slow [(O) = N] When Compared With Data Structure Like Binary Search Tree Which Has [(O) = Log n]

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.

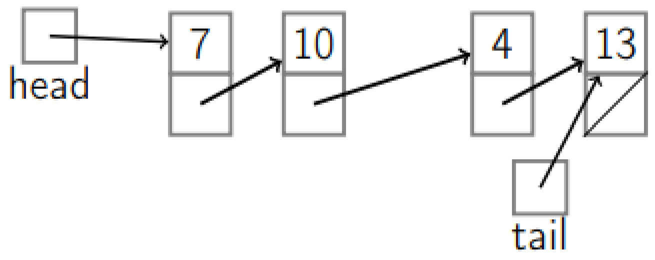
Types Of Linked List :

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

Singly Linked List

What Is a Singly Linked List ?

Singly Linked List is a One way chain or 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.



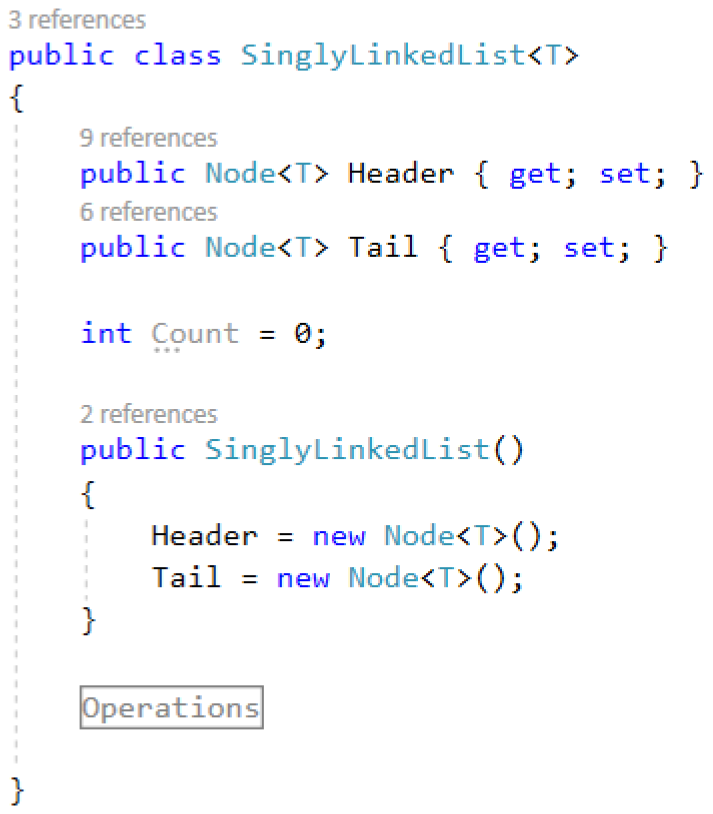
Singly Linked List Structure :

1- <u>Header</u> : Used To Point To The First Node In The Linked List
2- <u>Tail</u> : Used To Point To The Last Node In The Linked List
3- <u>Data Nodes</u> : 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.

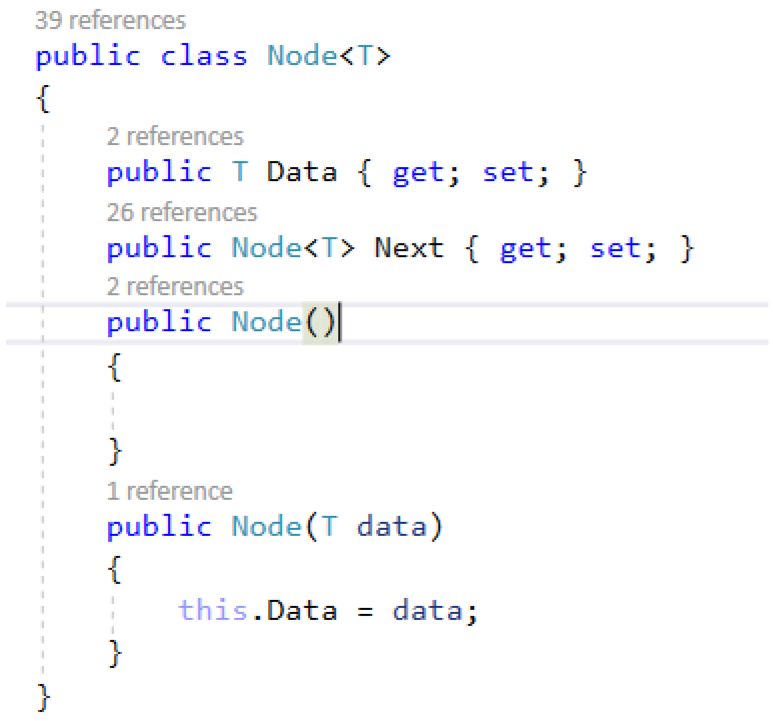
Singly Linked List Structure Code :

Singly Linked List Class :



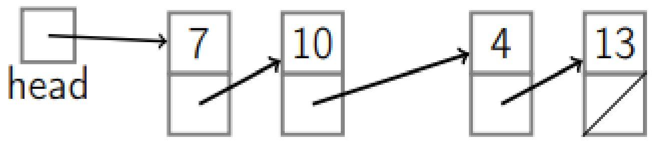
```
3 references
public class SinglyLinkedList<T>
{
    9 references
    public Node<T> Header { get; set; }
    6 references
    public Node<T> Tail { get; set; }

    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



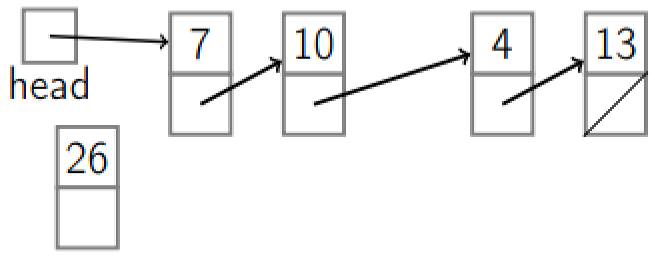
1- Insert Operations

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

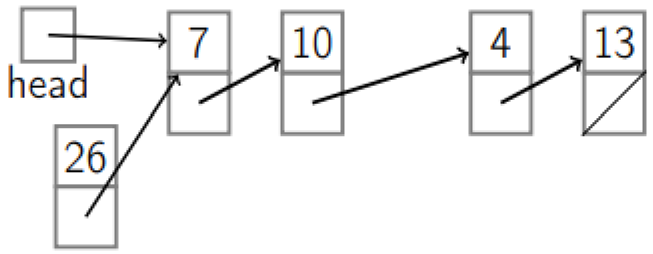
- Steps :

1. Create New Node

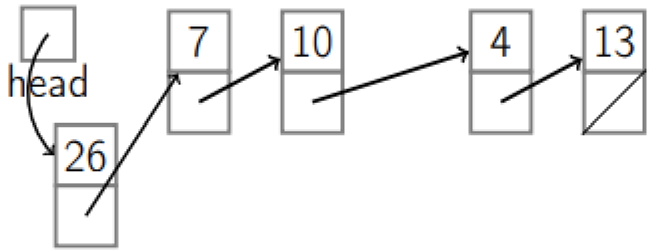


2.   If LinkedList Is Not Empty

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

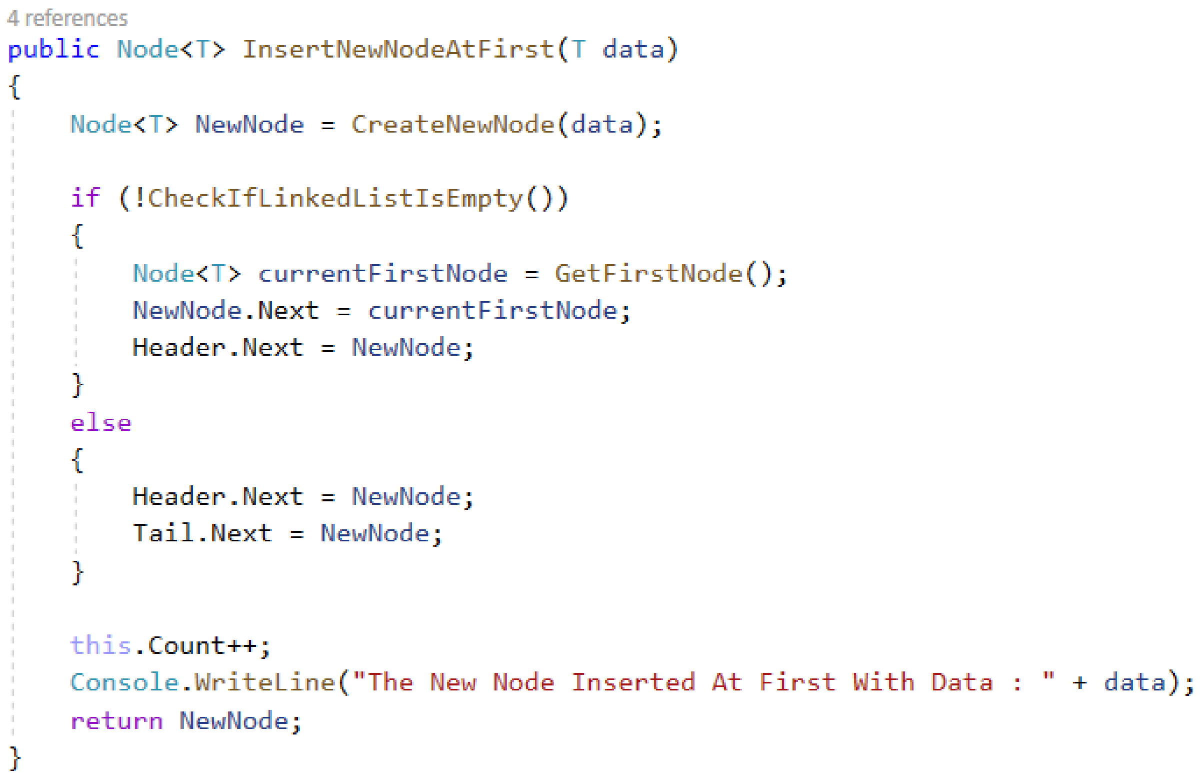


3. Else (When LinkedList Is Empty)

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 :



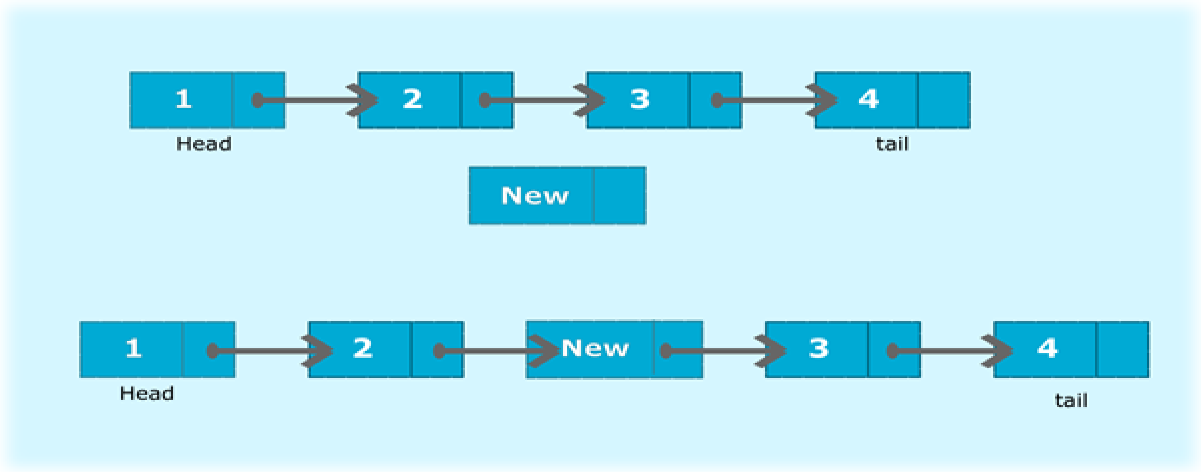
```
4 references
public Node<T> InsertNewNodeAtFirst(T data)
{
    Node<T> NewNode = CreateNewNode(data);

    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;
}
```

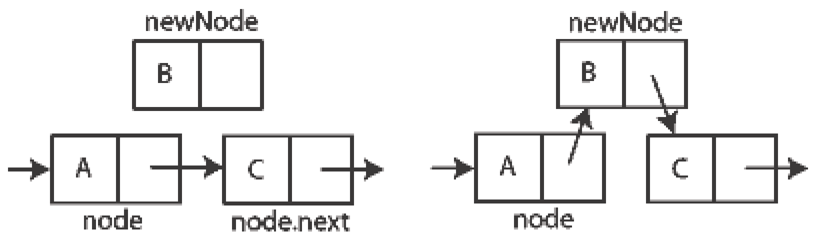
- Time Complexity :

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 :

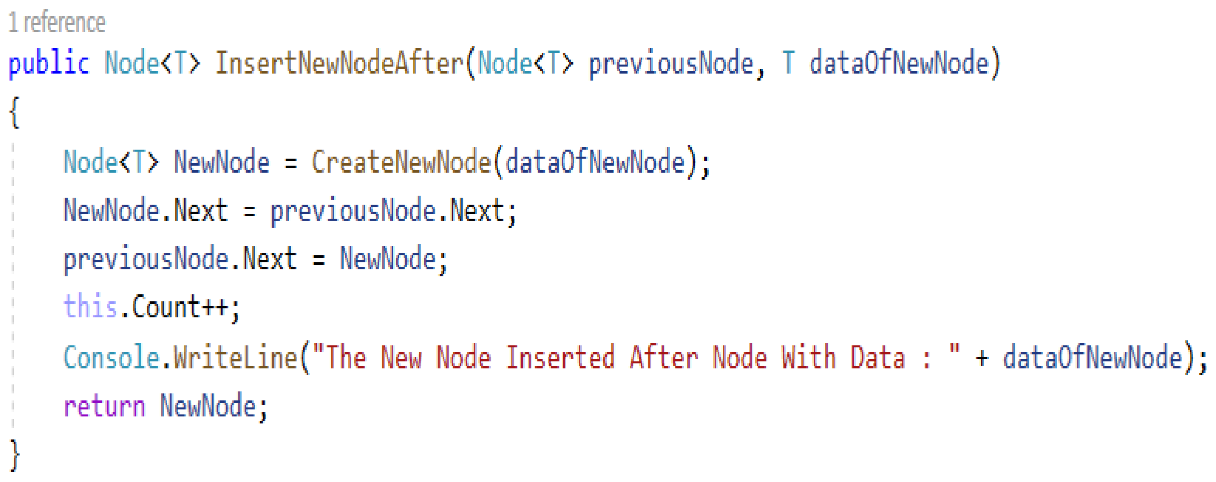




- Steps :

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 :



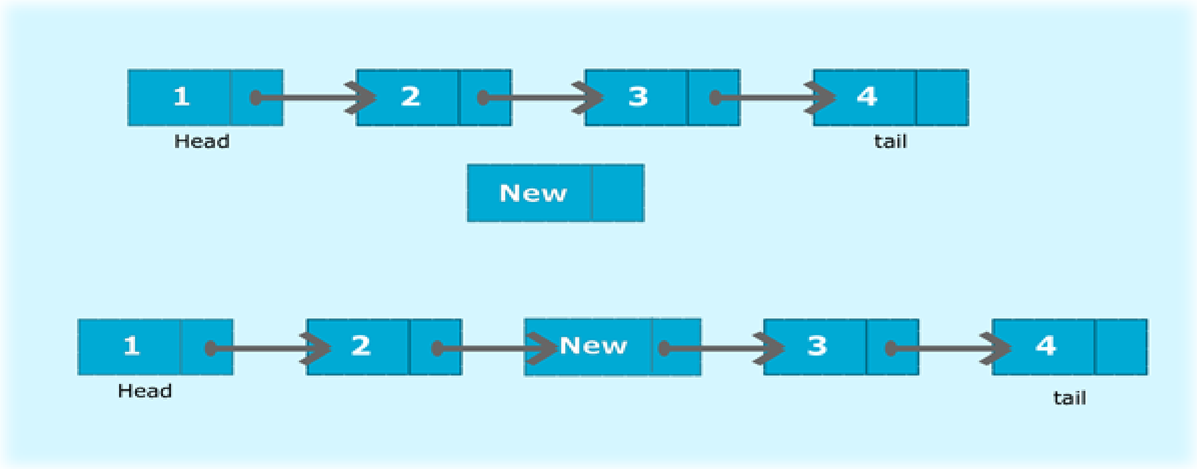
```
1 reference
public Node<T> InsertNewNodeAfter(Node<T> previousNode, T dataOfNewNode)
{
    Node<T> NewNode = CreateNewNode(dataOfNewNode);
    NewNode.Next = previousNode.Next;
    previousNode.Next = NewNode;
    this.Count++;
    Console.WriteLine("The New Node Inserted After Node With Data : "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

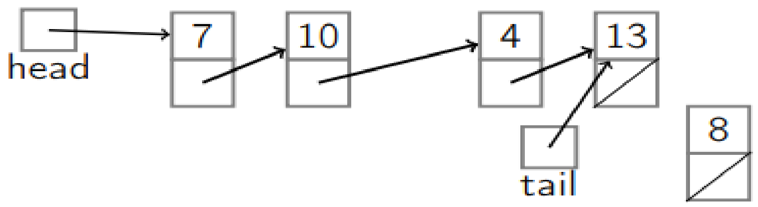
Steps :

1. If Linked List Is Empty :

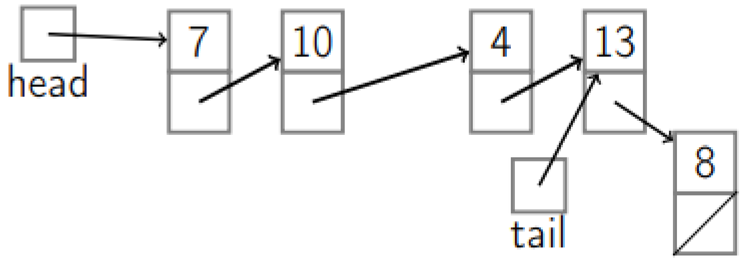
1.1 The New Node Will Insert At First As We See In The Previous Operation

2 Else (Linked List Is Not Empty)

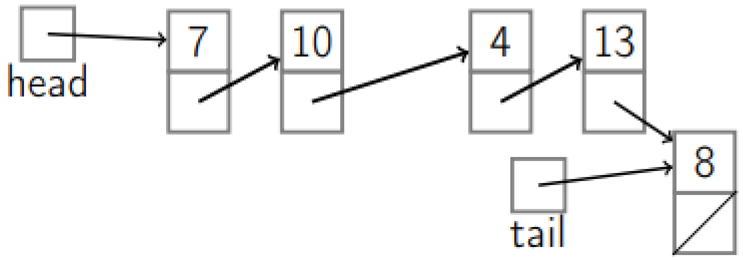
2.1 Create New Node



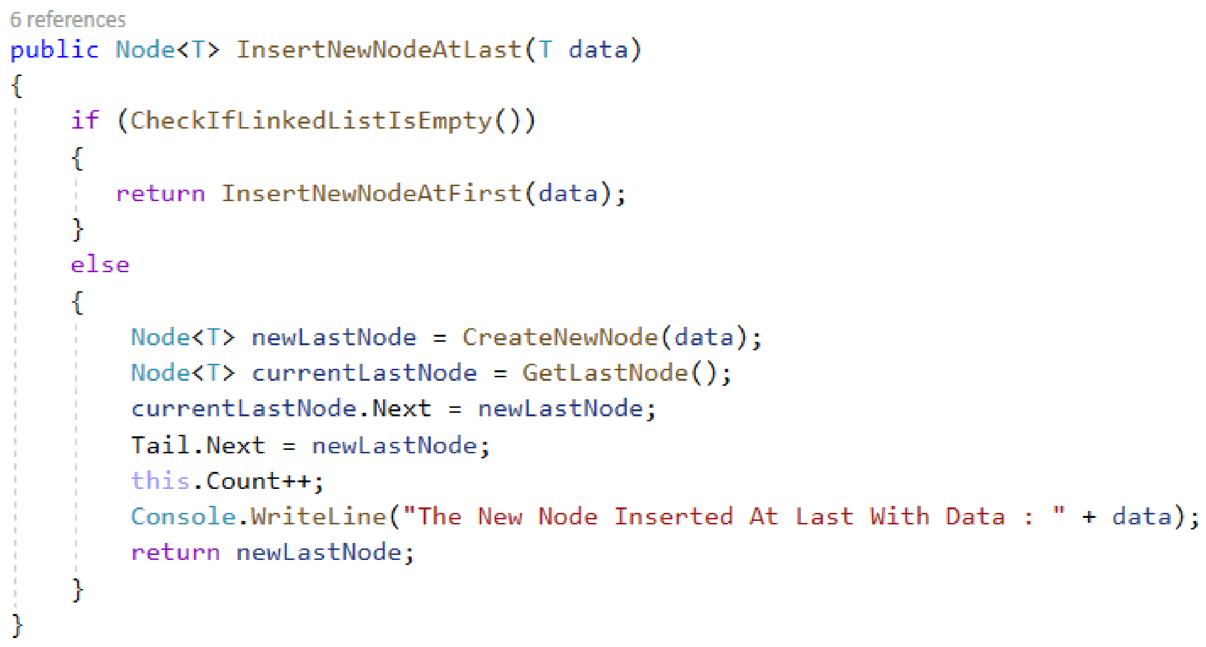
2.2 The Next Of Current Last Node Will Point To The New Node



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;
    }
}
```

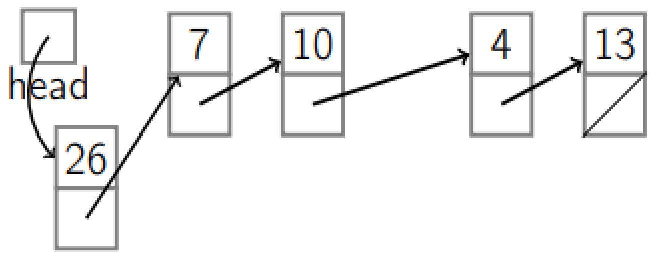
Time Complexity

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

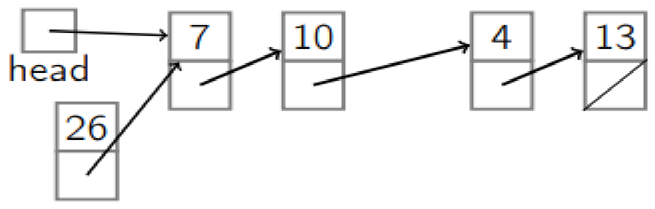
2.1 Delete The Node At The First

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

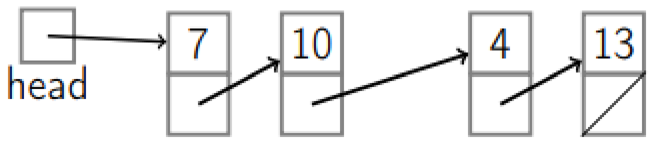


Steps :

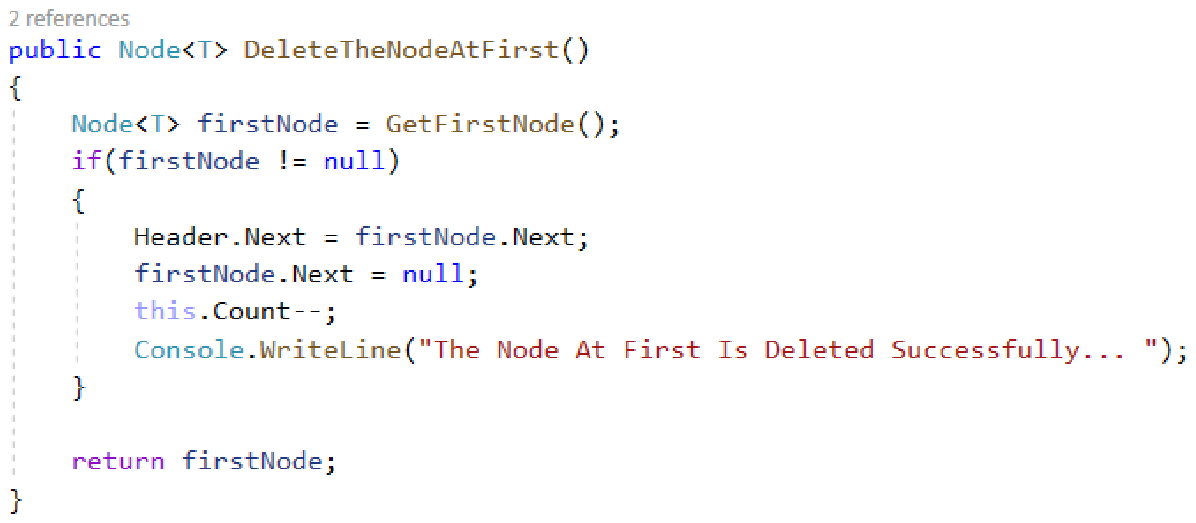
1. The Header Will Point To The Next Of Current First Node.



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 :

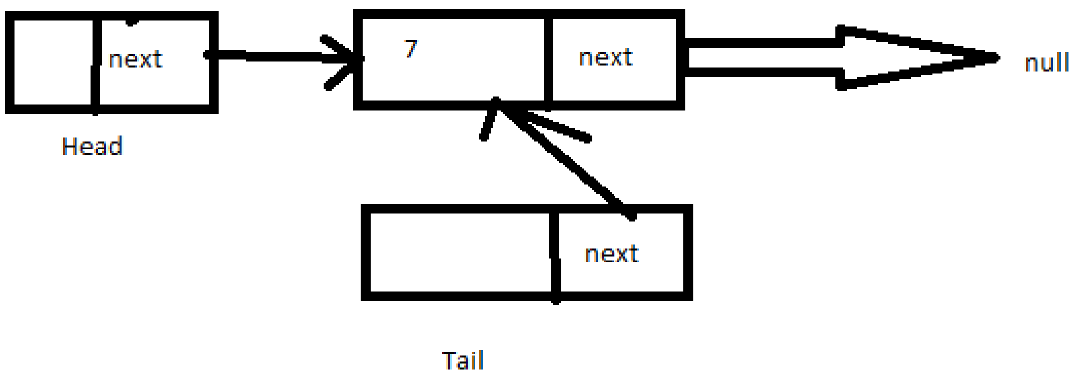
  Time Complexity Of Delete Node At First = O(1)

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.

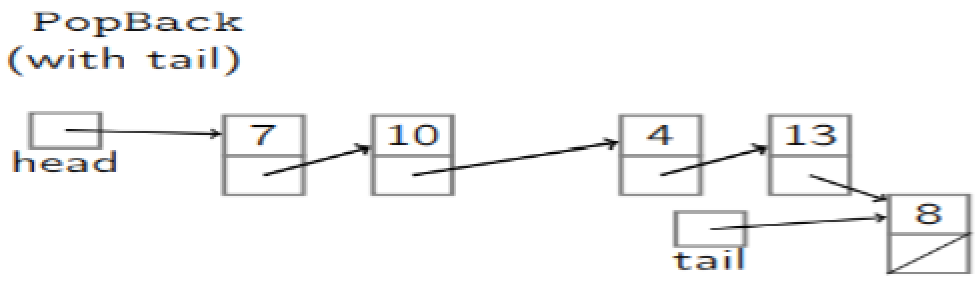
Steps

1.If Linked List Contain Only One Node

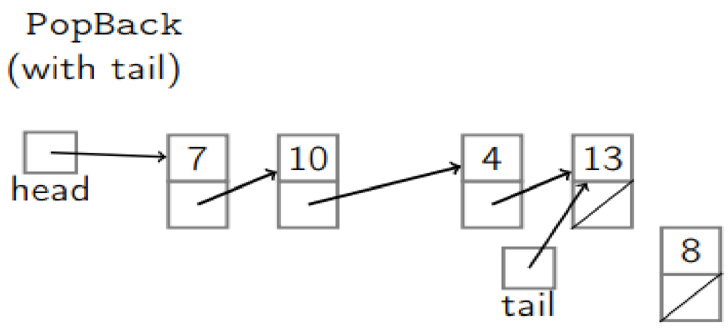


1.1 Delete The Node At First
1.2 Header Point To Null
1.3 Tail Point To Null

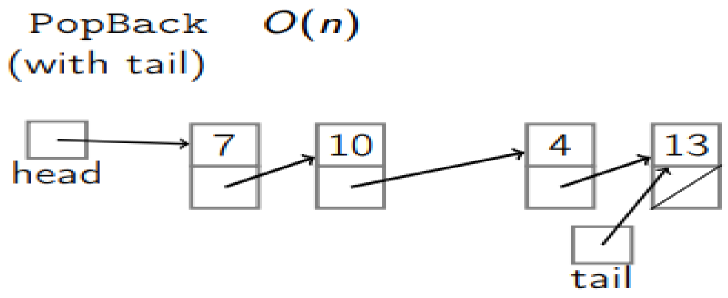
2. Else If Linked List Contain More Than One Node



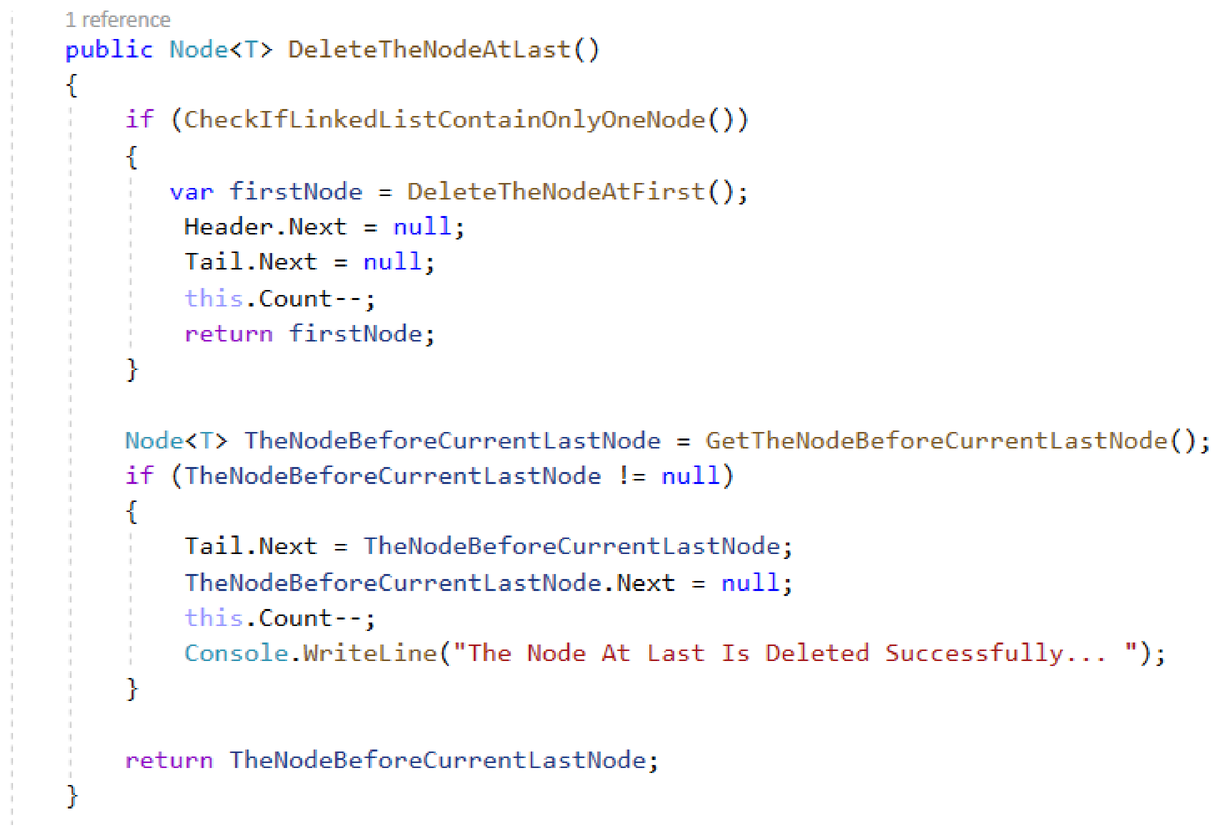
2.1 The Tail Point To The Node That Was Before The Current Last Node.



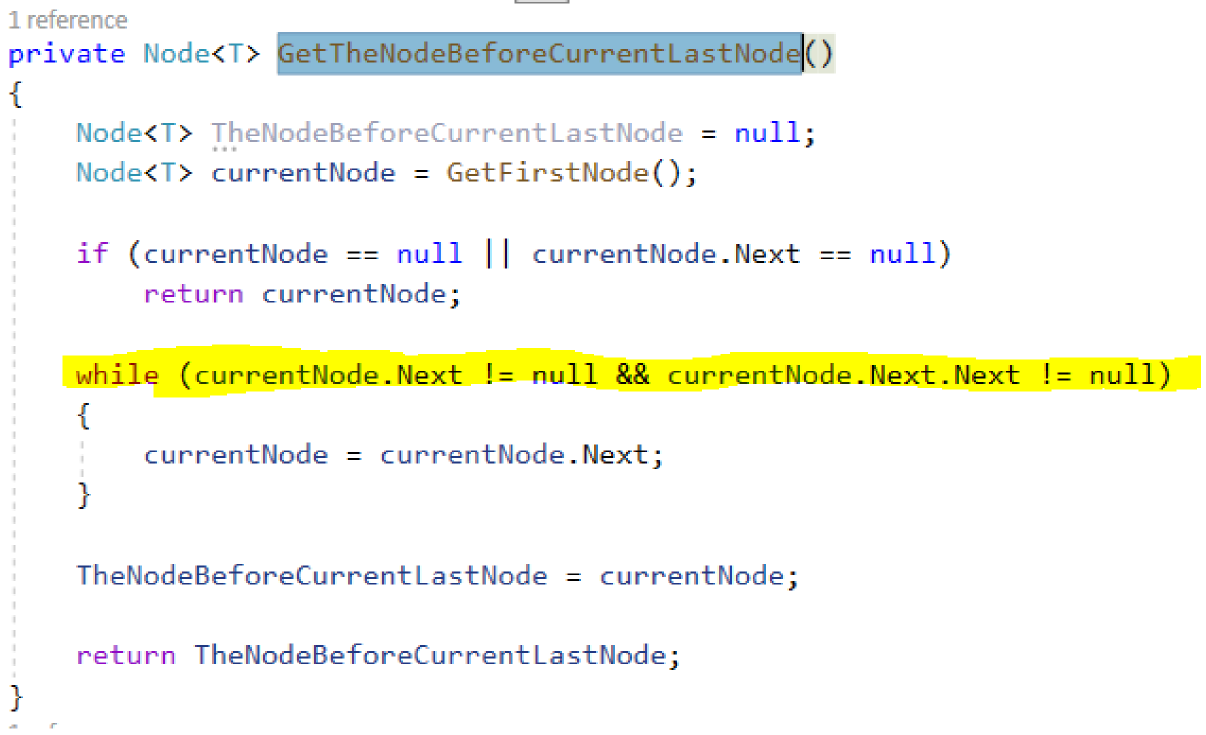
2.2 The Next Of The New Last Node Point To Null



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;
}
```



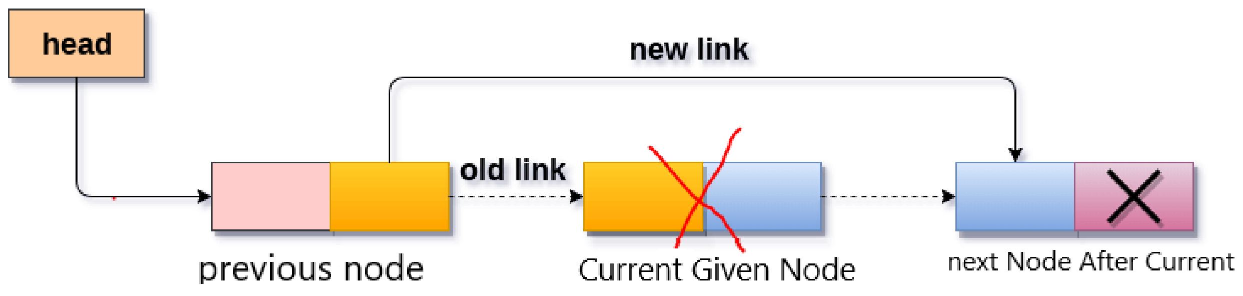
```
1 reference
private Node<T> GetTheNodeBeforeCurrentLastNode()
{
    Node<T> TheNodeBeforeCurrentLastNode = null;
    Node<T> currentNode = GetFirstNode();

    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.



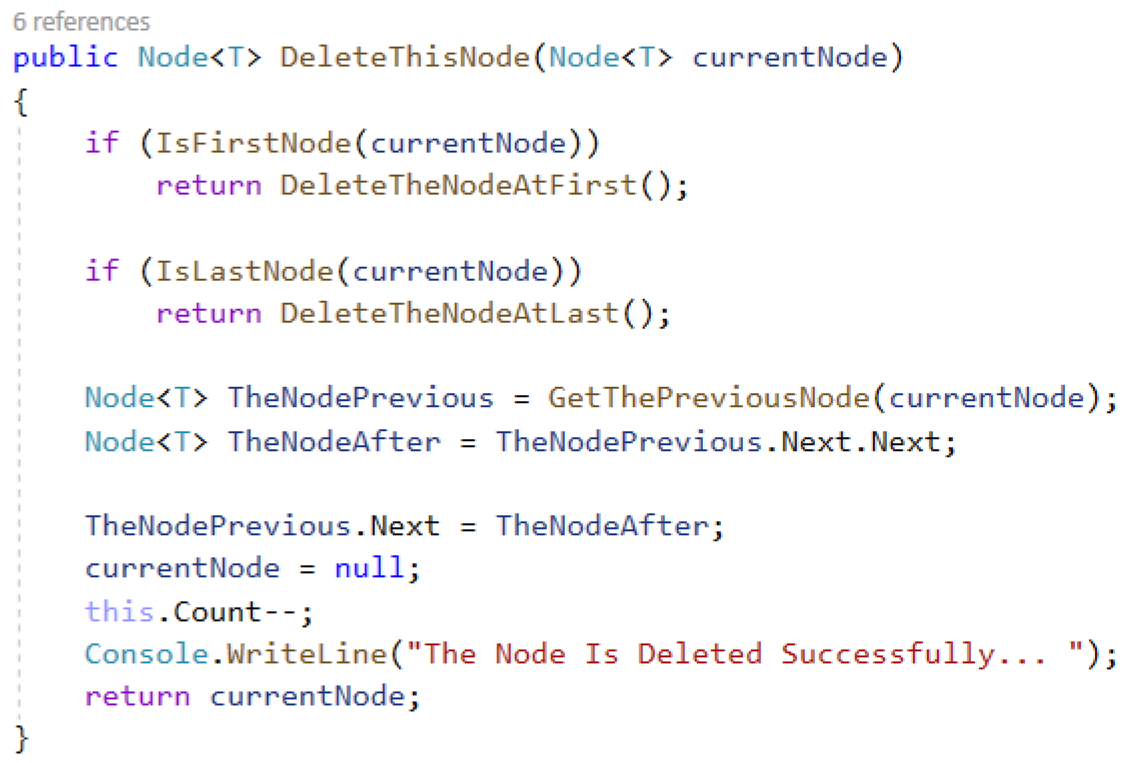
Steps :

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

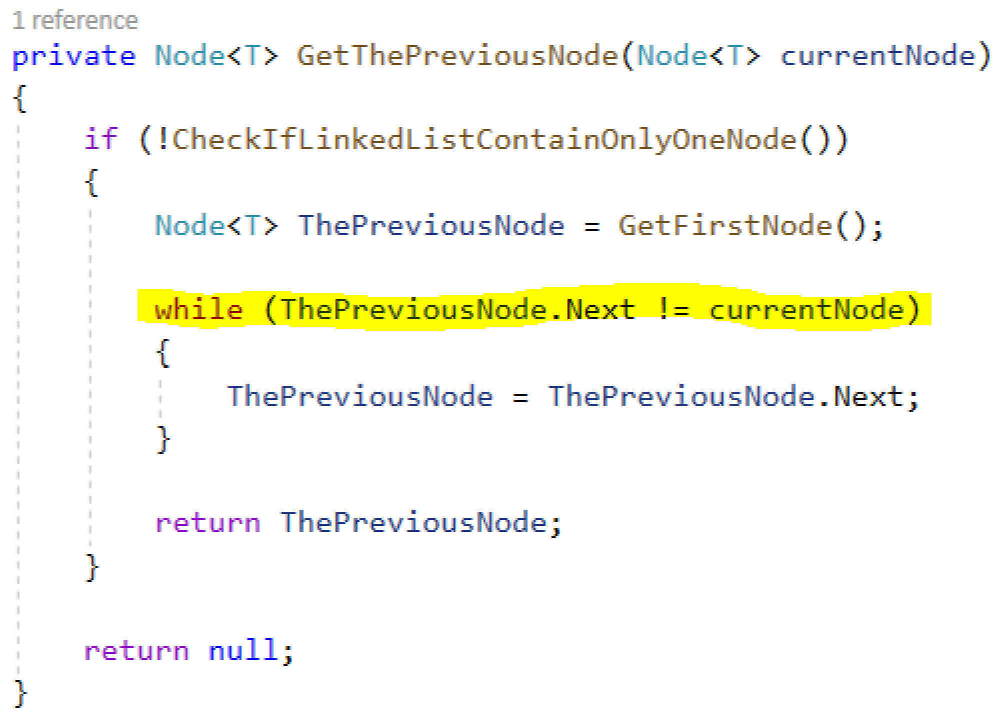
2.If Current Node Is Last Node
2.1 Delete Node At Last
2.2 RETURN

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;
}
```

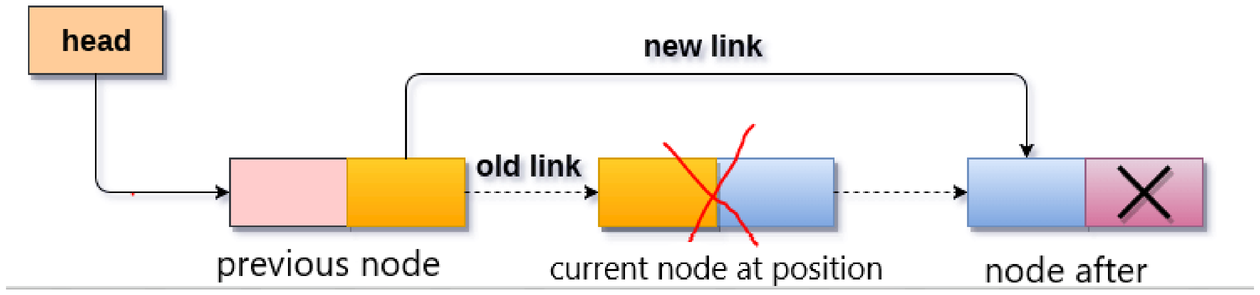
Time Complexity

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

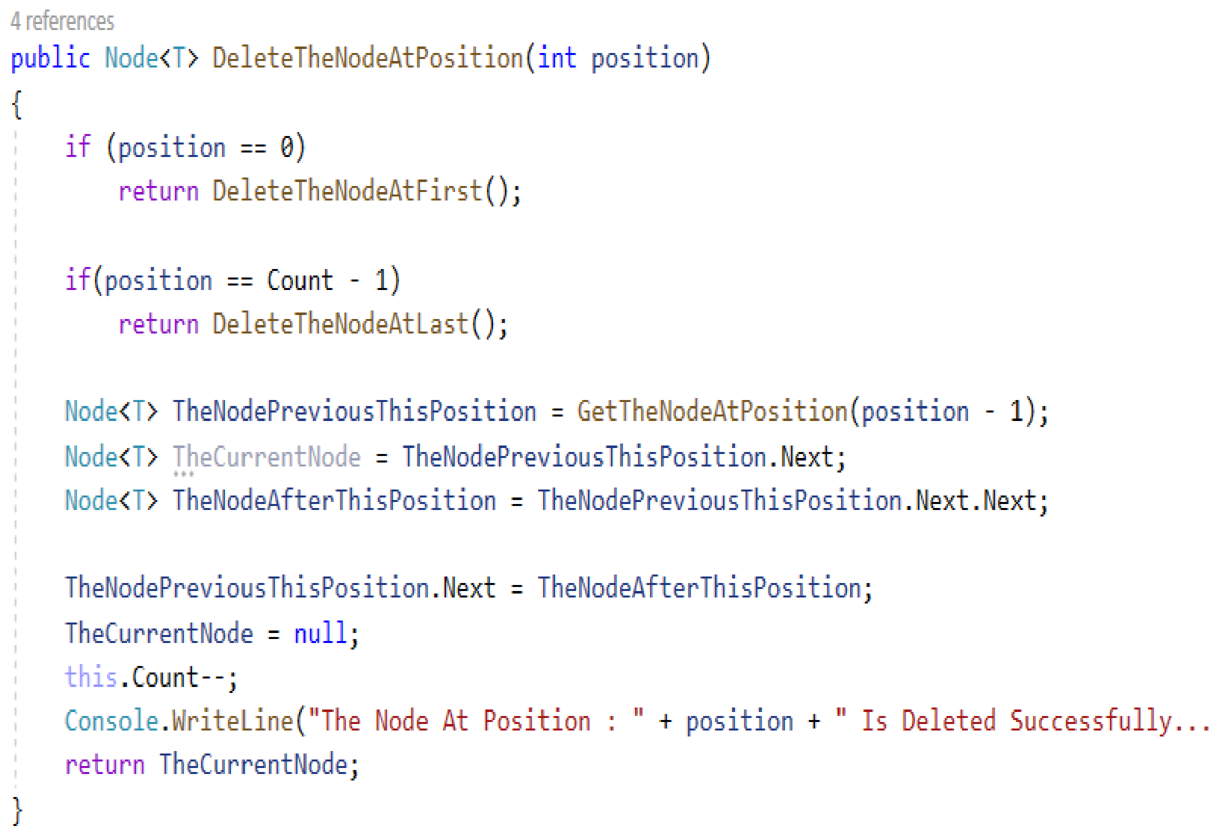
Steps :

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

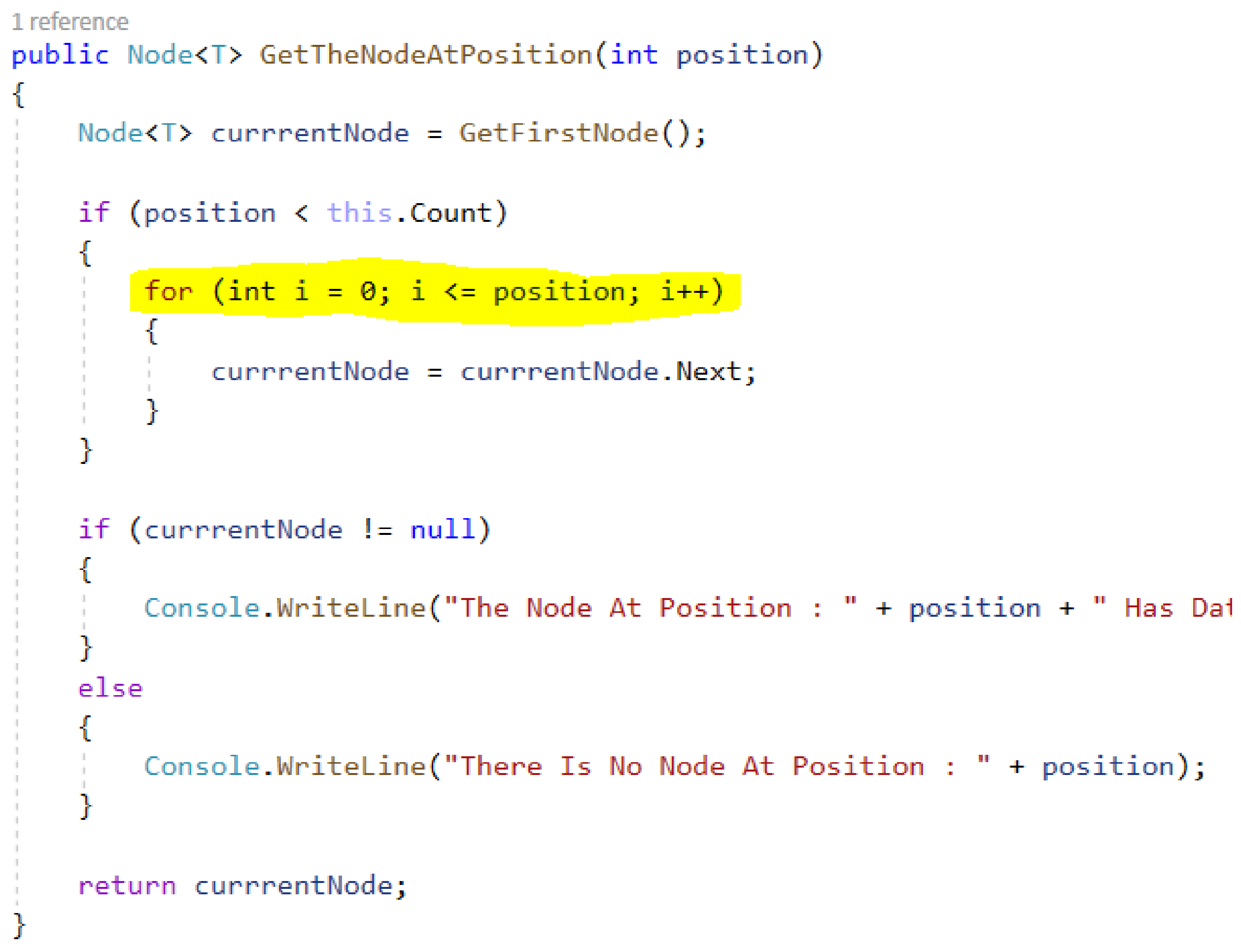


3.1 Get The Previous Node Of This Position
3.2 The Next Of The Previous Node Point To The Node After This Position

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;
}
```



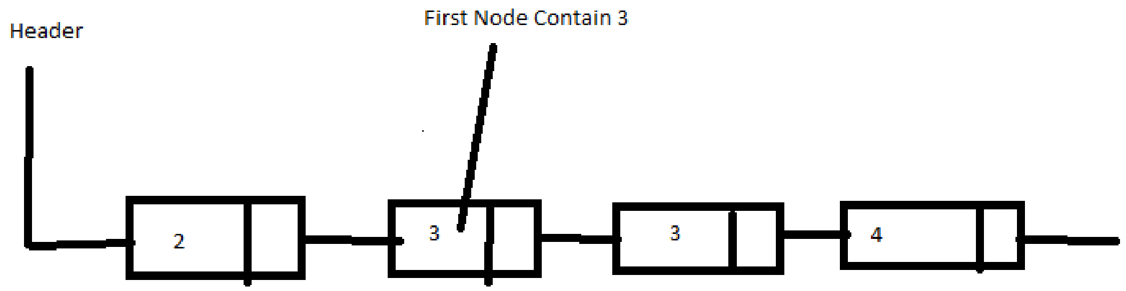
```
1 reference
public Node<T> GetTheNodeAtPosition(int position)
{
    Node<T> currrentNode = GetFirstNode();

    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.

3- Search Operation (Find Node)

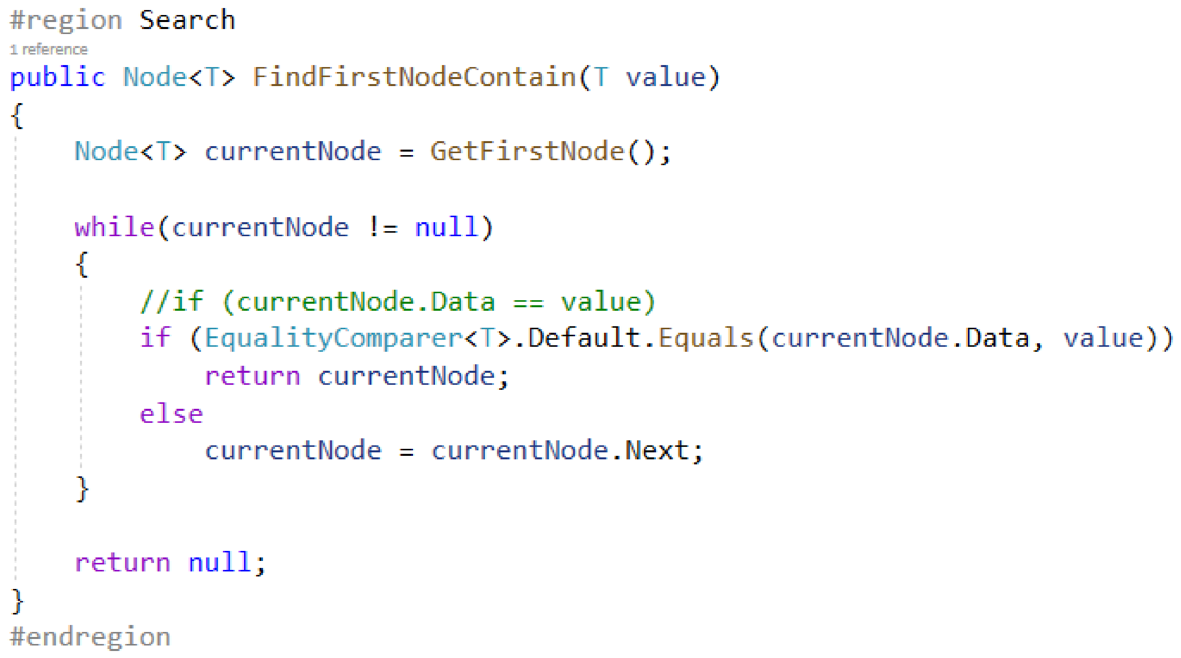
Find First Node Contain given Value



Steps :

```
1- Define Variable to Store The Current Node
2- Get First Node In The Linked List
3- Current Node = First Node
3- While ( Current Node != null )
3.1 If Data Of Current Node ==  Value
3.2 Return Current Node
3.3 Else (  Data Of Current Node !=  Value )
3.4 Current Node = Next Node
4- Return Null
```

Code :



```
#region Search
1 reference
public Node<T> FindFirstNodeContain(T value)
{
    Node<T> currentNode = GetFirstNode();

    while(currentNode != null)
    {
        //if (currentNode.Data == value)
        if (EqualityComparer<T>.Default.Equals(currentNode.Data, value))
            return currentNode;
        else
            currentNode = currentNode.Next;
    }

    return null;
}
#endregion
```

Time Complexity :

Time Complexity Of Find First Node Contain Given Value = O(n).

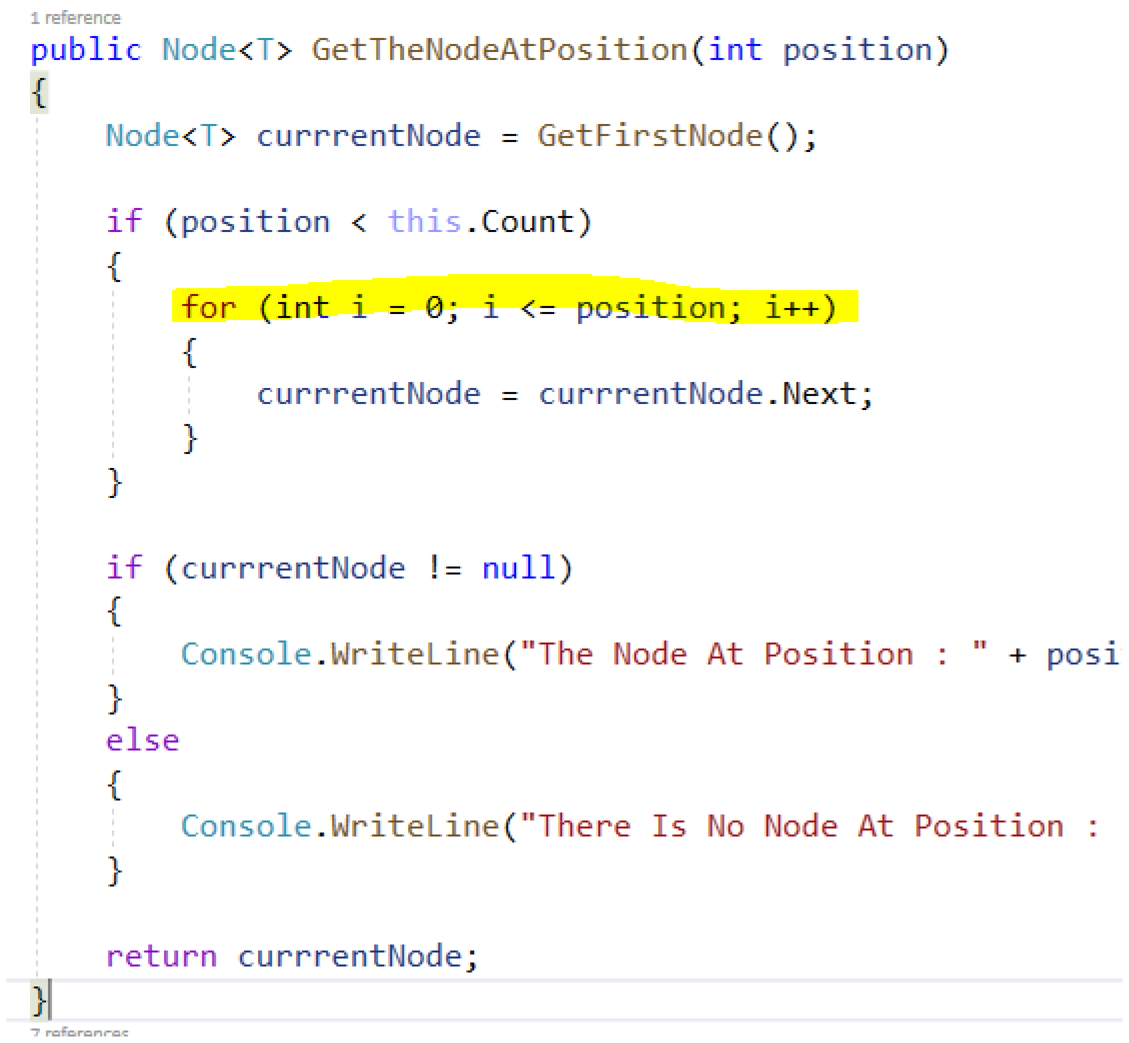
4- Read Operations

4.1 Get Node At Position

Steps :

```
1- Define Variable to Store The Current Node
2- Get First Node In The Linked List
3-  If  Position < Linked List Length
3.1 Loop  (int i = 0; i <= position; i++)
3.2 Current Node = Next Node
3.3 Return Current Node
4. Else Position >= Linked List Length
4.1 Return Null
```

Code :



```
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 : " + posi
    }
    else
    {
        Console.WriteLine("There Is No Node At Position :
    }

    return currrentNode;
}
```

Time Complexity :

Time Complexity Of Get Node At Position = O(n).

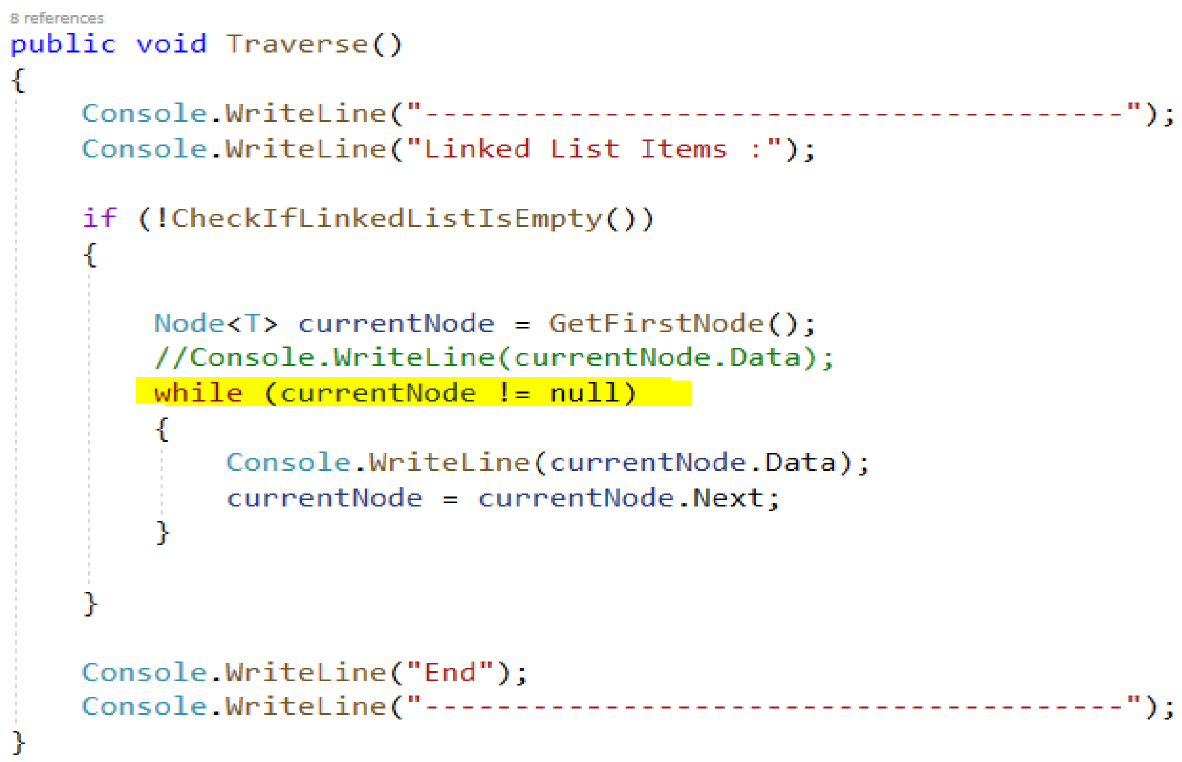
4.2 Traverse Linked List

Traverse Linked List To Get Or Print Data Of All Nodes.

Steps :

1- Define Variable to Store The Current Node
2- Get First Node In The Linked List
3- If Linked List Not Empty
3.1 Print Current Node Data
3.2 Current Node = Next Node

Code :



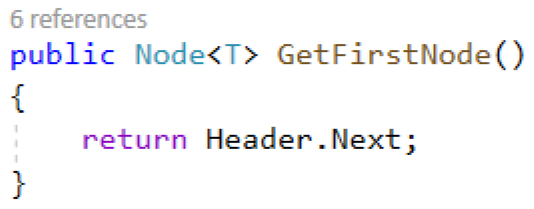
```
8 references
public void Traverse()
{
    Console.WriteLine("---------------------------------------");
    Console.WriteLine("Linked List Items :");

    if (!CheckIfLinkedListIsEmpty())
    {

        Node<T> currentNode = GetFirstNode();
        //Console.WriteLine(currentNode.Data);
        while (currentNode != null)
        {
            Console.WriteLine(currentNode.Data);
            currentNode = currentNode.Next;
        }

    }

    Console.WriteLine("End");
    Console.WriteLine("---------------------------------------");
}
```

Time Complexity :

Time Complexity Of Traverse Operation = O(n).

4.3 Get Node At First

Code :



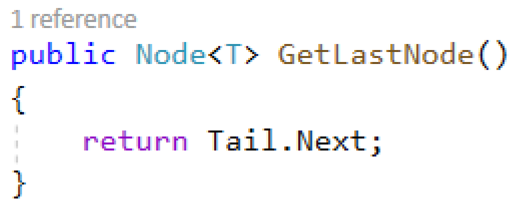
```
6 references
public Node<T> GetFirstNode()
{
    return Header.Next;
}
```

Time Complexity :

Time Complexity Of Get Node At First = O(1).

4.4 Get Node At Last

Code :



```
1 reference
public Node<T> GetLastNode()
{
    return Tail.Next;
}
```

Time Complexity :

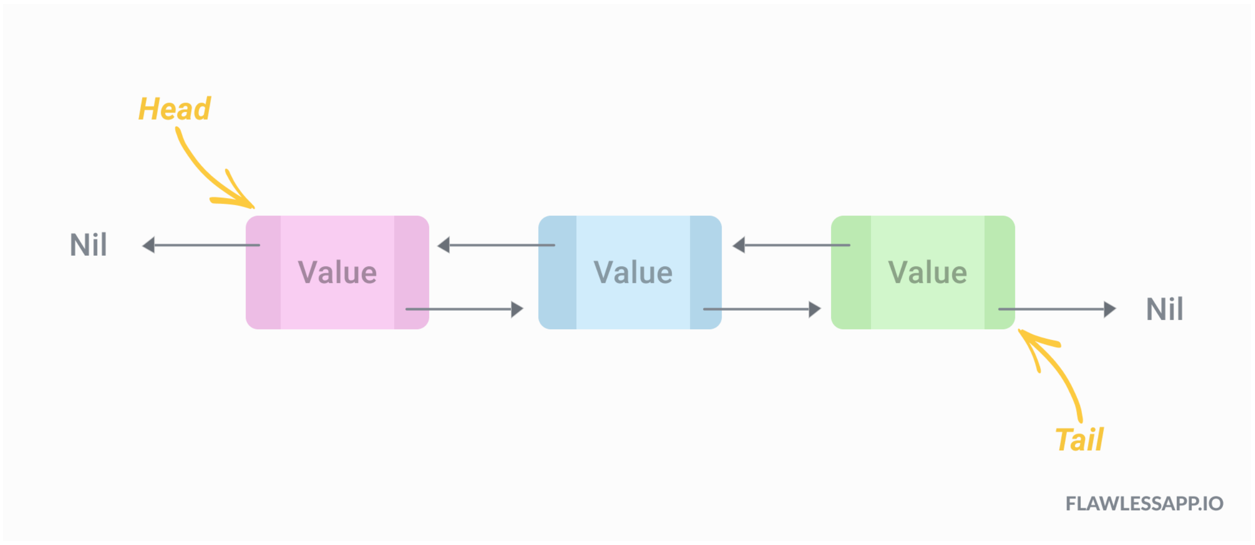
Time Complexity Of Get Node At Last = O(1).

Doubly Linked List

What Is Doubly Linked List ?

In Doubly Linked List Every Node Contains Two Pointers :

- Next Pointer
- Previous Pointer



Advantages Of Doubly Linked List Over Singly Linked List :

<u>1- Previous Pointer Help To Reduce Cost For Some Operations That Have O(n) On Singly Linked List To O(1).</u>

Operations in singly linked list like :

- Insert new node Before
- Delete node

These operations have O(n) Because they need the previous node of the current node and the node in singly linked list has only pointer to the next node <u>so singly linked list need to traverse from first node until required node</u>.
But in doubly linked list , the node contains previous pointer <u>so doubly linked list can find the previous node of current node in one step.</u>

<u>2- Previous Pointer Help Doubly Linked List To Traverse In The Opposite direction (From Tail To Header)</u>

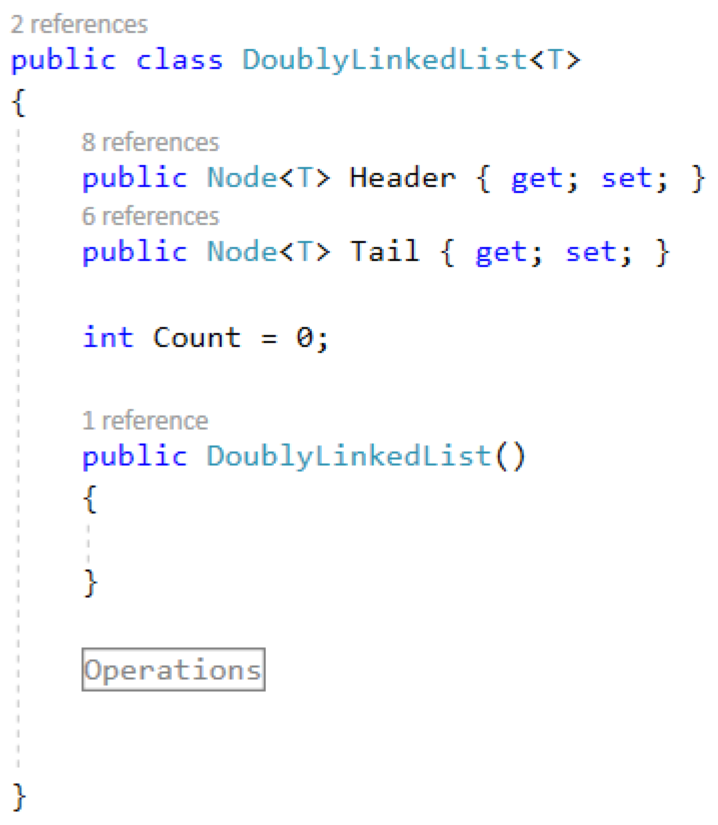
Singly Linked List Can Traverse only on one direction(from header to tail).
But doubly linked list can traverse from header to tail or from tail to header.

Disadvantages Of Doubly Linked List :

1- Doubly linked list need extra space than singly linked list to store previous pointer.

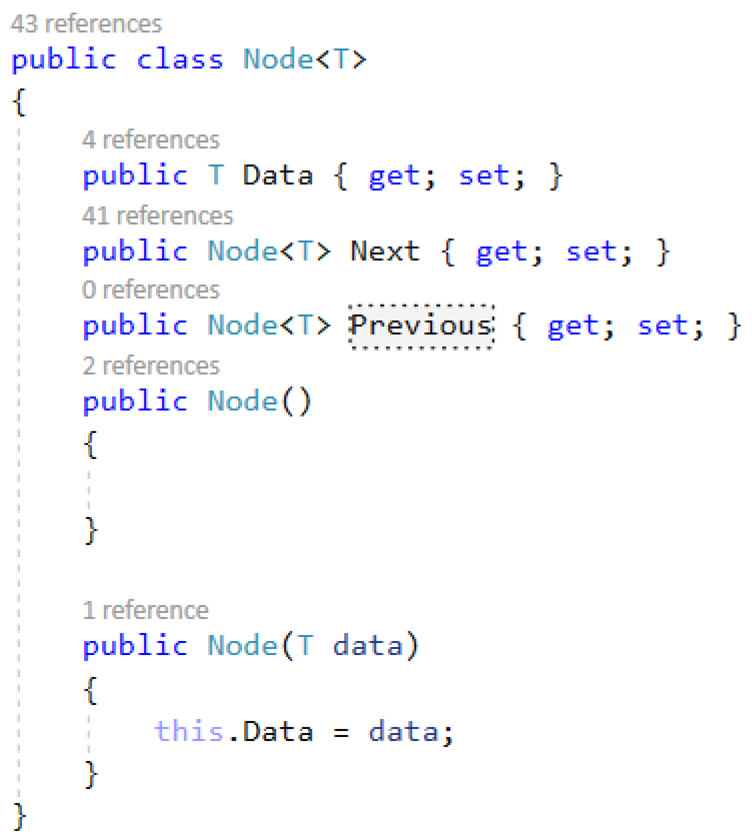
2- Code Of Doubly linked list is more complex than singly linked to maintain previous pointer.

Doubly Linked List Structure Code :



```
2 references
public class DoublyLinkedList<T>
{
        8 references
        public Node<T> Header { get; set; }
        6 references
        public Node<T> Tail { get; set; }

        int Count = 0;

        1 reference
        public DoublyLinkedList()
        {

        }

        Operations


}
```

Doubly Node Code :



```
43 references
public class Node<T>
{
    4 references
    public T Data { get; set; }
    41 references
    public Node<T> Next { get; set; }
    0 references
    public Node<T> Previous { get; set; }
    2 references
    public Node()
    {

    }

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

Operations Of Doubly Linked List :

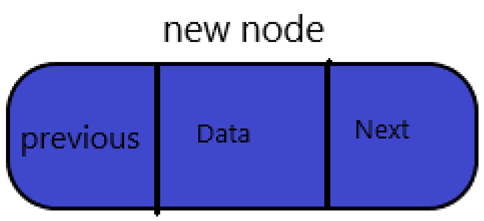
1- Insert Operations

1.1 Insert Node At First
1.2 Insert Node At Last
1.3 insert Node After
1.4 insert Node Before

1.1 Insert Node At First

Steps :

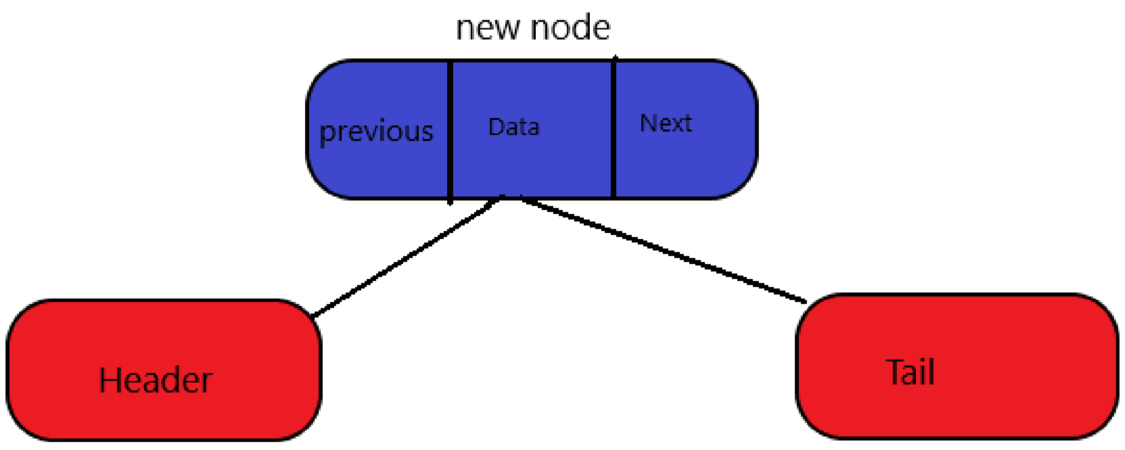
1.Create New Node



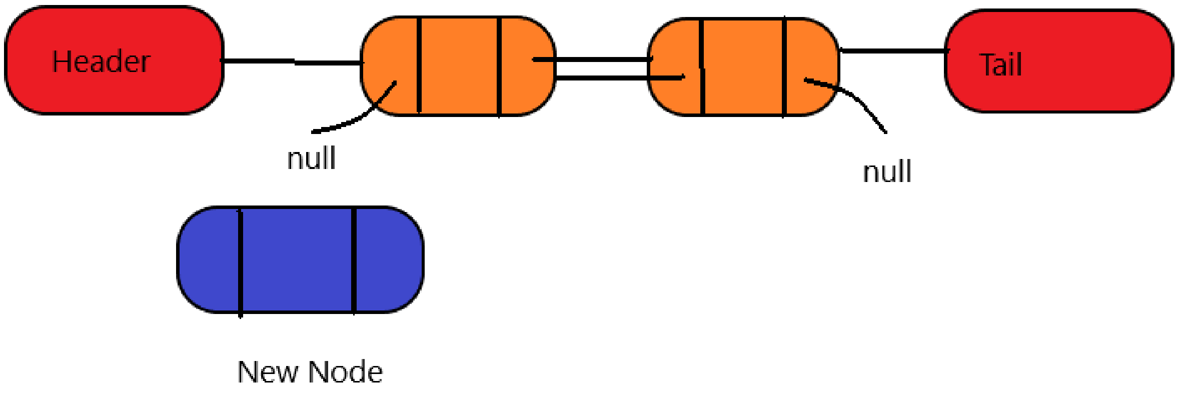
2.If linked list is empty

Header
Tail

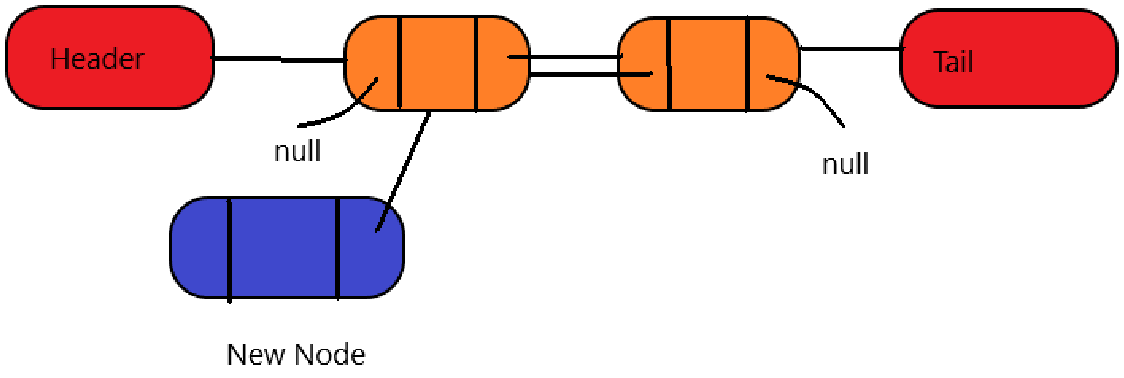
2.1 Header And Tail Point To The New Node.



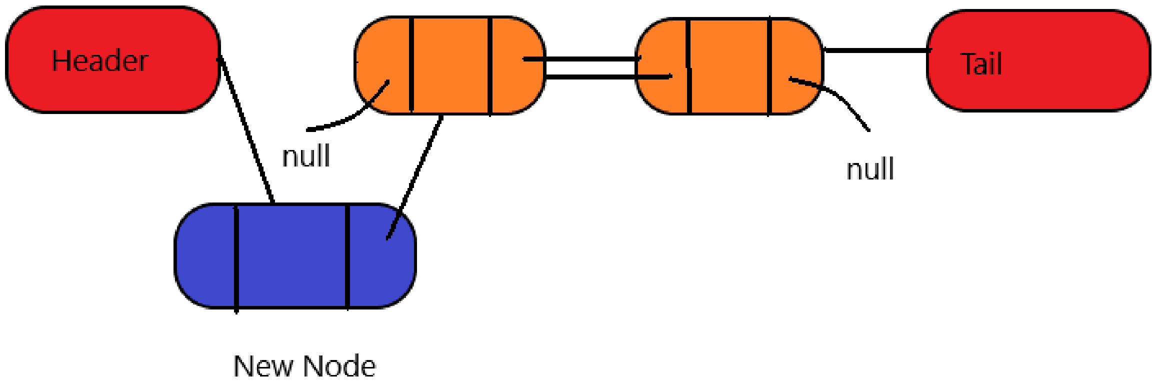
3.Else (Linked List is Not Empty)



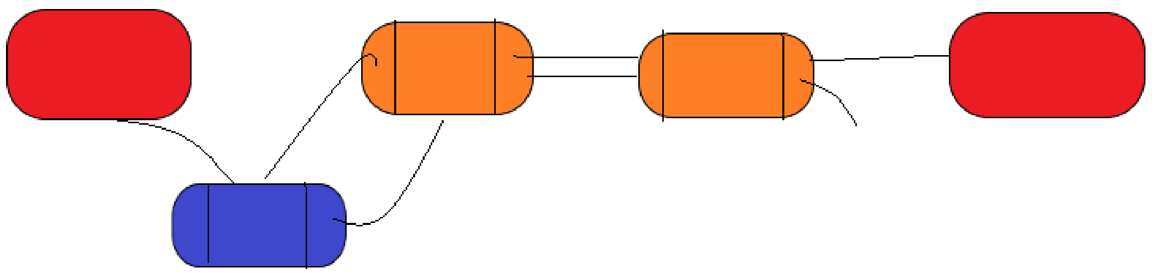
3.1 Next Of New Node Point To Current First Node



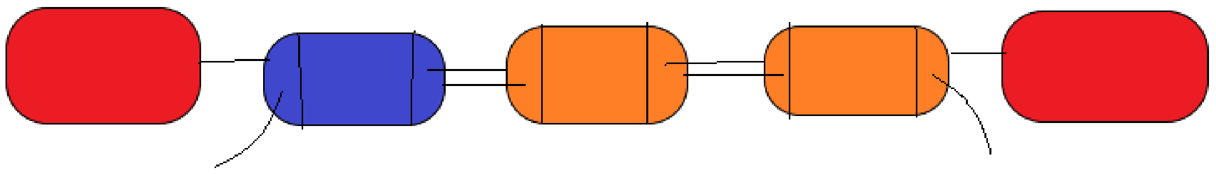
3.2 Header Point To New Node



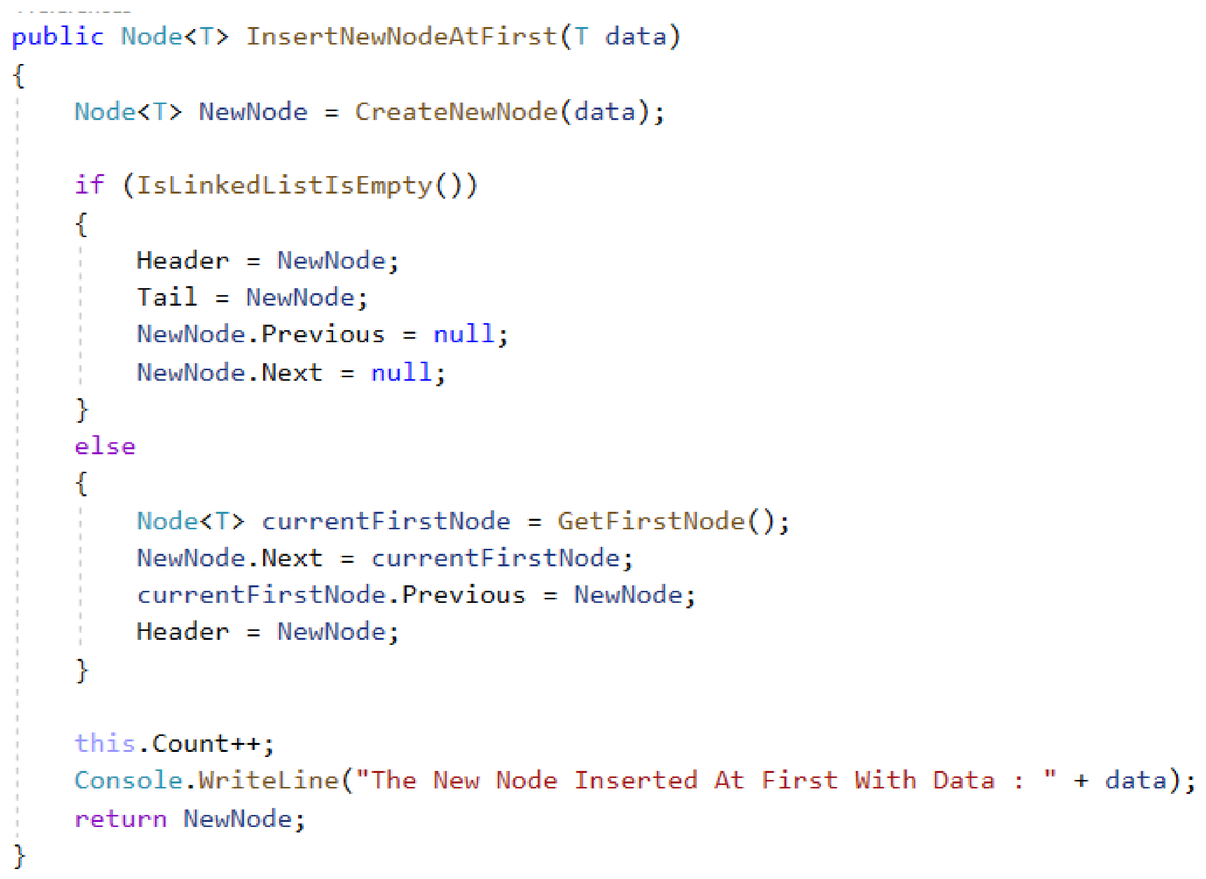
3.3 Previous Of Current First Node Point To New Node



Finally



Code



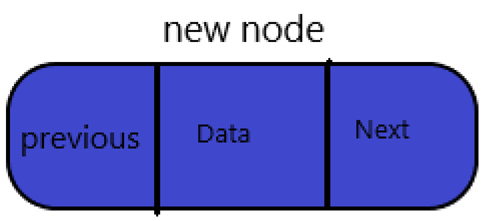
```
public Node<T> InsertNewNodeAtFirst(T data)
{
    Node<T> NewNode = CreateNewNode(data);

    if (IsLinkedListIsEmpty())
    {
        Header = NewNode;
        Tail = NewNode;
        NewNode.Previous = null;
        NewNode.Next = null;
    }
    else
    {
        Node<T> currentFirstNode = GetFirstNode();
        NewNode.Next = currentFirstNode;
        currentFirstNode.Previous = NewNode;
        Header = NewNode;
    }

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

Time Complexity = O(1)

1.2 Insert Node At Last

Steps :

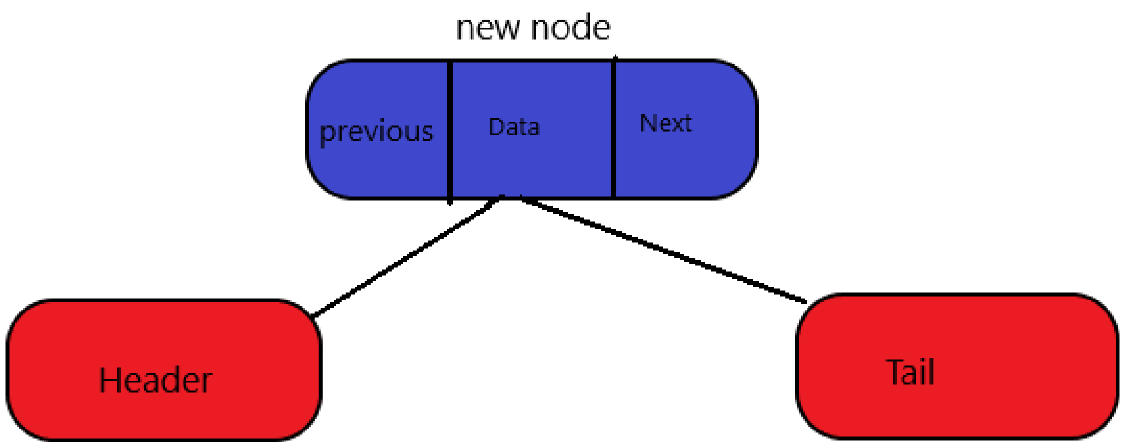
1.Create New Node



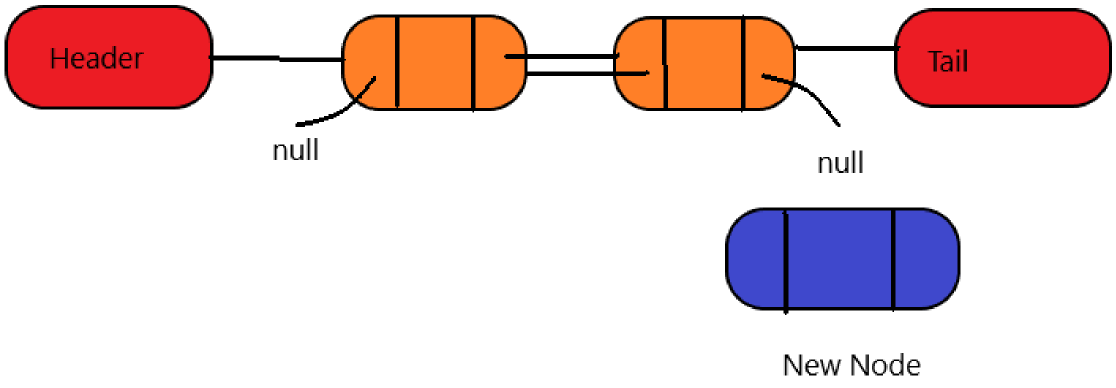
2.If linked list is empty

Header
Tail

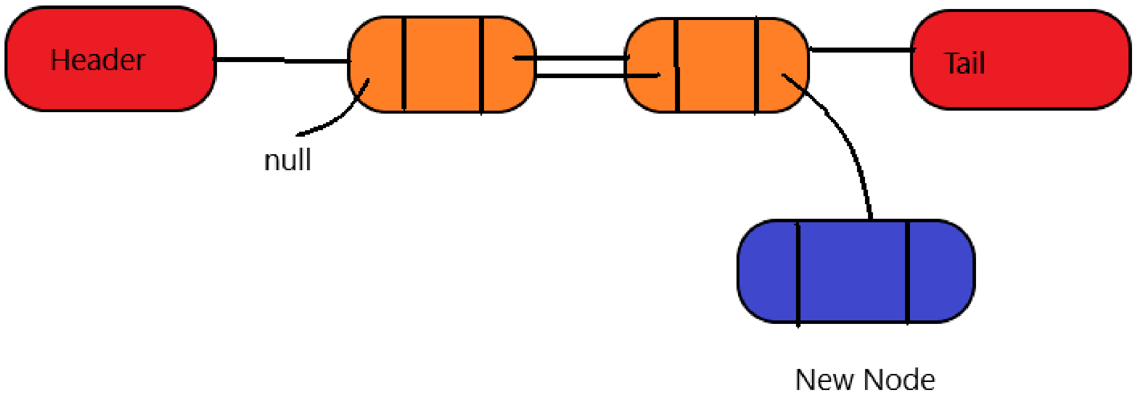
2.1 Insert New Node At First



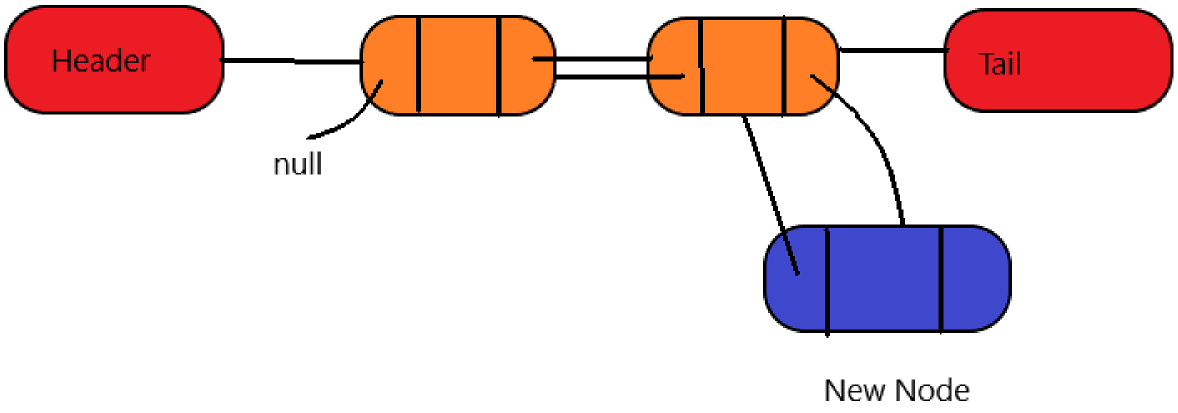
3.Else (Linked List is Not Empty)



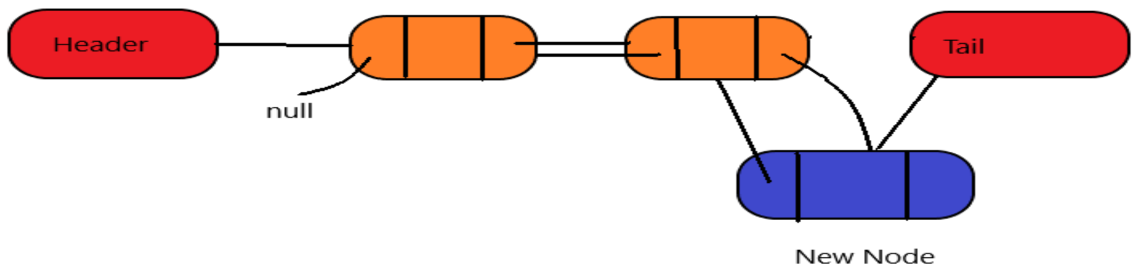
3.1 The Next Of Current Last Node Point To New Node



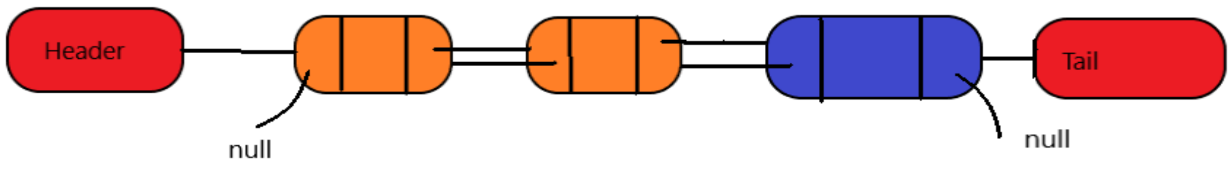
3.2 Previous Of New Node Point To Current Last Node



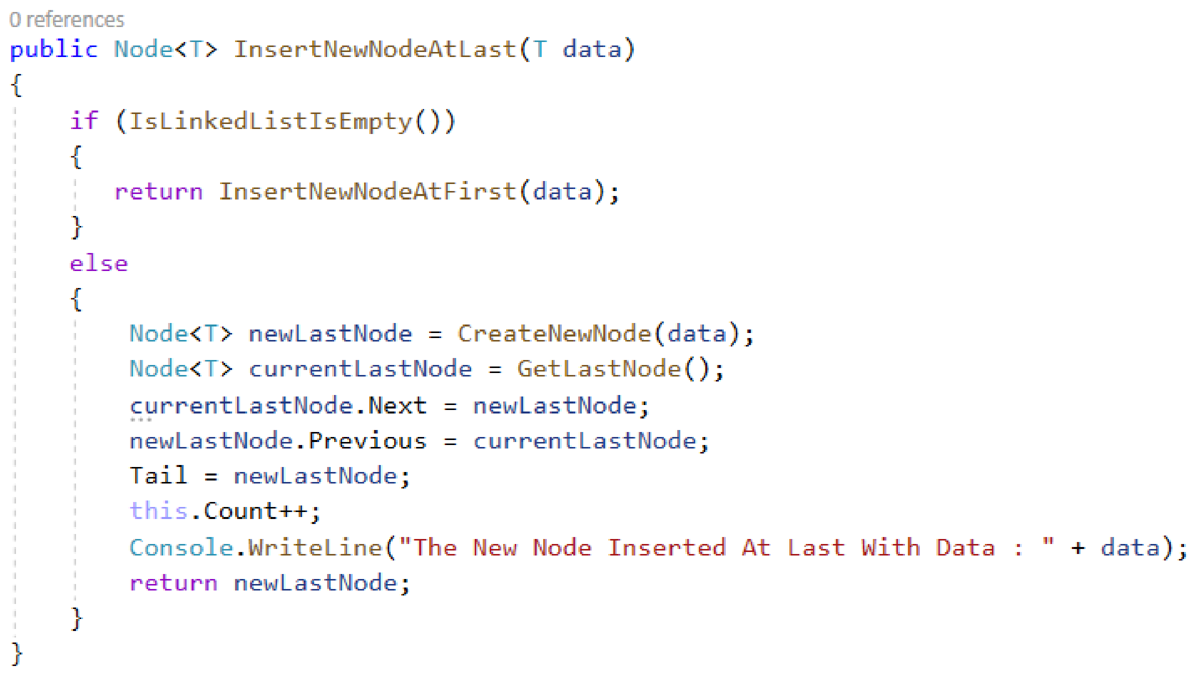
3.3 Tail Point To New Node



Finally :



Code

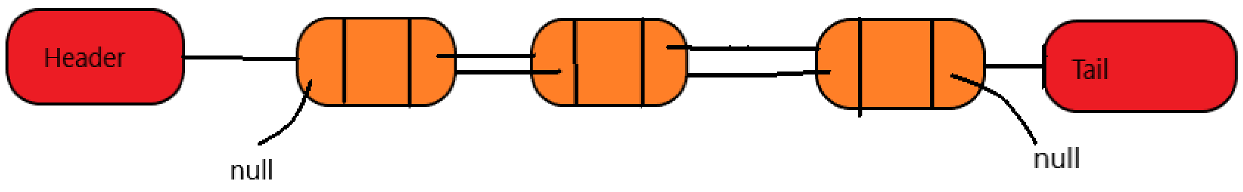


```
0 references
public Node<T> InsertNewNodeAtLast(T data)
{
    if (IsLinkedListIsEmpty())
    {
       return InsertNewNodeAtFirst(data);
    }
    else
    {
        Node<T> newLastNode = CreateNewNode(data);
        Node<T> currentLastNode = GetLastNode();
        currentLastNode.Next = newLastNode;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

Time Complexity = O(1)

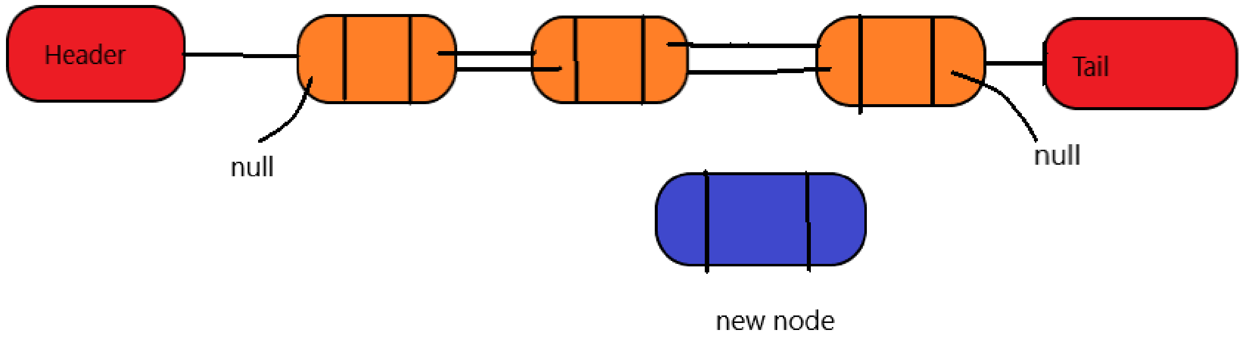
1.3 Insert Node After

Ex : Insert new node after second node

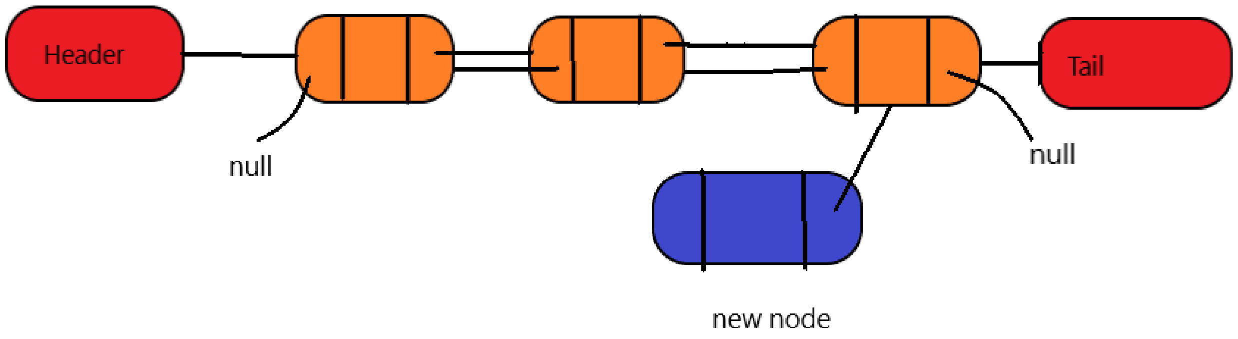


Steps :

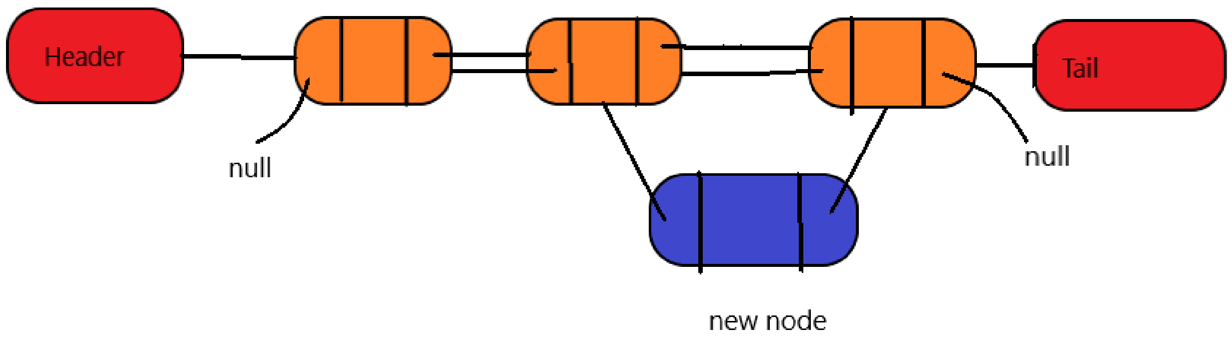
1.Create New Node



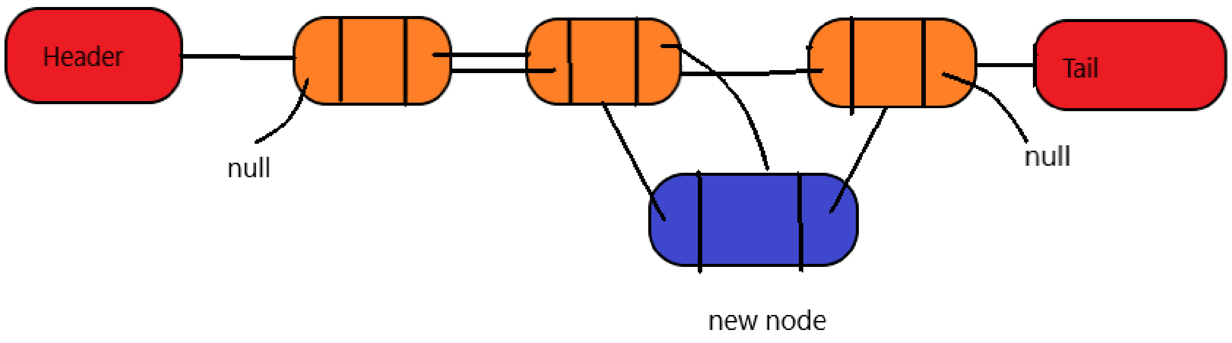
2.The Next Of New Node Point To Next Of Previous Node



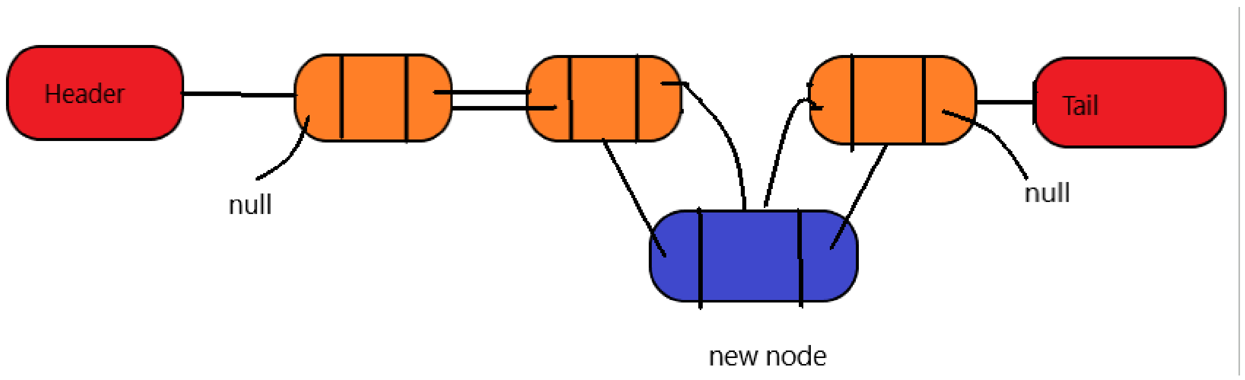
3. The Previous Of New Node Point To Of Previous Node



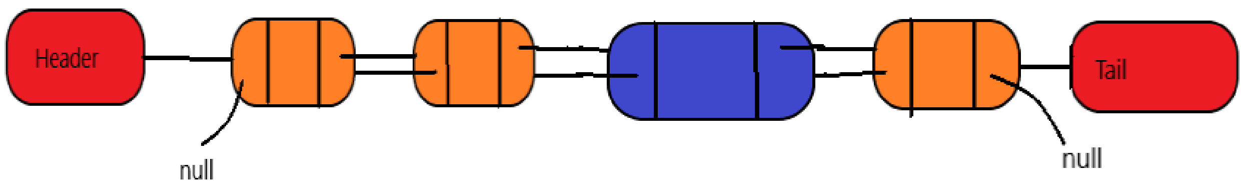
4. The Next Of Previous Node Point To New Node



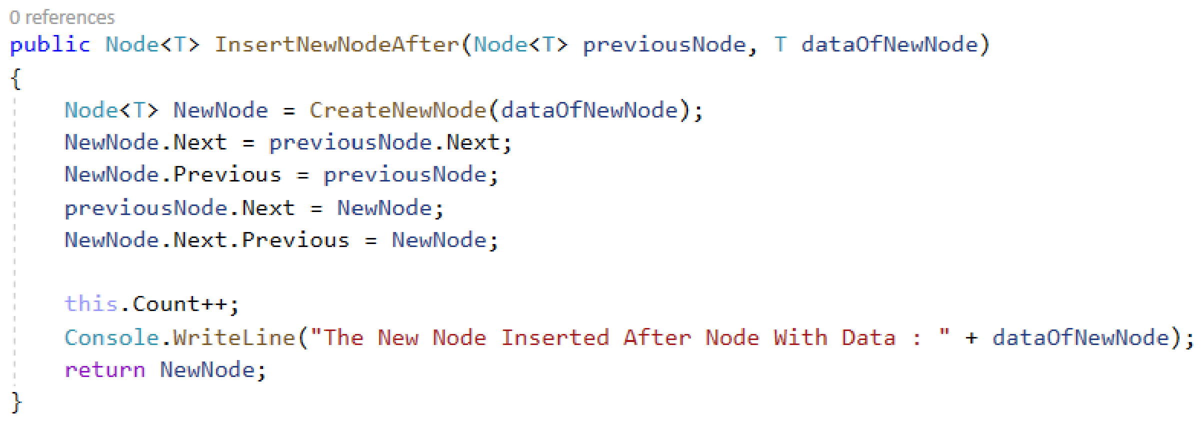
5. Previous Of Next Node Point To New Node



Finally :



Code



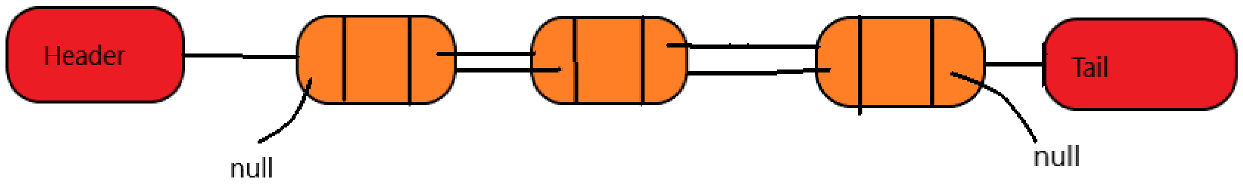
0 references

```
public Node<T> InsertNewNodeAfter(Node<T> previousNode, T dataOfNewNode)
{
    Node<T> NewNode = CreateNewNode(dataOfNewNode);
    NewNode.Next = previousNode.Next;
    NewNode.Previous = previousNode;
    previousNode.Next = NewNode;
    NewNode.Next.Previous = NewNode;

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

Time Complexity = O(1)

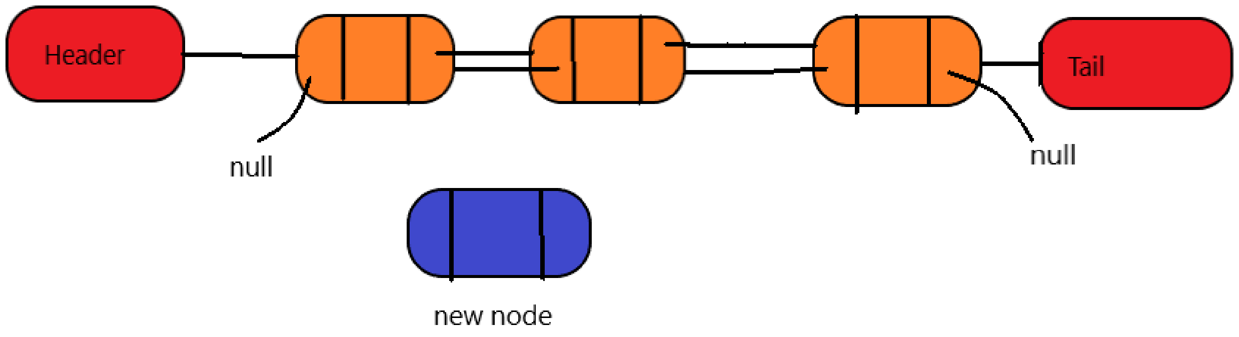
1.4 Insert Node Before

Ex : Insert new node before second node

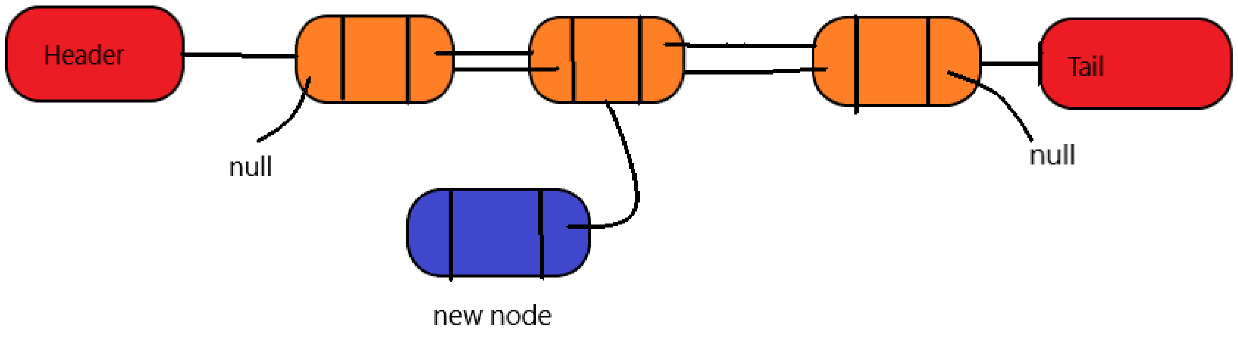


Steps :

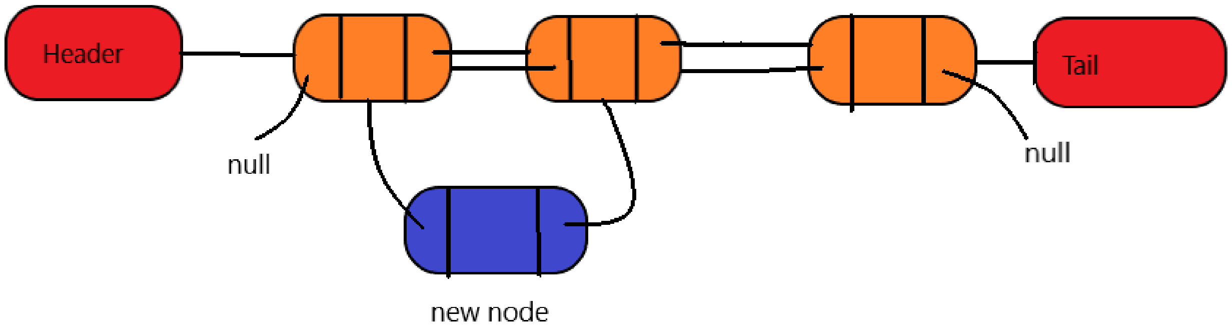
1.Create New Node



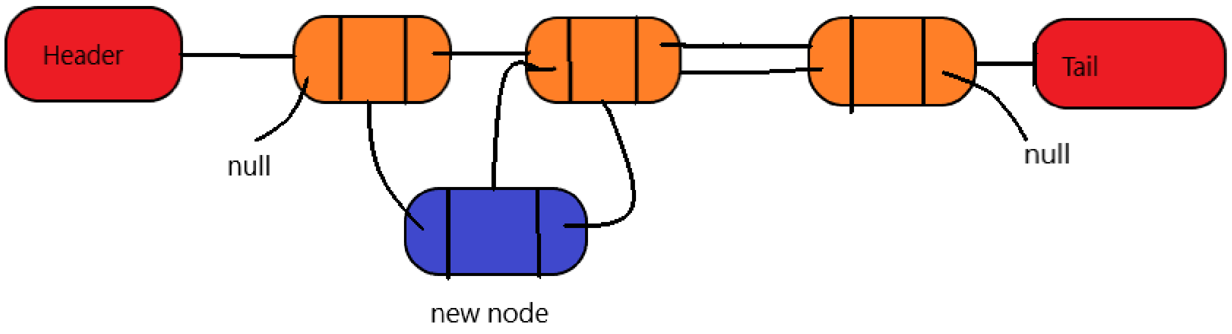
2.The Next Of New Node Point To Next Node



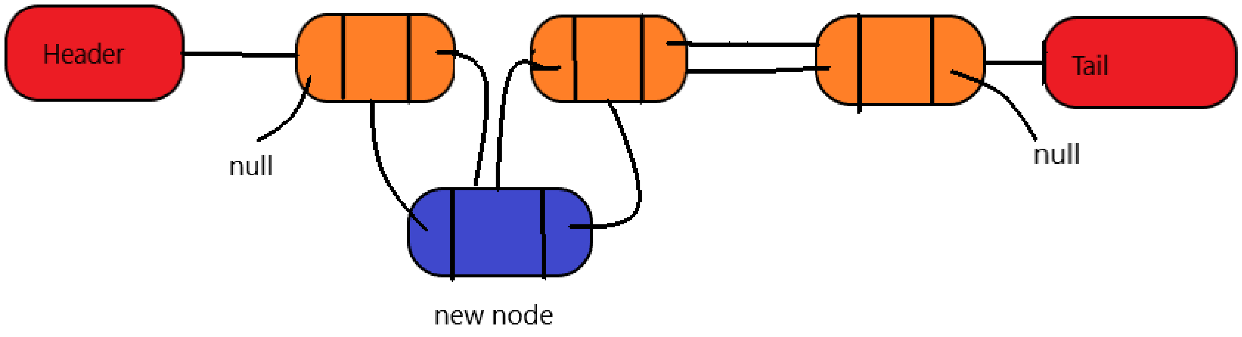
3. The Previous Of New Node Point To Of Previous Of Next Node



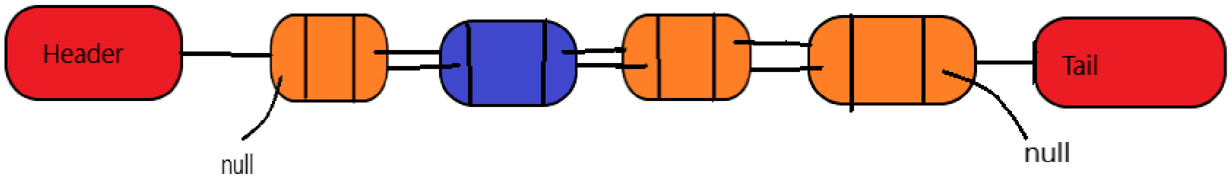
4. The Previous Of Next Node Point To New Node



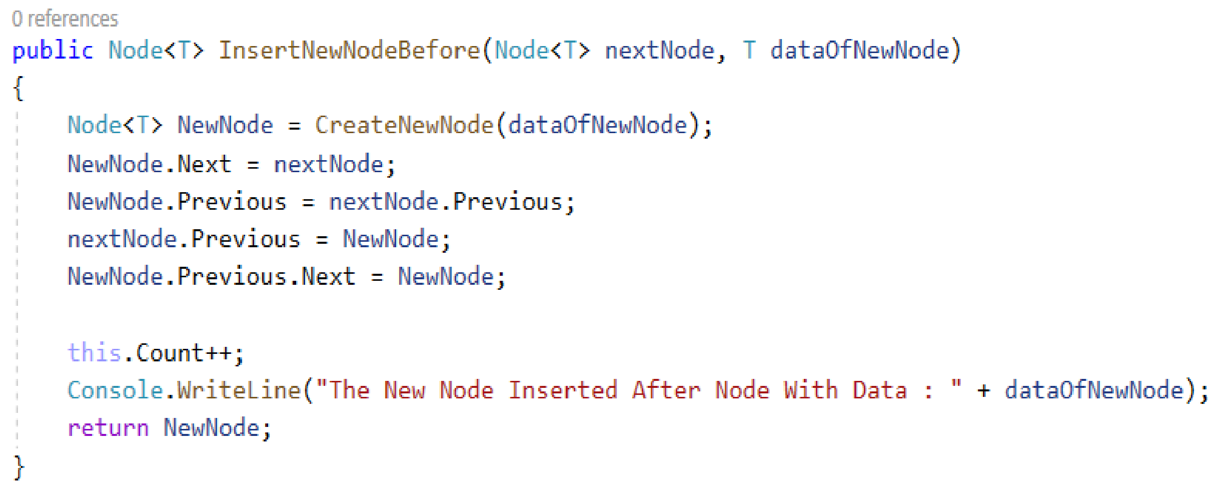
5. Next Of Previous Node Point To New Node



Finally :



Code



```
0 references
public Node<T> InsertNewNodeBefore(Node<T> nextNode, T dataOfNewNode)
{
    Node<T> NewNode = CreateNewNode(dataOfNewNode);
    NewNode.Next = nextNode;
    NewNode.Previous = nextNode.Previous;
    nextNode.Previous = NewNode;
    NewNode.Previous.Next = NewNode;

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

Time Complexity = O(1)

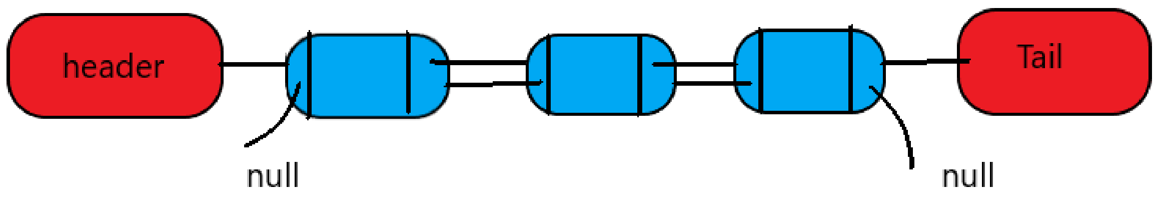
2- Delete Operations

2.1 Delete Node At First
2.2 Delete Node At Last
2.3 Delete Given Node

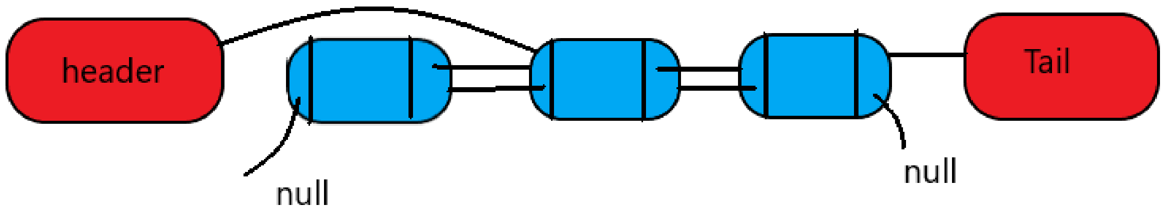
2.1 Delete Node At First

Steps :

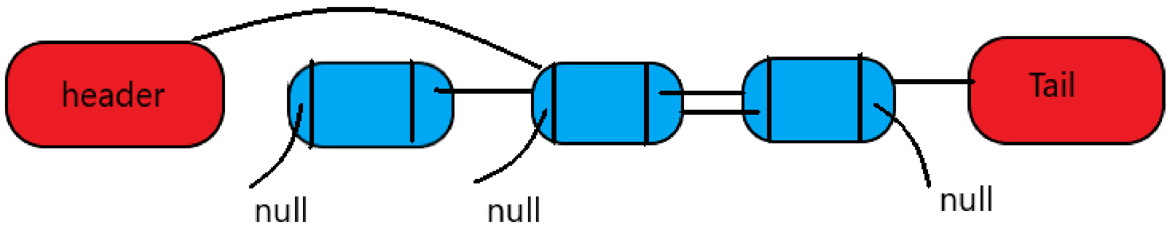
1.If Linked List Is Not Empty
1.1 If Linked List Contain MoreThan One Node



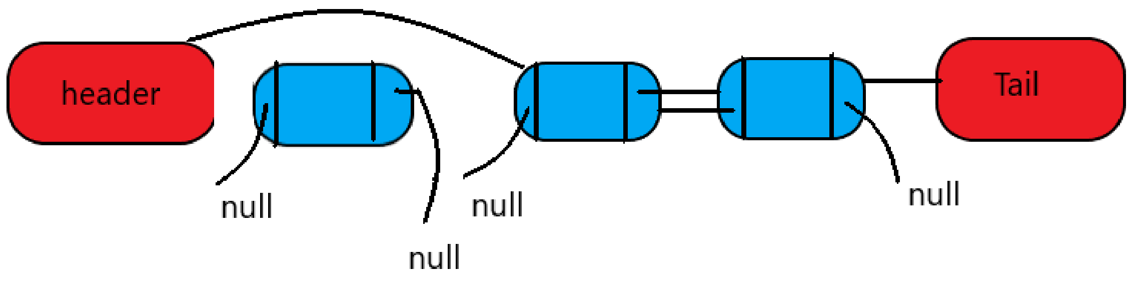
1.1.1 Header Point To The Next Of Current First Node (Second Node)



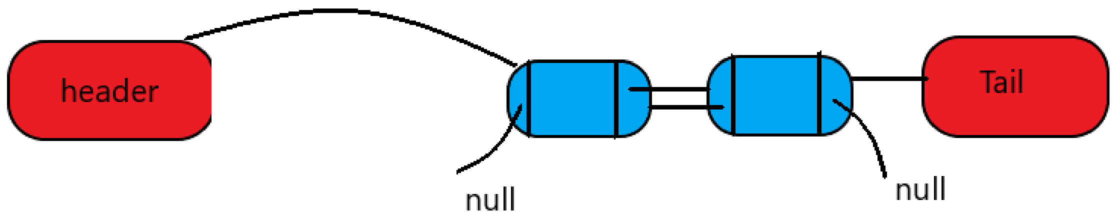
1.1.2 The Previous Of Second Node Point To Null



1.1.3 The Next Of Current First Node Point To Null

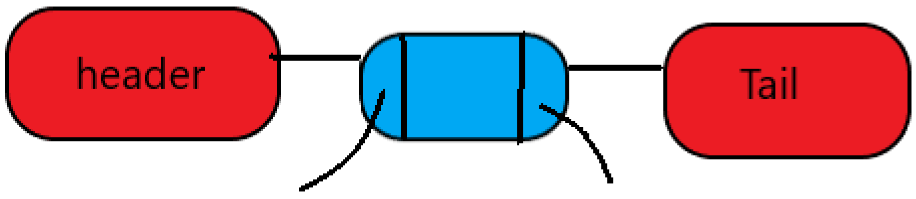


Finally

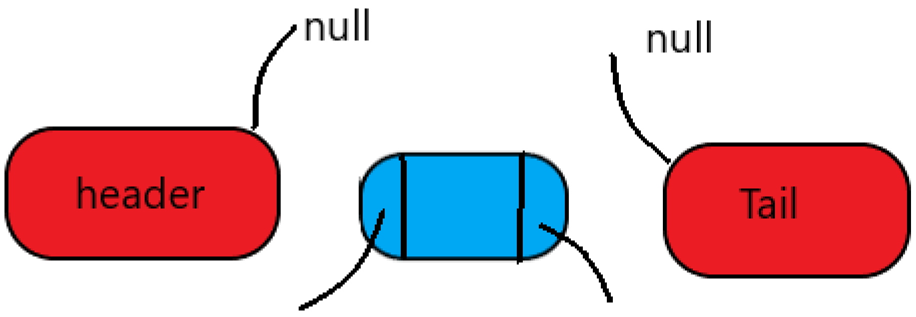


,

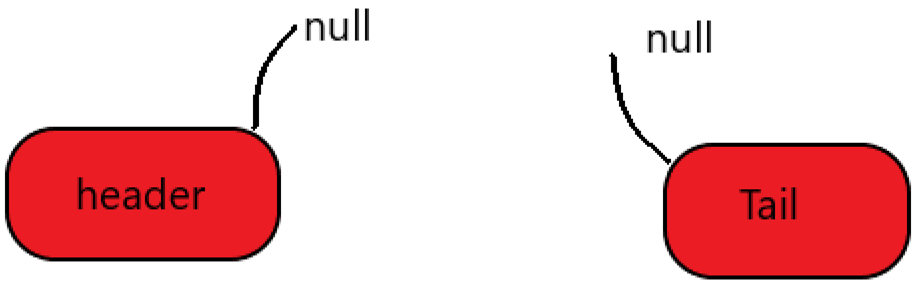
1.2 Else If (Linked List Contain Only One Node)



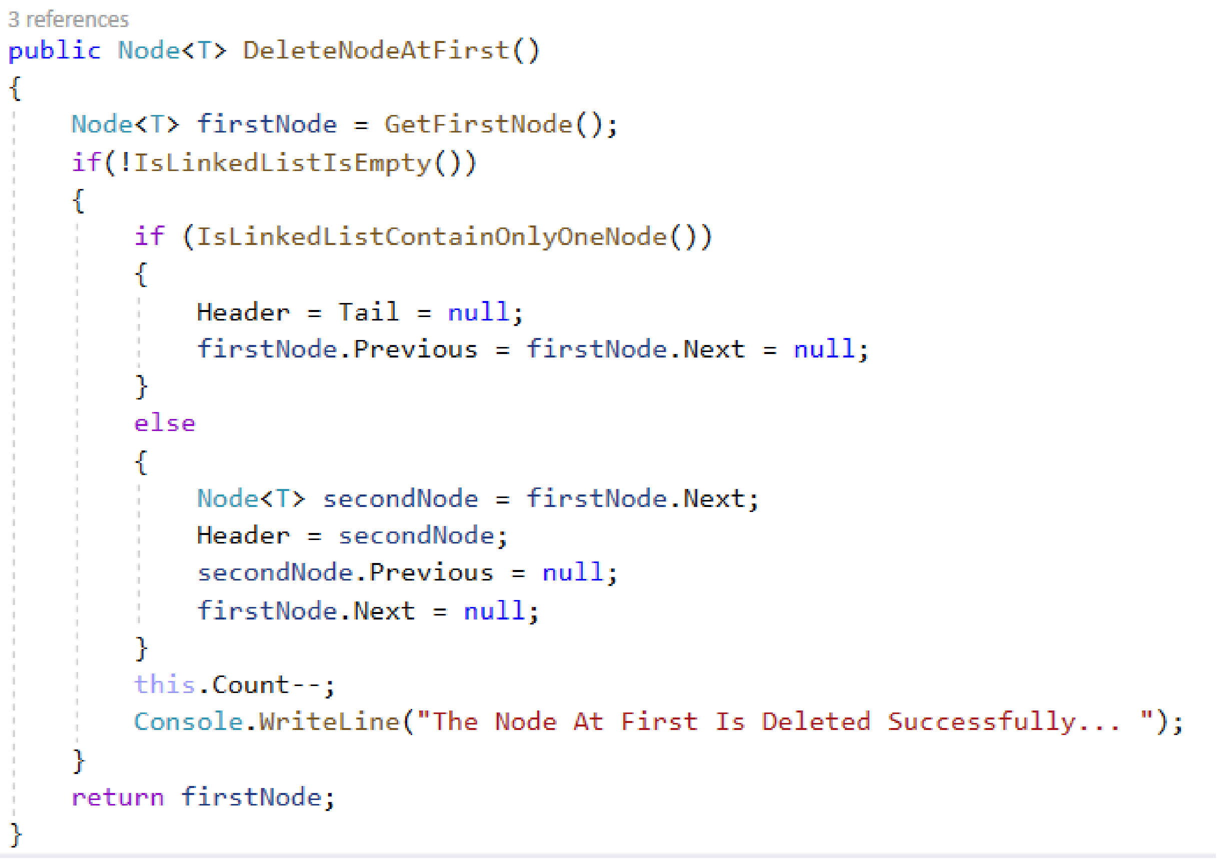
1.2.1 Header And Tail Point To Null



Finally



Code



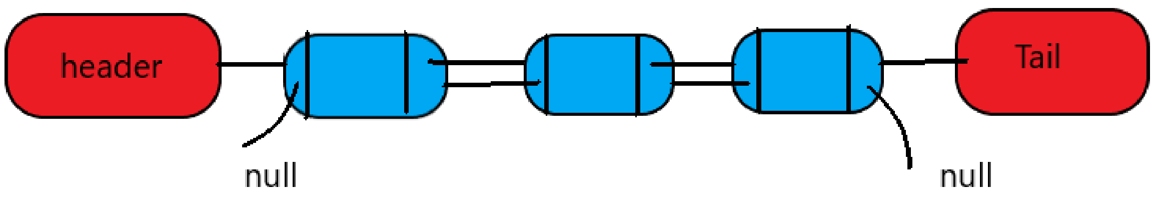
```
3 references
public Node<T> DeleteNodeAtFirst()
{
    Node<T> firstNode = GetFirstNode();
    if(!IsLinkedListIsEmpty())
    {
        if (IsLinkedListContainOnlyOneNode())
        {
            Header = Tail = null;
            firstNode.Previous = firstNode.Next = null;
        }
        else
        {
            Node<T> secondNode = firstNode.Next;
            Header = secondNode;
            secondNode.Previous = null;
            firstNode.Next = null;
        }
        this.Count--;
        Console.WriteLine("The Node At First Is Deleted Successfully... ");
    }
    return firstNode;
}
```

Time Complexity = O(1)

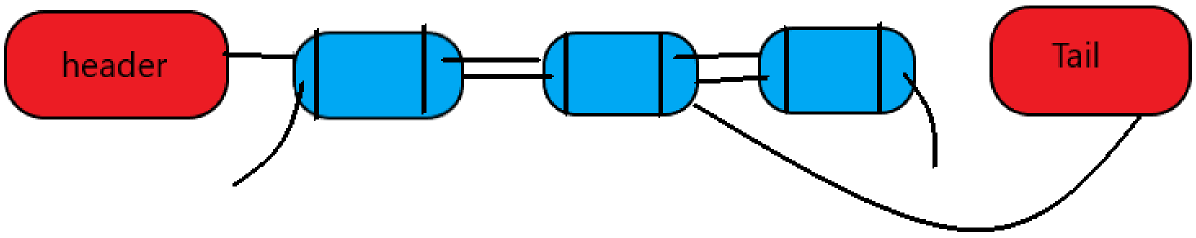
2.2 Delete Node At Last

Steps :

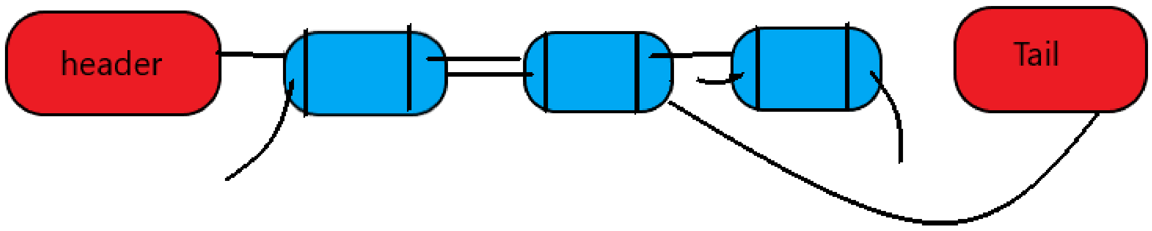
1.If Linked List Is Not Empty
1.1 If Linked List Contain MoreThan One Node



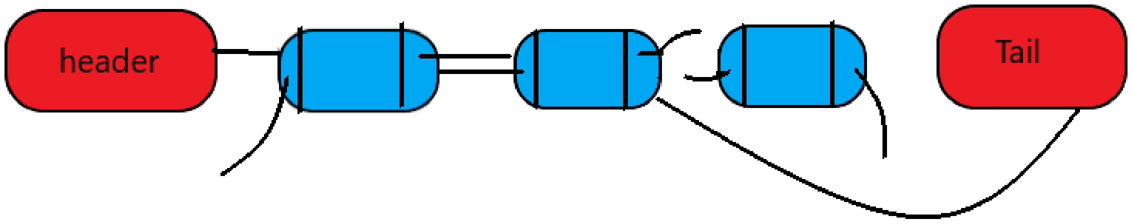
1.1.1 Tail Point To The Previous Of Current Last Node (Second Last Node)



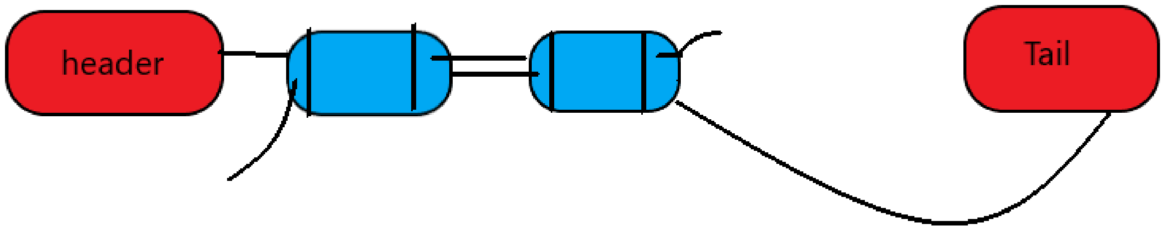
1.1.2 The Previous Of Current Last Node Point To Null



1.1.3 The Next Of Second Last Node Point To Null

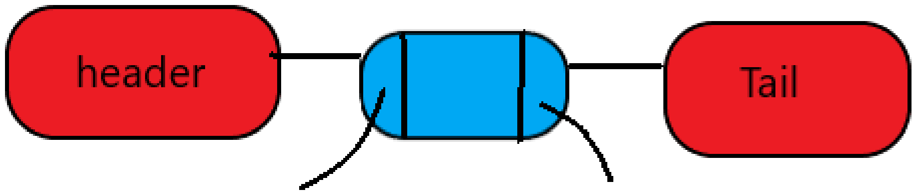


Finally

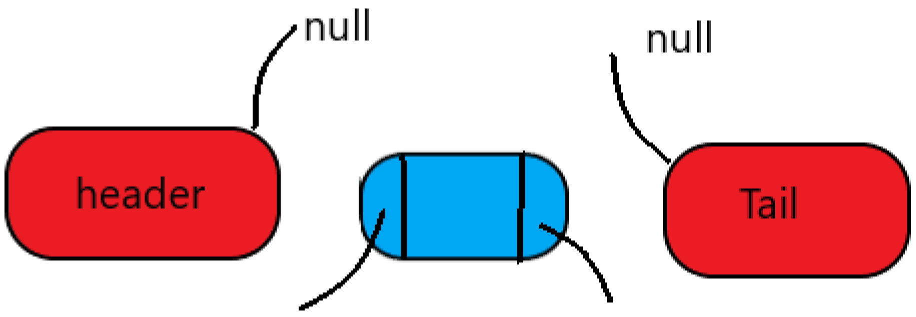


,

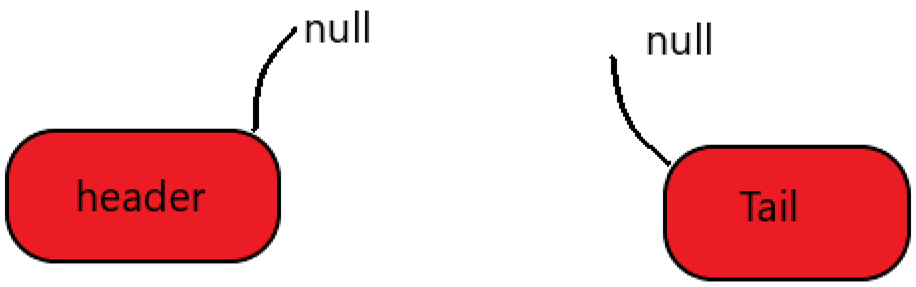
1.2 Else If (Linked List Contain Only One Node)



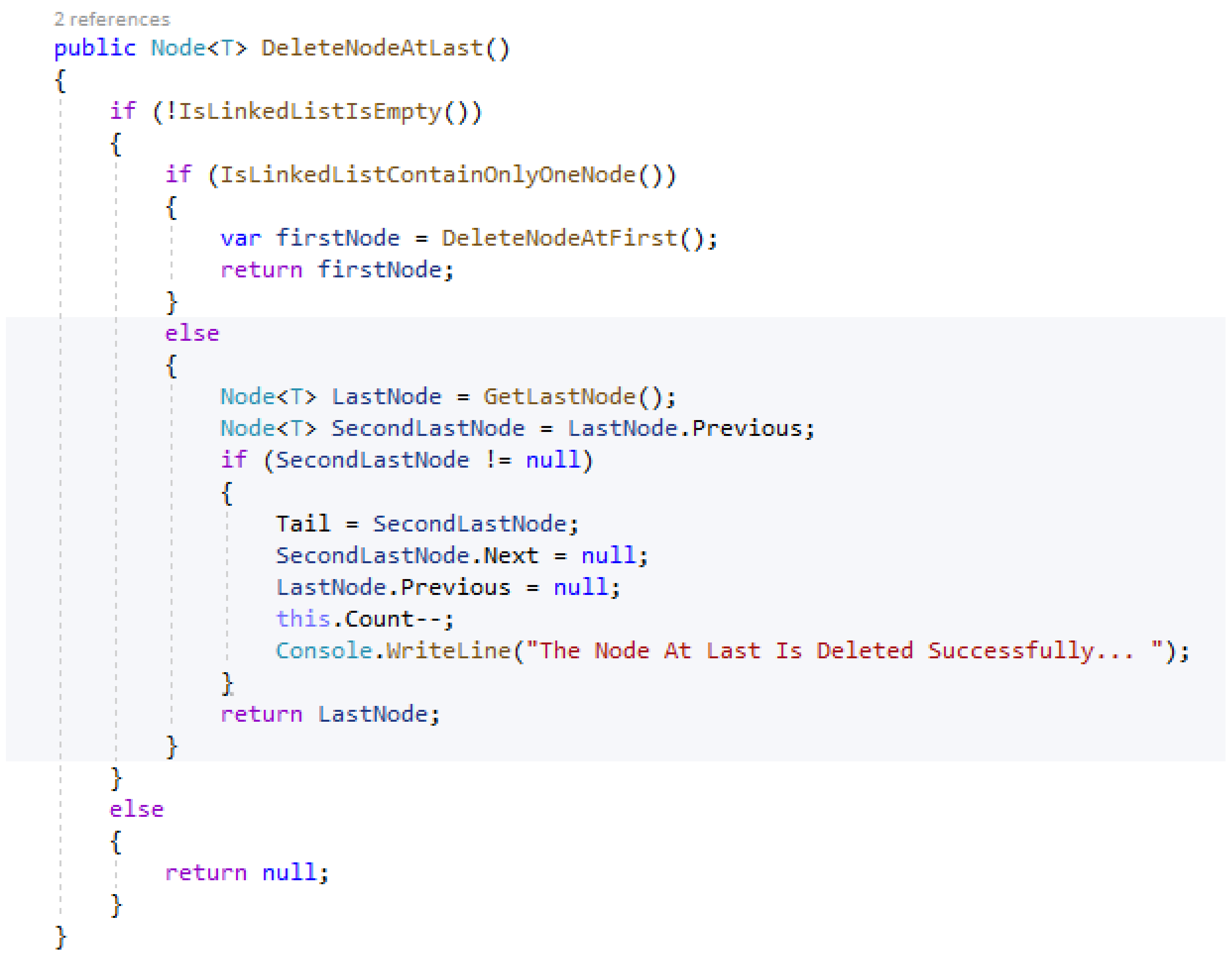
1.2.1 Delete The Node At First



Finally



Code



```
public Node<T> DeleteNodeAtLast()
{
    if (!IsLinkedListIsEmpty())
    {
        if (IsLinkedListContainOnlyOneNode())
        {
            var firstNode = DeleteNodeAtFirst();
            return firstNode;
        }
        else
        {
            Node<T> LastNode = GetLastNode();
            Node<T> SecondLastNode = LastNode.Previous;
            if (SecondLastNode != null)
            {
                Tail = SecondLastNode;
                SecondLastNode.Next = null;
                LastNode.Previous = null;
                this.Count--;
                Console.WriteLine("The Node At Last Is Deleted Successfully... ");
            }
            return LastNode;
        }
    }
    else
    {
        return null;
    }
}
```

Time Complexity = O(1)

2.3 Delete Given Node

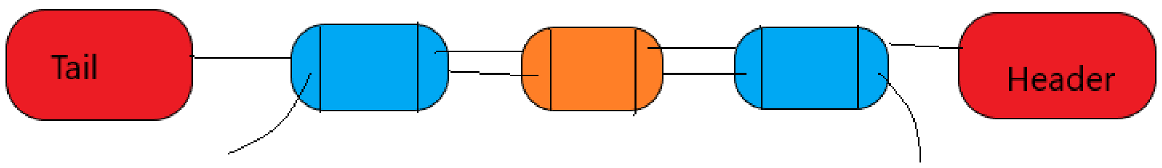
Ex : Delete Second Node

Steps :

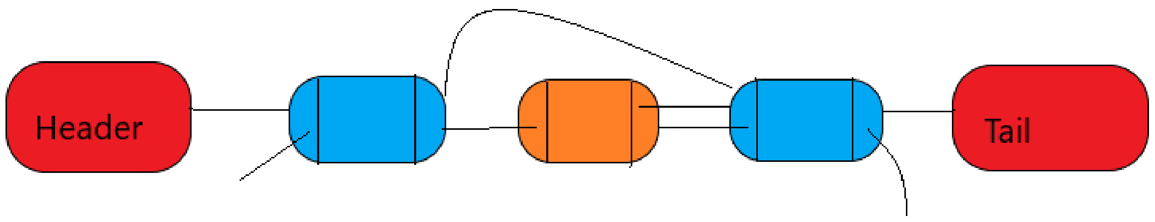
1.If Node Is First Node
1.1 Delete Node At First

2.If Node Is Last Node
2.1 Delete Node At Last

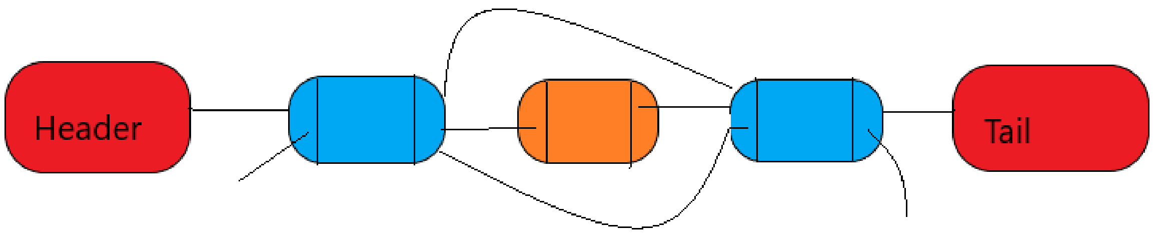
3.1 If Linked List Is Not Empty



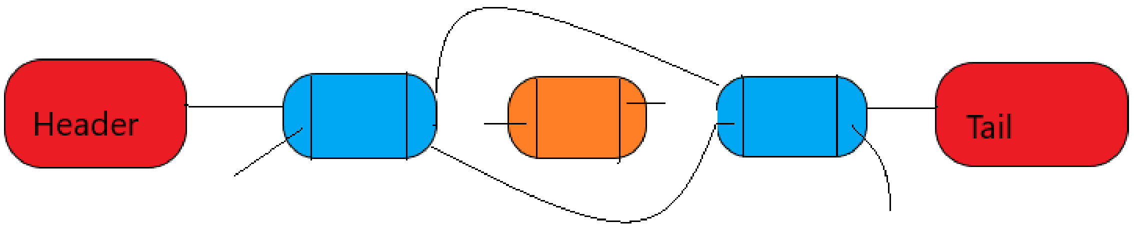
3.1 The Next Of Previous Node Point To Next Node



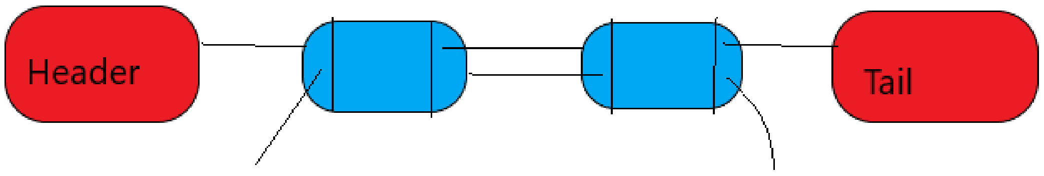
3.2 The Previous Of Next Node Point To Previous Node



3.3 The Next And Previous Of Current Node Point To Null

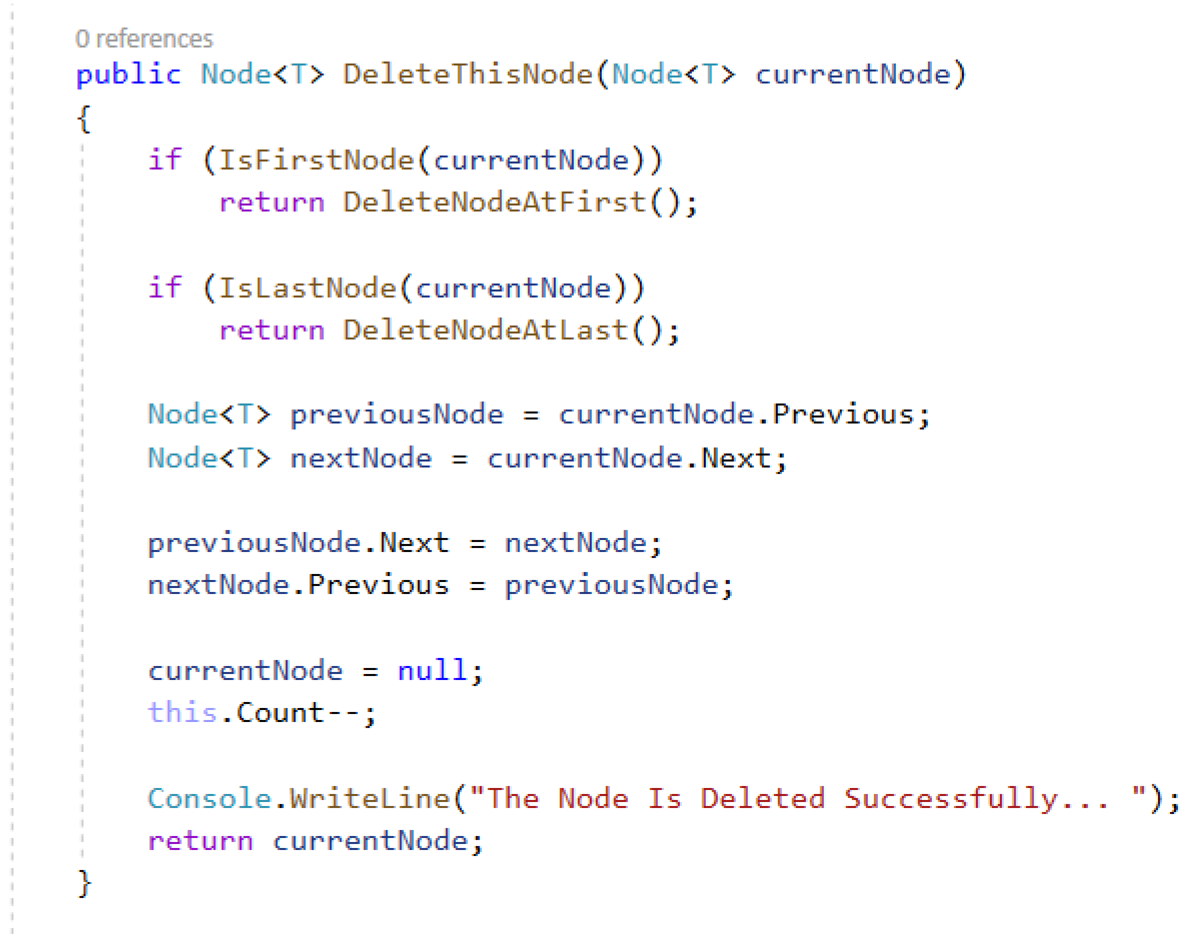


Finally



,

Code



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

    if (IsLastNode(currentNode))
        return DeleteNodeAtLast();

    Node<T> previousNode = currentNode.Previous;
    Node<T> nextNode = currentNode.Next;

    previousNode.Next = nextNode;
    nextNode.Previous = previousNode;

    currentNode = null;
    this.Count--;

    Console.WriteLine("The Node Is Deleted Successfully... ");
    return currentNode;
}
```

Time Complexity = O(1)

3- Search Operations

Same As Singly Linked List.

4- Read Operations

4.1 Get Node At First

Same As Singly Linked List.

4.2 Get Node At Last

Same As Singly Linked List.

4.3 Get Node At Position

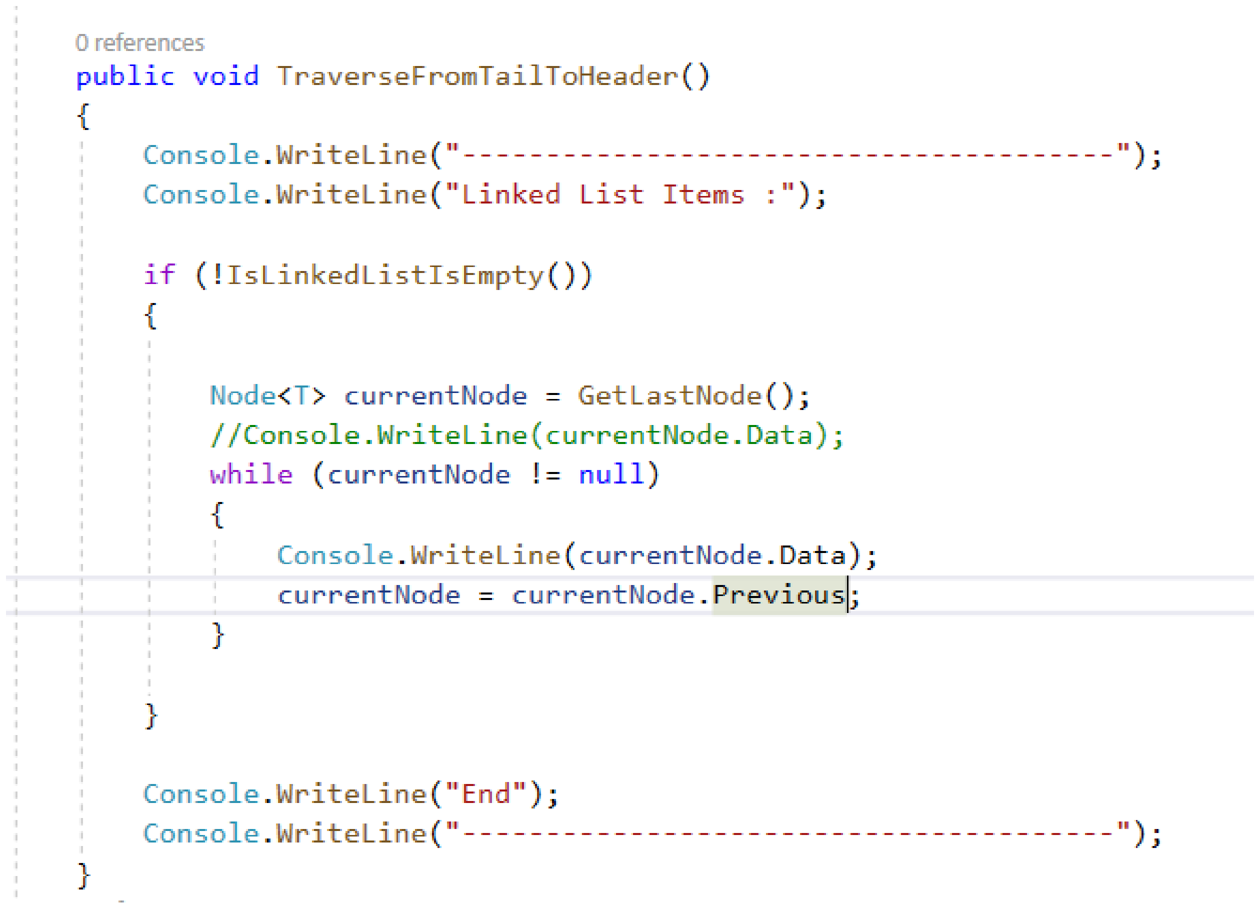
Same As Singly Linked List.

4.4 Traverse Linked List (From Header To Tail)

Same As Singly Linked List.

4.4 Traverse Linked List (From Tail To Header)

Code :



```
public void TraverseFromTailToHeader()
{
    Console.WriteLine("---------------------------------------");
    Console.WriteLine("Linked List Items :");

    if (!IsLinkedListIsEmpty())
    {

        Node<T> currentNode = GetLastNode();
        //Console.WriteLine(currentNode.Data);
        while (currentNode != null)
        {
            Console.WriteLine(currentNode.Data);
            currentNode = currentNode.Previous;
        }

    }

    Console.WriteLine("End");
    Console.WriteLine("---------------------------------------");
}
```

Time Complexity = O(n)