Implementing Linked List In C#

Ankit Sharma Updated date Sep 22, 2019

402.4k 11 13

LinkedList.zip

Introduction

In this article, I am going to discuss one of the most important Data Structures- Linked List.

I will be explaining about

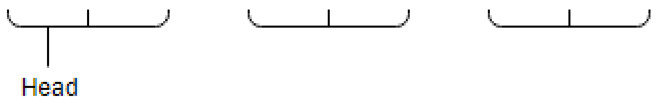
- Linked List
- Advantage of Linked List
- Types of Linked List
- How to Create a Linked List
- Various operations on Linked list.

What is a Linked List?

Linked List is a linear data structure which consists of a group of nodes in a sequence. Each node contains two parts.

- Data− Each node of a linked list can store a data.
- Address − Each node of a linked list contains an address to the next node, called "Next".

The first node of a Linked List is referenced by a pointer called Head

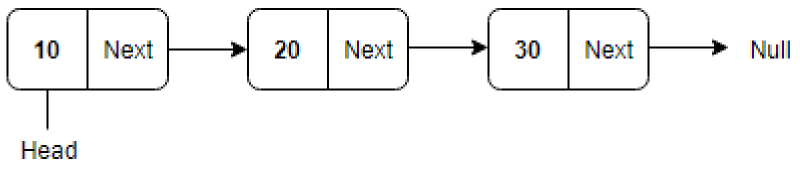


Advantages of Linked List

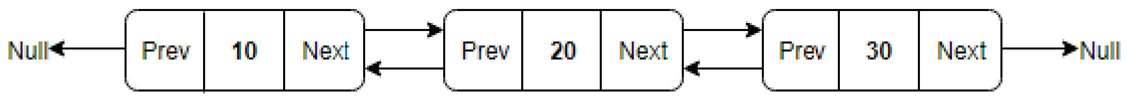
- They are dynamic in nature and allocate memory as and when required.
- Insertion and deletion is easy to implement.
- Other data structures such as Stack and Queue can also be implemented easily using Linked List.
- It has faster access time and can be expanded in constant time without memory overhead.
- Since there is no need to define an initial size for a linked list, hence memory utilization is effective.
- Backtracking is possible in doubly linked lists.

Types of Linked List

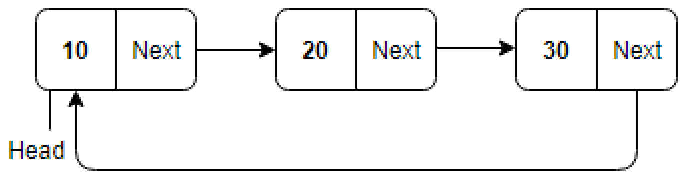
- **Singly Linked List:** Singly linked lists contain nodes which have a data part and an address part, i.e., Next, which points to the next node in the sequence of nodes. The next pointer of the last node will point to null.



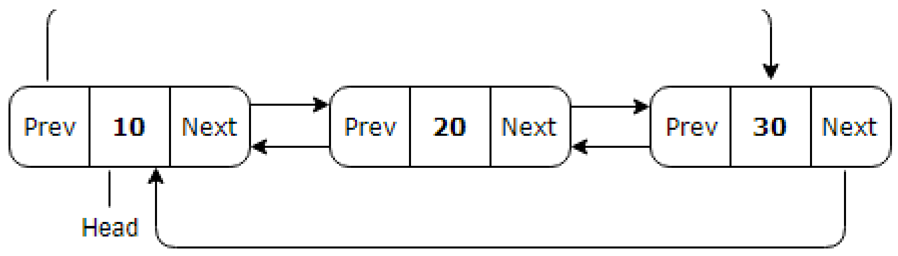
- **Doubly Linked List:** In a doubly linked list, each node contains two links - the first link points to the previous node and the next link points to the next node in the sequence.The prev pointer of the first node and next pointer of the last node will point to null.



- **Circular Linked List:** In the circular linked list, the next of the last node will point to the first node, thus forming a circular chain.



- **Doubly Circular Linked List:** In this type of linked list, the next of the last node wi point to the first node and the previous pointer of the first node will point to the la



Creating a Linked List

The node of a singly linked list contains a data part and a link part. The link will contain the address of next node and is initialized to null. So, we will create class definition of node for singly linked list as follows -

```
internal class Node {
    internal int data;
    internal Node next;
    public Node(int d) {
        data = d;
        next = null;
    }
}
```

The node for a Doubly Linked list will contain one data part and two link parts - previous link and next link. Hence, we create a class definition of a node for the doubly linked list as shown below.

```
internal class DNode {
    internal int data;
    internal DNode prev;
    internal DNode next;
    public DNode(int d) {
        data = d;
        prev = null;
        next = null;
    }
}
```

Now, our node has been created, so, we will create a linked list class now. When a new Linked List is instantiated, it just has the head, which is Null.The SinglyLinkedList class will contain nodes of type Node class. Hence, SinglyLinkedList class definition will look like below.

```
internal class SingleLinkedList {
    internal Node head;
}
```

The DoublyLinkedList class will contain nodes of type DNode class. Hence, DoublyLinkedList class will look like this.

```
}
```

Various operations on Linked list

Insert data at front of the Linked List

- The first node, head, will be null when the linked list is instantiated. When we want to add any node at the front, we want the head to point to it.
- We will create a new node. The next of the new node will point to the head of the Linked list.
- The previous Head node is now the second node of Linked List because the new node is added at the front. So, we will assign head to the new node.

```
internal void InsertFront(SingleLinkedList singlyList, int new_data) {
    Node new_node = new Node(new_data);
    new_node.next = singlyList.head;
    singlyList.head = new_node;
}
```

To insert the data at front of the doubly linked list, we have to follow one extra step .i.e point the previous pointer of head node to the new node. So, the method will look like this.

```
internal void InsertFront(DoubleLinkedList doubleLinkedList, int data)
    DNode newNode = new DNode(data);
    newNode.next = doubleLinkedList.head;
    newNode.prev = null;
    if (doubleLinkedList.head != null) {
        doubleLinkedList.head.prev = newNode;
    }
    doubleLinkedList.head = newNode;
}
```

Insert data at the end of Linked List

- If the Linked List is empty, then we simply add the new node as the Head of the Linked List.
- If the Linked List is not empty, then we find the last node and make next of the last node to the new node, hence the new node is the last node now.

```
internal void InsertLast(SingleLinkedList singlyList, int new_data)
{
    Node new_node = new Node(new_data);
    if (singlyList.head == null) {
        singlyList.head = new_node;
        return;
    }
    Node lastNode = GetLastNode(singlyList);
    lastNode.next = new_node;
}
```

```
internal void InsertLast(DoubleLinkedList doubleLinkedList, int data) {
    DNode newNode = new DNode(data);
    if (doubleLinkedList.head == null) {
        newNode.prev = null;
        doubleLinkedList.head = newNode;
        return;
    }
    DNode lastNode = GetLastNode(doubleLinkedList);
    lastNode.next = newNode;
    newNode.prev = lastNode;
}
```

The last node will be the one with its next pointing to null. Hence we will traverse the list until we find the node with next as null and return that node as last node. Therefore the method to get the last node will be

```
internal Node GetLastNode(SingleLinkedList singlyList) {
    Node temp = singlyList.head;
    while (temp.next != null) {
        temp = temp.next;
    }
    return temp;
}
```

In the above mentioned method, pass doubleLinkedList object to get last node for Doubly Linked List.

Insert data after a given node of Linked List

- We have to insert a new node after a given node.
- We will set the next of new node to the next of given node.
- Then we will set the next of given node to new node

So the method for singly Linked List will look like this,

```
internal void InsertAfter(Node prev_node, int new_data)
{
    if (prev_node == null) {
        Console.WriteLine("The given previous node Cannot be null");
        return;
    }
    Node new_node = new Node(new_data);
    new_node.next = prev_node.next;
    prev_node.next = new_node;
}
```

To perform this operation on doubly linked list we need to follow two extra steps

1. Set the previous of new node to given node.
2. Set the previous of the next node of given node to the new node.

```
internal void InsertAfter(DNode prev_node, int data)
{
    if (prev_node == null) {
        Console.WriteLine("The given prevoius node cannot be null");
        return;
    }
    DNode newNode = new DNode(data);
    newNode.next = prev_node.next;
    prev_node.next = newNode;
    newNode.prev = prev_node;
    if (newNode.next != null) {
        newNode.next.prev = newNode;
    }
}
```

Delete a node from Linked List using a given key value

- First step is to find the node having the key value.
- We will traverse through the Linked list, and use one extra pointer to keep track of the previous node while traversing the linked list.
- If the node to be deleted is the first node, then simply set the Next pointer of the Head to point to the next element from the Node to be deleted.
- If the node is in the middle somewhere, then find the node before it, and make the Node before it point to the Node next to it.
- If the node to be deleted is last node, then find the node before it, and set it to point to null.

So, the method for singly linked list will look like this,

```
internal void DeleteNodebyKey(SingleLinkedList singlyList, int key)
{
    Node temp = singlyList.head;
    Node prev = null;
    if (temp != null && temp.data == key) {
        singlyList.head = temp.next;
        return;
    }
    while (temp != null && temp.data != key) {
        prev = temp;
        temp = temp.next;
    }
    if (temp == null) {
        return;
    }
    prev.next = temp.next;
}
```

To perform this operation on doubly linked list we don't need any extra pointer for previous node as Doubly linked list already have a pointer to previous node.so the delete method will be,

```
internal void DeleteNodebyKey(DoubleLinkedList doubleLinkedList, in
    if (temp != null && temp.data == key) {
        doubleLinkedList.head = temp.next;
        doubleLinkedList.head.prev = null;
        return;
    }
    while (temp != null && temp.data != key) {
        temp = temp.next;
    }
    if (temp == null) {
        return;
    }
    if (temp.next != null) {
        temp.next.prev = temp.prev;
    }
    if (temp.prev != null) {
        temp.prev.next = temp.next;
    }
}
```

Reverse a Singly Linked list

This is one of the most famous interview questions. We need to reverse the links of each node to point to its previous node, and the last node should be the head node.This can be achieved by iterative as well as recursive methods. Here I am explaining the iterative method.

- We need two extra pointers to keep track of previous and next node, initialize them to null.
- Start traversing the list from head node to last node and reverse the pointer of one node in each iteration.
- Once the list is exhausted, set last node as head node.

The method will look like this,

```
public void ReverseLinkedList(SingleLinkedList singlyList)
{
    Node prev = null;
    Node current = singlyList.head;
    Node temp = null;
    while (current != null) {
        temp = current.next;
        current.next = prev;
        prev = current;
        current = temp;
    }
    singlyList.head = prev;
}
```

Conclusion

understanding. You will find a few more methods in the attached code such as finding middle element and searching a linked list.

Please give your valuable feedback in the comment section.

See Also

- Implementation Of Stack And Queue Using Linked List

Here are recommended articles on collections:

1. C# Dictionary
2. C# Queue
3. C# Stack
4. C# List
5. C# Arrays
6. C# HashTable
7. C# StringCollection
8. C# ArrayList
9. C# HashSet
10. C# LinkedList

C# Data Structure Linked List In C# LinkedList

Next Recommended Reading
Merging Two Sorted Linked Lists In C#

Speaker | Passionate Programmer

https://ankitsharmablogs.com/

106 **2.9m** **4** **1**

View Previous Comments

13 **11**

Type your comment here and press Enter Key (Minimum 10 characters)

Thanks for Sharing Ankit, really a well explained article
Shivaji Kakad Jul 03, 2021
1962 **71** **0** 0 0 Reply

How Come InsertAfter() method has node_prev for singly linked list ?
Ravi Kiran Jan 21, 2021
2010 **23** **0** 0 0 Reply

One of the best explained articles to understand linked list data structure.
vishal rajasekaran Jun 12, 2020
1987 **46** **0** 0 0 Reply

Well explained,Thanks for sharing.
BEAUTY SINHA Dec 27, 2019
2017 **16** **0** 0 0 Reply

Nice article
Pankaj Patel Sep 27, 2019
95 **22.2k** **865.1k** 1 0 Reply

Thank You Ankit, Good Job :)
Muhammad Ramadan Sep 21, 2019
2013 **20** **877** 0 0 Reply

Great job Ankit!
reggie nijor Aug 15, 2019
1999 **34** **0** 0 0 Reply

Why you taken LinkedList as parameter in insertBefore(). You just have taken the data only.
karthik d Nov 29, 2018
2031 **2** **0** 0 0 Reply

I have a question in insertfront for single linked list, why didn't we check if head is null and assigned the new node to it like we did in insertend
rawan alzalam Oct 18, 2018
2030 **3** **0** 0 0 Reply

Very good concept of LinkedList in c#. Also well explained with description. thanks to shar

FEATURED ARTICLES

Using Certificates For API Authentication In .NET 5

List Of Commonly Used GitHub Commands

How To Get Started With The Radzen Blazor Components

Write Your First Smart Contract On Stratis Blockchain

Getting Started With Azure Bastion

View All

TRENDING UP

01 Commonly Used Angular Commands

02 What's New In Java 16?

03 Building Serverless API's Using Azure Functions And C#

04 List Of Commonly Used GitHub Commands

05 What Is New In Visual Studio 2022

06 Create And Validate JWT Token In .NET 5.0

07 Microservices Architecture Pattern - SAGA

08 Creating And Training Custom ML Model to Read Sales Receipts Using AI-Powered Azure Form Recognizer

09 Overview Of Vue.js

10 Why Choose Flutter For Cross-Platform Development

View All