



Linked List Implementation in C#

by Ankit Sharma MVB · Dec. 28, 17 · Web Dev Zone · Tutorial

Have you seen our [HERE Twitch channel to livestream our Developer Waypoints series?](#)

Presented by HERE

Introduction

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

I Will Be Explaining:

- Linked lists.
- Advantage of a linked list.
- Types of linked lists.
- How to create a linked list.
- Various operations on a linked list.

Before proceeding further, I would recommend you download the source code from GitHub.

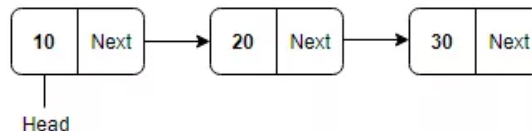
What Is a Linked List?

A 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”

- **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**

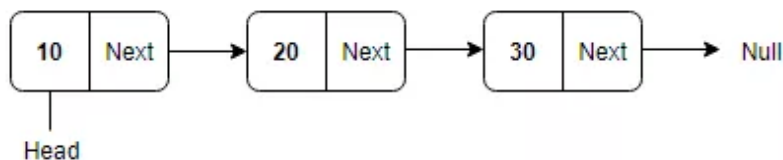


Advantages of a Linked List

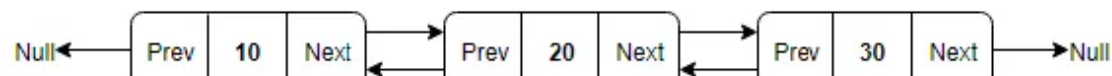
- They are dynamic in nature and allocate memory as and when required.
- Insertion and deletion are easy to implement.
- Other data structures such as Stack and Queue can also be implemented easily using a linked list.
- It has faster access time and can be expanded in constant time without memory overhead.
- 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 Lists

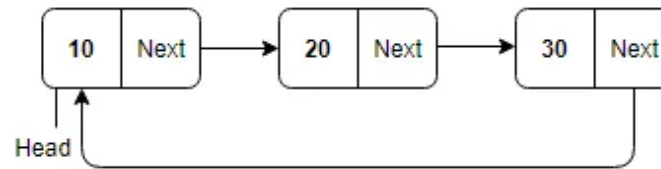
- **Singly Linked List:** Singly linked lists contain nodes which have a data part and an address part, i.e., **Next**, which points to the next node in the sequence of nodes. The next pointer of the last node will point to null.



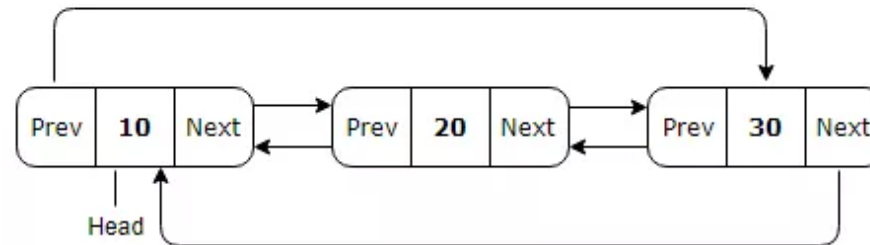
- **Doubly Linked List:** In a doubly linked list, each node contains two links – the first link points to the previous node and the next link points to the next node in the sequence. The previous pointer of the first node and next pointer of the last node will point to null.



- **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 will point to the first node and the previous pointer of the first node will point to the last node.



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 the Next node and is initialized to null. So, we will create a class definition of the node for a singly linked list as follows:

```
1 internal class Node {  
2     internal int data;  
3     internal Node next;  
4     public Node(int d) {  
5         data = d;  
6         next = null;  
7     }  
8 }
```

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

```
1  internal class DNode {  
2      internal int data;  
3      internal DNode prev;  
4      internal DNode next;  
5      public DNode(int d) {  
6          data = d;  
7          prev = null;  
8          next = null;  
9      }  
10 }
```

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, the `SinglyLinkedList` class definition will look like this:

```
1  internal class SingleLinkedList {  
2      internal Node head;  
3  }
```

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

```
1  internal class DoubleLinkedList {  
2      internal DNode head;  
3  }
```

Various Operations on Linked List

1. 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 the Linked List because the new node is added at the front. So, we will assign head to

the new node.

```
1 internal void InsertFront(SingleLinkedList singlyList, int new_data) {  
2     Node new_node = new Node(new_data);  
3     new_node.next = singlyList.head;  
4     singlyList.head = new_node;  
5 }
```

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

```
1 internal void InsertFront(DoubleLinkedList doubleLinkedList, int data) {  
2     DNode newNode = new DNode(data);  
3     newNode.next = doubleLinkedList.head;  
4     newNode.prev = null;  
5     if (doubleLinkedList.head != null) {  
6         doubleLinkedList.head.prev = newNode;  
7     }  
8     doubleLinkedList.head = newNode;  
9 }
```

2. Insert Data at the End of a 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 the Next of the last node go to the new node, hence the new node is the last node now.

```
1 internal void InsertLast(SingleLinkedList singlyList, int new_data)  
2 {  
3     Node new_node = new Node(new_data);  
4     if (singlyList.head == null) {  
5         singlyList.head = new_node;  
6         return;  
7     }  
8     Node lastNode = GetLastNode(singlyList);
```

```
9     lastNode.next = new_node;  
10 }
```

To insert the data at the end of a doubly linked list, we have to follow one extra step: point the previous pointer of the new node to the last node. The method will look like this:

```
1  internal void InsertLast(DoubleLinkedList doubleLinkedList, int data) {  
2      DNode newNode = new DNode(data);  
3      if (doubleLinkedList.head == null) {  
4          newNode.prev = null;  
5          doubleLinkedList.head = newNode;  
6          return;  
7      }  
8      DNode lastNode = GetLastNode(doubleLinkedList);  
9      lastNode.next = newNode;  
10     newNode.prev = lastNode;  
11 }
```

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 the last node. Therefore the method to get the last node will be:

```
1  internal Node GetLastNode(SingleLinkedList singlyList) {  
2      Node temp = singlyList.head;  
3      while (temp.next != null) {  
4          temp = temp.next;  
5      }  
6      return temp;  
7  }
```

In the above-mentioned method, pass the `doubleLinkedList` object to get the last node for the doubly linked list.

3. 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 the new node to the Next of the given node.
- Then we will set the Next of the given node to the new node.

So the method for a singly linked list will look like this:

```
1 internal void InsertAfter(Node prev_node, int new_data)
2 {
3     if (prev_node == null) {
4         Console.WriteLine("The given previous node Cannot be null");
5         return;
6     }
7     Node new_node = new Node(new_data);
8     new_node.next = prev_node.next;
9     prev_node.next = new_node;
10 }
```

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

- Set the Previous of the new node to the given node.
- Set the Previous of the next node of the given node to the new node.

So, the method for a doubly linked list will look like this:

```
1 internal void InsertAfter(DNode prev_node, int data)
2 {
3     if (prev_node == null) {
4         Console.WriteLine("The given prevoius node cannot be null");
5         return;
6     }
7     DNode newNode = new DNode(data);
8     newNode.next = prev_node.next;
9     prev_node.next = newNode;
10    newNode.prev = prev_node;
11    if (newNode.next != null) {
12        newNode.next.prev = newNode;
13    }
14 }
```

14 }

4. Delete a Node From a Linked List Using a Given Key Value

- The first step is to find the node that has the key value.
- We will traverse 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, 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 the last node, then find the node before it, and set it to point to null.

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

```
1  internal void DeleteNodeByKey(SingleLinkedList singlyList, int key)
2  {
3      Node temp = singlyList.head;
4      Node prev = null;
5      if (temp != null && temp.data == key) {
6          singlyList.head = temp.next;
7          return;
8      }
9      while (temp != null && temp.data != key) {
10         prev = temp;
11         temp = temp.next;
12     }
13     if (temp == null) {
14         return;
15     }
16     prev.next = temp.next;
17 }
```

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

```
1  internal void DeleteNodeByKey(DoubleLinkedList doubleLinkedList, int key)
```



```
2  {
3      DNode temp = doubleLinkedList.head;
4      if (temp != null && temp.data == key) {
5          doubleLinkedList.head = temp.next;
6          doubleLinkedList.head.prev = null;
7          return;
8      }
9      while (temp != null && temp.data != key) {
10         temp = temp.next;
11     }
12     if (temp == null) {
13         return;
14     }
15     if (temp.next != null) {
16         temp.next.prev = temp.prev;
17     }
18     if (temp.prev != null) {
19         temp.prev.next = temp.next;
20     }
21 }
```

5. 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 the Previous and Next nodes and initialize them to null.
- Start traversing the list from the head node to the last node and reverse the pointer of one node in each iteration.
- Once the list is exhausted, set the last node as the head node.

The method will look like this,

```
1  public void ReverseLinkedList(SingleLinkedList singlyList)
2  {
```

```
3     Node prev = null;
4     Node current = singlyList.head;
5     Node temp = null;
6     while (current != null) {
7         temp = current.next;
8         current.next = prev;
9         prev = current;
10        current = temp;
11    }
12    singlyList.head = prev;
13 }
```

Conclusion

We have implemented a singly linked list and doubly linked list using C#. We have also performed various operations on them. Please refer to the attached code for better understanding. You will find a few more methods in the attached code such as finding the middle element and searching a linked list.

Please give your valuable feedback in the comment section.

You can also find this article at C# Corner.

You can find my other articles on data structures here.

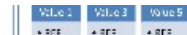
Get the fastest and most feature-rich [React Grid](#) on the market. Benefit from extensive documentation with examples, developer API and ease of customization.

Presented by ag-Grid

Like This Article? Read More From DZone



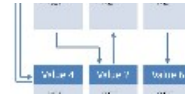
A Few Quick Links: Invisible Talent and Silicon



Introduction to Linked Lists in C#



At DZone, we're committed to diverse talent and growth. **Valley Diversity**



Introduction to Linked Lists in C#




Autoscale Your API Builder Docker Application in the Axway API Runtime Service



Free DZone Refcard Low-Code Application Development

Topics: NODE , LINKED LIST , C# , WEB DEV

Published at DZone with permission of Ankit Sharma , DZone MVB. [See the original article here.](#) 
Opinions expressed by DZone contributors are their own.