Evaluation/Generalizationin Linked List

- In a linked-list, elements contain links that reference the previous and the successor elements in the list.
- Each element is a node that consists of a value and a reference (link) to the next node in the sequence.
- A node with its two fields can reside anywhere in memory.

- The list maintains a reference variable, front/head that identifies the first element in the sequence. The list ends when the link is null.
- A singly-linked list is not a direct access structure. It must be accessed sequentially by moving forward one node at a time.
- Elements in a linked-list are called nodes. These are NODE objects that have two instant variables. The first variable is a value for a node and the second variable is a node reference called next that provides a link to the next node.

Advantages of Linked List

- Linked list are dynamic data structure. That is, they can grow or shrink during the execution of a program.
- Efficient memory utilization
- Insertion and deletion are easier and efficient.
- Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
- Many complex applications can be easily carried out with linked list.

Disadvantages of Linked List

- More memory: to store an integer number, a node with integer data and address field is allocated. That is more memory space is needed.
- Access to an arbitrary data item is little bit cumbersome and also time consuming.

Implementation of LinkedList in C#

A **linked list** is a linear data structure which stores element in the non- contiguous location. The elements in a linked list are linked with each other pointers. Or in other words, linked list consists of nodes where each node contains a data field and reference (link) to the next node in the list.

In C#, LinkedList is the generic type of collection which is defined in **System.Collections.Generic** namespace

Important Points:

- The LinkedList class implements the: Icollection<T>, Ienumerable<T>, IReadOnlyCollection<T>, Icollection, Ienumerable, IDeserializationCallback, Iserializable interfaces.
- It also supports enumerators.
- You can remove nodes and reinsert them, either in the same list or in another list, which results in no additional objects allocated on the heap.
- Every node in a LinkedList<T> object is of the type LinkedListNode<T>.
- It does not support chaining, splitting, cycles, or other features that can leave the list in an inconsistent state.
- If the LinkedList is empty, the First and Last properties contain null.
- The capacity of a LinkedList is the number of elements the LinkedList can hold.
- In LinkedList, it is allowed to store duplicate elements but of the same type.

How to create a LinkedList in C#?

A Linked List has 3 constructors which are used to create a LinkedList which are as follows:

- **LinkedList():** This constructor is used to create an instance of the LinkedList class that is empty
- **Linked List(IEnumerable):** This constructor is used to create an instance of the LinkedList class that contains elements copied from the specified IEnumerable and has sufficient capacity to accommodate the number of elements copied.
- **LinkedList(SerializationInfo,StreamingContext():**This constructor is used to create an instance of the LinkedList class that is serializable with the specified SerializationInfo and StreamingContext.

Create a LinkedList using *LinkedList()* constructor:

Step 1: Include *System.Collections.Generic* namespace in your program with the help of *using* keyword:
Step 2: Create a LinkedList using LinkedList class as shown below:

```
LinkedList <Type_of_linkedlist> linkedlist_name =
        new LinkedList
<Type_of_linkedlist>();
```

Step 3: LinkedList provides 4 different methods to add nodes

and these methods are:

Methods	Usage
AddAfter	This method is used to add a new node or value after an existing node in the LinkedList.
AddBefore	This method is used to add a new node or value before an existing node in the LinkedList.
AddFirst	This method is used to add a new node or value at the start of the LinkedList.
AddLast	This method is used to add a new node or value at the end of the LinkedList.

Step 4: The elements of the LinkedList is accessed by using a foreach loop or by using for loop

Example Program 1

```
//Creating a linkedlist Using LinkedList
// Using LinkedList Class

  LinkedList<String> my_list = new LinkedList<String>();

 //adding elements in the LinkedList
 //Using AddLast ( ) method
 my_list.AddLast("Zoya");
 my_list.AddLast("Shilpa");
 my_list.AddLast("Rohit");
 my_list.AddLast("Rohan");
 my_list.AddLast("Juhi");
 my_list.AddLast("Zoya");
 Console.WriteLine("Best Students of XYZ University:");
 //Accessing the elements of
 //LinkedList Using foreach loop
  Foreach(string str in my_list)
{
 Console.WriteLine(str);
```

```
Best Student of XYZ University
Zoya
Shilpa
Rohit
Rohan
Juhi
Zoya
```

How to remove elements from the LinkedList?

In linkedList, it is allowed to remove elements from the linkedList. LinkedList<T> class provides 5 different methods to remove elements and the methods are:

Methods	Usage
Clear():	This method is used to remove all nodes from the LinkedList.
Remove(LinkedListNode):	This method is used to remove the specified node from the LinkedList.
Remove(T):	This method is used to remove the first occurrence of the specified value from the LinkedList.
RemoveFirst():	This method is used to remove the node at the start of the LinkedList.
RemoveLast():	This method is used to remove the node at the end of the LinkedList.

Example Program 2

```
//Creating a LinkedList Using LinkedList
// Using LinkedList Class

 LinkedList<String> my_list = new LinkedList<String> ();
 //adding elements in the lInkedList
 //Using AddLast () method
 My_list.AddLast("Zoya");
 My_list.AddLast("Shilpa");
 My_list.AddLast("Rohit");
 My_list.AddLast("Rohan");
 My_list.AddLast("Juhi");
 My_list.AddLast("Zoya");

 // Initial number of elements
 Console.WriteLine("Best Students of XYZ University initially");
 //Accessing the elements of
 //LinkedList Using foreach loop
  Foreach(string str in my_list)
 {
 Console.WriteLine(str);
 }
// After using Remove (LinkedListNode)
 // method
 Console.WriteLine("Best Students of ZYZ University in 2000:");
 my_list.Remove (my_list.First);
  Foreach(string str in my_list)
 {
      Console.WriteLine(str);
 }
// After using Remove (LinkedListNode)
 // method
 Console.WriteLine("Best Students of ZYZ University in 2000:");
 my_list.Remove (my_list.First);
  Foreach(string str in my_list)
 {
      Console.WriteLine(str);
 }
```

```
// After using Remove (T) method
   Console.WriteLine("Best Students of ZYZ University in 2001:");

    my_list.Remove("Rohit");
    Foreach(string str in my_list)
    {
          Console.WriteLine(str);
    }

// After using RemoveFirst () method
     // method
     Console.WriteLine("Best Students of ZYZ University in 2002:");
     my_list.RemoveFirst ();


   Foreach(string str in my_list)
   {
        Console.WriteLine(str);
    }
```

```
Best Student of XYZ University
initially:  Zoya
Shilpa
Rohit
Roha
n
Juhi
Zoya
Best Student of XYZ University in
2000:  Shilpa
Rohit
Roha
n
Juhi
Zoya
Best Student of XYZ University in
2001  Shilpa
Roha
n
Juhi
Zoya
Best Student of XYZ University in
2002  Rohan
Juhi
Zoya
Best Student of XYZ University in
2003  Rohan
```

How to check the availability of the elements in the LinkedList?

In linkedList, you can check whether the given value is present or not using the Contains(T) method.

Contains(T): This method is used to remove all nodes from the LinkedList.

Example Program 3

```
 //creating a lInkedList
 //Using LinkedList class
LinkedList<String> my_list = new LinkedList<String> ();

// adding elements in the LinkedList
// using AddLast () method
 my_list.AddLast("Zoya");
 my_list.AddLast("Shilpa");
 my_list.AddLast("Rohit");
 my_list.AddLast("Rohan");
 my_list.AddLast("Juhi");

//check if the given element
// is available or not
 if (my_list.Contains ("Shilpa") ==true)
Console.WriteLine("Element Found...!!");
```