C# | List Class

Read Discuss Courses Practice

List<T> class represents the list of objects which can be accessed by index. It comes under the **System.Collections.Generic** namespace. List class can be used to create a collection of different types like integers, strings etc. List<T> class also provides the methods to search, sort, and manipulate lists.

Characteristics:

- It is different from the arrays. A **List<T> can be resized dynamically** but arrays cannot.
- List<T> class can accept null as a valid value for reference types and it also allows duplicate elements.
- If the Count becomes equals to Capacity, then the capacity of the List increased automatically by reallocating the internal array. The existing elements will be copied to the new array before the addition of the new element.
- List<T> class is the generic equivalent of ArrayList class by implementing the IList<T> generic interface.
- This class can use both equality and ordering comparer.
- List<T> class is not sorted by default and elements are accessed by zero-based index.
- For very large List<T> objects, you can increase the **maximum capacity to 2 billion elements** on a 64-bit system by setting the enabled attribute of the configuration element to true in the run-time environment.

Constructors

Constructor	Description
List<T>()	Initializes a new instance of the List<T> class that is empty and has the default initial capacity.
List<T> (IEnumerable<T>)	Initializes a new instance of the List<T> class that contains elements copied from the specified collection and has sufficient capacity to accommodate the number of elements copied.
List<T>(Int32)	Initializes a new instance of the List<T> class that is empty and has the specified initial capacity.

Example:

```
// C# program to create a List<T>
using System;
using System.Collections.Generic;

class Geeks {

    // Main Method
    public static void Main(String[] args)
    {

        // Creating a List of integers
        List<int> firstlist = new List<int>();

        // displaying the number
        // of elements of List<T>
        Console.WriteLine(firstlist.Count);
    }
}
```

Output:

```
0
```

Properties

Property	Description
Capacity	Gets or sets the total number of elements the internal data structure can hold without resizing.
Count	Gets the number of elements contained in the List<T>.
Item[Int32]	Gets or sets the element at the specified index.

Example:

```
// C# program to illustrate the
// Capacity Property of List<T>
using System;
using System.Collections.Generic;

class Geeks {

    // Main Method
    public static void Main(String[] args)
    {

        // Creating a List of integers
        // Here we are not setting
        // Capacity explicitly
```

```
            List<int> firstlist = new List<int>();

            // adding elements in firstlist
            firstlist.Add(1);
            firstlist.Add(2);
            firstlist.Add(3);
            firstlist.Add(4);

            // Printing the Capacity of firstlist
            Console.WriteLine("Capacity Is: " + firstlist.Capacity);

            // Printing the Count of firstlist
            Console.WriteLine("Count Is: " + firstlist.Count);

            // Adding some more
            // elements in firstlist
            firstlist.Add(5);
            firstlist.Add(6);

            // Printing the Capacity of firstlist
            // It will give output 8 as internally
            // List is resized
            Console.WriteLine("Capacity Is: " + firstlist.Capacity);

            // Printing the Count of firstlist
            Console.WriteLine("Count Is: " + firstlist.Count);
        }
    }
```

Output:

```
Capacity Is: 4
Count Is: 4
Capacity Is: 8
Count Is: 6
```

Methods

Method	Description
Add(T)	Adds an object to the end of the List<T>.
AddRange(IEnumerable<T>)	Adds the elements of the specified collection to the end of the List<T>.
AsReadOnly()	Returns a read-only ReadOnlyCollection<T> wrapper for the current collection.
BinarySearch()	Uses a binary search algorithm to locate a specific element in the sorted List<T> or a portion of it.
Clear()	Removes all elements from the List<T>.
Contains(T)	Determines whether an element is in the List<T>.
ConvertAll(Converter)	Converts the elements in the current List<T> to another type, and returns a list containing the

CopyTo()	Copies the List<T> or a portion of it to an array.
Equals(Object)	Determines whether the specified object is equal to the current object.
Exists(Predicate<T>)	Determines whether the List<T> contains elements that match the conditions defined by the specified predicate.
Find(Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the first occurrence within the entire List<T>.
FindAll(Predicate<T>)	Retrieves all the elements that match the conditions defined by the specified predicate.
FindIndex()	Searches for an element that matches the conditions defined by a specified predicate, and returns the zero-based index of the first occurrence within the List<T> or a portion of it. This method returns -1 if an item that matches the conditions is not found.
FindLast(Predicate<T>)	Searches for an element that matches the conditions defined by the specified predicate, and returns the last occurrence within the entire List<T>.

FindLastIndex()	Searches for an element that matches the conditions defined by a specified predicate, and returns the zero-based index of the last occurrence within the List<T> or a portion of it.
ForEach(Action<T>)	Performs the specified action on each element of the List<T>.
GetEnumerator()	Returns an enumerator that iterates through the List<T>.
GetHashCode()	Serves as the default hash function.
GetRange(Int32, Int32)	Creates a shallow copy of a range of elements in the source List<T>.
GetType()	Gets the Type of the current instance.
IndexOf()	Returns the zero-based index of the first occurrence of a value in the List<T> or in a portion of it.
Insert(Int32, T)	Inserts an element into the List<T> at the specified index.
InsertRange(Int32, IEnumerable<T>)	Inserts the elements of a collection into the List<T> at the specified index.
LastIndexOf()	Returns the zero-based index of the last occurrence of a value in the List<T> or in a portion of it.
MemberwiseClone()	Creates a shallow copy of the current Object.
Remove(T)	Removes the first occurrence of a specific object from the List<T>.
RemoveAll(Predicate<T>)	Removes all the elements that match the conditions defined by the specified predicate.
RemoveAt(Int32)	Removes the element at the specified index of the List<T>.
RemoveRange(Int32, Int32)	Removes a range of elements from the List<T>.
Reverse()	Reverses the order of the elements in the List<T> or a portion of it.
Sort()	Sorts the elements or a portion of the elements in the List<T> using either the specified or default

	IComparer<T> implementation or a provided Comparison<T> delegate to compare list elements.
ToArray()	Copies the elements of the List<T> to a new array.
ToString()	Returns a string that represents the current object.
TrimExcess()	Sets the capacity to the actual number of elements in the List<T>, if that number is less than a threshold value.
TrueForAll(Predicate<T>)	Determines whether every element in the List<T> matches the conditions defined by the specified predicate.

Example 1:

```
// C# Program to check whether the
// element is present in the List
// or not
using System;
using System.Collections.Generic;

class Geeks {

    // Main Method
    public static void Main(String[] args)
    {

        // Creating an List<T> of Integers
        List<int> firstlist = new List<int>();

        // Adding elements to List
        firstlist.Add(1);
        firstlist.Add(2);
        firstlist.Add(3);
        firstlist.Add(4);
        firstlist.Add(5);
        firstlist.Add(6);
        firstlist.Add(7);

        // Checking whether 4 is present
        // in List or not
        Console.Write(firstlist.Contains(4));
    }
}
```

Output:

```
True
```

Example 2:

```
// C# Program to remove the element at
// the specified index of the List<T>
using System;
using System.Collections.Generic;

class Geeks {

    // Main Method
    public static void Main(String[] args)
    {

        // Creating an List<T> of Integers
        List<int> firstlist = new List<int>();

        // Adding elements to List
        firstlist.Add(17);
        firstlist.Add(19);
        firstlist.Add(21);
        firstlist.Add(9);
        firstlist.Add(75);
        firstlist.Add(19);
        firstlist.Add(73);

        Console.WriteLine("Elements Present in List:\n");

        int p = 0;

        // Displaying the elements of List
        foreach(int k in firstlist)
        {
            Console.Write("At Position {0}: ", p);
            Console.WriteLine(k);
            p++;
        }

        Console.WriteLine(" ");

        // removing the element at index 3
        Console.WriteLine("Removing the element at index 3\n");

        // 9 will remove from the List
        // and 75 will come at index 3
        firstlist.RemoveAt(3);

        int p1 = 0;

        // Displaying the elements of List
        foreach(int n in firstlist)
        {
            Console.Write("At Position {0}: ", p1);
            Console.WriteLine(n);
            p1++;
        }
    }
}
```

Output:

```
Elements Present in List:

At Position 0: 17
At Position 1: 19
At Position 2: 21
At Position 3: 9
At Position 4: 75
At Position 5: 19
At Position 6: 73

Removing the element at index 3

At Position 0: 17
At Position 1: 19
At Position 2: 21
At Position 3: 75
At Position 4: 19
At Position 5: 73
```

Reference:

- https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netframework-4.7.2

Last Updated : 25 Nov, 2022

Similar Reads

- C# Program to Check a Class is a Sub-Class of a Specified Class or Not
- C# Program to Check a Specified class is a Serializable class or not
- C# Program to Inherit an Abstract Class and Interface in the Same Class
- C# Program to Check a Specified Class is an Abstract Class or Not
- C# Program to Check a Specified Class is a Sealed Class or not
- How to sort a list in C# | List.Sort() Method Set -1
- How to sort a list in C# | List.Sort() Method Set -2
- C# Program to Find the List of Students whose Name Starts with 'S' using where() Metho…