Programming List in C#

This free book is provided by courtesy of C# Corner and its authors. Feel free to share this book with your friends and co-workers. Please do not reproduce, republish, edit or copy this book.

Mahesh Chand

July 2012, Garnet Valley PA

Message from the Author

Thank you for being a part of C# Corner, a free online community for IT developers and professionals.

I've always been a big believer and advocate of free knowledge sharing and education to all. C# Corner is all about helping each other.

I will have to say a big thank you to you and our top members who made all this possible. There is a saying in Hindi, AKELA CHANA BHAD NAHIN FOD SAKTA. This is all possible because of you and you believing in sharing your code and knowledge.

Again, thank you to our top members and their hard work and dedication! This is why we're here today. Please feel free to share this book with your friends and co-workers.

Also, do not forget to share your knowledge and spread the word around about C# Corner and the Mindcracker Network.

Cheers!

Mahesh Chand

Microsoft MVP, Visual C#
Founder, C# Corner and Mindcracker Network

Introduction

A list is a collection of items that can be accessed by index and provides functionality to search, sort and manipulate list items.

The List<T> class defined in the System.Collections.Generic namespace is a generic class and can store any data types to create a list. Before you use the List class in your code, you must import the System.Collections.Generic namespace using the following line.

```
using System.Collections.Generic;
```

Creating a List

The List class constructor takes a key data type. The data type can be any .NET data type.

The following code snippet creates a List of string types.

```
List<string> AuthorList = new List<string>();
```

The following code snippet adds items to the list.

```
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");
```

Alternatively, we can also pass an array of objects to create a List object. The following code snippet creates a List object from an array of strings.

```
// Create a List using Range
string[] authors = { "Mike Gold", "Don Box",
                     "Sundar Lal", "Neel Beniwal" };
List<string> authorsRange = new List<string>(authors);
```

The following code snippet creates a list of integer type.

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

The following code snippet adds items to the dictionary.

```
AgeList.Add(35);
AgeList.Add(25);
AgeList.Add(29);
AgeList.Add(21);
AgeList.Add(84);
```

We can also limit the size of a list. The following code snippet creates a list where the key type is float and the total number of items it can hold is 3.

```
List<float> PriceList = new List<float>(3);
```

The following code snippet adds items to the list.

```
PriceList.Add(3.25f);
PriceList.Add(2.76f);
PriceList.Add(1.15f);
```

Creating a List of Objects

The List class can be used to create any type including a class. In this article, we will see how to create a list of a class with several properties.

We have a class named Author that has five pubic properties Name, Age, BookTitle, IsMVP and PublishedDate of type string, short, string, bool, and DateTime respectively.

```
public class Author
{
    private string name;
    private short age;
    private string title;
    private bool mvp;
    private DateTime pubdate;

    public Author(string name, short age, string title, bool mvp, DateTime pubdate)
    {
        this.name = name;
        this.age = age;
        this.title = title;
        this.mvp = mvp;
        this.pubdate = pubdate;
    }

    public string Name
    {
        get { return name;  }
        set { name = value; }
    }

    public short Age
    {
        get { return age; }
        set { age = value; }
    }
    public string BookTitle
    {
        get { return title; }
        set { title = value; }
    }
    public bool IsMVP
    {
        get { return mvp; }
        set { mvp = value; }
```

```
    }
    public DateTime PublishedDate
    {
        get { return pubdate; }
        set { pubdate = value; }
    }
}
```

The following code snippet creates a List of Author type.

```
List<Author> AuthorList = new List<Author>();
```

The following code snippet creates the Author objects and adds them to the List.

```
AuthorList.Add(new Author("Mahesh Chand", 35, "A Prorammer's Guide to ADO.NET", true, new
DateTime(2003,7,10)));
AuthorList.Add(new Author("Neel Beniwal", 18, "Graphics Development with C#", false, new
DateTime(2010, 2, 22)));
AuthorList.Add(new Author("Praveen Kumar", 28, "Mastering WCF", true, new DateTime(2012,
01, 01)));
AuthorList.Add(new Author("Mahesh Chand", 35, "Graphics Programming with GDI+", true, new
DateTime(2008, 01, 20)));
AuthorList.Add(new Author("Raj Kumar", 30, "Building Creative Systems", false, new
DateTime(2011, 6, 3)));
```

The following code snippets loop through the List and prints out all the items of the List.

```
foreach (var author in AuthorList)
{
    Console.WriteLine("Author: {0},{1},{2},{3},{4}", author.Name, author.Age,
author.BookTitle, author.IsMVP, author.PublishedDate);
}
```

Reading List Items

The List<T> is a collection. We can use the foreach loop to go through all the items and read them. The following code snippet reads all items of a list and prints them on the console.

```
foreach (string author in AuthorList)
{
    Console.WriteLine(author);
}
```

We can also use a var keyword for any kind of data type.

```
foreach (var author in AuthorList)
{
    Console.WriteLine(author);
}
```

The following code snippet creates a new list and reads all of its items and displays on the console.

```
public void CreateList()
{
    // Create a list of strings
    List<string> AuthorList = new List<string>();
    AuthorList.Add("Mahesh Chand");
    AuthorList.Add("Praveen Kumar");
    AuthorList.Add("Raj Kumar");
    AuthorList.Add("Nipun Tomar");
    AuthorList.Add("Dinesh Beniwal");

    // Read all data
    Console.WriteLine("Authors List");
    foreach (var author in AuthorList)
    {
        Console.WriteLine(author);
    }
}
```

List Properties

The List class has three properties – Capacity, Count, and Item.

Capacity

The Capacity property gets and sets the number of items a list can hold without resizing. Capacity is always greater than or equal to the Count value.

The following code snippet creates a List of authors and displays the original capacity. After that, code removes the excess capacity by calling TrimExcess method. After that, capacity is set to 20.

```
// Create a list of strings
List<string> AuthorList = new List<string>();
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");

// Original Capacity
Console.WriteLine("Original Capacity: {0}", AuthorList.Capacity);
// Trim excess
AuthorList.TrimExcess();
Console.WriteLine("Trimmed Capacity: {0}", AuthorList.Capacity);
// Update Capacity
AuthorList.Capacity = 20;
Console.WriteLine(AuthorList.Capacity);
Console.WriteLine("Updated Capacity: {0}", AuthorList.Capacity);
```

Count

The Count property gets the total actual number of items in list.

The following code snippet display number of items in a list.

```
// Create a list of strings
List<string> AuthorList = new List<string>();
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");

// Count
Console.WriteLine("Count: {0}", AuthorList.Count);
```

Item

The Item property gets and sets the value associated with the specified index.

The following code snippet gets and sets the first item in a list.

```
// Set Item value
AuthorList["Mahesh Chand"] = 20;
// Get Item value
Int16 age = Convert.ToInt16(AuthorList["Mahesh Chand"]);

// Create a list of strings
List<string> AuthorList = new List<string>();
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");

// Get first item of a List
string auth = AuthorList[0];
Console.WriteLine(auth);
// Set first item of a List
AuthorList[0] = "New Author";
```

List Methods

The List class is a generic collection and provides all common methods to add, remove, find and replace items in the collection.

Add Items

The Add method adds an item to a List. The following code snippet creates a List and adds items to it by using the Add method.

```
// Create a list
List<string> AuthorList = new List<string>();

// Add items using Add method
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");
```

The AddRange method is used to add a collection of items. The following code snippet adds a collection of items to a List.

```
// Add a range of items
string[] authors = { "Mike Gold", "Don Box",
                     "Sundar Lal", "Neel Beniwal" };
AuthorList.AddRange(authors);
```

Insert Items

The Insert method inserts an item to a List at the specified position. The following code snippet creates a List and inserts an item to the List by using the Insert method.

```
// Create a list
List<string> AuthorList = new List<string>();

// Add items using Add method
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");

// Insert an item at position 2
AuthorList.Insert(1, "Second Author");
```

The AddRange method is used to add a collection of items. The following code snippet adds a collection of items to a List.

```
// Insert a range of items
string[] authors = { "Mike Gold", "Don Box",
                     "Sundar Lal", "Neel Beniwal" };
AuthorList.InsertRange(4, authors);
```

Remove Items

The Remove method removes the first occurrence of a specific object from a List. The Remove method takes an item as its parameter. The following code snippet removes an item from a List.

```
// Create a list of strings
```

```
List<string> AuthorList = new List<string>();
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");

AuthorList.Remove("Mahesh Chand");
```

The RemoveAt method removes an item at the specified zero based index. The following code snippet removes an item at 2nd position in the List.

```
AuthorList.RemoveAt(2);
```

The RemoveRange method removes a number of items based on the specified starting index and number of items. The RemoveRange method takes first parameter as the starting position and second parameter as the number of the items to be removed from the List. The following code snippet removes 2 items starting at the 3rd position.

```
AuthorList.RemoveRange(3, 2);
```

The Clear method removes all items from the List. The following code snippet removes all items by calling the Clear method.

The Clear method removes all items from the collection. The following code snippet removes all items by calling the Clear method.

```
AuthorList.Clear();
```

Searching

The BinarySearch method users a binary search algorithm to find an item in the sorted List.

The following code snippet finds an item in a List.

```
// Create a list of strings
List<string> AuthorList = new List<string>();
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");

int itemPosition = AuthorList.BinarySearch("Raj Kumar");
Console.WriteLine("Item found at position: {0}", itemPosition + 1);
```

Sorting

The Sort method sorts all the items of a List. The following code snippet creates and sorts a List.

```
List<string> AuthorList = new List<string>();
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");
// Read all data
Console.WriteLine("Original Authors List");
foreach (var author in AuthorList)
{
    Console.WriteLine(author);
}

AuthorList.Sort();
Console.WriteLine("Sorted Authors List");
foreach (var author in AuthorList)
{
    Console.WriteLine(author);
}
```

Reverse

The Reverse method reverses the items of a List. The following code snippet creates and reverses a List.

```
List<string> AuthorList = new List<string>();
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");
// Read all data
Console.WriteLine("Original Authors List");
foreach (var author in AuthorList)
{
    Console.WriteLine(author);
}

AuthorList.Reverse();
Console.WriteLine("Reversed Authors List");
foreach (var author in AuthorList)
{
    Console.WriteLine(author);
}
```

Copy a List

The CopyTo method copies a List into a one-dimensional array. The CopyTo method has three overloaded forms.

The following code snippet copies the entire list into an Array.

```
List<string> AuthorList = new List<string>();
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Dinesh Beniwal");

// Create an array of strings
string[] authorArray = new string[15];
// Copy entire List
AuthorList.CopyTo(authorArray);
// Copy items starting at index = 4
AuthorList.CopyTo(authorArray, 4);
// Copy 4 items starting at index 2 in List and copying
// to array starting at index 10
AuthorList.CopyTo(2, authorArray, 3, 4);
foreach (string author in authorArray)
{
    Console.WriteLine(author);
}
```

Find an Item

The Contains method checks if the specified item is already exists in the List. The following code snippet checks if an item is already exits.

```
if (AuthorList.Contains("Mahesh Chand"))
    AuthorList.Remove("Mahesh Chand");
```

The IndexOf method returns the first index of an item if found in the List.

```
int idx = AuthorList.IndexOf("Nipun Tomar");
```

The LastIndexOf method returns the last index of an item if found in the List.

```
idx = AuthorList.LastIndexOf("Mahesh Chand");
```

The following code snippet shows how to use the Contains, the IndexOf and the LastIndexOf methods.

```
List<string> AuthorList = new List<string>();
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Praveen Kumar");
AuthorList.Add("Raj Kumar");
AuthorList.Add("Nipun Tomar");
AuthorList.Add("Mahesh Chand");
AuthorList.Add("Dinesh Beniwal");
```

```
// Contains - Check if an item is in the list
if (AuthorList.Contains("Mahesh Chand"))
{
    Console.WriteLine("Author found!");
}

// Find an item and replace it with new item
int idx = AuthorList.IndexOf("Nipun Tomar");
if (idx >= 0)
{
    AuthorList[idx] = "New Author";
}
Console.WriteLine("\nIndexOf ");
foreach (var author in AuthorList)
{
    Console.WriteLine(author);
}


// Find Last index of
idx = AuthorList.LastIndexOf("Mahesh Chand");
if (idx >= 0)
{
    AuthorList[idx] = "New Mahesh";
}
Console.WriteLine("\nLastIndexOf ");
foreach (var author in AuthorList)
{
    Console.WriteLine(author);
}
```