

Contents

The List collections	1
Introduction	1
Syntax	1
Creation	1
Adding Elements	1
Accessing Elements	2
Removing Elements	2
A Custom Implementation of Lists	3

The List collections

Introduction

The List class serves a similar purpose than arrays, but with a few notable differences:

- Lists do not need to have a number of elements fixed ahead of time,
- Lists automatically expand when elements are added,
- Lists automatically shrink when elements are removed,
- Lists require to have the **using** `System.Collections.Generic;` statement at the beginning of the file,
- Lists have many built-in methods.

Syntax

Creation

The syntax to create an empty list of `string` named `nameList` and a list of `int` named `valueList` containing 1, 2 and 3 is:

```
List<string> nameList = new List<string>();  
List<int> valueList = new List<int>() { 1, 2, 3 };
```

Adding Elements

Adding an element to the list is done using the `Add` method, and counting the number of elements is done using the `Count` property:

```
Console.WriteLine("nameList has " + nameList.Count + "  
↪ element.");  
nameList.Add("Bob");  
Console.WriteLine("nameList has " + nameList.Count + "  
↪ element.");
```

```
nameList.Add("Sandrine");
Console.WriteLine("nameList has " + nameList.Count + "
↪ elements.");
```

Note that we did not need to resize the `nameList` manually: its size went from 0 to 1 after we added "Bob", and from 1 to 2 after we added "Sandrine".

Accessing Elements

Using the [] operator Accessing an element can be done using the same operator as with arrays (the `[]` operator):

```
Console.Write(nameList[0]);
```

will display "Bob". Note that this syntax can be used to change the value of an element that already exist. For example,

```
nameList[0] = "Robert";
```

would replace the first value in the list ("Bob") with "Robert".

Note that while accessing or replacing an element using the `[]` operator inside a list is fine, *you cannot add new elements to the list using this syntax*. For example,

```
nameList[2] = "Sandrine";
```

would raise an exception since there is no third element to our list.

Using foreach Another way of accessing the elements in a list is to use **foreach** loops:

```
foreach (string name in nameList)
{
    Console.WriteLine(name);
}
```

Removing Elements

An element can be removed from the list using the `RemoveAt` method. If `nameList` contains "Robert, Sandrine", then after the following statement,

```
nameList.RemoveAt(0);
```

it would only contain "Sandrine" and its size would be 1. That is, the first element would be deleted and the list would shrink.

Another way of removing an element is to use the `Remove` method. Suppose we have the following list:

```
List<int> valueList = new List<int>() {-1, 0, 1, 2, 3, 2,  
↪ 5 };
```

then using

```
valueList.Remove(1);
```

would remove "1" from the list, and the list would become -1, 0, 2, 3, 2, 5.

Observe that Remove returns a **bool**, so that for instance the following

```
if(valueList.Remove(0)){  
    Console.WriteLine("0 was removed.");  
}
```

would not only remove 0 from the list, but also display "0 was removed".

Finally, if the value is present multiple times in the list, then only its first occurrence is removed. For example, if the list is -1, 2, 3, 2, 5, then after executing

```
valueList.Remove(2);
```

it would become -1, 3, 2, 5.

A Custom Implementation of Lists

A "custom" implementation of list can be found in this project¹.

¹<https://princomp.github.io/code/projects/CList.zip>