UNIT 6. LISTS.

# Lists



## 1. Introduction

In this unit we are going to introduce a new structured data type that will allow us to work with **data collections** in a similar way as we do with arrays but with greater funcionality. Among other things, it will allow us to work with dynamic data lists, that is, their size may vary throughout the program execution, unlike arrays whose size is defined at the beginning.

## 2. The List Class

Collection type objects created with this class are of the same type, but unlike arrays, the number of elements can be modified **dynamically**.

We can add elements to the list or delete them.

Before we can use **the List Class**, we must have its library by putting at the top:

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

A dynamic collection is created in a similar way to an array. To create a dynamic collection the **new** keyword is used, followed by the **List** keyword.

```
List<type> numbers = new List<type>();
```

Note that it's necessary to indicate the type of data that the list will store.

### Add values to a List

We can add values to a list by using the following methods:

- **Add(value)**. Adds the value element to the end of the list.
- **Insert(Position, Value)**. Inserts the value at the position of the array, shifting the rest of values one position forward.

```
int num, pos;
```

```
            // We add to the end of the list
            num = int.Parse(InputBox("Enter the number"));
            numbers.Add(num);


            // We insert in the position indicated by pos
            num = int.Parse(InputBox("Enter the number"));
            pos = int.Parse(InputBox("Enter the position you want
insert."));
            numbers.Insert(pos, num);
```

### Size, accessing and looping through Lists

As we have said, a list is a dynamic array, that is, its size varies throughout the execution of the program.

**Count** property returns the number of items contained in the List.

We can access the elements of a list in Visual C# with indexes, just as we do with arrays.

```
            int i;
            string text;

            text = "The items in the list are: ";
            for (i = 0; i < numbers.Count; i++)
                text = text + numbers[i] + ", ";

            MessageBox.Show(text);
```

For example, if we want to add up all the numbers contained in a list, it would be done in the fowolling way (it's not necessary to do any casting since the compiler knows what type of data is stored in the list).

```
            int sum = 0;

            for (int i = 0; i < numbers.Count; i++)
                sum = sum + numbers[i];

            MessageBox.Show("The sum is " + sum);
```

### Looping through a List with foreach

Another possible way to loop through a list is by using the foreach sentence. The foreach sentence works in a similar way to the for sentence but without the need to use indexes.

What we do is to loop though a list using a variable **that takes the value of each of the elements** in the list.

The foreach syntax is:

```
foreach(type variable in numbers)
        actions with variable
```

For example, if we want to loop through a list and print its data:

```
string text;

text = "The items in the list are: ";
foreach(int num in numbers)
    text = text + num + ", ";

MessageBox.Show(text);
```

If we want to do the sum of all the elements in the list we do it like this (it is not necessary to cast in this case):

```
int sum = 0;

foreach(int num in numbers)
    sum = sum + num;

MessageBox.Show("The sum is " + sum);
```

**Removing elements from a List.**

As we have said, lists are dynamic data structures to which we can add elements during the execution, and logically we can also remove elements using any of the fowolling functions:

- **Remove(value)**. Removes the first occurrence from the list that **corresponds to the value.**
- **RemoveAt(position)**. Removes the element located at the **position index** from the list.
- **Clear()**. Removes all elements from the list.

```
int num, pos;

num = int.Parse(InputBox("Enter the number to remove."));
// We remove the num number (first occurrence)
numbers.Remove(num);

pos = int.Parse(InputBox("Enter the position you want remove."));
// We remove the element at the specified index
numbers.RemoveAt(pos);

// We removes all the elements
numbers.Clear();
```

**Searching and sorting a List**

Lists have methods or functions to find an element within the list and to sort their elements.

- **IndexOf(value).** Returns the first position the value is in.
- **LastIndexOf(value)**. Returns the last position the value is in.
- **Contains(value)**. Returns if the list contains the value (true, false).
- **Sort()**. Sorts the elements in the list.

It's important to be careful when working with lists and their limits as we do with arrays. To do this, before calling some methods, we must check (using Count) that we dont'exceed the limits of the list.

For example, if we want to remove an element by position, we must check that this position doesn't exceed the number of elements entered. We must do the same if we want to insert a value in a position with Insert.

Let's imagine we have the following dynamic list:

| | | | | |
|---|---|---|---|---|
| 6 | 28 | 1 | 300 | 2 |

and we do **numbers.Add(50)**

| | | | | | |
|---|---|---|---|---|---|
| 6 | 28 | 1 | 300 | 2 | 50 |

and then **numbers.Insert(2, 100)**

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 28 | 100 | 1 | 300 | 2 | 50 |

and then **numbers.Remove(300)**

| | | | | | |
|---|---|---|---|---|---|
| 6 | 28 | 100 | 1 | 2 | 50 |

and then **numbers.RemoveAt(2)**

| | | | | |
|---|---|---|---|---|
| 6 | 28 | 1 | 2 | 50 |

If at this point we try to do **numbers.RemoveAt(5)** the program will give us an execution error, since we have 5 elements in a range of 0 to 4.

It will also give us an error if we try to do something like **numbers.Insert(10, 20).**