Tema 6. Listas

1. Introducción

En este tema vamos a introducir un nuevo tipo de datos estructurado que nos va a permitir trabajar con **colecciones de datos** de forma similar a como lo hacemos con los vectores pero con mayor funcionalidad. Entre otras cosas nos va a permitir trabajar con listas de datos dinámicas, es decir que su tamaño podrá variar a lo largo de la ejecución de programa, a diferencia de los vectores cuyo tamaño se define al principio.

2. La clase List

Los objetos de tipo colección creados con esta clase son del mismo tipo, pero a diferencia de los arrays el número de elementos puede modificarse **dinámicamente**.

Podremos añadir elementos a la lista o eliminarlos.

Para utilizar la clase **List** debemos tener su librería en la parte de arriba:

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

Una colección dinámica se crea de manera similar a un array. El modo de creación de una colección dinámica consiste en utilizar la palabra clave **new**, seguida de la palabra clave List.

```
List<tipo> numeros = new List<tipo>();
```

Es necesario en una lista de tipo List indicar de qué tipo son los datos que va a contener ya que serán datos del mismo tipo.

Agregar valores a una Lista

Podemos añadir valores a una lista mediante los métodos siguientes:

- Add(valor). Añade al final de la lista el elemento valor.
- Insert(Posición, Valor). Inserta el valor en la posición del array, desplazando el resto de valores una posición adelante.

```
int num, pos;

// Añadimos al final de la Lista
num = int.Parse(InputBox("Introduzca el número."));
numeros.Add(num);


// Insertamos en la posición que indica pos.
num = int.Parse(InputBox("Introduzca el número."));
pos = int.Parse(InputBox("Introduzca la posición en la que quiere
              insertar."));
numeros.Insert(pos, num);
```

Tamaño, acceso y recorrido de una Lista

Como hemos dicho una lista es un vector dinámico, es decir, su tamaño varía a lo largo de la ejecución del programa.

Count nos devuelve el número de elementos que tiene la lista.

Podemos acceder a los elementos de una lista en Visual C# con índices, de igual manera a como lo hacemos con un vector.

```
int i;
string texto;

texto = "Los elementos de la lista son: ";
for (i = 0; i < numeros.Count; i++)
    texto = texto + numeros[i] + ", ";

MessageBox.Show(texto);
```

Por ejemplo, si quisiéramos sumar los enteros que forman parte de una lista se haría de la siguiente forma, no hace falta hacer ningún casting ya que el compilador sabe de qué tipo es el elemento que forma parte de la lista.

```
int suma = 0;

for (int i = 0; i < lista.Count; i++)
    suma = suma + lista[i];

MessageBox.Show("La suma es " + suma);
```

Recorrido de elementos utilizando foreach

Otra posible forma de recorrer una lista es utilizando la instrucción foreach. La instrucción foreach trabaja de manera similiar al for pero sin necesidad de utilizar los índices. Lo que hacemos es navegar con una variable que **va tomando el valor de cada uno de los elementos** de la lista

La sintaxis de foreach es:

```
foreach(tipo variable in List)
        acciones con variable
```

Por ejemplo, si queremos recorrer y mostrar los datos de la lista:

```
string texto;

texto = "Los elementos de la lista son: ";
foreach(int num in lista)
    texto = texto + num + ", ";

MessageBox.Show(texto);
```

Si queremos hacer la suma de los elementos de la lista lo hacemos así (tampoco es necesario hacer casting en este caso):

```
int suma = 0;

foreach(int num in lista)
    suma = suma + num;

MessageBox.Show("La suma es " + suma);
```

Borrado de elementos en una colección List

Como hemos dicho, List es una lista dinámica a la cual podemos ir añadiendo elementos durante la ejecución, y lógicamente también podemos eliminar elementos mediante alguna de las funciones siguientes:

- Remove(valor). Elimina el elemento del array que **corresponde a valor**.
- RemoveAt(posición). Elimina el elemento del array **situado en el índice** posición.
- Clear(). Elimina todos los elementos del objeto.

```
int num, pos;

num = int.Parse(InputBox("Introduzca el número a borrar."));
// Eliminamos el número num.
numeros.Remove(num);

pos = int.Parse(InputBox("Introduzca la posición que quiere
                          borrar."));
// Eliminamos el elemento que hay en la posición pos
numeros.RemoveAt(pos);

// Eliminamos todos los elementos de la lista.
numeros.Clear();
```

Búsqueda y ordenación en List.

Las listas tienen métodos o funciones para buscar un elemento dentro de la lista y para ordenar los elementos.

- IndexOf(valor). Devuelve la primera posición en la que se encuentra el valor.
- LastIndexOf(Valor). Devuelve la última posición en la que se encuentra el Valor.
- Contains(Valor). Devuelve si la lista contiene al valor (true, false).
- Sort(). Ordena la lista.

Es importante que, al trabajar con listas, al igual que con los arrays, tengamos cuidado con los **límites de la lista**. Para ello, según con qué métodos, debemos comprobar (utilizando **Count**) que no nos pasamos de los límites de la lista.

Por ejemplo, si queremos borrar un elemento por posición, deberemos comprobar que esa posición no excede el número de elementos introducidos. Lo mismo deberemos hacer si queremos insertar un valor en una posición con Insert.

Imaginemos que tenemos la siguiente lista dinámica:

6	28	1	300	2

y hacemos **lista.Add(50)**

6	28	1	300	2	50

y a continuación **lista.Insert(2, 100)**

6	28	100	1	300	2	50

y a continuación **lista.Remove(300)**

6	28	100	1	2	50

y a continuación **lista.RemoveAt(2)**

6	28	1	2	50

si en este punto intentáramos hacer **lista.RemoveAt(5)** nos daría error de ejecución, ya que tenemos 5 elementos que van del 0 al 4.

También nos daría error si intentáramos hacer algo como **lista.Insert(10, 20)**