

# C# Y .NET 8

## Parte 6. LINQ

2024-07

Rafael Alberto Moreno Parra  
ramsoftware@gmail.com

## Contenido

Tabla de ilustraciones.....	3
Acerca del autor.....	4
Licencia de este libro .....	4
Licencia del software .....	4
Marcas registradas .....	5
Dedicatoria .....	6
Filtrar de un arreglo.....	7
Filtrar y poner en un List .....	8
Ordenación .....	9
Ordenación descendente .....	10
Consulta con salida a texto personalizado .....	11
Contar los registros .....	12
Máximo, mínimo y suma .....	13
Máximo, mínimo y suma con condiciones .....	15
Consulta con elementos tipo string.....	17
Consulta con objetos .....	18
Consulta con objetos y resultado en un List.....	20
Determinación de tipo de dato.....	22
Ordenación por un campo y luego por otro .....	24
Agrupación por un campo .....	26
Hacer un "join" entre listas .....	28
Un "join" con resultado personalizado .....	30
Extraer los datos de una lista que no están en otra.....	32
Intersección de dos listas .....	33
Unir dos listas sin repetir elementos.....	34
Consulta de texto por algún patrón .....	35
Ordenar internamente una cadena .....	37
Ordenar internamente una cadena con diversos caracteres alfanuméricos .....	38
Ordenamiento según tamaño de la palabra.....	39
Ordenamiento por la segunda letra de cada palabra .....	40
Invertir el ordenamiento.....	41
Métricas: Comparativa entre usar LINQ e implementación tradicional .....	42

## Tabla de ilustraciones

Ilustración 1: LINQ: Filtrar de un arreglo.....	7
Ilustración 2: LINQ: Filtrar y poner en un List .....	8
Ilustración 3: LINQ: Ordenación .....	9
Ilustración 4: LINQ: Ordenación descendente .....	10
Ilustración 5: LINQ: Consulta con salida a texto personalizado .....	11
Ilustración 6: LINQ: Contar los registros .....	12
Ilustración 7: LINQ: Máximo, mínimo y suma .....	14
Ilustración 8: LINQ: Máximo, mínimo y suma con condiciones .....	15
Ilustración 9: LINQ: Consulta con elementos tipo string .....	17
Ilustración 10: LINQ: Consulta con objetos.....	19
Ilustración 11: LINQ: Consulta con objetos y resultado en un List .....	21
Ilustración 12: LINQ: Determinación de tipo de dato.....	23
Ilustración 13: LINQ: Ordenación por un campo y luego por otro .....	25
Ilustración 14: LINQ: Agrupación por un campo .....	27
Ilustración 15: LINQ: Hacer un "join" entre listas .....	29
Ilustración 16: LINQ: Un "join" con resultado personalizado .....	31
Ilustración 17: LINQ: Extraer los datos de una lista que no están en otra .....	32
Ilustración 18: LINQ: Intersección de dos listas.....	33
Ilustración 19: LINQ: Unir dos listas sin repetir elementos .....	34
Ilustración 20: LINQ: Consulta de texto por algún patrón .....	36
Ilustración 21: LINQ: Ordenar internamente una cadena .....	37
Ilustración 22: LINQ: Ordenar internamente una cadena .....	38
Ilustración 23: LINQ: Ordenamiento según tamaño de la palabra .....	39
Ilustración 24: LINQ: Ordenamiento por la segunda letra de cada palabra .....	40
Ilustración 25: LINQ: Invertir el ordenamiento .....	41
Ilustración 26: Comparativa desempeño de LINQ .....	44

## Acerca del autor

Rafael Alberto Moreno Parra

[ramsoftware@gmail.com](mailto:ramsoftware@gmail.com) o [enginelifelife@hotmail.com](mailto:enginelifelife@hotmail.com)

Sitio Web: <http://darwin.50webs.com> (dedicado a la investigación de algoritmos evolutivos y vida artificial).

Github: <https://github.com/ramsoftware>

Youtube: <https://www.youtube.com/@RafaelMorenoP>

## Licencia de este libro



## Licencia del software

Todo el software desarrollado aquí tiene licencia LGPL "Lesser General Public License" [1]



## Marcas registradas

En este libro se hace uso de las siguientes tecnologías registradas:

Microsoft ® Windows ® Enlace: <http://windows.microsoft.com/en-US/windows/home>

Microsoft ® Visual Studio 2022 ® Enlace: <https://visualstudio.microsoft.com/es/vs/>

## Dedicatoria

A mis padres, a mi hermana....

Y a mi tropa gatuna: Sally, Suini, Grisú, Capuchina, Milú,  
Arián, Frac y mis recordados Tinita, Tammy, Vikingo y  
Michu.

# Filtrar de un arreglo

F/001.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3];

            //Consulta: Extrayendo solo los números impares
            //¡OJO! Sólo crea la instrucción de consulta pero
            //todavía no la hace
            IEnumerable<int> Consulta =
                from numero in Lista
                where (numero % 2) == 1
                select numero;

            //Ejecuta la consulta y la imprime
            foreach (int Valor in Consulta)
                Console.Write(Valor.ToString() + ", ");
        }
    }
}
```



Ilustración 1: LINQ: Filtrar de un arreglo

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8];

            //Consulta: Extrayendo solo los números pares
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            List<int> Resultados = (from numero in Lista
                                   where (numero % 2) == 0
                                   select numero).ToList();

            //Ejecuta la consulta y la imprime
            for (int cont = 0; cont < Resultados.Count; cont++) {
                Console.Write(Resultados[cont].ToString() + ", ");
            }
        }
    }
}
```

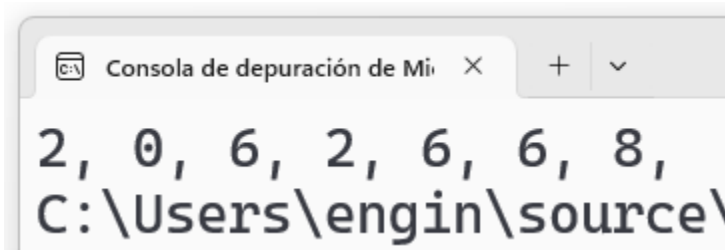


Ilustración 2: LINQ: Filtrar y poner en un List



```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Fuente de datos, un arreglo unidimensional  
            int[] Lista = new int[] { 1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8 };  
  
            //Consulta: Extrayendo solo los números pares en orden ascendente  
            //¡OJO! Aquí si se ejecuta la consulta de una vez  
            List<int> Resultados = (from numero in Lista  
                                   where (numero % 2) == 0  
                                   orderby numero select numero).ToList();  
  
            //Ejecuta la consulta y la imprime  
            for (int cont = 0; cont < Resultados.Count; cont++) {  
                Console.Write(Resultados[cont].ToString() + ", ");  
            }  
        }  
    }  
}
```

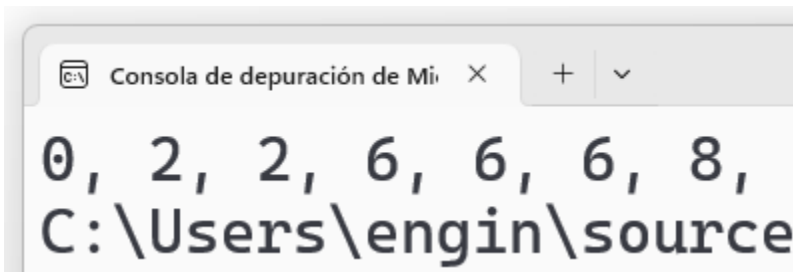


Ilustración 3: LINQ: Ordenación

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8];

            //Consulta: Extrayendo solo los números pares en orden
            //descendente. ¡OJO! Aquí si se ejecuta la consulta
            //de una vez
            List<int> Resultados = (from numero in Lista
                                    where (numero % 2) == 0
                                    orderby numero descending
                                    select numero).ToList();

            //Ejecuta la consulta y la imprime
            for (int contador = 0; contador < Resultados.Count; contador++) {
                Console.Write(Resultados[contador].ToString() + ", ");
            }
        }
    }
}
```

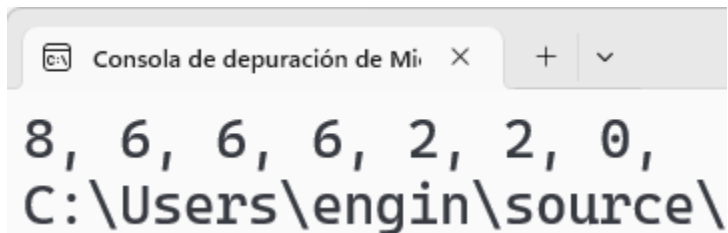


Ilustración 4: LINQ: Ordenación descendente

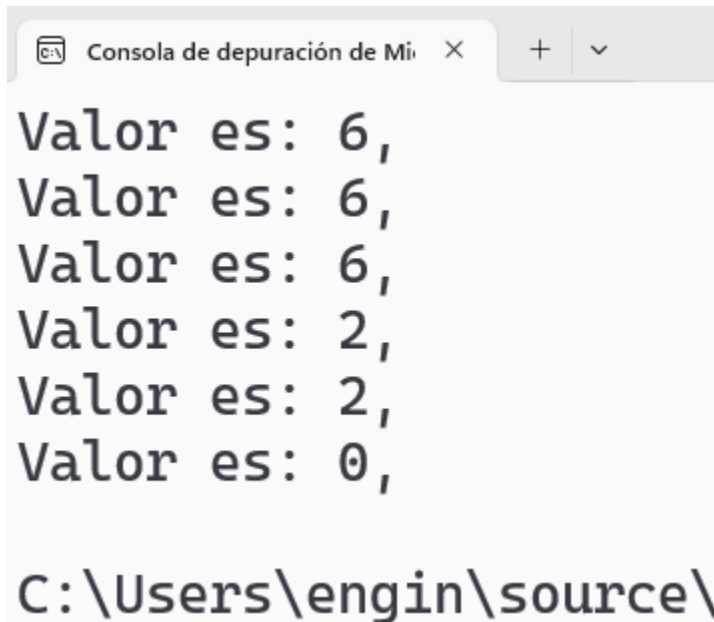
## Consulta con salida a texto personalizado

F/005.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6];

            //Consulta: Extrayendo solo los números pares en
            //orden descendente
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            List<string> Resultados =
                (from numero in Lista
                 where (numero % 2) == 0
                 orderby numero descending
                 select $"Valor es: {numero}, ").ToList();

            //Ejecuta la consulta y la imprime
            for (int cont = 0; cont < Resultados.Count; cont++) {
                Console.WriteLine(Resultados[cont]);
            }
        }
    }
}
```



```
C:\Users\engin\source\
```

Ilustración 5: LINQ: Consulta con salida a texto personalizado

# Contar los registros

F/006.cs

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Fuente de datos, un arreglo unidimensional  
            int[] Lista = [9, 2, 9, 2, 3, 8, 6, 1];  
  
            //Consulta: Contando los números pares  
            //¡OJO! Aquí si se ejecuta la consulta de una vez  
            int TotalRegistros =  
                (from numero in Lista  
                 where (numero % 2) == 0  
                 select numero).Count();  
  
            //Ejecuta la consulta y la imprime  
            Console.WriteLine(TotalRegistros);  
        }  
    }  
}
```

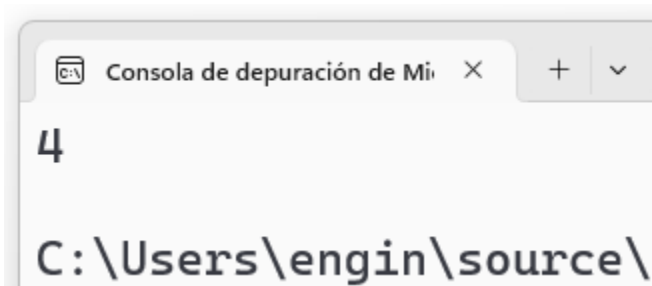


Ilustración 6: LINQ: Contar los registros

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = [8, -3, 2, 10, -7, 3, 0, 4];

            //Consulta: Máximo, mínimo y suma
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            int Maximo = (from numero in Lista
                          select numero).Max();

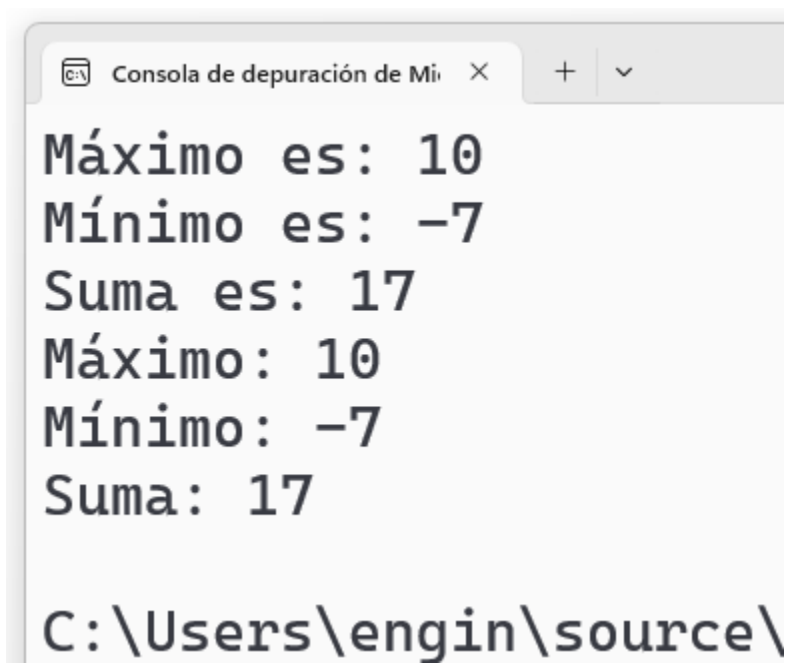
            int Minimo = (from numero in Lista
                          select numero).Min();

            int Suma = (from numero in Lista
                        select numero).Sum();

            //Ejecuta la consulta y la imprime
            Console.WriteLine("Máximo es: " + Maximo);
            Console.WriteLine("Mínimo es: " + Minimo);
            Console.WriteLine("Suma es: " + Suma);

            //Otra forma de hacerlo
            int maximo = Lista.Max();
            int minimo = Lista.Min();
            int suma = Lista.Sum();

            //Imprime
            Console.WriteLine("Máximo: " + maximo);
            Console.WriteLine("Mínimo: " + minimo);
            Console.WriteLine("Suma: " + suma);
        }
    }
}
```



A screenshot of a Visual Studio debug console window. The title bar reads 'Consola de depuración de Mi' with a close button. The console contains the following text: 'Máximo es: 10', 'Mínimo es: -7', 'Suma es: 17', 'Máximo: 10', 'Mínimo: -7', 'Suma: 17', and the file path 'C:\Users\engin\source\'.

```
Consola de depuración de Mi × + v
Máximo es: 10
Mínimo es: -7
Suma es: 17
Máximo: 10
Mínimo: -7
Suma: 17
C:\Users\engin\source\
```

*Ilustración 7: LINQ: Máximo, mínimo y suma*

# Máximo, mínimo y suma con condiciones

F/008.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            int[] Lista = [8, -3, 2, 10, -7, 3];

            //Consulta: Máximo y mínimo
            //;OJO! Aquí si se ejecuta
            //la consulta de una vez
            int Maximo = (from numero in Lista
                          where numero % 2 == 0
                          select numero).Max();

            int Minimo = (from numero in Lista
                          where numero % 2 == 0
                          select numero).Min();

            int Suma = (from numero in Lista
                        where numero % 2 == 0
                        select numero).Sum();

            //Ejecuta la consulta y la imprime
            Console.WriteLine("Máximo es: " + Maximo);
            Console.WriteLine("Mínimo es: " + Minimo);
            Console.WriteLine("Suma es: " + Suma);
        }
    }
}
```

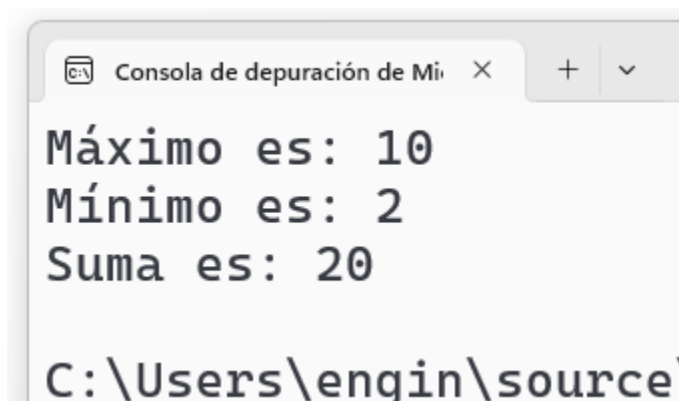


Ilustración 8: LINQ: Máximo, mínimo y suma con condiciones





## Consulta con elementos tipo string

F/009.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Fuente de datos, un arreglo unidimensional
            string[] Lista = [ "búho", "loro", "gaviota",
                               "azulejo", "bichofue", "canario" ];

            //Consulta, especies de aves que tengan
            //la letra 'a'
            //¡OJO! Aquí si se ejecuta la consulta de una vez
            int Cuenta = (from aves in Lista
                          where aves.Contains("a")
                          select aves).Count();

            //Ejecuta la consulta y la imprime
            Console.WriteLine("Aves con letra a: " + Cuenta);
        }
    }
}
```

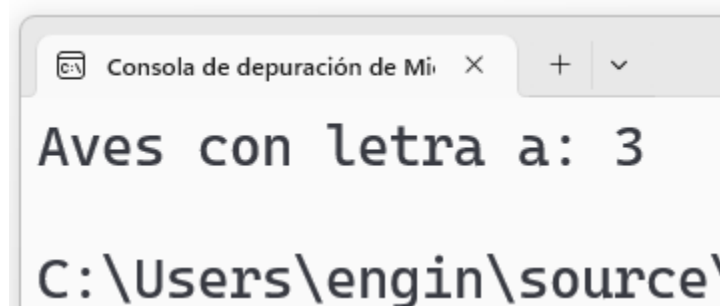


Ilustración 9: LINQ: Consulta con elementos tipo string

```
namespace Ejemplo {
    internal class Mascota {
        public intCodigo;
        public string Nombre;
        public int FechaNace; //Formato: aaaammdd

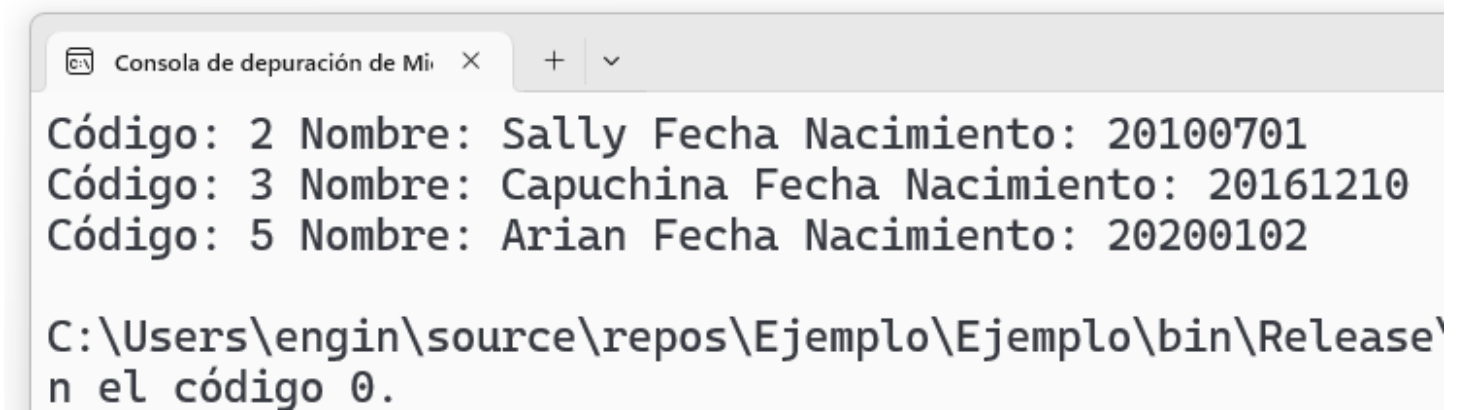
        public Mascota(intCodigo, string Nombre, int FechaNace) {
            this.Codigo = Codigo;
            this.Nombre = Nombre;
            this.FechaNace = FechaNace;
        }

        public void Imprime() {
            Console.WriteLine("Código: " + Codigo);
            Console.WriteLine(" Nombre: " + Nombre);
            Console.WriteLine(" Fecha Nacimiento: " + FechaNace);
        }
    }

    internal class Program {
        static void Main() {
            //Fuente de datos, una lista
            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota(1, "Suini", 20121012));
            listaMascotas.Add(new Mascota(2, "Sally", 20100701));
            listaMascotas.Add(new Mascota(3, "Capuchina", 20161210));
            listaMascotas.Add(new Mascota(4, "Grisú", 20161120));
            listaMascotas.Add(new Mascota(5, "Arian", 20200102));
            listaMascotas.Add(new Mascota(6, "Milú", 20160706));

            //Extraiga los registros donde el nombre tenga la letra 'a'
            List<Mascota> Resultados = (from animal in listaMascotas
                                        where animal.Nombre.Contains("a")
                                        select animal).ToList();

            //Ejecuta la consulta y la imprime
            for (int cont = 0; cont < Resultados.Count; cont++) {
                Resultados[cont].Imprime();
            }
        }
    }
}
```



The image shows a screenshot of a Visual Studio debug console window. The title bar at the top reads 'Consola de depuración de Mi' followed by a close button (X) and a dropdown menu with a plus sign and a downward arrow. The console contains three lines of text representing query results, each on a new line: 'Código: 2 Nombre: Sally Fecha Nacimiento: 20100701', 'Código: 3 Nombre: Capuchina Fecha Nacimiento: 20161210', and 'Código: 5 Nombre: Arian Fecha Nacimiento: 20200102'. Below these, there is a line of code: 'C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\' followed by a line break and 'n el código 0.'

```
Consola de depuración de Mi X + v
Código: 2 Nombre: Sally Fecha Nacimiento: 20100701
Código: 3 Nombre: Capuchina Fecha Nacimiento: 20161210
Código: 5 Nombre: Arian Fecha Nacimiento: 20200102

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release'
n el código 0.
```

*Ilustración 10: LINQ: Consulta con objetos*

## Consulta con objetos y resultado en un List

F/011.cs

```
namespace Ejemplo {
    internal class Mascota {
        public int Codigo { get; set; }
        public string Nombre { get; set; }
        public int FechaNace { get; set; } //Formato: aaaammdd

        public Mascota(int Codigo, string Nombre, int FechaNace) {
            this.Codigo = Codigo;
            this.Nombre = Nombre;
            this.FechaNace = FechaNace;
        }

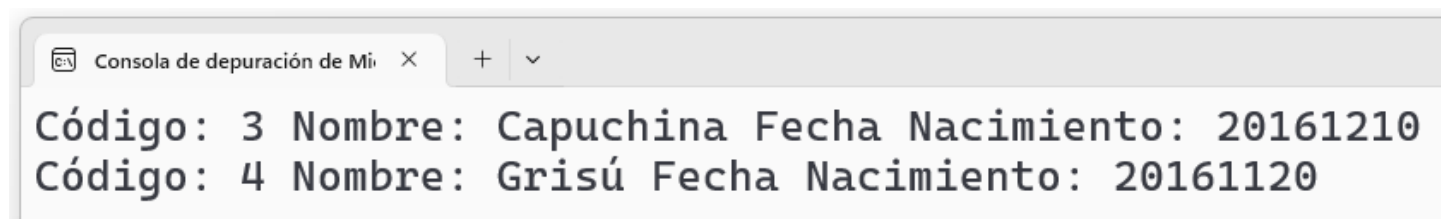
        public void Imprime() {
            Console.WriteLine("Código: " + Codigo);
            Console.WriteLine(" Nombre: " + Nombre);
            Console.WriteLine(" Fecha Nacimiento: " + FechaNace);
        }
    }

    internal class Program {
        static void Main() {
            //Fuente de datos, una lista
            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota(1, "Suini", 20121012));
            listaMascotas.Add(new Mascota(2, "Sally", 20100701));
            listaMascotas.Add(new Mascota(3, "Capuchina", 20161210));
            listaMascotas.Add(new Mascota(4, "Grisú", 20161120));
            listaMascotas.Add(new Mascota(5, "Arian", 20200102));
            listaMascotas.Add(new Mascota(6, "Milú", 20100706));

            //Extraiga los registros donde la fecha de nacimiento
            //esté en un rango
            List<Mascota> Resultados = (from animal in listaMascotas
                                       where animal.FechaNace > 20150101
                                       && animal.FechaNace <= 20161231
                                       select animal).ToList();

            //Ejecuta la consulta y la imprime
            for (int cont = 0; cont < Resultados.Count; cont++) {
                Resultados[cont].Imprime();
            }
        }
    }
}
```

}



The image shows a screenshot of a Visual Studio debug console window. The title bar at the top reads 'Consola de depuración de Mi' followed by a close button (X) and a dropdown menu with a plus sign and a downward arrow. The console output displays two lines of text, each representing an object from a LINQ query. The first line is 'Código: 3 Nombre: Capuchina Fecha Nacimiento: 20161210' and the second line is 'Código: 4 Nombre: Grisú Fecha Nacimiento: 20161120'. The text is in a monospaced font.

```
Código: 3 Nombre: Capuchina Fecha Nacimiento: 20161210  
Código: 4 Nombre: Grisú Fecha Nacimiento: 20161120
```

*Ilustración 11: LINQ: Consulta con objetos y resultado en un List*

```
using System.Collections;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            ArrayList Varios = new ArrayList();

            Varios.Add(1822);
            Varios.Add('M');
            Varios.Add(true);
            Varios.Add(639.9);
            Varios.Add("Rafael");

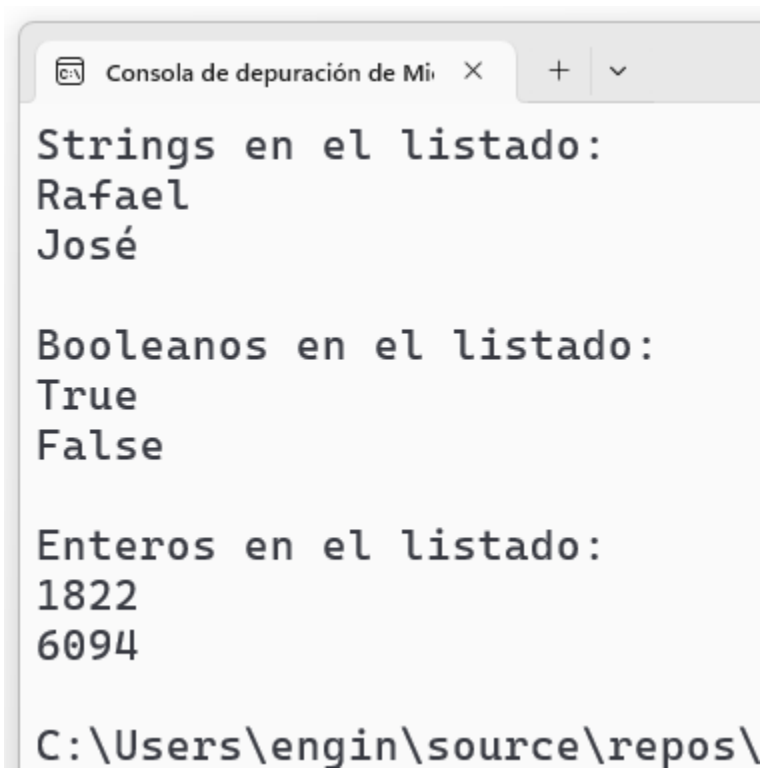
            Varios.Add(6094);
            Varios.Add('J');
            Varios.Add(false);
            Varios.Add(55.5);
            Varios.Add("José");

            //Muestra los ítems que son strings
            Console.WriteLine("Strings en el listado:");
            List<string> Cadenas = (from nombre in
                                   Varios.OfType<string>()
                                   select nombre).ToList();
            for (int Cont = 0; Cont < Cadenas.Count; Cont++) {
                Console.WriteLine(Cadenas[Cont]);
            }

            //Muestra los ítems que son booleanos
            Console.WriteLine("\r\nBooleanos en el listado:");
            List<bool> Booleanos = (from valorbool in
                                    Varios.OfType<bool>()
                                    select valorbool).ToList();
            for (int Cont = 0; Cont < Cadenas.Count; Cont++) {
                Console.WriteLine(Booleanos[Cont]);
            }

            //Muestra los ítems que son enteros
            Console.WriteLine("\r\nEnteros en el listado:");
            List<int> Enteros = (from valorentero in
                                 Varios.OfType<int>()
                                 select valorentero).ToList();
            for (int Cont = 0; Cont < Cadenas.Count; Cont++) {
                Console.WriteLine(Enteros[Cont]);
            }
        }
    }
}
```

```
}  
}  
}  
}
```



The screenshot shows a debug console window titled 'Consola de depuración de Mi'. It displays the results of a LINQ query in a monospaced font. The output is organized into three sections: 'Strings en el listado:' followed by 'Rafael' and 'José'; 'Booleanos en el listado:' followed by 'True' and 'False'; and 'Enteros en el listado:' followed by '1822' and '6094'. At the bottom, the file path 'C:\Users\engin\source\repos\' is visible.

```
Consola de depuración de Mi  X  +  v  
  
Strings en el listado:  
Rafael  
José  
  
Booleanos en el listado:  
True  
False  
  
Enteros en el listado:  
1822  
6094  
  
C:\Users\engin\source\repos\
```

*Ilustración 12: LINQ: Determinación de tipo de dato*

## Ordenación por un campo y luego por otro

F/013.cs

```
namespace Ejemplo {
    internal class Mascota {
        public string Especie { get; set; }
        public string Nombre { get; set; }
        public Mascota(string Especie, string Nombre) {
            this.Especie = Especie;
            this.Nombre = Nombre;
        }

        public void Imprime() {
            Console.Write("Especie: " + Especie);
            Console.WriteLine(" Nombre: " + Nombre);
        }
    }

    internal class Program {
        static void Main() {
            //Fuente de datos, una lista
            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota("gato", "Suini"));
            listaMascotas.Add(new Mascota("gato", "Gris"));
            listaMascotas.Add(new Mascota("gato", "Sally"));
            listaMascotas.Add(new Mascota("gato", "Tinita"));
            listaMascotas.Add(new Mascota("conejo", "Krousky"));
            listaMascotas.Add(new Mascota("gato", "Capuchina"));
            listaMascotas.Add(new Mascota("gato", "Tammy"));
            listaMascotas.Add(new Mascota("gato", "Grisú"));
            listaMascotas.Add(new Mascota("ave", "Lua"));
            listaMascotas.Add(new Mascota("conejo", "Copo"));
            listaMascotas.Add(new Mascota("gato", "Vikingo"));
            listaMascotas.Add(new Mascota("gato", "Arian"));
            listaMascotas.Add(new Mascota("gato", "Milú"));
            listaMascotas.Add(new Mascota("ave", "Azulin"));
            listaMascotas.Add(new Mascota("gato", "Frac"));
            listaMascotas.Add(new Mascota("ave", "Negro"));
            listaMascotas.Add(new Mascota("conejo", "Clopa"));

            //Ordene primero por especie y luego por nombre
            List<Mascota> Resultados = (from animal in listaMascotas
                                        orderby animal.Especie,
                                        animal.Nombre
                                        select animal).ToList();

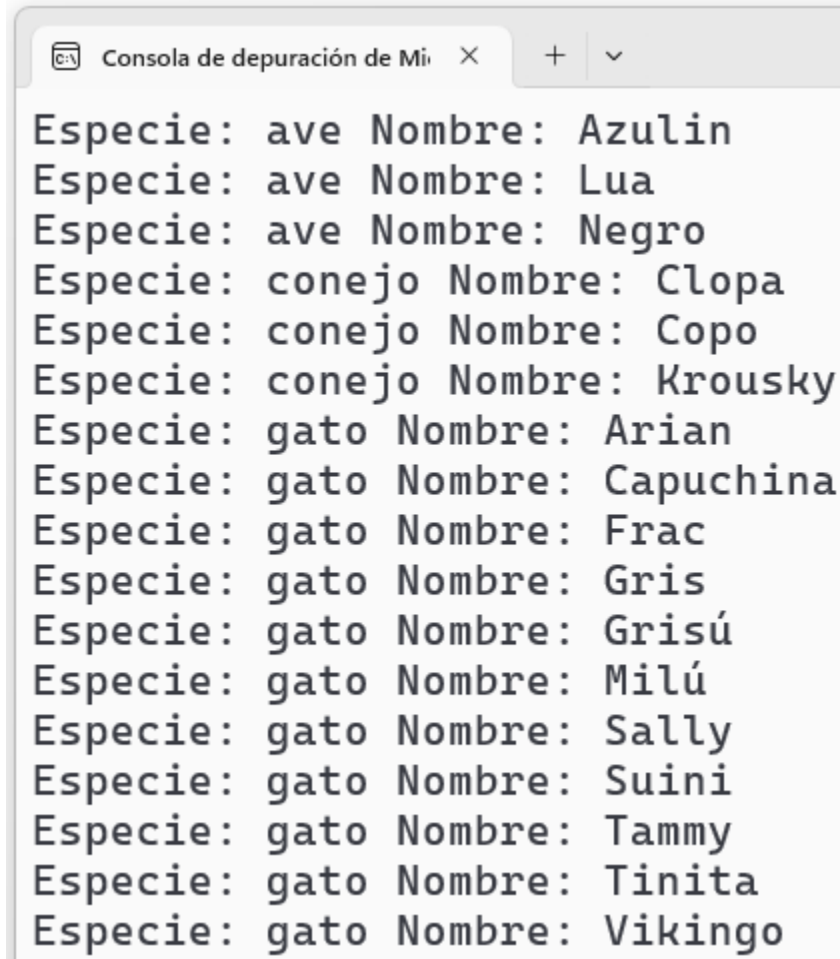
            //Ejecuta la consulta y la imprime
        }
    }
}
```



```

        for (int Cont = 0; Cont < Resultados.Count; Cont++) {
            Resultados[Cont].Imprime();
        }
    }
}

```



```

Especie: ave Nombre: Azulín
Especie: ave Nombre: Lua
Especie: ave Nombre: Negro
Especie: conejo Nombre: Clopa
Especie: conejo Nombre: Copo
Especie: conejo Nombre: Krousky
Especie: gato Nombre: Arian
Especie: gato Nombre: Capuchina
Especie: gato Nombre: Frac
Especie: gato Nombre: Gris
Especie: gato Nombre: Grisú
Especie: gato Nombre: Milú
Especie: gato Nombre: Sally
Especie: gato Nombre: Suini
Especie: gato Nombre: Tammy
Especie: gato Nombre: Tinita
Especie: gato Nombre: Vikingo

```

*Ilustración 13: LINQ: Ordenación por un campo y luego por otro*

```
namespace Ejemplo {
    internal class Mascota {
        public string Especie { get; set; }
        public string Nombre { get; set; }
        public Mascota(string Especie, string Nombre) {
            this.Especie = Especie;
            this.Nombre = Nombre;
        }
    }

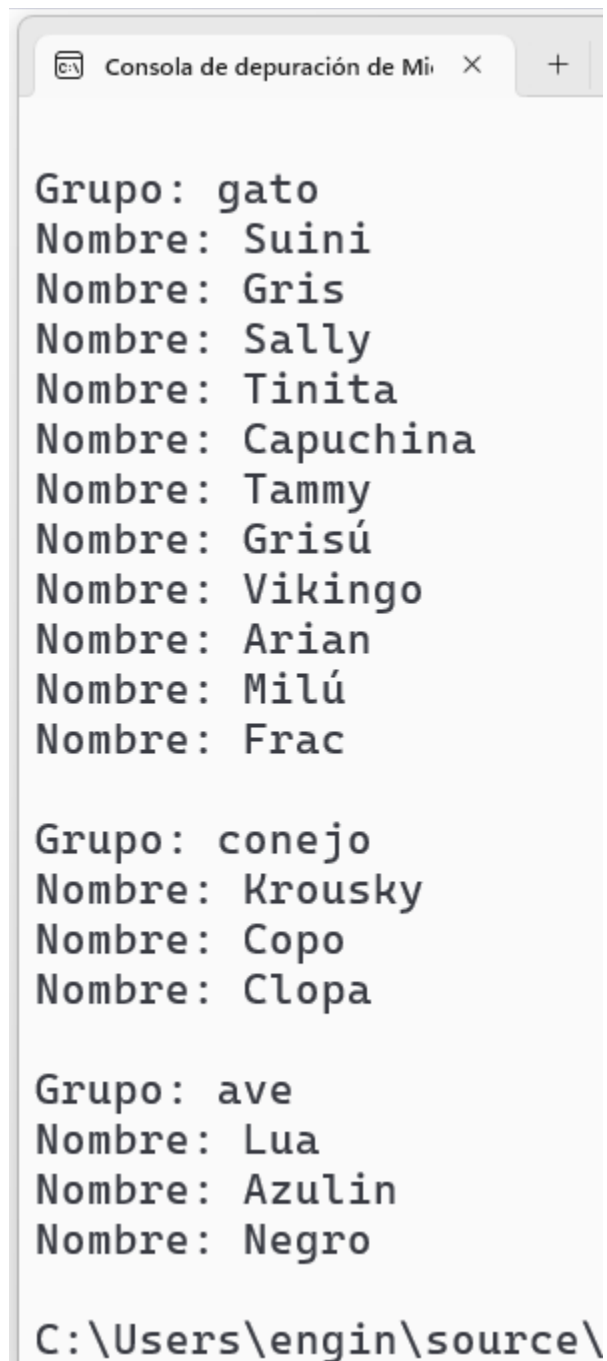
    internal class Program {
        static void Main() {
            //Fuente de datos, una lista
            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota("gato", "Suini"));
            listaMascotas.Add(new Mascota("gato", "Gris"));
            listaMascotas.Add(new Mascota("gato", "Sally"));
            listaMascotas.Add(new Mascota("gato", "Tinita"));
            listaMascotas.Add(new Mascota("conejo", "Krousky"));
            listaMascotas.Add(new Mascota("gato", "Capuchina"));
            listaMascotas.Add(new Mascota("gato", "Tammy"));
            listaMascotas.Add(new Mascota("gato", "Grisú"));
            listaMascotas.Add(new Mascota("ave", "Lua"));
            listaMascotas.Add(new Mascota("conejo", "Copo"));
            listaMascotas.Add(new Mascota("gato", "Vikingo"));
            listaMascotas.Add(new Mascota("gato", "Arian"));
            listaMascotas.Add(new Mascota("gato", "Milú"));
            listaMascotas.Add(new Mascota("ave", "Azulin"));
            listaMascotas.Add(new Mascota("gato", "Frac"));
            listaMascotas.Add(new Mascota("ave", "Negro"));
            listaMascotas.Add(new Mascota("conejo", "Clopa"));

            var ConjuntoGrupos = from animal in listaMascotas
                                group animal by animal.Especie;

            //Itera por grupo
            //Cada grupo tiene una llave
            foreach (var grupo in ConjuntoGrupos) {
                Console.WriteLine("\r\nGrupo: {0}", grupo.Key);

                // Cada grupo tiene una colección interna
                foreach (Mascota individuo in grupo)
                    Console.WriteLine("Nombre: {0}", individuo.Nombre);
            }
        }
    }
}
```

```
}  
}  
}
```



```
C:\> Consola de depuración de Mi... X +  
  
Grupo: gato  
Nombre: Suini  
Nombre: Gris  
Nombre: Sally  
Nombre: Tinita  
Nombre: Capuchina  
Nombre: Tammy  
Nombre: Grisú  
Nombre: Vikingo  
Nombre: Arian  
Nombre: Milú  
Nombre: Frac  
  
Grupo: conejo  
Nombre: Krousky  
Nombre: Copo  
Nombre: Clopa  
  
Grupo: ave  
Nombre: Lua  
Nombre: Azulín  
Nombre: Negro  
  
C:\Users\engin\source\
```

Ilustración 14: LINQ: Agrupación por un campo

## Hacer un “join” entre listas

F/015.cs

```
namespace Ejemplo {
    internal class Especie {
        public int Codigo { get; set; }
        public string Nombre { get; set; }

        public Especie(int Codigo, string Nombre) {
            this.Codigo = Codigo;
            this.Nombre = Nombre;
        }
    }

    internal class Mascota {
        public int Especie { get; set; }
        public string Nombre { get; set; }
        public Mascota(int Especie, string Nombre) {
            this.Especie = Especie;
            this.Nombre = Nombre;
        }
    }

    internal class Program {
        static void Main() {
            List<Especie> listaEspecies = new List<Especie>();
            listaEspecies.Add(new Especie(1, "Gato"));
            listaEspecies.Add(new Especie(2, "Conejo"));
            listaEspecies.Add(new Especie(3, "Ave"));

            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota(1, "Suini"));
            listaMascotas.Add(new Mascota(1, "Gris"));
            listaMascotas.Add(new Mascota(1, "Sally"));
            listaMascotas.Add(new Mascota(1, "Tinita"));
            listaMascotas.Add(new Mascota(2, "Krousky"));
            listaMascotas.Add(new Mascota(1, "Capuchina"));
            listaMascotas.Add(new Mascota(1, "Tammy"));
            listaMascotas.Add(new Mascota(1, "Grisú"));
            listaMascotas.Add(new Mascota(3, "Lua"));
            listaMascotas.Add(new Mascota(2, "Copo"));
            listaMascotas.Add(new Mascota(1, "Vikingo"));
            listaMascotas.Add(new Mascota(1, "Arian"));
            listaMascotas.Add(new Mascota(1, "Milú"));
            listaMascotas.Add(new Mascota(3, "Azulin"));
            listaMascotas.Add(new Mascota(1, "Frac"));
            listaMascotas.Add(new Mascota(3, "Negro"));
        }
    }
}
```

```

listaMascotas.Add(new Mascota(2, "Clopa"));

var Consulta = from mascota in listaMascotas
                join especie in listaEspecies
                on mascota.Especie equals especie.Codigo
                select new {
                    Especie = especie.Nombre,
                    Mascota = mascota.Nombre
                };

foreach (var item in Consulta) {
    Console.WriteLine($"La mascota {item.Mascota}");
    Console.WriteLine($"es de la especie {item.Especie}");
}
}
}
}

```

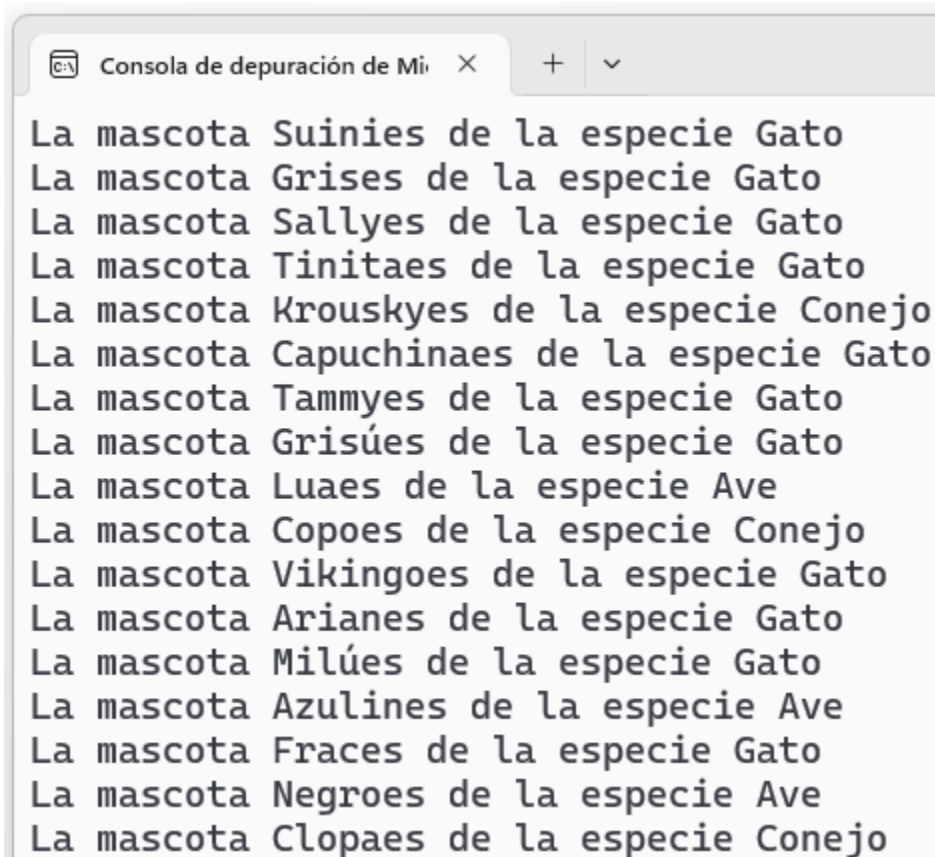


Ilustración 15: LINQ: Hacer un "join" entre listas

## Un "join" con resultado personalizado

F/016.cs

```
namespace Ejemplo {
    internal class Especie {
        public int Codigo { get; set; }
        public string Nombre { get; set; }

        public Especie(int Codigo, string Nombre) {
            this.Codigo = Codigo;
            this.Nombre = Nombre;
        }
    }

    internal class Mascota {
        public int Especie { get; set; }
        public string Nombre { get; set; }
        public Mascota(int Especie, string Nombre) {
            this.Especie = Especie;
            this.Nombre = Nombre;
        }
    }

    internal class Program {
        static void Main() {
            List<Especie> listaEspecies = new List<Especie>();
            listaEspecies.Add(new Especie(1, "Gato"));
            listaEspecies.Add(new Especie(2, "Conejo"));
            listaEspecies.Add(new Especie(3, "Ave"));

            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota(1, "Suini"));
            listaMascotas.Add(new Mascota(1, "Gris"));
            listaMascotas.Add(new Mascota(1, "Sally"));
            listaMascotas.Add(new Mascota(1, "Tinita"));
            listaMascotas.Add(new Mascota(2, "Krousky"));
            listaMascotas.Add(new Mascota(1, "Capuchina"));
            listaMascotas.Add(new Mascota(1, "Tammy"));
            listaMascotas.Add(new Mascota(1, "Grisú"));
            listaMascotas.Add(new Mascota(3, "Lua"));
            listaMascotas.Add(new Mascota(2, "Copo"));
            listaMascotas.Add(new Mascota(1, "Vikingo"));
            listaMascotas.Add(new Mascota(1, "Arian"));
            listaMascotas.Add(new Mascota(1, "Milú"));
            listaMascotas.Add(new Mascota(3, "Azulin"));
            listaMascotas.Add(new Mascota(1, "Frac"));
            listaMascotas.Add(new Mascota(3, "Negro"));
        }
    }
}
```

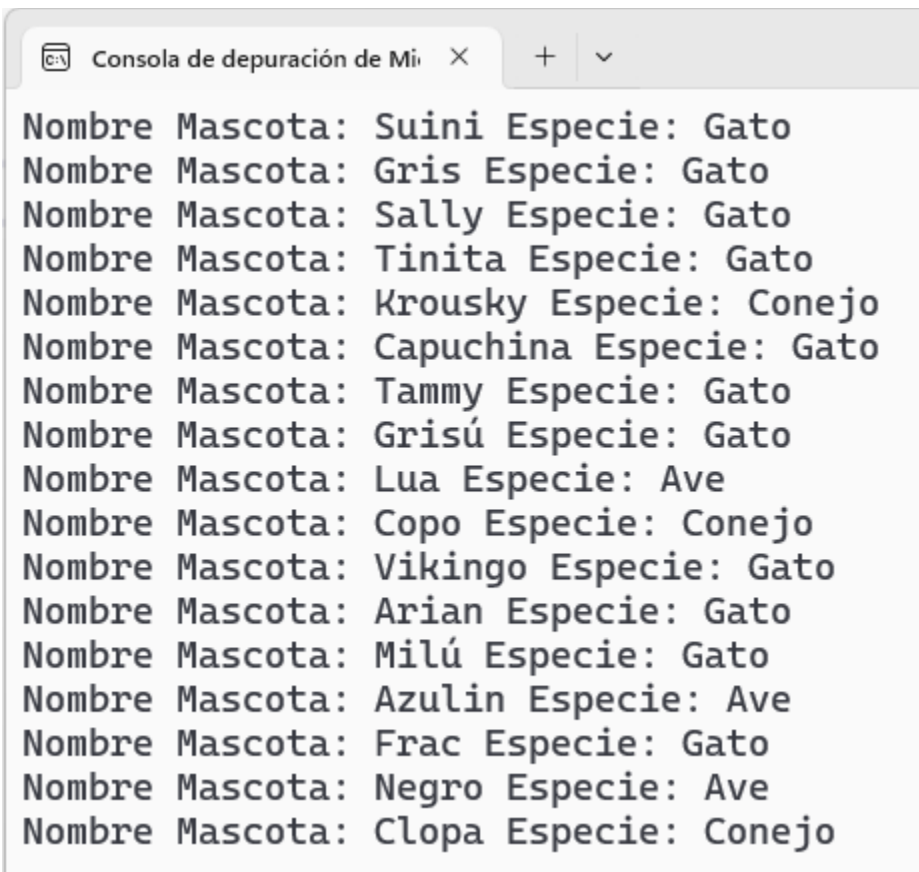
```

listaMascotas.Add(new Mascota(2, "Clopa"));

var Consulta = from mascota in listaMascotas
                join especie in listaEspecies
                on mascota.Especie equals especie.Codigo
                select new {
                    Mascota = "Nombre Mascota: " + mascota.Nombre,
                    Especie = "Especie: " + especie.Nombre
                };

foreach (var item in Consulta) {
    Console.WriteLine($"{item.Mascota} {item.Especie}");
}
}
}
}

```



```

Nombre Mascota: Suini Especie: Gato
Nombre Mascota: Gris Especie: Gato
Nombre Mascota: Sally Especie: Gato
Nombre Mascota: Tinita Especie: Gato
Nombre Mascota: Krousky Especie: Conejo
Nombre Mascota: Capuchina Especie: Gato
Nombre Mascota: Tammy Especie: Gato
Nombre Mascota: Grisú Especie: Gato
Nombre Mascota: Lua Especie: Ave
Nombre Mascota: Copo Especie: Conejo
Nombre Mascota: Vikingo Especie: Gato
Nombre Mascota: Arian Especie: Gato
Nombre Mascota: Milú Especie: Gato
Nombre Mascota: Azulin Especie: Ave
Nombre Mascota: Frac Especie: Gato
Nombre Mascota: Negro Especie: Ave
Nombre Mascota: Clopa Especie: Conejo

```

Ilustración 16: LINQ: Un "join" con resultado personalizado

## Extraer los datos de una lista que no están en otra

F/017.cs

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            List<string> Animales = new List<string>()  
                { "Gato", "Condor", "Perro", "Conejo", "Loro" };  
  
            List<string> Mamiferos = new List<string>()  
                { "Perro", "Conejo", "Gato" };  
  
            /* Extrae los animales que no están en mamíferos */  
            var Resultado = Animales.Except(Mamiferos);  
  
            foreach(string Cadena in Resultado)  
                Console.WriteLine(Cadena);  
        }  
    }  
}
```

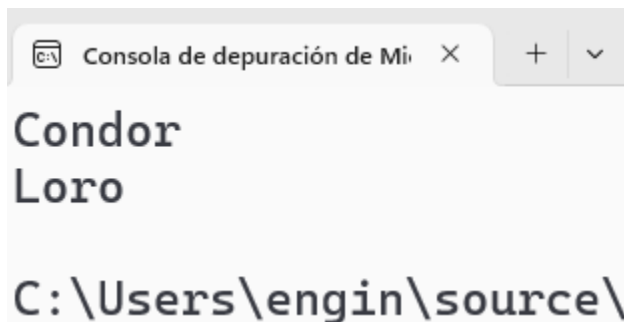


Ilustración 17: LINQ: Extraer los datos de una lista que no están en otra



## Intersección de dos listas

F/018.cs

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            List<string> ColoresA = new List<string>()  
                { "Azul", "Rojo", "Verde", "Violeta" };  
  
            List<string> ColoresB = new List<string>()  
                { "Violeta", "Azul", "Marrón" };  
  
            /* Muestra los colores comunes a ambas listas */  
            var Resultado = ColoresA.Intersect(ColoresB);  
  
            foreach(string Cadena in Resultado)  
                Console.WriteLine(Cadena);  
        }  
    }  
}
```

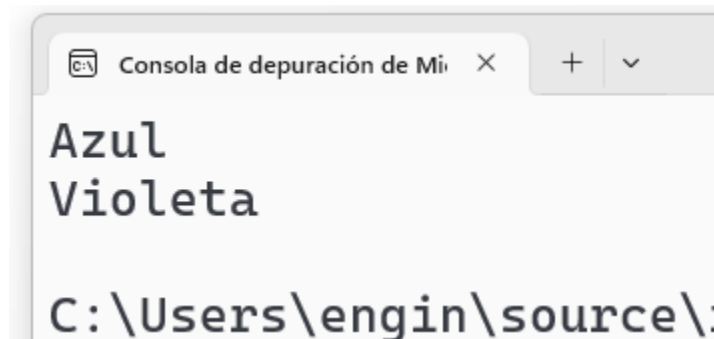


Ilustración 18: LINQ: Intersección de dos listas

## Unir dos listas sin repetir elementos

F/019.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            List<string> ColoresA = new List<string>()
                { "Azul", "Rojo", "Verde", "Marrón", "Violeta" };

            List<string> ColoresB = new List<string>()
                { "Violeta", "Azul", "Marrón", "Naranja" };

            /* Une los valores de ambas listas evitando repetir */
            var Resultado = ColoresA.Union(ColoresB);

            foreach(string Cadena in Resultado)
                Console.WriteLine(Cadena);
        }
    }
}
```

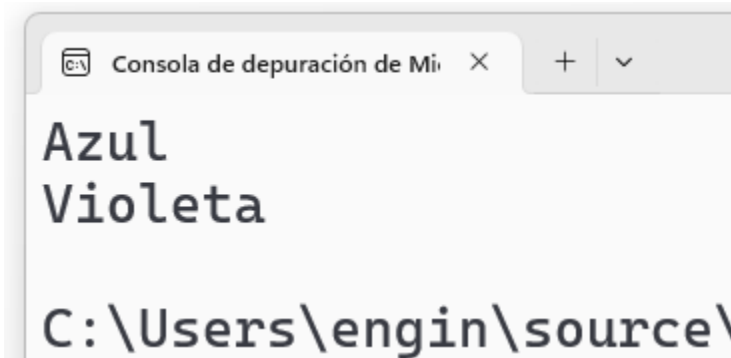


Ilustración 19: LINQ: Unir dos listas sin repetir elementos

## Consulta de texto por algún patrón

F/020.cs

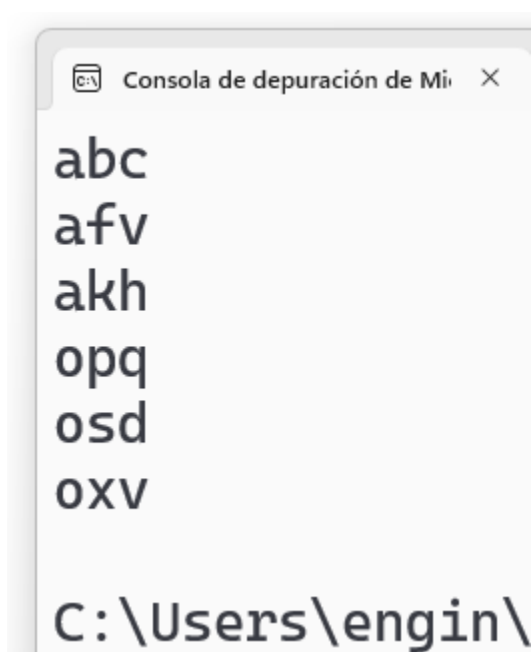
```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            List<string> Textos =
                [ "abc", "Opq", "Afv", "Tkl", "qaz",
                  "Akh", "oSd", "uyt", "oxv" ];

            //Extrae las palabras que empiezan con "a"
            var PalabraMinusculaA = from palabra in Textos
                                     where palabra.ToLower().StartsWith("a")
                                     select palabra.ToLower();

            foreach (string Cadena in PalabraMinusculaA)
                Console.WriteLine(Cadena);

            //Extrae las palabras que empiezan con "o"
            //usando la instrucción Let
            var PalabraMinusculaB = from palabra in Textos
                                     let minuscula = palabra.ToLower()
                                     where minuscula.StartsWith("o")
                                     select minuscula;

            foreach (string Cadena in PalabraMinusculaB)
                Console.WriteLine(Cadena);
        }
    }
}
```



*Ilustración 20: LINQ: Consulta de texto por algún patrón*

# Ordenar internamente una cadena

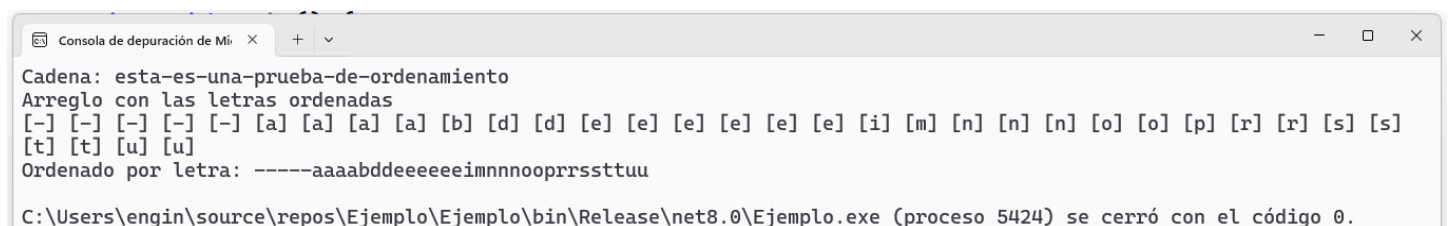
F/021.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string Cadena = "esta-es-una-prueba-de-ordenamiento";
            Console.WriteLine("Cadena: " + Cadena);

            /* Se ordena usando Linq
             * Si desea ordenar los elementos dentro de una secuencia
             * deberá pasar un método keySelector de identidad que
             * indique que cada elemento de
             * la secuencia es, en sí mismo, una clave. */
            IEnumerable<char> resultado = Cadena.OrderBy(str => str);
            Console.WriteLine("Arreglo con las letras ordenadas");
            foreach (int valor in resultado) {
                Console.Write "[" + (char)valor + " ] ");
            }

            // Y convierte ese arreglo en cadena
            string Ordenado = String.Concat(resultado);

            //Imprime
            Console.WriteLine("\r\nOrdenado por letra: " + Ordenado);
        }
    }
}
```



```
Consola de depuración de Mi
Cadena: esta-es-una-prueba-de-ordenamiento
Arreglo con las letras ordenadas
[-] [-] [-] [-] [-] [a] [a] [a] [a] [b] [d] [d] [e] [e] [e] [e] [e] [e] [i] [m] [n] [n] [n] [o] [o] [p] [r] [r] [s] [s]
[t] [t] [u] [u]
Ordenado por letra: -----aaaabddeeeeeimnnnooprsttuu
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0\Ejemplo.exe (proceso 5424) se cerró con el código 0.
```

Ilustración 21: LINQ: Ordenar internamente una cadena

## Ordenar internamente una cadena con diversos caracteres alfanuméricos

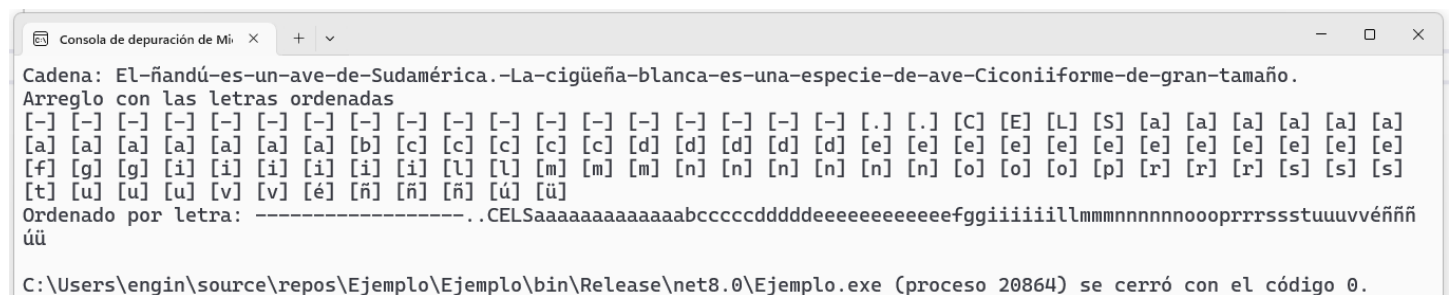
F/022.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string Cadena = "El-ñandú-es-un-ave-de-Sudamérica.";
            Cadena += "-La-cigüeña-blanca-es-una-especie-de-ave";
            Cadena += "-Ciconiiforme-de-gran-tamaño.";
            Console.WriteLine("Cadena: " + Cadena);

            /* Se ordena usando Linq
             * Si desea ordenar los elementos dentro de una secuencia,
             * deberá pasar un método keySelector de identidad que
             * indique que cada elemento de
             * la secuencia es, en sí mismo, una clave. */
            IEnumerable<char> resultado = Cadena.OrderBy(str => str);
            Console.WriteLine("Arreglo con las letras ordenadas");
            foreach (int valor in resultado) {
                Console.Write "[" + (char)valor + " ] ");
            }

            // Y convierte ese arreglo en cadena
            string Ordenado = String.Concat(resultado);

            //Imprime
            Console.WriteLine("\r\nOrdenado por letra: " + Ordenado);
        }
    }
}
```

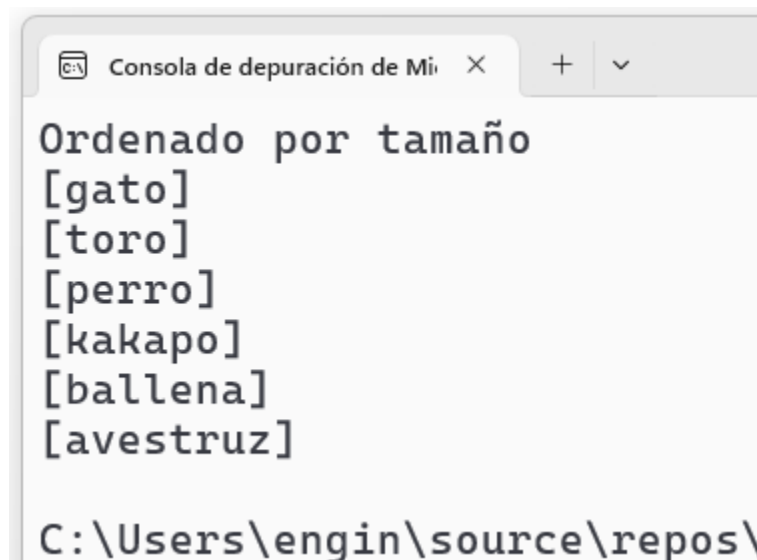


*Ilustración 22: LINQ: Ordenar internamente una cadena*

## Ordenamiento según tamaño de la palabra

F/023.cs

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Una cadena  
            string[] Cad = { "gato", "perro", "avestruz",  
                            "toro", "ballena", "kakapo" };  
  
            /* Se ordena usando Linq  
             * Ordena por tamaño de las cadenas. */  
            IEnumerable<string> Res = Cad.OrderBy(str => str.Length);  
            Console.WriteLine("Ordenado por tamaño");  
            foreach (string valor in Res) {  
                Console.WriteLine "[" + valor + " ] ");  
            }  
        }  
    }  
}
```



```
Consola de depuración de Mi X + v  
Ordenado por tamaño  
[gato]  
[toro]  
[perro]  
[kakapo]  
[ballena]  
[avestruz]  
C:\Users\engin\source\repos\
```

Ilustración 23: LINQ: Ordenamiento según tamaño de la palabra

## Ordenamiento por la segunda letra de cada palabra

F/024.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string[] Cad = { "gato", "perro", "avestruz", "toro",
                            "ballena", "kakapo" };

            /* Se ordena usando Linq
             * Ordena por la segunda letra. */
            IEnumerable<string> resultado = Cad.OrderBy(str => str[1]);
            Console.WriteLine("Ordenado por la segunda letra");
            foreach (string valor in resultado) {
                Console.WriteLine "[" + valor + " ] ");
            }
        }
    }
}
```

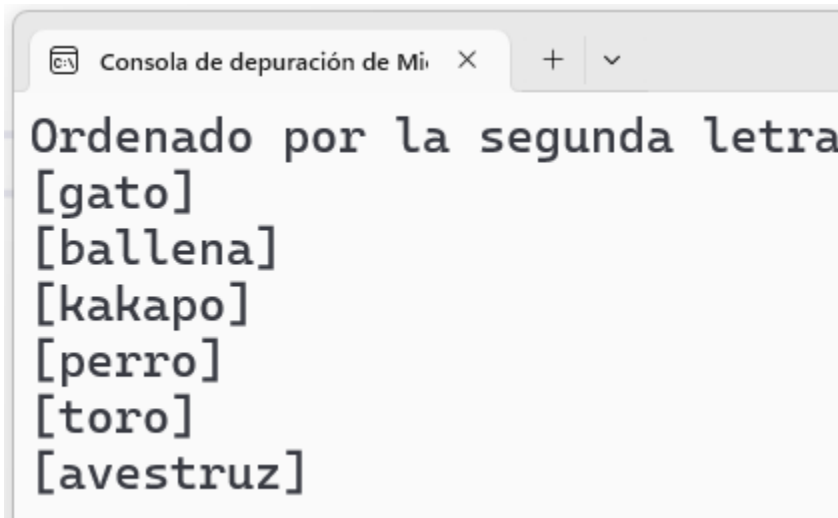


Ilustración 24: LINQ: Ordenamiento por la segunda letra de cada palabra



## Invertir el ordenamiento

F/025.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Una cadena
            string[] Cadenas = { "gato", "perro", "avestruz", "toro",
                                "ballena", "kakapo" };

            /* Invierte el orden en que fueron declarados */
            IEnumerable<string> resultado = Cadenas.Reverse();
            Console.WriteLine("Invierte el orden");
            foreach (string valor in resultado) {
                Console.WriteLine "[" + valor + " ] ");
            }
        }
    }
}
```

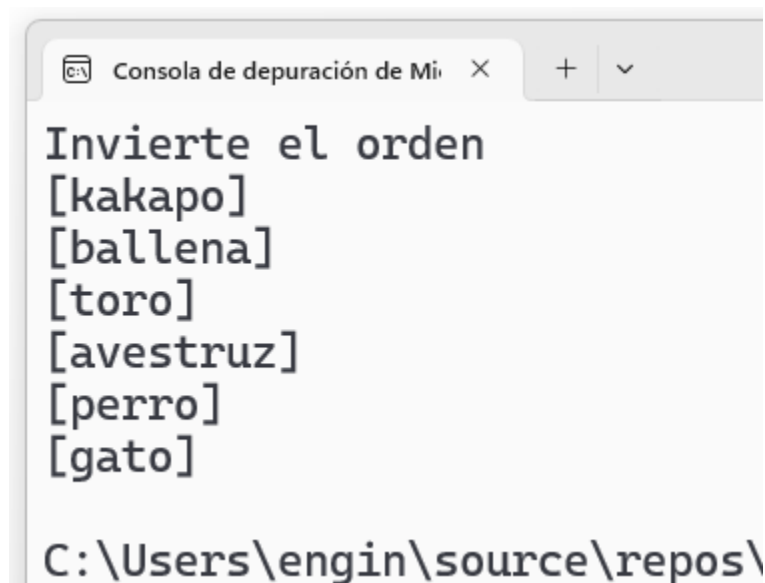


Ilustración 25: LINQ: Invertir el ordenamiento

## Métricas: Comparativa entre usar LINQ e implementación tradicional

LINQ es una herramienta poderosa porque ahorra la escritura de líneas de código, haciendo más fácil el mantenimiento. Pero ¿cuál es el precio que pagar por esta nueva funcionalidad? A continuación, se muestra una comparativa de desempeño entre usar LINQ y escribir el algoritmo tradicional.

F/026.cs

```
using System.Diagnostics;
/* Comparativa entre LINQ e implementación tradicional */

namespace Ejemplo {

    class Program {
        static void Main() {
            Console.WriteLine("Comparativa LINQ vs Clásico\r\n");

            //Generador de números aleatorios único
            Random Azar = new();
            for (int Probar = 1; Probar <= 20; Probar++)
                Pruebas(Azar);
        }

        static public void Pruebas(Random Azar) {
            //Llenar un List<int> con valores al azar
            List<int> Enteros = [];
            for (int Cont = 1; Cont <= 1000000; Cont++) {
                Enteros.Add(Azar.Next(-50, 50));
            }
            List<int> ResultadosLINQ = [];

            //=====
            //Prueba con LINQ
            //=====

            //Medidor de tiempos
            Stopwatch cronometro = new();
            cronometro.Reset();
            cronometro.Start();

            IEnumerable<int> Consulta =
                from numero in Enteros
                where (numero % 2) == 1
                select numero;
```

```

//Ejecuta la consulta y guarda el resultado en una lista
foreach (int Valor in Consulta)
    ResultadosLINQ.Add(Valor);

long TiempoLINQ = cronometro.ElapsedMilliseconds;

//Imprime el tiempo transcurrido y un valor de la lista
Console.Write("Tiempo LINQ: " + TiempoLINQ);

//=====
//Prueba sin LINQ
//=====
List<int> ResultadosNOLINQ = [];
cronometro.Reset();
cronometro.Start();

//Ejecuta la consulta y guarda el resultado en una lista
foreach (int Valor in Enteros)
    if (Valor % 2 == 1) {
        ResultadosNOLINQ.Add(Valor);
    }

long TiempoNOLINQ = cronometro.ElapsedMilliseconds;

//Imprime el tiempo transcurrido y un valor de la lista
Console.Write("Tiempo NO LINQ: " + TiempoNOLINQ);

if (ResultadosNOLINQ.Sum() == ResultadosLINQ.Sum())
    Console.WriteLine("Coinciden");
else
    Console.WriteLine("¡OJO! No hay coincidencia");
}
}
}

```

```
Consola de depuración de Mi X + v
Tiempo LINQ: 23  Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 23  Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 23  Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 15  Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 7   Tiempo NO LINQ: 6 Coinciden
Tiempo LINQ: 6   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 6   Tiempo NO LINQ: 6 Coinciden
Tiempo LINQ: 6   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 5   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 6   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 6   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 6   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 5   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 6   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 5   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 6   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 5   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 6   Tiempo NO LINQ: 5 Coinciden
Tiempo LINQ: 5   Tiempo NO LINQ: 5 Coinciden
C:\Users\engin\source\repos\Ejemplo\Ejemplo\I
```

*Ilustración 26: Comparativa desempeño de LINQ*

Como se puede observar, al principio LINQ es notablemente más lento, pero a medida que se hacen más pruebas, el tiempo consumido por LINQ está llegando prácticamente a igualarse con la implementación algorítmica tradicional.

¡Qué emocionante proyecto! Aquí tienes una lista de temas esenciales que podrías cubrir en tu libro sobre LINQ:

**1. Introducción a LINQ:**

- Historia y evolución de LINQ.
- Beneficios y aplicaciones de LINQ.

**2. Fundamentos de LINQ:**

- Sintaxis básica de LINQ.
- Operadores de consulta estándar (Standard Query Operators).

**3. Tipos de datos y colecciones:**

- Trabajando con colecciones en memoria (List, Array, etc.).
- LINQ to Objects.

**4. Consultas LINQ:**

- Consultas básicas y avanzadas.
- Filtrado, proyección, ordenación y agrupación de datos.

**5. LINQ to SQL:**

- Introducción a LINQ to SQL.
- Mapeo de objetos a bases de datos.
- Consultas y operaciones CRUD (Create, Read, Update, Delete).

**6. LINQ to XML:**

- Trabajando con datos XML.
- Consultas y manipulaciones de XML.

**7. LINQ to Entities (Entity Framework):**

- Introducción a Entity Framework.
- Consultas y operaciones con Entity Framework.

**8. LINQ to JSON:**

- Trabajando con datos JSON.
- Consultas y manipulaciones de JSON.

**9. Evaluación de consultas:**

- Evaluación diferida vs. evaluación inmediata.
- Impacto en el rendimiento.

**10. Expresiones Lambda:**

- Sintaxis y uso de expresiones lambda.
- Funciones anónimas y delegados.

#### **11.Métodos de extensión:**

- Creación y uso de métodos de extensión.
- Aplicaciones prácticas en LINQ.

#### **12.Consultas asincrónicas:**

- Introducción a la programación asincrónica.
- Uso de async y await en consultas LINQ.

#### **13.Optimización y rendimiento:**

- Mejores prácticas para optimizar consultas LINQ.
- Herramientas y técnicas de profiling.

#### **14.Casos de estudio y ejemplos prácticos:**

- Ejemplos del mundo real.
- Soluciones a problemas comunes usando LINQ.

#### **15.Recursos adicionales:**

- Libros, artículos y documentación recomendada.
- Comunidades y foros de discusión.

Estos temas te proporcionarán una base sólida para cubrir todos los aspectos importantes de LINQ y ofrecer a tus lectores un recurso completo y útil. ¡Buena suerte con tu libro! ¿Hay algún tema específico que te gustaría profundizar más?