

C# Y .NET 10

Parte 6. LINQ y LAMBDA

2026-01

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones.....	4
Acerca del autor.....	5
Licencia de este libro	5
Licencia del software	5
Marcas registradas	6
En memoria	7
Filtrar de un List. Ejemplo 1	8
Filtrar de un List. Ejemplo 2	9
Ordenación	10
Ordenación descendente	11
Consulta con salida a texto personalizado	12
Contar los registros	13
Máximo, mínimo y suma	14
Máximo, mínimo y suma con condiciones	16
Consulta con elementos tipo string.....	17
Consulta con objetos. Ejemplo 1.....	18
Consulta con objetos. Ejemplo 2.....	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	29
Un "join" con resultado personalizado	31
Extraer los datos de una lista que no están en otra	33
Intersección de dos listas	34
Unir dos listas sin repetir elementos	35
Consulta de texto por algún patrón	36
Ordenar internamente una cadena	38
Ordenar internamente una cadena con diversos caracteres alfanuméricos	39
Ordenamiento según tamaño de la palabra.....	41
Ordenamiento por la segunda letra de cada palabra	43
Invertir el ordenamiento.....	44
Métricas: Comparativa entre usar LINQ e implementación tradicional	45
Expresiones LAMBDA	48

Expresión Lambda: Uso de varias líneas.....	49
Expresión Lambda: Retornando tipo de dato.....	50
Expresión Lambda: Acceso a variable externa.....	51
Expresión Lambda: Listas. Pero no funciona.	52
Expresión Lambda: Listas. Ahora si funciona.....	53
Expresiones LAMBDA: Listas y variables externas no funciona	54
Expresiones LAMBDA: Uso de static	55
Comparativa entre LINQ y Lambda. Ejemplo 1.	56
Comparativa entre LINQ y Lambda. Ejemplo 2.	57
Comparativa entre LINQ y Lambda. Ejemplo 3.	59
Expresiones Lambda: Promedio	61
JSON y LINQ	62

Tabla de ilustraciones

Ilustración 1: LINQ: Filtrar de un arreglo.....	8
Ilustración 2: LINQ: Filtrar y poner en un List	9
Ilustración 3: LINQ: Ordenación	10
Ilustración 4: LINQ: Ordenación descendente	11
Ilustración 5: Consulta con salida a texto personalizado	12
Ilustración 6: LINQ: Contar los registros	13
Ilustración 7: LINQ: Máximo, mínimo y suma	15
Ilustración 8: LINQ: Máximo, mínimo y suma con condiciones	16
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	28
Ilustración 15: LINQ: Hacer un "join" entre listas	30
Ilustración 16: LINQ: Un "join" con resultado personalizado	32
Ilustración 17: LINQ: Extraer los datos de una lista que no están en otra	33
Ilustración 18: LINQ: Intersección de dos listas.....	34
Ilustración 19: LINQ: Unir dos listas sin repetir elementos	35
Ilustración 20: LINQ: Consulta de texto por algún patrón	37
Ilustración 21: LINQ: Ordenar internamente una cadena	38
Ilustración 22: LINQ: Ordenar internamente una cadena	40
Ilustración 23: LINQ: Ordenamiento según tamaño de la palabra	42
Ilustración 24: LINQ: Ordenamiento por la segunda letra de cada palabra	43
Ilustración 25: LINQ: Invertir el ordenamiento	44
Ilustración 26: Comparativa desempeño de LINQ	47
Ilustración 27: Ejecuta expresión Lambda	48
Ilustración 28: Expresiones Lambda usando varias líneas	49
Ilustración 29: Retornando tipo de dato	50
Ilustración 30: Acceso a variable externa	51
Ilustración 31: No funciona	52
Ilustración 32: Ahora si funciona	53
Ilustración 33: No funciona	54
Ilustración 34: Error al tratar de usar una variable externa	55
Ilustración 35: Comparativa LINQ y Lambda	56
Ilustración 36: Uso de LINQ con y sin expresiones Lambda	58
Ilustración 37: Mismos resultados	60
Ilustración 38: Promedio	61
Ilustración 39: JSON y LINQ	63

Acerca del autor

Rafael Alberto Moreno Parra

ramsoftware@gmail.com o 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 2026 ® Enlace: <https://visualstudio.microsoft.com/es/vs/>

En memoria

De mi gato "Vikingo"



Filtrar de un List. Ejemplo 1

F/001.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Fuente de datos
        List<int> Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3];

        //LINQ, extrae los impares
        List<int> ListaImpares = (from numero in Lista
                                where (numero % 2) == 1
                                select numero).ToList();

        for (int Cont = 0; Cont < ListaImpares.Count; Cont++)
            Console.Write(ListaImpares[Cont] + ", ");
    }
}
```

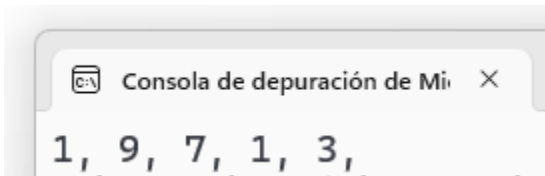


Ilustración 1: LINQ: Filtrar de un arreglo

Filtrar de un List. Ejemplo 2

F/002.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Fuente de datos
        List<int> Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3];

        //LINQ, extrae los mayores o iguales a 5
        List<int> ListaNumeros = (from numero in Lista
                                where numero >= 5
                                select numero).ToList();

        for (int Cont = 0; Cont < ListaNumeros.Count; Cont++)
            Console.Write(ListaNumeros[Cont] + ", ");
    }
}
```

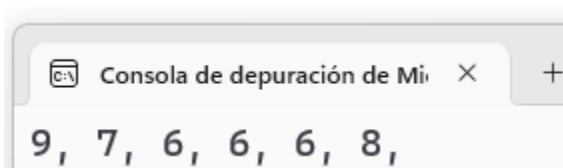


Ilustración 2: LINQ: Filtrar y poner en un List

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Fuente de datos
        List<int> Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3];

        //Consulta: Extrayendo solo los números pares
        //en orden ascendente
        List<int> Resultados = (from numero in Lista
                                where (numero % 2) == 0
                                orderby numero
                                select numero).ToList();

        for (int Cont = 0; Cont < Resultados.Count; Cont++)
            Console.Write(Resultados[Cont] + ", ");
    }
}
```

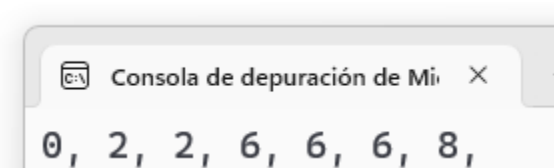


Ilustración 3: LINQ: Ordenación

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Fuente de datos
        List<int> Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3];

        //Consulta: Extrayendo solo los números pares en orden
        //descendente.
        List<int> Resultados = (from numero in Lista
                                where (numero % 2) == 0
                                orderby numero descending
                                select numero).ToList();

        for (int Cont = 0; Cont < Resultados.Count; Cont++)
            Console.Write(Resultados[Cont] + ", ");
    }
}
```

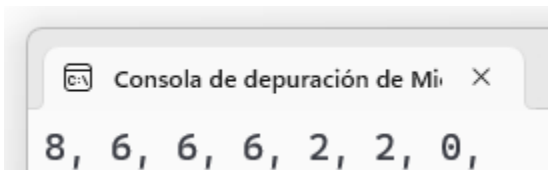


Ilustración 4: LINQ: Ordenación descendente

Consulta con salida a texto personalizado

F/005.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Fuente de datos
        List<int> Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3];

        //Consulta: Extrayendo solo los números pares en
        //orden descendente
        List<string> Resultados =
            (from numero in Lista
             where (numero % 2) == 0
             orderby numero descending
             select $"Valor es: {numero}, ").ToList();

        for (int Cont = 0; Cont < Resultados.Count; Cont++)
            Console.WriteLine(Resultados[Cont]);
    }
}
```

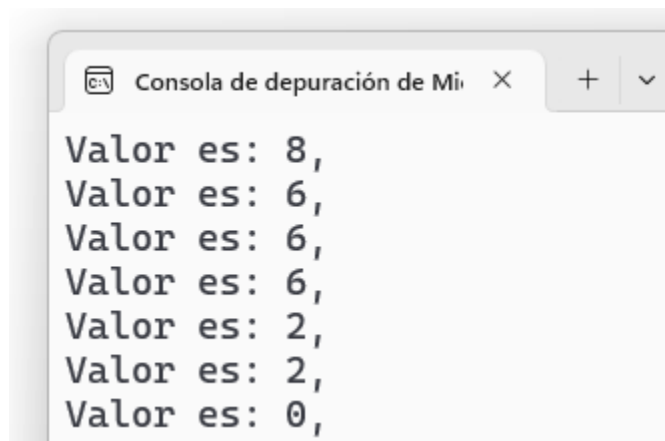


Ilustración 5: Consulta con salida a texto personalizado

Contar los registros

F/006.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Fuente de datos
        List<int> Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3];

        //Consulta: Contando los números pares
        int TotalRegistros =
            (from numero in Lista
             where (numero % 2) == 0
             select numero).Count();

        //Ejecuta la consulta y la imprime
        Console.WriteLine(TotalRegistros);
    }
}
```

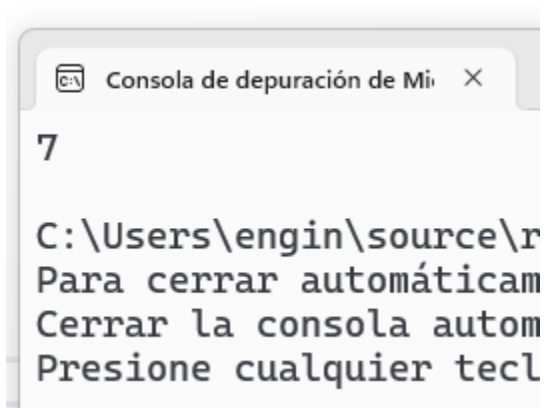


Ilustración 6: LINQ: Contar los registros

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Fuente de datos
        List<int> Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3];

        //Consulta: Máximo, mínimo y suma
        int Maximo = (from numero in Lista
                      select numero).Max();

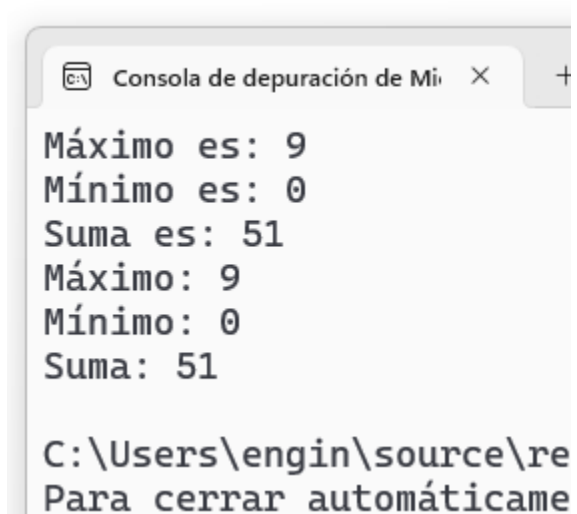
        int Minimo = (from numero in Lista
                      select numero).Min();

        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);
    }
}
```



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 and a plus sign. The console output displays the results of LINQ queries for a sequence of numbers. It shows the maximum value is 9, the minimum value is 0, and the sum is 51. This information is presented twice. At the bottom, a file path 'C:\Users\engin\source\re' is visible, along with the text 'Para cerrar automáticamente'.

```
Máximo es: 9
Mínimo es: 0
Suma es: 51
Máximo: 9
Mínimo: 0
Suma: 51

C:\Users\engin\source\re
Para cerrar automáticamente
```

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
        List<int> Lista = [1, 9, 7, 2, 0, 6, 2, 6, 1, 6, 8, 3];

        //Consulta: Máximo y mínimo
        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);
    }
}
```

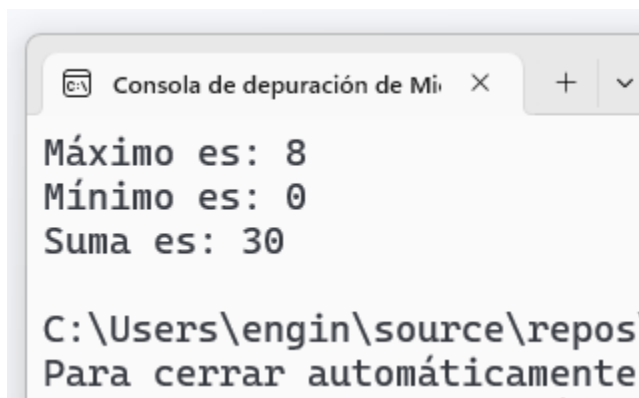


Ilustración 8: LINQ: Máximo, mínimo y suma con condiciones

Consulta con elementos tipo string

F/009.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Fuente de datos
        List<string> Lista = [ "búho", "loro", "gaviota",
                               "azulejo", "bichofue", "canario" ];

        //Consulta, especies de aves que tengan
        //la letra 'a'
        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);
    }
}
```

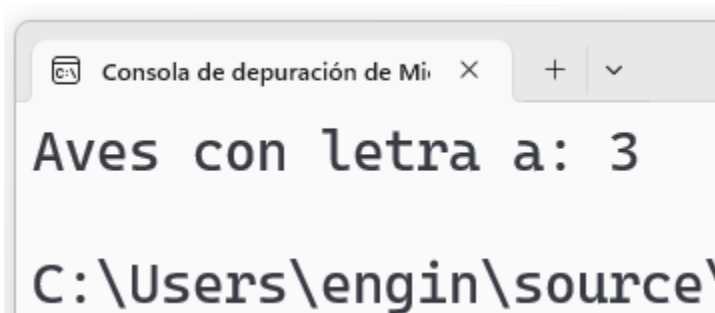


Ilustración 9: LINQ: Consulta con elementos tipo string

Consulta con objetos. Ejemplo 1.

F/010.cs

```
namespace Ejemplo;

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

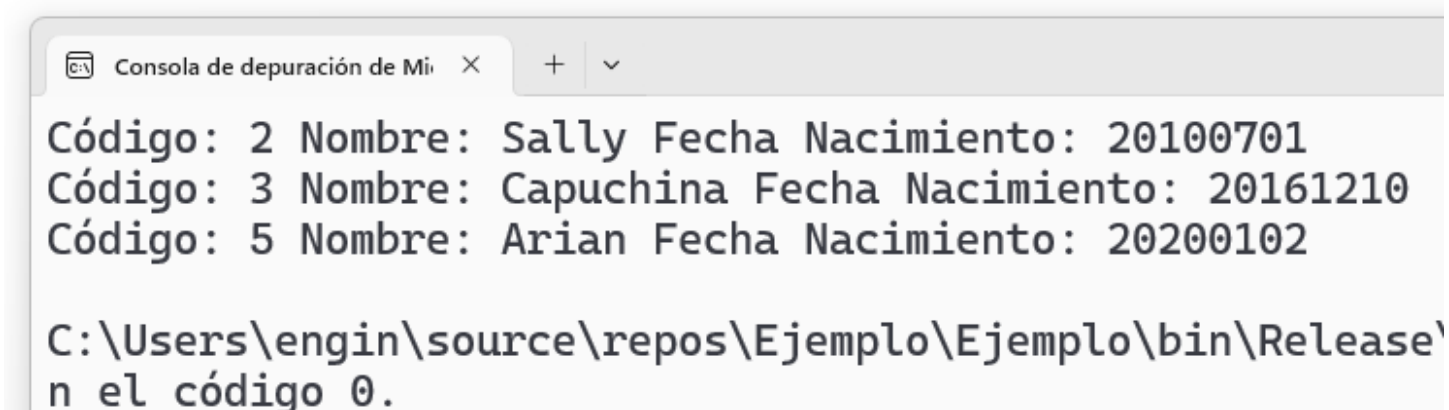
    public Mascota(intCodigo, string Nombre, int FechaNace) {
        this.Codigo = Codigo;
        this.Nombre = Nombre;
        this.FechaNace = FechaNace;
    }

    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 screenshot shows a Visual Studio debug console window titled 'Consola de depuración de Mi'. It displays the output of a LINQ query, which consists of three lines of text, each representing an object with three properties: Código, Nombre, and Fecha Nacimiento. The first line is 'Código: 2 Nombre: Sally Fecha Nacimiento: 20100701', the second is 'Código: 3 Nombre: Capuchina Fecha Nacimiento: 20161210', and the third is 'Código: 5 Nombre: Arian Fecha Nacimiento: 20200102'. Below these lines, there is a message: 'C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\'n el código 0.'

```
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. Ejemplo 2.

F/011.cs

```
namespace Ejemplo;

internal class Mascota {
    public intCodigo { get; set; }
    public string Nombre { get; set; }
    public int FechaNace { get; set; } //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", 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();
        }
    }
}
```

}

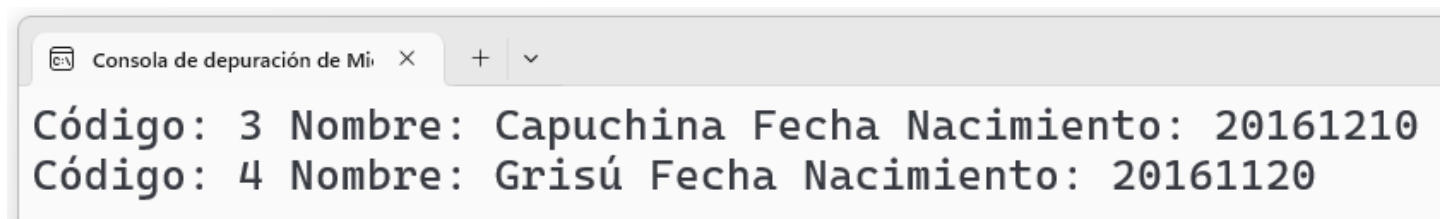


Ilustración 11: LINQ: Consulta con objetos y resultado en un List

Determinación de tipo de dato

F/012.cs

```
using System.Collections;

namespace Ejemplo;

internal class Program {
    static void Main() {
        ArrayList Varios =
        [
            1822,
            'M',
            true,
            639.9,
            "Rafael",
            6094,
            'J',
            false,
            55.5,
            "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]);  
    }  
}  
}
```

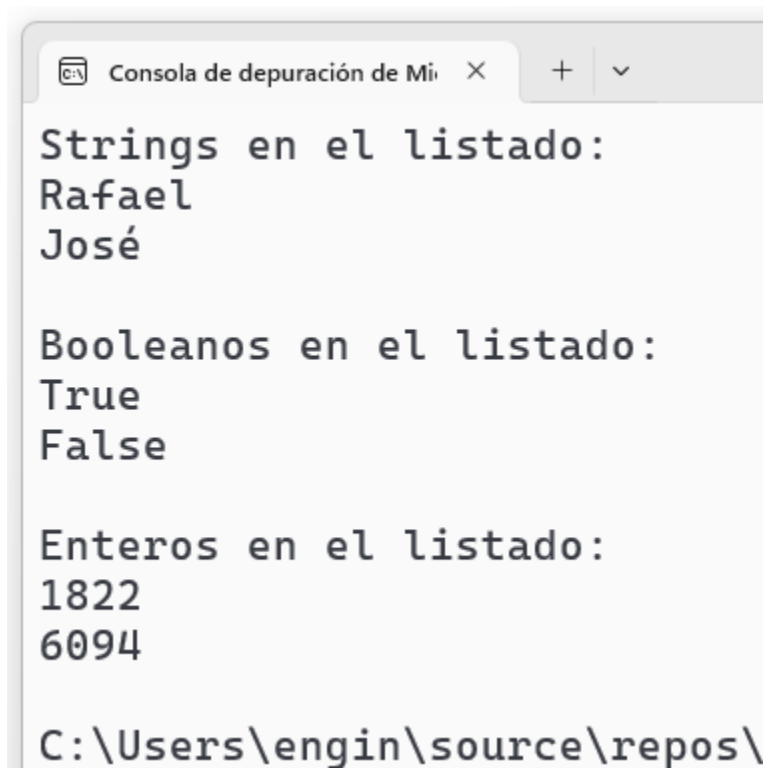


Ilustración 12: LINQ: Determinación de tipo de dato

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;
    }
}

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

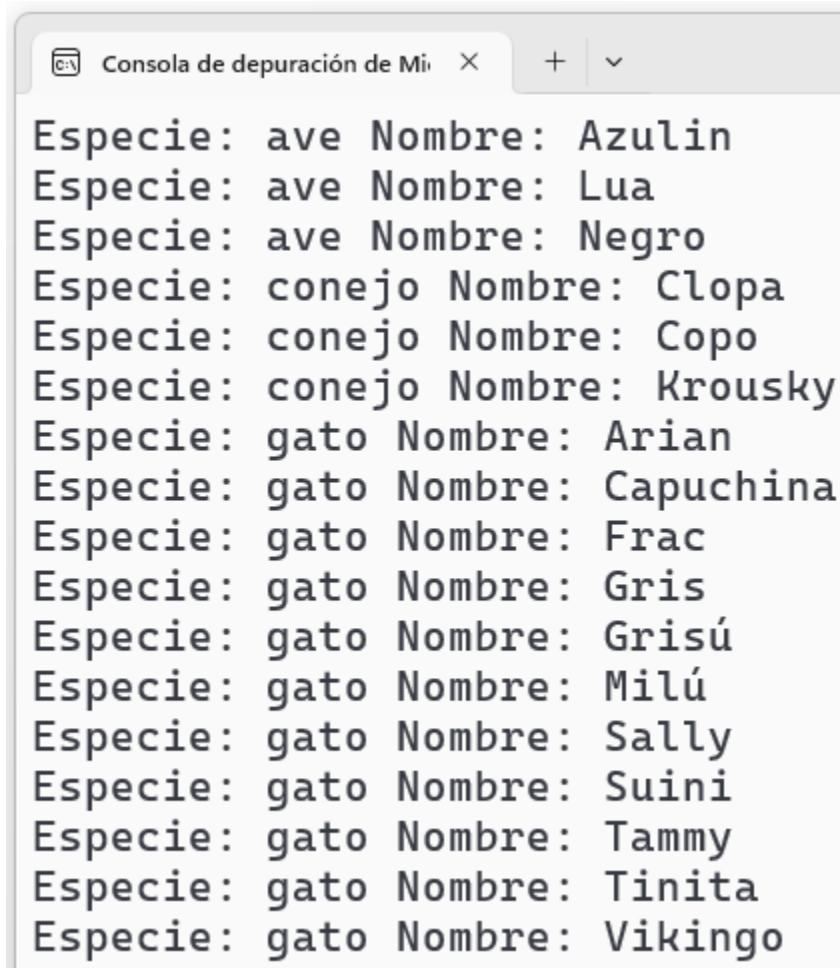
        var ConjuntoGrupos = from animal in listaMascotas
                              group animal by animal.Especie;

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

            // Cada grupo tiene una colección interna
            foreach (Mascota individuo in grupo)
```



```
        Console.WriteLine("Nombre: " + individuo.Nombre);  
    }  
}
```



```
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

Agrupación por un campo

F/014.cs

```
namespace Ejemplo;

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

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

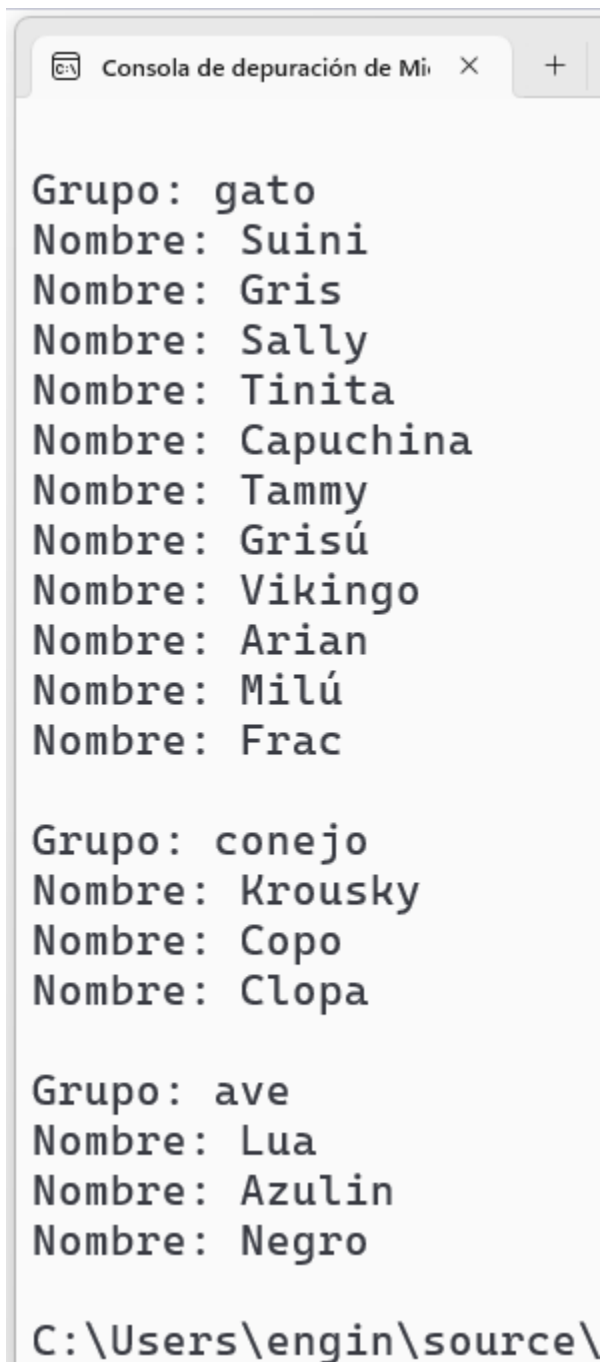
internal class Program {
    static void Main() {
        List<Especie> listaEspecies =
        [
            new Especie(1, "Gato"),
            new Especie(2, "Conejo"),
            new Especie(3, "Ave"),
        ];

        List<Mascota> listaMascotas =
        [
            new Mascota(1, "Suini"),
            new Mascota(1, "Gris"),
            new Mascota(1, "Sally"),
            new Mascota(1, "Tinita"),
            new Mascota(2, "Krousky"),
            new Mascota(1, "Capuchina"),
            new Mascota(1, "Tammy"),
            new Mascota(1, "Grisú"),
            new Mascota(3, "Lua"),
            new Mascota(2, "Copo"),
            new Mascota(1, "Vikingo"),
            new Mascota(1, "Arian"),
            new Mascota(1, "Milú"),
            new Mascota(3, "Azulin"),
            new Mascota(1, "Frac"),
            new Mascota(3, "Negro"),
            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.Write("La mascota " + item.Mascota);
        Console.WriteLine(" es de la especie " + item.Especie);
    }
}
```



The image shows a screenshot of a Visual Studio debug console window titled "Consola de depuración de Mi...". The console displays the results of a LINQ query, grouped by animal type. The output is as follows:

```
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(intCodigo, string Nombre) {
    public int Codigo { get; set; } = Codigo;
    public string Nombre { get; set; } = Nombre;
}

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

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

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

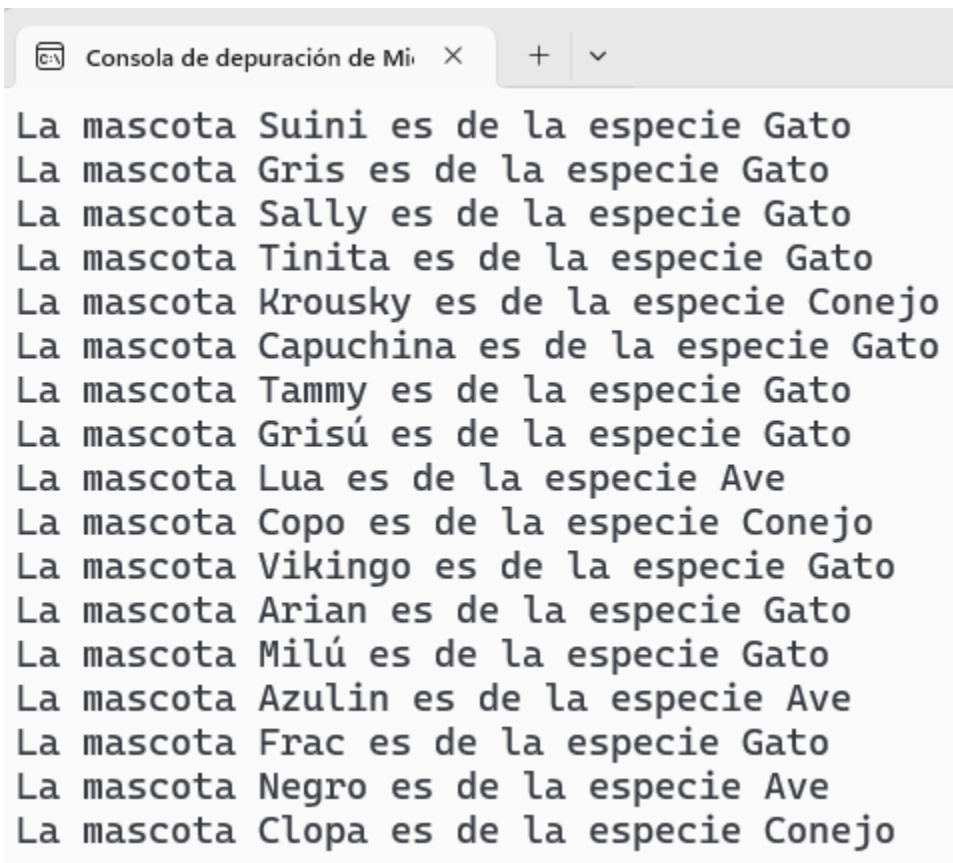
        var Consulta = from mascota in listaMascotas
                        join especie in listaEspecies
```

```

        on mascota.Especie equals especie.Codigo
        select new {
            Especie = especie.Nombre,
            Mascota = mascota.Nombre
        };

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

```



```

La mascota Suini es de la especie Gato
La mascota Gris es de la especie Gato
La mascota Sally es de la especie Gato
La mascota Tinita es de la especie Gato
La mascota Krousky es de la especie Conejo
La mascota Capuchina es de la especie Gato
La mascota Tammy es de la especie Gato
La mascota Grisú es de la especie Gato
La mascota Lua es de la especie Ave
La mascota Copo es de la especie Conejo
La mascota Vikingo es de la especie Gato
La mascota Arian es de la especie Gato
La mascota Milú es de la especie Gato
La mascota Azulin es de la especie Ave
La mascota Frac es de la especie Gato
La mascota Negro es de la especie Ave
La mascota Clopa es de la especie Conejo

```

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

Un "join" con resultado personalizado

F/016.cs

```
namespace Ejemplo;

internal class Especie(intCodigo, string Nombre) {
    public int Codigo { get; set; } = Codigo;
    public string Nombre { get; set; } = Nombre;
}

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

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

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

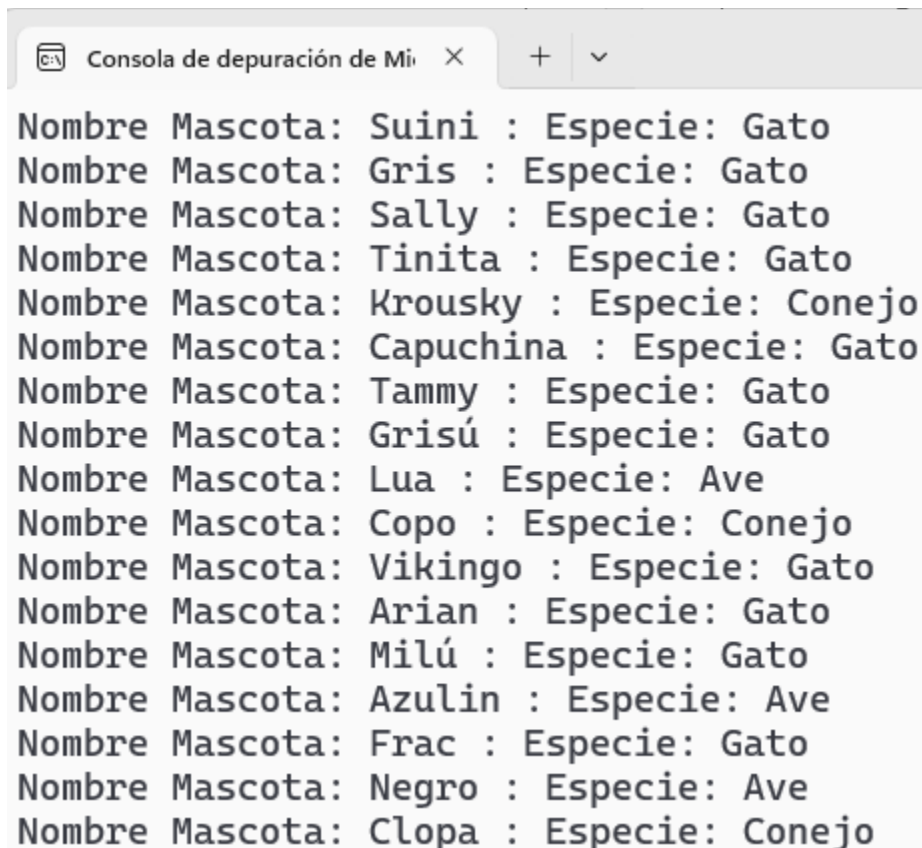
        var Consulta = from mascota in listaMascotas
                        join especie in listaEspecies
```

```

        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: Azulín : 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 = ["Gato", "Condor", "Perro", "Conejo",
                                "Loro"];

        List<string> Mamiferos = ["Perro", "Conejo", "Gato"];

        /* Extrae los animales que no están en mamíferos */
        List<string> Resultado = Animales.Except(Mamiferos).ToList();

        for (int Cont = 0; Cont < Resultado.Count; Cont++) {
            Console.WriteLine(Resultado[Cont]);
        }
    }
}
```

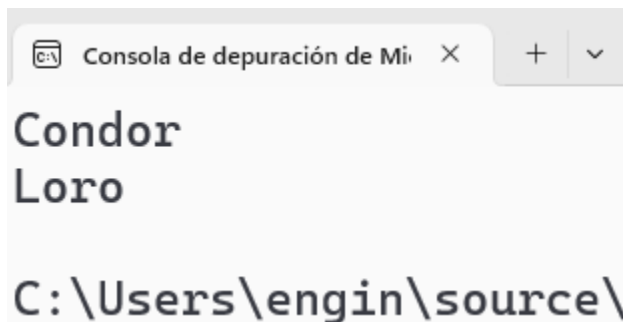


Ilustración 17: LINQ: Extraer los datos de una lista que no están en otra

Intersección de dos listas

F/018.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        List<string> ColoresA = ["Azul", "Rojo", "Verde", "Violeta"];

        List<string> ColoresB = ["Violeta", "Azul", "Marrón"];

        /* Intersecta los animales que están en ambas listas */
        List<string> Resultado = ColoresA.Intersect(ColoresB).ToList();

        for (int Cont = 0; Cont < Resultado.Count; Cont++) {
            Console.WriteLine(Resultado[Cont]);
        }
    }
}
```

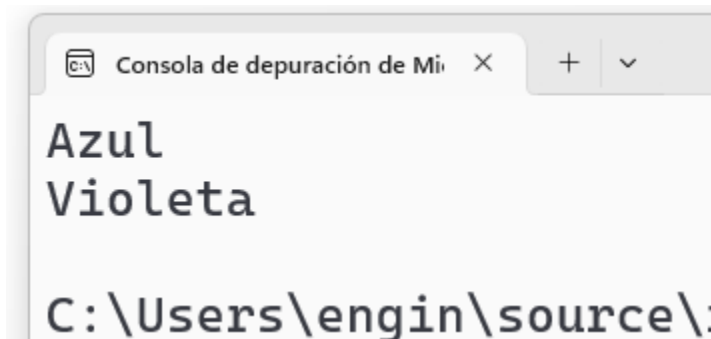


Ilustración 18: LINQ: Intersección de dos listas

Unir dos listas sin repetir elementos

F/019.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        List<string> ColoresA = ["Azul", "Rojo", "Verde", "Marrón",
"Violeta"];

        List<string> ColoresB = ["Violeta", "Azul", "Marrón", "Naranja"];

        /* Une los valores de ambas listas evitando repetir */
        List<string> Resultado = ColoresA.Union(ColoresB).ToList();

        for (int Cont = 0; Cont < Resultado.Count; Cont++) {
            Console.WriteLine(Resultado[Cont]);
        }
    }
}
```



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"
        List<string> PalabraMinusculaA = (from palabra in Textos
                                           where palabra.ToLower().StartsWith("a")
                                           select palabra.ToLower()).ToList();

        for (int Cont = 0; Cont < PalabraMinusculaA.Count; Cont++)
            Console.WriteLine(PalabraMinusculaA[Cont]);

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

        for (int Cont = 0; Cont < PalabraMinusculaB.Count; Cont++)
            Console.WriteLine(PalabraMinusculaB[Cont]);
    }
}
```

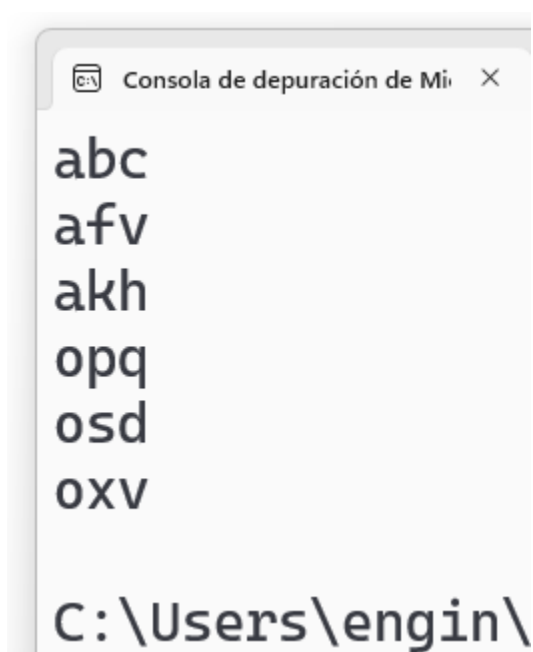


Ilustración 20: LINQ: Consulta de texto por algún patrón

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. */
        List<char> Resultado = Cadena.OrderBy(str => str).ToList();
        Console.WriteLine("Arreglo con las letras ordenadas");
        for (int Cont = 0; Cont < Resultado.Count; Cont++)
            Console.Write(Resultado[Cont]);

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

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

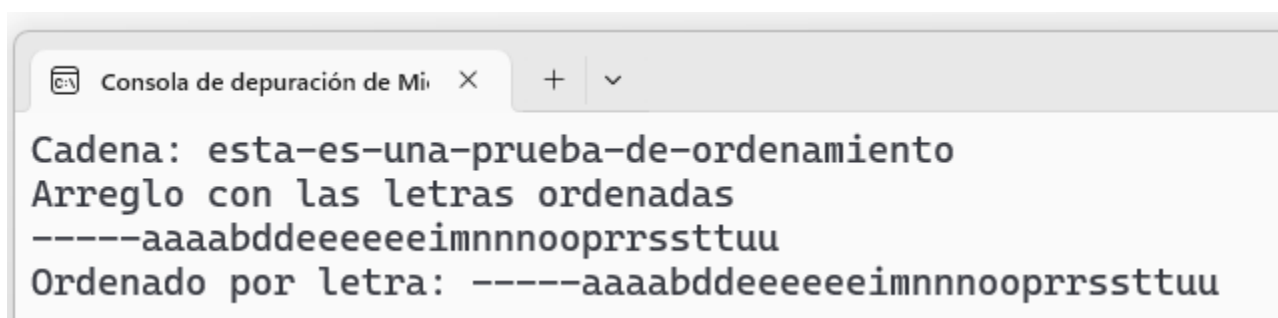


Ilustración 21: LINQ: Ordenar internamente una cadena

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. */
        List<char> Resultado = Cadena.OrderBy(str => str).ToList();

        Console.WriteLine("Arreglo con las letras ordenadas");
        for (int Cont = 0; Cont < Resultado.Count; Cont++)
            Console.Write "[" + Resultado[Cont] + " ] ");

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

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

```
Consola de depuración de Mi X + v - □ X
Cadena: El-ñandú-es-un-ave-de-Sudamérica.-La-cigüeña-blanca-es-una-esp
ecie-de-ave-Ciconiiforme-de-gran-tamaño.
Arreglo con las letras ordenadas
[-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-]
[.] [.] [C] [E] [L] [S] [a] [a] [a] [a] [a] [a] [a] [a] [a] [a] [a] [a]
[a] [a] [b] [c] [c] [c] [c] [c] [d] [d] [d] [d] [d] [e] [e] [e] [e] [e]
[e] [e] [e] [e] [e] [e] [e] [f] [g] [g] [i] [i] [i] [i] [i] [i] [l]
[l] [m] [m] [m] [n] [n] [n] [n] [n] [n] [o] [o] [o] [p] [r] [r] [r] [s]
[s] [s] [t] [u] [u] [u] [v] [v] [é] [ñ] [ñ] [ñ] [ú] [ü]
Ordenado por letra: -----..CELSaaaaaaaaaaaaabccccddddddee
eeeeeeeeefggiiiiiiillmmnnnnnnnooprrrrsstuuuvvéñññüü
```

Ilustración 22: LINQ: Ordenar internamente una cadena

Ordenamiento según tamaño de la palabra

F/023.cs

```
namespace Ejemplo;

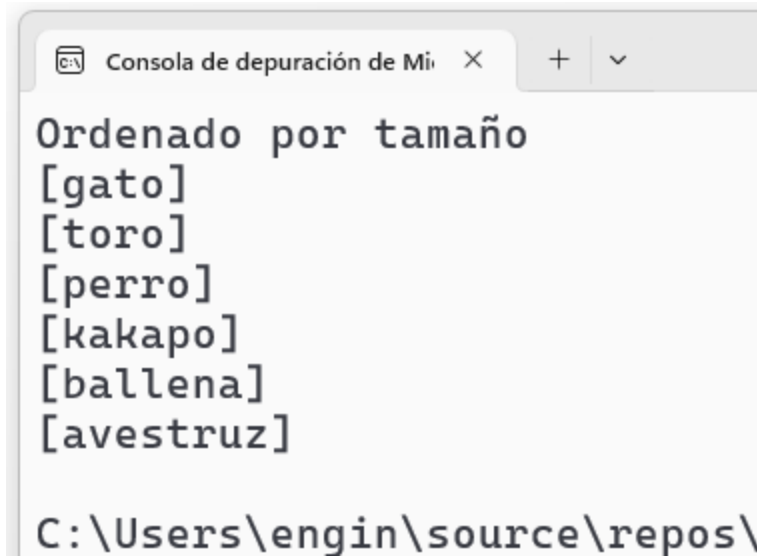
internal class Program {
    static void Main() {
        //Una cadena
        string Cadena = "El-ñandú-es-un-ave-de-Sudamérica.";
        Cadena += "-La-cigüeña-blanca-es-una-especie-de-ave";
        Cadena += "-Ciconiiforme-de-gran-tamaño.";
        Console.WriteLine("Cadena: " + Cadena);

        /* 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. */
        List<char> Resultado = Cadena.OrderBy(str => str).ToList();

        Console.WriteLine("Arreglo con las letras ordenadas");
        for (int Cont = 0; Cont < Resultado.Count; Cont++)
            Console.Write "[" + Resultado[Cont] + " ] ");

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

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



The image shows a screenshot of a Visual Studio console window. The title bar at the top reads 'Consola de depuración de Mi' followed by a close button and a dropdown menu with a plus sign and a downward arrow. The console output is as follows:

```
Ordenado por tamaño  
[gato]  
[toro]  
[perro]  
[kakapo]  
[ballena]  
[avestruz]  
  
C:\Users\engin\source\repos\
```

Ilustración 23: LINQ: Ordenamiento según tamaño de la palabra

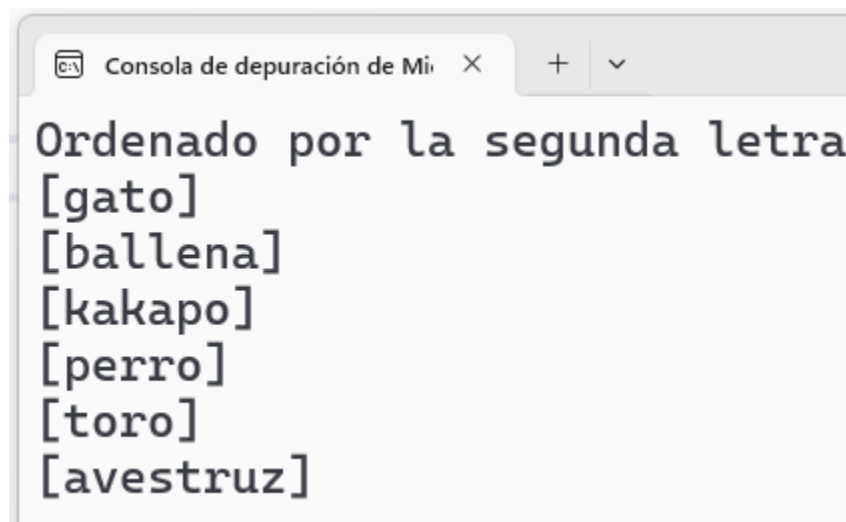
Ordenamiento por la segunda letra de cada palabra

F/024.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Una lista de cadenas
        List<string> Cadenas = [ "gato", "perro", "avestruz", "toro",
                                "ballena", "kakapo" ];

        /* Se ordena usando LINQ
         * Ordena por la segunda letra. */
        List<string> Resultado = Cadenas.OrderBy(str => str[1]).ToList();
        Console.WriteLine("Ordenado por la segunda letra");
        for (int Cont = 0; Cont < Resultado.Count; Cont++)
            Console.WriteLine "[" + Resultado[Cont] + " ] ";
    }
}
```



```
Consola de depuración de Mi...
Ordenado por la segunda letra
[gato]
[ballena]
[kakapo]
[perro]
[toro]
[avestruz]
```

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

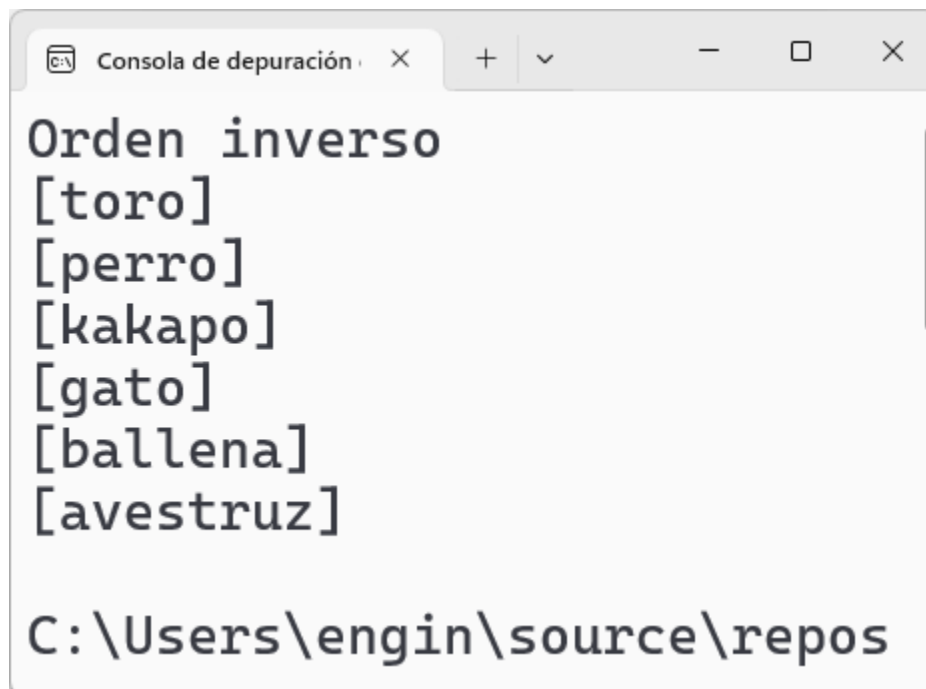
Invertir el ordenamiento

F/025.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        //Una lista de cadenas
        List<string> Cadenas = [ "gato", "perro", "avestruz", "toro",
                                "ballena", "kakapo" ];

        /* Se ordena usando LINQ
           * Ordena por la segunda letra. */
        List<string> Resultado = Cadenas.OrderBy(str => str).ToList();
        Resultado.Reverse();
        Console.WriteLine("Orden inverso");
        for (int Cont = 0; Cont < Resultado.Count; Cont++)
            Console.WriteLine "[" + Resultado[Cont] + " ] ";
    }
}
```



```
Consola de depuración
Orden inverso
[toro]
[perro]
[kakapo]
[gato]
[ballena]
[avestruz]
C:\Users\engin\source\repos
```

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() {
#ifdef DEBUG
        Console.WriteLine("Modo DEBUG detectado. Las pruebas se deben hacer en RELEASE");
        Environment.Exit(0);
#endif

        Console.WriteLine("Comparativa LINQ vs Programación Clásica\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();
    }
}
```

```

ResultadosLINQ =
    (from numero in Enteros
     where (numero % 2) == 1
     select numero).ToList();

long TiempoLINQ = cronometro.ElapsedMilliseconds;

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

//Compara ambas listas
for (int Cont = 0; Cont < ResultadosLINQ.Count; Cont++) {
    if (ResultadosLINQ[Cont] != ResultadosNOLINQ[Cont])
        Console.WriteLine("Fallo en el proceso");
}

Console.WriteLine(" ");
}
}

```

```
Consola de depuración de Mi  X  +  v  -  □  X

Comparativa LINQ vs Programación Clásica

Tiempo LINQ (ms): 7  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 3
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 3
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 3
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 3  Tiempo NO LINQ (ms): 2
Tiempo LINQ (ms): 3  Tiempo NO LINQ (ms): 3
Tiempo LINQ (ms): 3  Tiempo NO LINQ (ms): 3
Tiempo LINQ (ms): 3  Tiempo NO LINQ (ms): 3
Tiempo LINQ (ms): 3  Tiempo NO LINQ (ms): 3
Tiempo LINQ (ms): 4  Tiempo NO LINQ (ms): 3
Tiempo LINQ (ms): 2  Tiempo NO LINQ (ms): 3

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net10.0\Ejemplo.exe (proceso 21860) se cerró con el código 0 (0x0).
```

Ilustración 26: Comparativa desempeño de LINQ

Como se puede observar, LINQ se desempeña muy similar a la programación clásica.

Expresiones LAMBDA

Una expresión lambda en C# es una forma concisa de escribir funciones anónimas. Se utilizan principalmente para simplificar el código cuando se trabaja con LINQ.

F/027.cs

```
namespace Ejemplo;

internal class Program {
    static void Main(string[] args) {
        //=====
        //Expresión lambda
        //=====
        var AlCubo = (int valor) => valor * valor * valor;

        //Uso de la expresión lambda
        int Resultado = AlCubo(13);
        Console.WriteLine("Elevado al cubo: " + Resultado);
    }
}
```

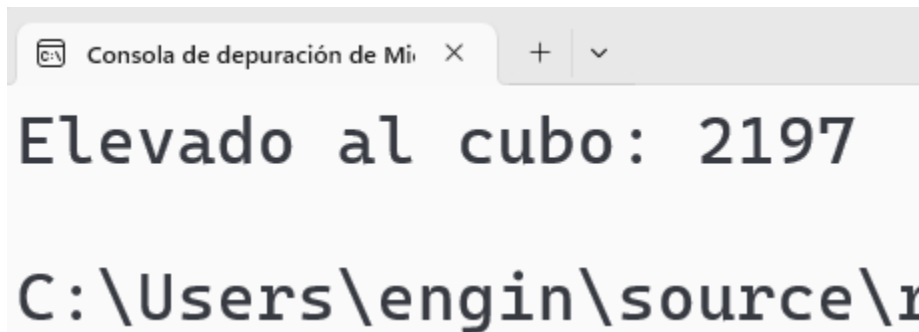


Ilustración 27: Ejecuta expresión Lambda

Expresión Lambda: Uso de varias líneas

F/028.cs

```
namespace Ejemplo;

internal class Program {
    static void Main(string[] args) {
        //=====
        //Expresión Lambda: Uso del return y { } porque hay
        //más de dos líneas
        //=====
        var AreaTriangulo = (double ladoA, double ladoB, double ladoC) => {
            double s = (ladoA + ladoB + ladoC) / 2;
            return Math.Sqrt(s * (s - ladoA) * (s - ladoB) * (s - ladoC));
        };

        double Area = AreaTriangulo(3, 4, 5);
        Console.WriteLine("Área del triángulo es: " + Area);
    }
}
```

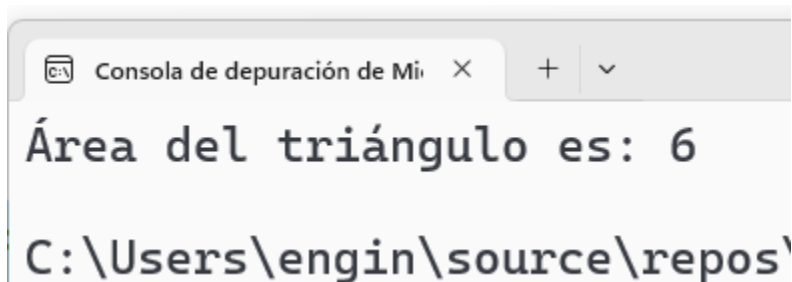


Ilustración 28: Expresiones Lambda usando varias líneas

Expresión Lambda: Retornando tipo de dato

F/029.cs

```
namespace Ejemplo;

internal class Program {
    static void Main(string[] args) {
        //=====
        //Expresion lambda retornando el tipo de valor
        //=====
        Func<int, long> Factorial = (int numero) => {
            long acumula = 1;
            for (int cont = 1; cont <= numero; cont++) {
                acumula *= cont;
            }
            return acumula;
        };
        long Probar = Factorial(5);
        Console.WriteLine("Factorial es: " + Probar);
    }
}
```

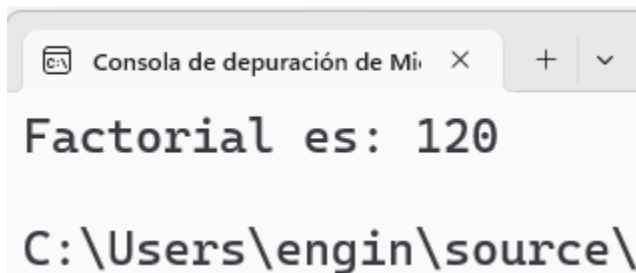


Ilustración 29: Retornando tipo de dato

Expresión Lambda: Acceso a variable externa

F/030.cs

```
namespace Ejemplo;

internal class Program {
    static void Main(string[] args) {
        //=====
        //Acceso a variable externa a la expresión lambda
        //=====
        double valorExterno = 12.345;

        //Expresión lambda haciendo uso de un valor externo
        var UnCalculo = (double Parametro) => Parametro * valorExterno;

        //Se usa la expresión
        double Resulta = UnCalculo(5);
        Console.WriteLine("Resultado es: " + Resulta);
    }
}
```

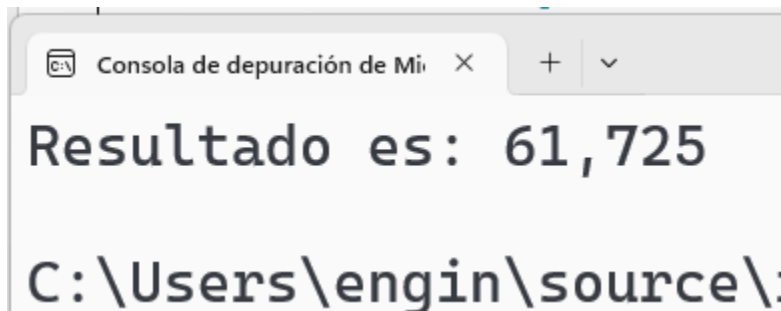


Ilustración 30: Acceso a variable externa

Expresión Lambda: Listas. Pero no funciona.

F/031.cs

```
namespace Ejemplo;

internal class Program {
    static void Main(string[] args) {
        //=====
        //Listas y variables externas
        //=====

        //Una lista de operaciones Lambda
        var Operaciones = new List<Func<int, int>>();

        //Agrega items a la lista
        for (int Cont = 5; Cont <= 10; Cont++) {
            Operaciones.Add((int Valor) => Valor + Cont);
        }

        //Pero no funciona porque no hace la operación
        foreach (var operacion in Operaciones) {
            int Resultados = operacion(300);
            Console.WriteLine(Resultados);
        }
    }
}
```

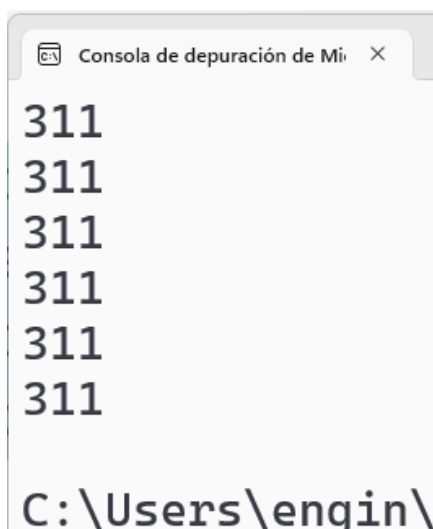


Ilustración 31: No funciona

Expresión Lambda: Listas. Ahora si funciona.

F/032.cs

```
namespace Ejemplo;

internal class Program {
    static void Main(string[] args) {
        //=====
        //Listas y variables externas
        //=====

        //Una lista de operaciones Lambda
        var Operaciones = new List<Func<int, int>>();

        //Agrega items a la lista, declara internamente a Suma
        for (int Cont = 5; Cont <= 10; Cont++) {
            int Suma = Cont;
            Operaciones.Add((int Valor) => Valor + Suma);
        }

        //En este caso SI funciona
        foreach (var operacion in Operaciones) {
            int Resultados = operacion(300);
            Console.WriteLine(Resultados);
        }
    }
}
```



Ilustración 32: Ahora si funciona

Expresiones LAMBDA: Listas y variables externas no funciona

F/033.cs

```
namespace Ejemplo;

internal class Program {
    static void Main(string[] args) {
        //=====
        //Listas y variables externas
        //=====

        //Una lista de operaciones Lambda
        var Operaciones = new List<Func<int, int>>();

        //Agrega items a la lista, declara externamente Suma
        int Suma;
        for (int Cont = 5; Cont <= 10; Cont++) {
            Suma = Cont;
            Operaciones.Add((int Valor) => Valor + Suma);
        }

        //En este caso NO va a funcionar
        foreach (var operacion in Operaciones) {
            int Resultados = operacion(300);
            Console.WriteLine(Resultados);
        }
    }
}
```

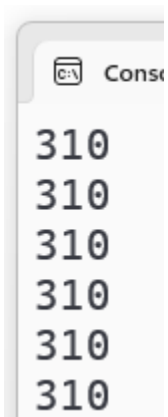


Ilustración 33: No funciona

Expresiones LAMBDA: Uso de static

F/034.cs

```
namespace Ejemplo;

internal class Program {
    static void Main(string[] args) {
        int ValorExterno = 13;

        /* El uso de static mejora el desempeño de la expresión lambda
         * pero ya NO se puede hacer uso de variables externas */
        var UnCalculo = static (int Numero) => Numero * ValorExterno;

        int Resultado = UnCalculo(17);
        Console.WriteLine(Resultado);
    }
}
```

Numero * ValorExterno;



(variable local) `int ValorExterno`

[Describir con Copilot](#)

CS8820: Una función anónima estática no puede contener una referencia a "ValorExterno".

[Mostrar posibles correcciones](#) (Alt+Entrar o Ctrl+.)

Ilustración 34: Error al tratar de usar una variable externa

Comparativa entre LINQ y Lambda. Ejemplo 1.

F/035.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, 2];

        Console.WriteLine("Consulta usando LINQ");
        List<int> Resultados = (from numero in Lista
                               where numero > 5
                               select numero).ToList();

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

        //Hacer lo mismo con una expresión lambda
        Console.WriteLine("\r\nConsulta usando Lambda");
        List<int> Datos = Lista.Where(x => x > 5).ToList();
        foreach (int UnValor in Datos) Console.Write(UnValor + ", ");
    }
}
```

Ambas instrucciones hacen lo mismo, salvo que con Lambda requiere menos líneas de código.

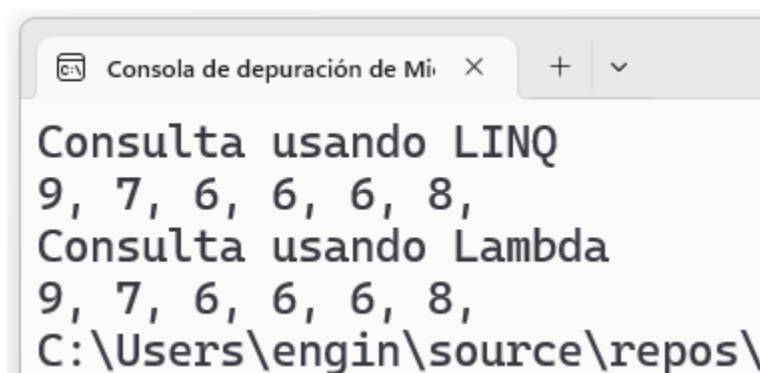


Ilustración 35: Comparativa LINQ y Lambda

Comparativa entre LINQ y Lambda. Ejemplo 2.

F/036.cs

```
namespace Ejemplo;

internal class Program {
    static int CalcularCuadrado(int num) {
        return num * num;
    }

    static void Main() {
        // Crear una lista de enteros
        List<int> numeros = [3, 1, 2, 7, 6, 9];

        // Usar LINQ con un método en lugar de expresiones lambda
        List<int> cuadrados = numeros.Select(CalcularCuadrado).ToList();

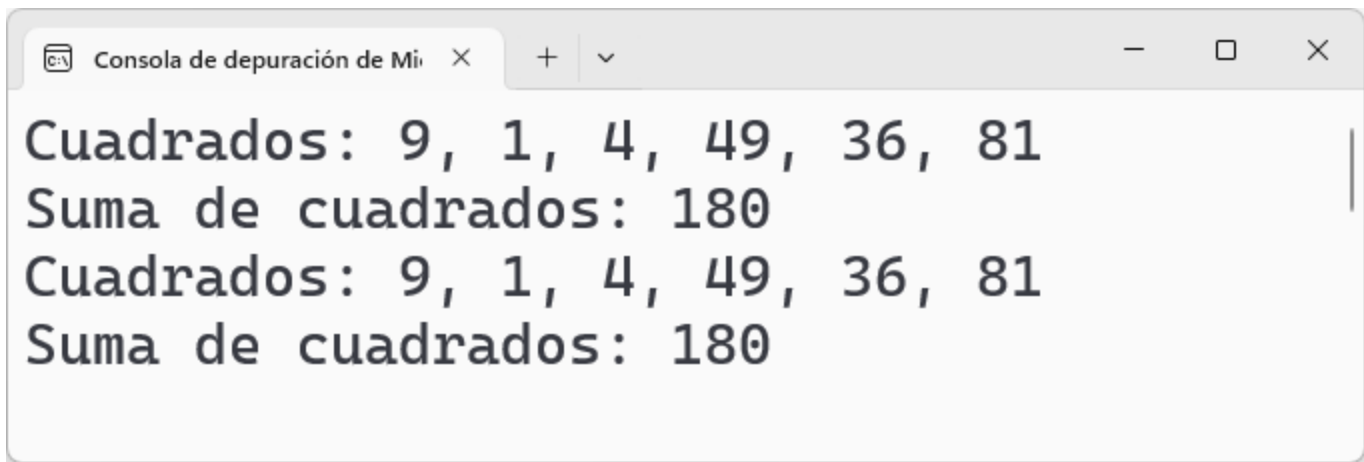
        // Usar LINQ para sumar los cuadrados
        int sumaCuadrados = cuadrados.Sum();

        // Mostrar los resultados
        Console.WriteLine("Cuadrados: " + string.Join(", ", cuadrados));
        Console.WriteLine("Suma de cuadrados: " + sumaCuadrados);

        // Usar LINQ con expresión Lambda
        var CuadradoLINQ = numeros.Select(num => num * num);

        // Usar LINQ para sumar todos los cuadrados
        int SumaCuadradosLINQ = CuadradoLINQ.Sum();

        // Mostrar los resultados
        Console.WriteLine("Cuadrados: " + string.Join(", ", CuadradoLINQ));
        Console.WriteLine("Suma de cuadrados: " + SumaCuadradosLINQ);
    }
}
```



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). Below the title bar, there are four lines of text output from a program. The first two lines are 'Cuadrados: 9, 1, 4, 49, 36, 81' and 'Suma de cuadrados: 180'. The next two lines are identical: 'Cuadrados: 9, 1, 4, 49, 36, 81' and 'Suma de cuadrados: 180'. The text is displayed in a monospaced font.

```
Consola de depuración de Mi X + v - □ X  
Cuadrados: 9, 1, 4, 49, 36, 81  
Suma de cuadrados: 180  
Cuadrados: 9, 1, 4, 49, 36, 81  
Suma de cuadrados: 180
```

Ilustración 36: Uso de LINQ con y sin expresiones Lambda

Comparativa entre LINQ y Lambda. Ejemplo 3.

Se presentan dos programas que hacen lo mismo, en uno hace uso de expresiones Lambda con LINQ y el otro sin expresiones Lambda.

Usando expresiones Lambda

F/037a.cs

```
namespace Ejemplo;

internal class Program {
    static void Main() {
        // Crear la lista de enteros
        List<int> numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

        // Agrupar usando LINQ con expresiones lambda
        var Agrupados = numeros.GroupBy(num => num % 2 == 0 ? "Pares" :
"Impares");

        // Mostrar los resultados
        foreach (var grupo in Agrupados) {
            Console.WriteLine($"{grupo.Key}: {string.Join(", ", grupo)}");
        }
    }
}
```

```
namespace Ejemplo;

internal class Program {
    // Método para clasificar números como "Pares" o "Impares"
    static string ClasificarParidad(int num) {
        return num % 2 == 0 ? "Pares" : "Impares";
    }

    static void Main() {
        // Crear la lista de enteros
        List<int> numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

        // Agrupar usando LINQ y un método en lugar de lambda
        var agrupados = numeros.GroupBy(ClasificarParidad);

        // Mostrar los resultados
        foreach (var grupo in agrupados) {
            Console.WriteLine($"{grupo.Key}: {string.Join(", ", grupo)}");
        }
    }
}
```

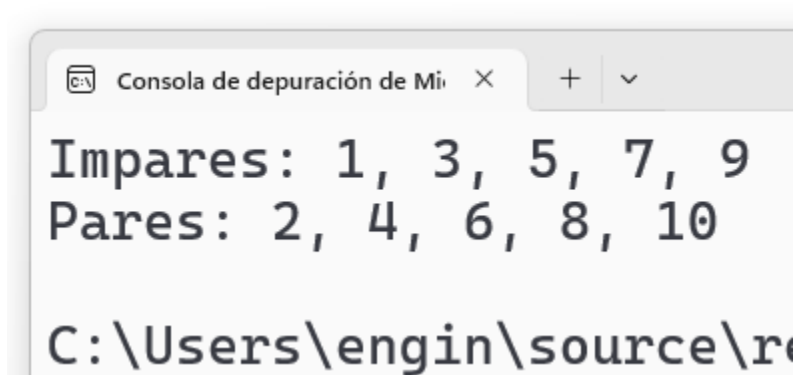


Ilustración 37: Mismos resultados

Expresiones Lambda: Promedio

F/038.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, 2, 9, 2, 9];

        //Promedio
        double Promedio = Lista.Where(x => x % 2 == 1).Average();
        Console.WriteLine("Promedio valores impares: " + Promedio);
    }
}
```

Consola de depuración de Mi X + v

Promedio valores impares: 5,571428571428571
C:\Users\engin\source\repos\Ejemplo\Ejemplo\b:
se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando

Ilustración 38: Promedio

```
using System.Text.Json.Nodes;

//JSON y LINQ
namespace Ejemplo;

class Program {
    static void Main() {
        string EjemploJSON = @"
        {
            ""Linea"": {
                ""titulo"": ""Ejemplo de LINQ to JSON"",
                ""items"": [
                    { ""titulo"": ""Teclado Mc"", ""descripcion"": ""Teclado
mecánico"" },
                    { ""titulo"": ""Teclado Mb"", ""descripcion"": ""Teclado
de membrana"" },
                    { ""titulo"": ""Mouse Ina"", ""descripcion"": ""Mouse
inalámbrico"" }
                ]
            }
        }";

        // Parsear el JSON
        JsonNode rss = JsonNode.Parse(EjemploJSON);

        // Consultar las descripciones de los artículos usando LINQ
        var titulos = rss["Linea"]["items"]
            .ToArray()
            .Select(item => item["descripcion"].ToString());

        // Mostrar los títulos
        foreach (var titulo in titulos) {
            Console.WriteLine(titulo);
        }
    }
}
```

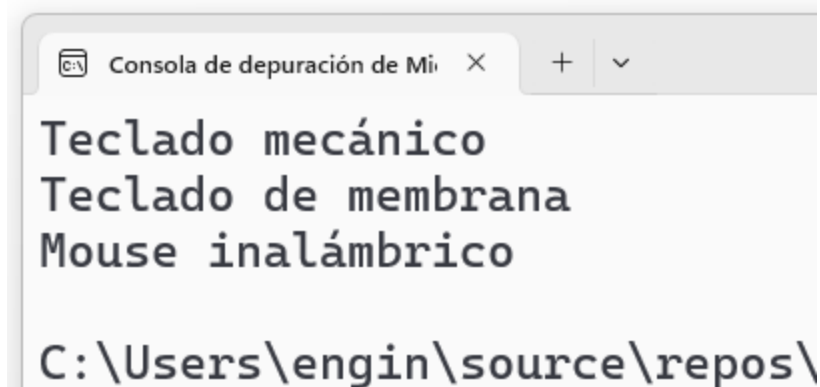


Ilustración 39: JSON y LINQ