

C# Y .NET 8

Parte 6. LINQ y LAMBDA

2024-12

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 arreglo.....	8
Filtrar y poner en un List	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.....	18
Consulta con objetos	19
Consulta con objetos y resultado en un List.....	21
Determinación de tipo de dato.....	23
Ordenación por un campo y luego por otro	25
Agrupación por un campo	27
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.....	40
Ordenamiento por la segunda letra de cada palabra.....	41
Invertir el ordenamiento.....	42
Métricas: Comparativa entre usar LINQ e implementación tradicional	43
Expresiones LAMBDA	46

Expresión Lambda: Uso de varias líneas.....	47
Expresión Lambda: Retornando tipo de dato.....	48
Expresión Lambda: Acceso a variable externa.....	49
Expresión Lambda: Listas. Pero no funciona.	50
Expresión Lambda: Listas. Ahora si funciona.....	51
Expresiones LAMBDA: Listas y variables externas no funciona	52
Expresiones LAMBDA: Uso de static	53
Comparativa entre LINQ y Lambda.....	54
Expresiones Lambda: Promedio	55
JSON y LINQ	56

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: LINQ: 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	18
Ilustración 10: LINQ: Consulta con objetos.....	20
Ilustración 11: LINQ: Consulta con objetos y resultado en un List	22
Ilustración 12: LINQ: Determinación de tipo de dato	24
Ilustración 13: LINQ: Ordenación por un campo y luego por otro	26
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	39
Ilustración 23: LINQ: Ordenamiento según tamaño de la palabra	40
Ilustración 24: LINQ: Ordenamiento por la segunda letra de cada palabra	41
Ilustración 25: LINQ: Invertir el ordenamiento	42
Ilustración 26: Comparativa desempeño de LINQ	45
Ilustración 27: Ejecuta expresión Lambda	46
Ilustración 28: Expresiones Lambda usando varias líneas	47
Ilustración 29: Retornando tipo de dato	48
Ilustración 30: Acceso a variable externa	49
Ilustración 31: No funciona	50
Ilustración 32: Ahora si funciona	51
Ilustración 33: No funciona	52
Ilustración 34: Error al tratar de usar una variable externa	53
Ilustración 35: Comparativa LINQ y Lambda	54
Ilustración 36: Promedio	55
Ilustración 37: JSON y LINQ	57

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

En memoria

De mi gato "Vikingo"



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() + ", ");
            }
        }
    }
}
```

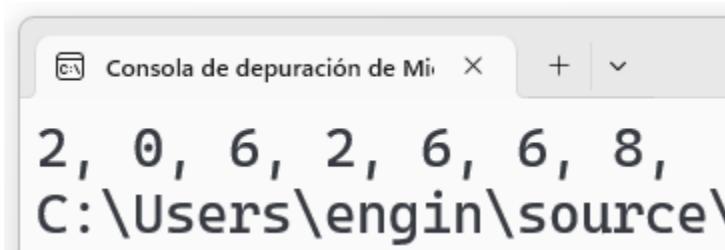


Ilustración 2: LINQ: Filtrar y poner en un List

```
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() + ", ");  
            }  
        }  
    }  
}
```

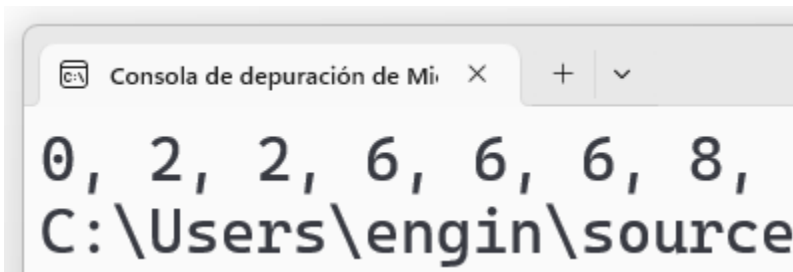


Ilustración 3: LINQ: Ordenación

```
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.WriteLine(Resultados[contador].ToString() + ", ");
            }
        }
    }
}
```

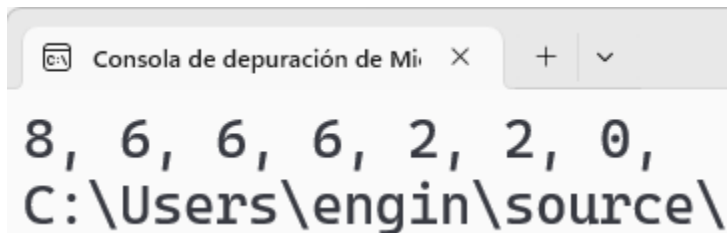


Ilustración 4: LINQ: Ordenación descendente

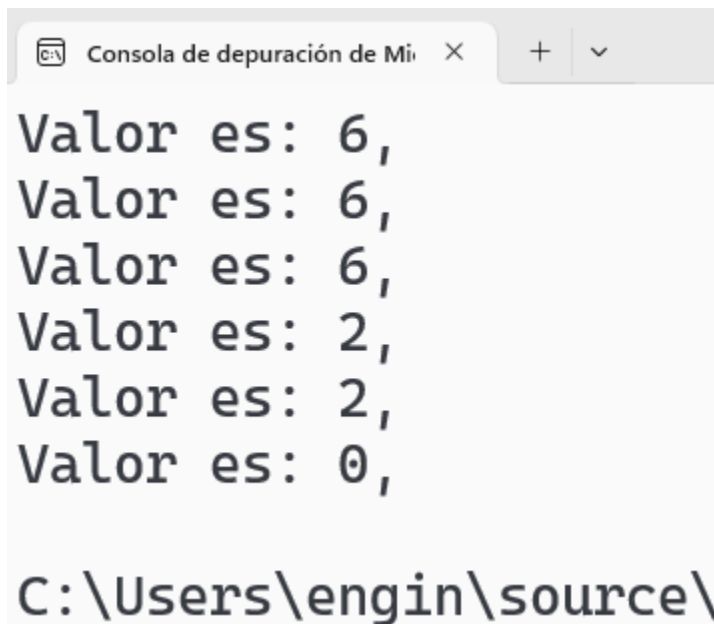
Consulta con salida a texto personalizado

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);  
        }  
    }  
}
```

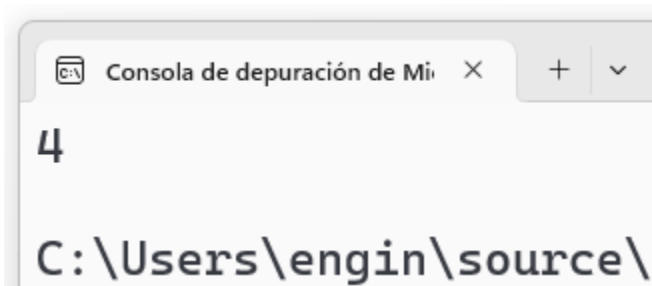


Ilustración 6: LINQ: Contar los registros

```
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();

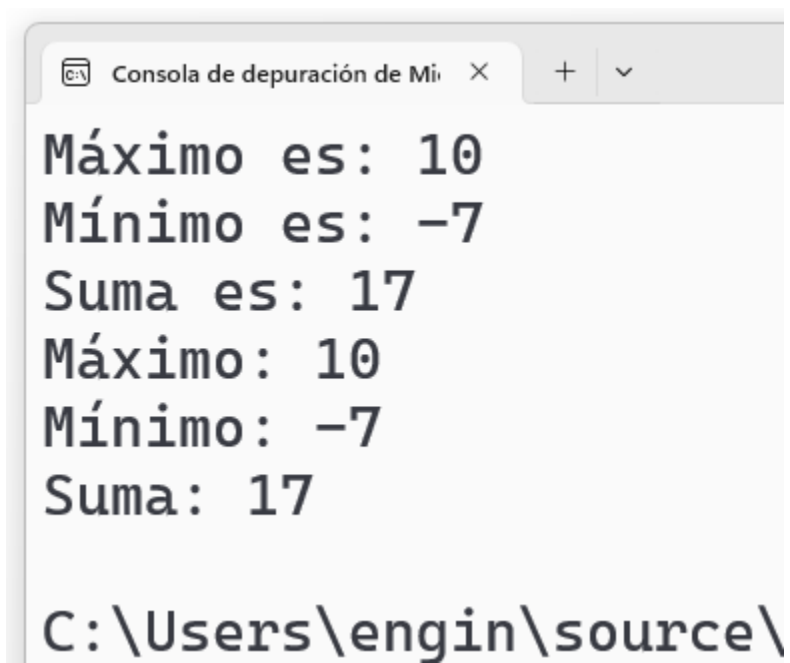
            int Minimo = (from numero in Lista
                          select numero).Min();

            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);
        }
    }
}
```



The image shows a screenshot of a Visual Studio debug console window. The title bar reads 'Consola de depuración de Mi' followed by a close button. The console contains the following output:

```
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);
        }
    }
}
```

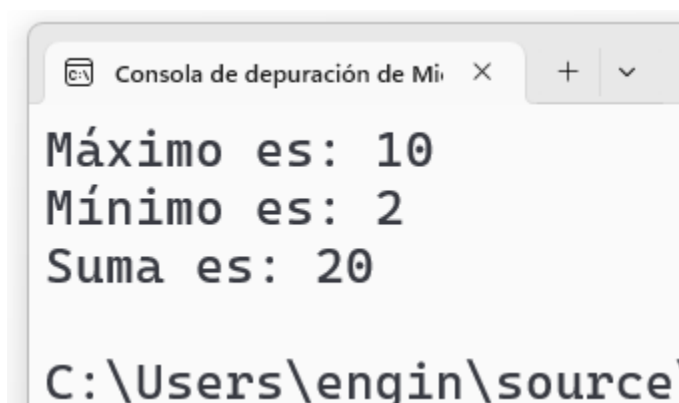


Ilustración 8: LINQ: Máximo, mínimo y suma con condiciones

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);  
        }  
    }  
}
```

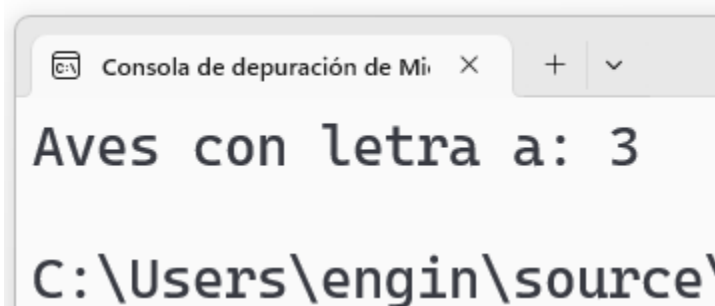


Ilustración 9: LINQ: Consulta con elementos tipo string

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

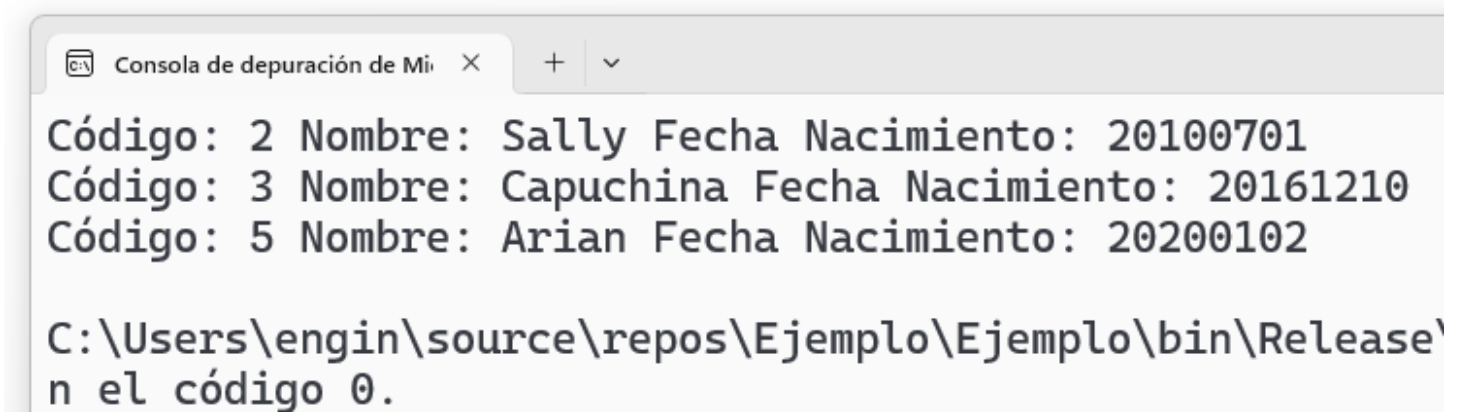
        public Mascota(intCodigo, string Nombre, int FechaNace) {
            this.Codigo = Codigo;
            this.Nombre = Nombre;
            this.FechaNace = FechaNace;
        }

        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'. Below the title bar, there are three lines of text representing query results: '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 message: '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 contains two lines of text, which are the results of 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 displayed 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");

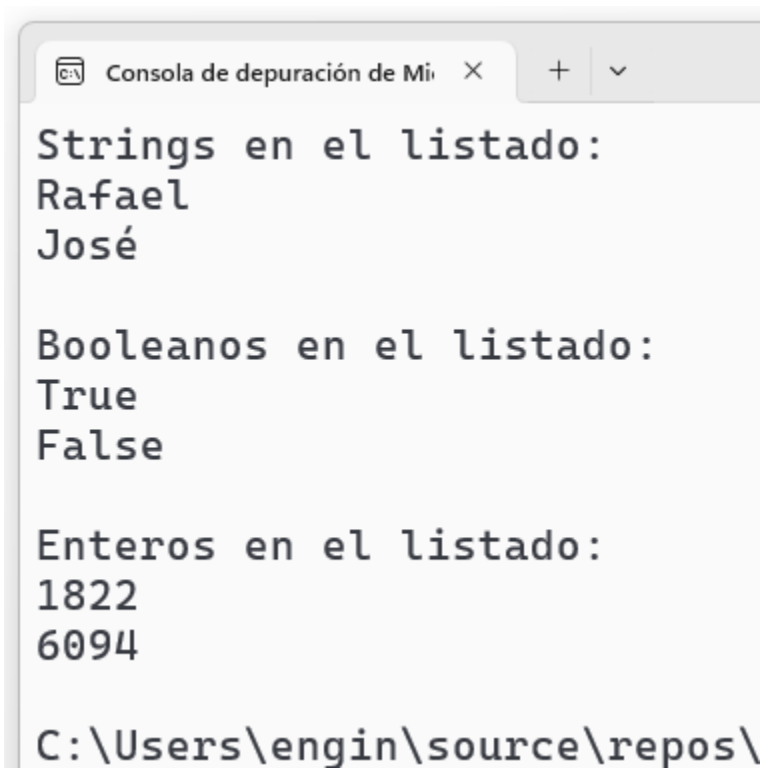
            Varios.Add(6094);
            Varios.Add('J');
            Varios.Add(false);
            Varios.Add(55.5);
            Varios.Add("José");

            //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 Visual Studio debug console window titled 'Consola de depuración de Mi'. It displays the output of a LINQ query. The output is organized into three sections: 'Strings en el listado:' with values 'Rafael' and 'José'; 'Booleanos en el listado:' with values 'True' and 'False'; and 'Enteros en el listado:' with values '1822' and '6094'. The console window has a tab bar at the top with a close button and a dropdown menu.

```
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"));

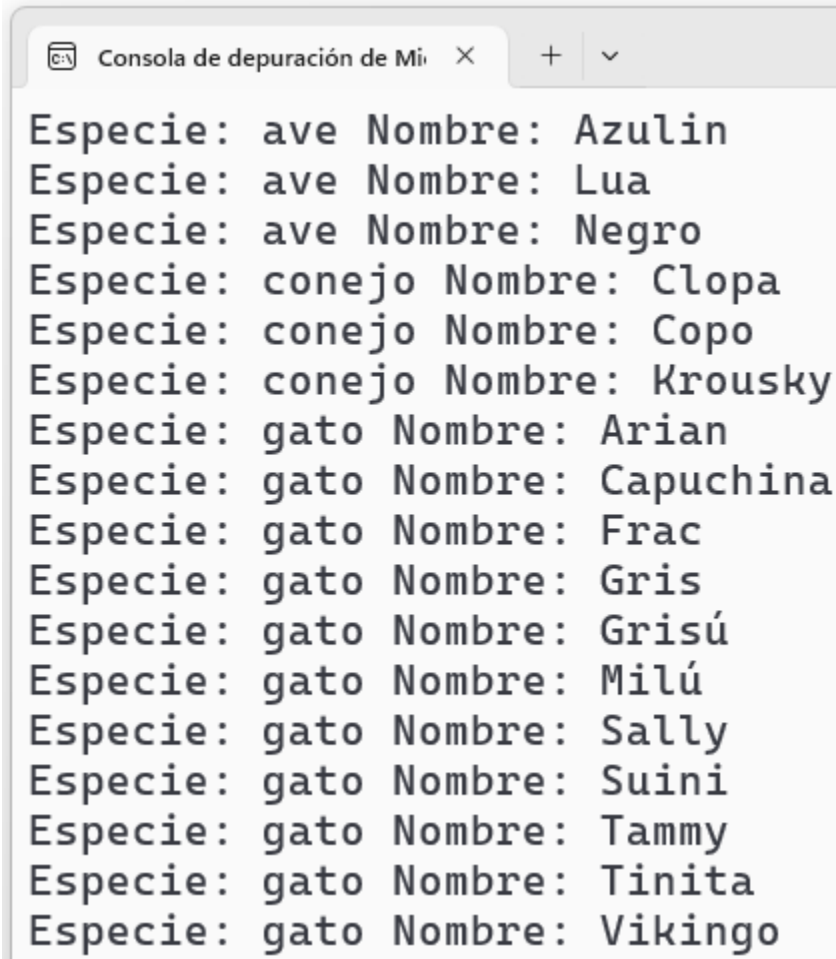
            //Ordene primero por especie y luego por nombre
            List<Mascota> Resultados = (from animal in listaMascotas
                                        orderby animal.Especie,
                                        animal.Nombre
                                        select animal).ToList();

            //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;
        }
    }

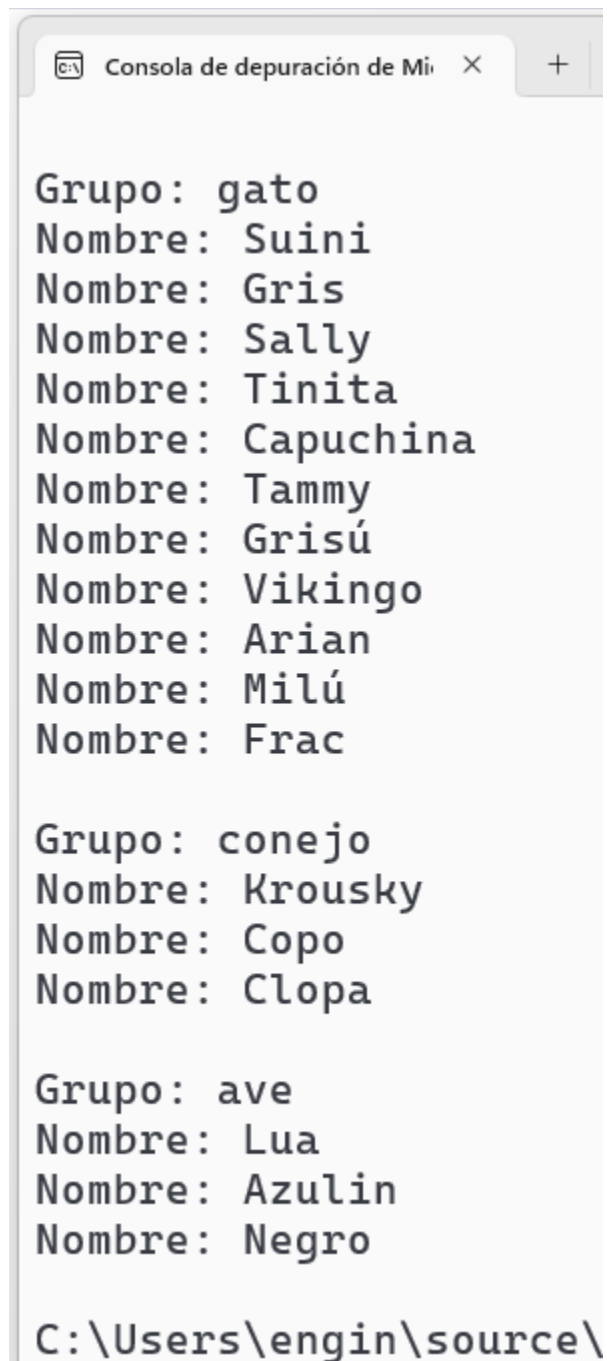
    internal class Program {
        static void Main() {
            //Fuente de datos, una lista
            List<Mascota> listaMascotas = new List<Mascota>();
            listaMascotas.Add(new Mascota("gato", "Suini"));
            listaMascotas.Add(new Mascota("gato", "Gris"));
            listaMascotas.Add(new 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}");
}
}
}
}

```

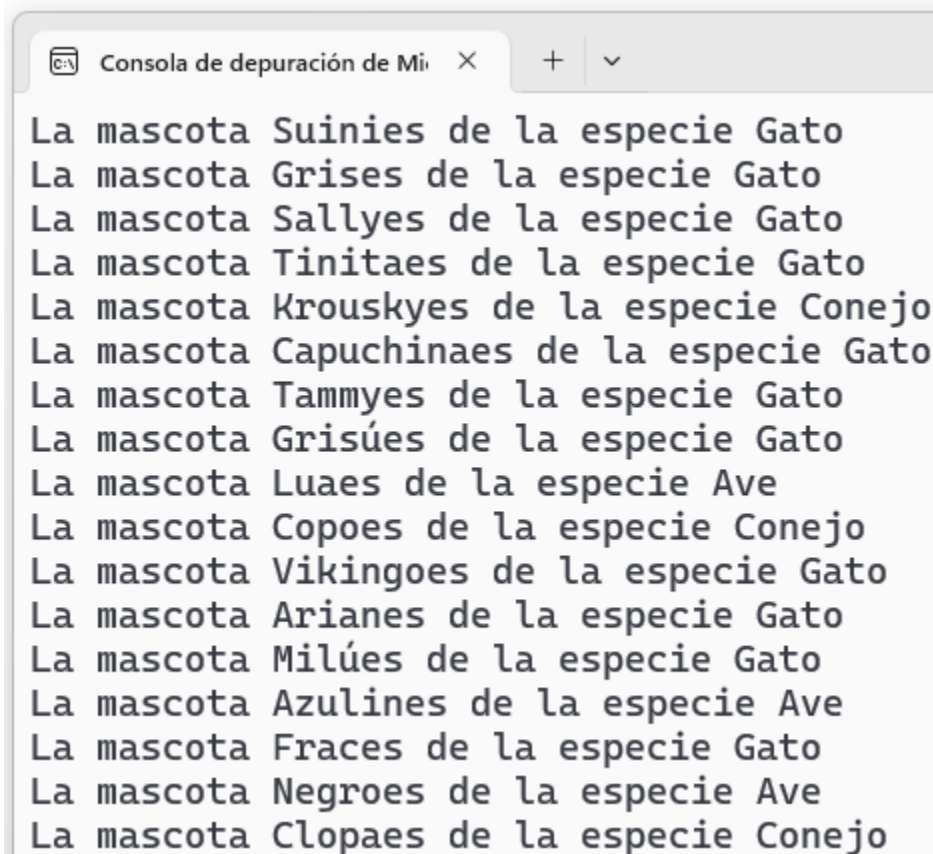


Ilustración 15: LINQ: Hacer un "join" entre listas

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"));
        }
    }
}
```

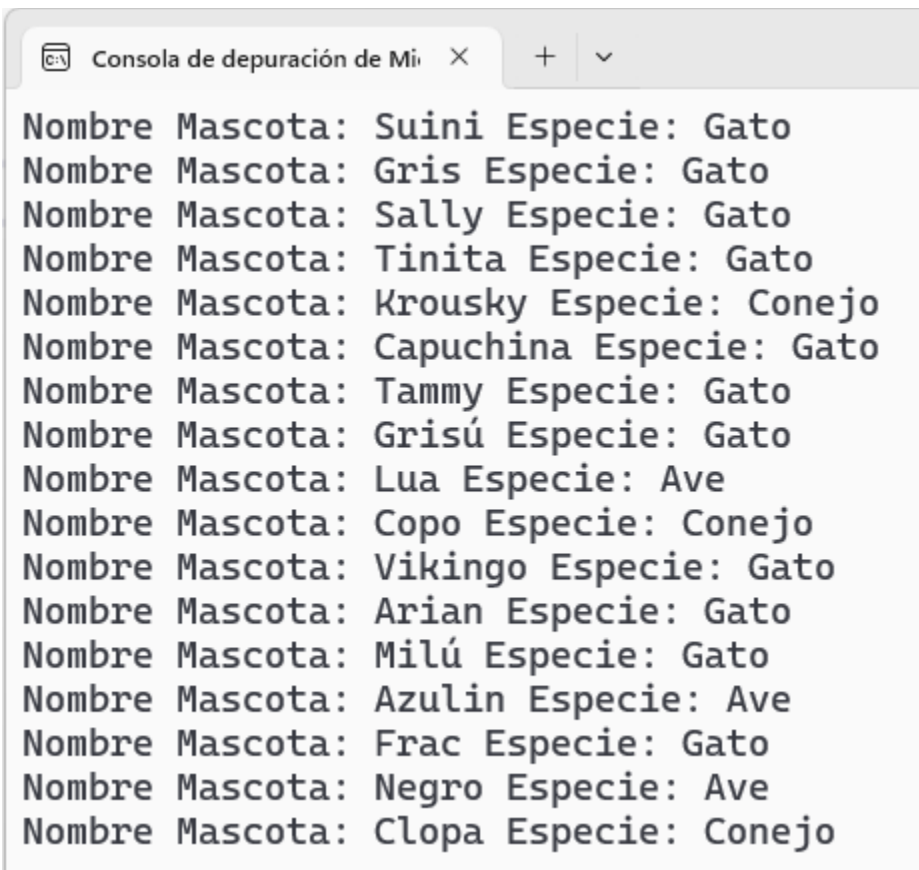
```

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);  
        }  
    }  
}
```

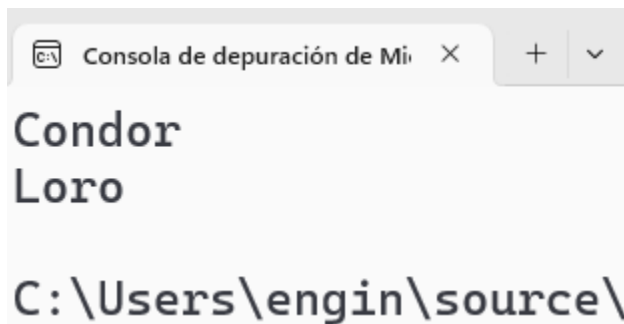


Ilustración 17: LINQ: Extraer los datos de una lista que no están en otra

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);  
        }  
    }  
}
```

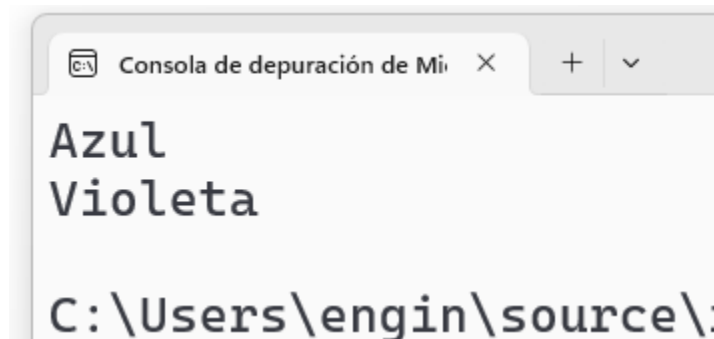


Ilustración 18: LINQ: Intersección de dos listas

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

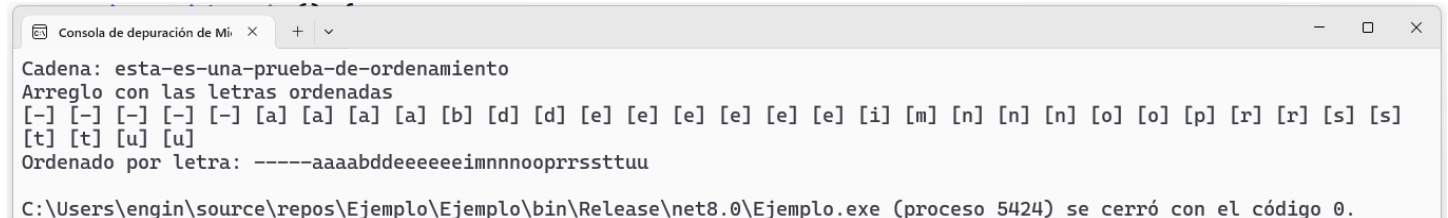
F/021.cs

```
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);
        }
    }
}
```



```
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: -----aaaabdddeeeeeimnnnooprsttuu
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.WriteLine("[" + (char)valor + "] ");
            }

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

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

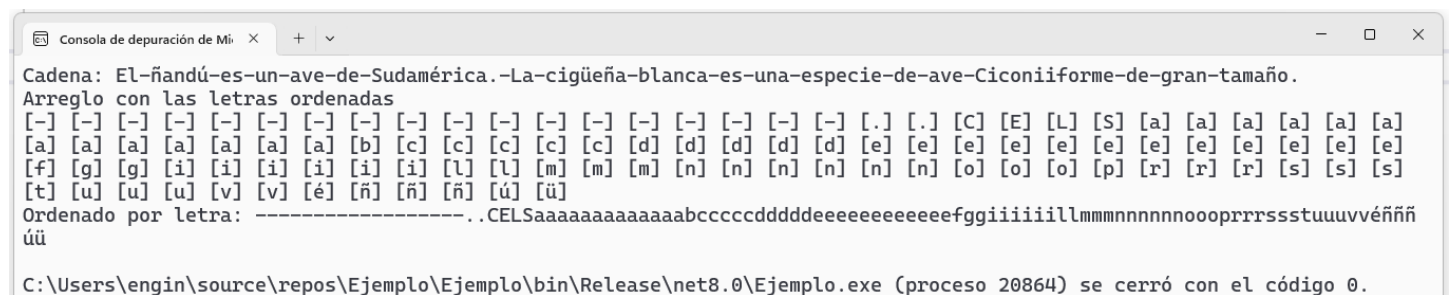


Ilustración 22: LINQ: Ordenar internamente una cadena

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 + " ] ");  
            }  
        }  
    }  
}
```

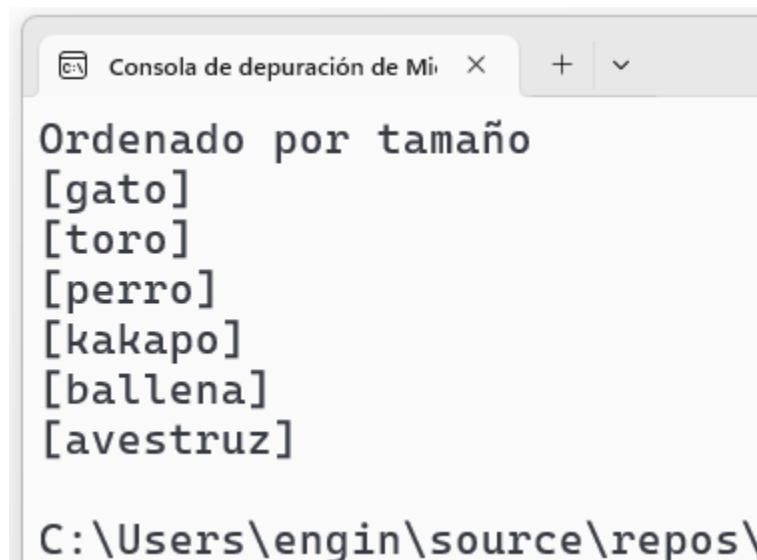


Ilustración 23: LINQ: Ordenamiento según tamaño de la palabra

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 + " ] ");
            }
        }
    }
}
```

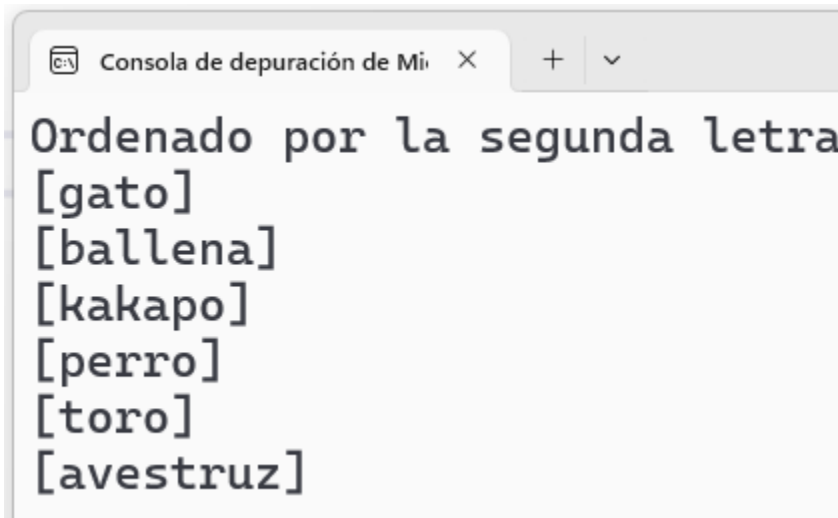


Ilustración 24: LINQ: Ordenamiento por la segunda letra de cada palabra

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 + " ] ");  
            }  
        }  
    }  
}
```

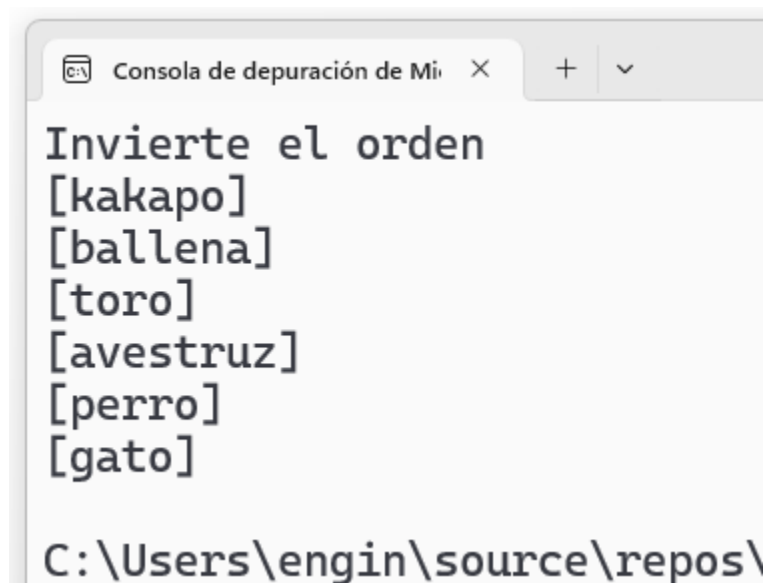


Ilustración 25: LINQ: Invertir el ordenamiento

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");
}
}
}

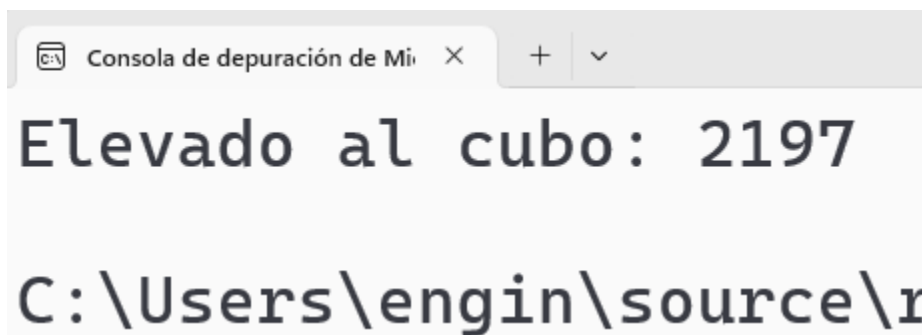
```

```
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.

```
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);  
        }  
    }  
}
```



The screenshot shows a Visual Studio console window with the title 'Consola de depuración de Mi'. The output text is 'Elevado al cubo: 2197' followed by the file path 'C:\Users\engin\source\r'.

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);
        }
    }
}
```

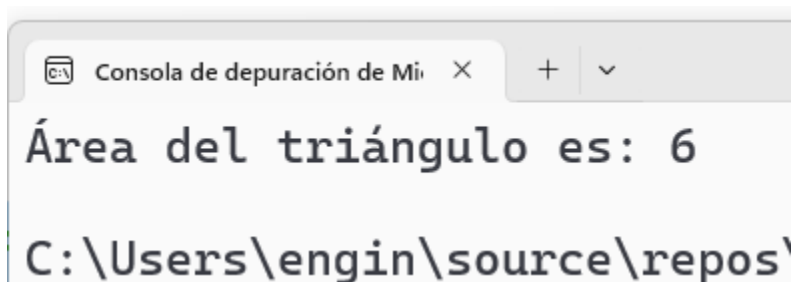


Ilustración 28: Expresiones Lambda usando varias líneas

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);
        }
    }
}
```

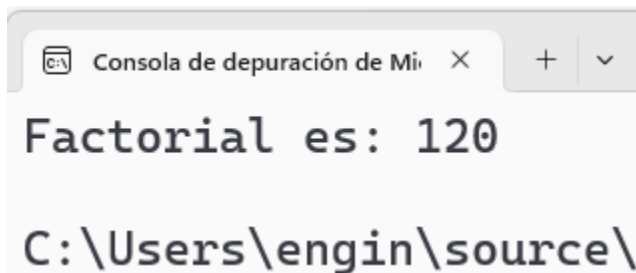


Ilustración 29: Retornando tipo de dato

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);  
        }  
    }  
}
```

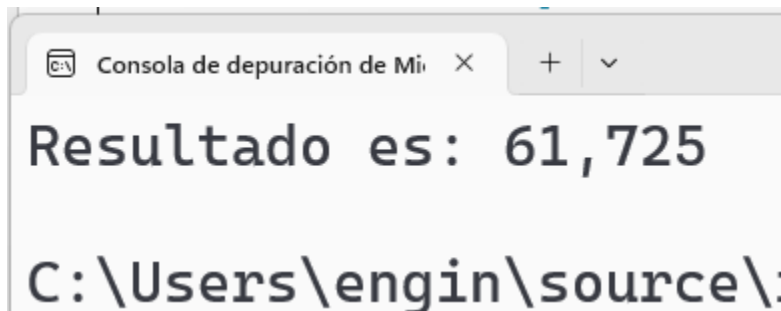


Ilustración 30: Acceso a variable externa

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);
            }
        }
    }
}
```

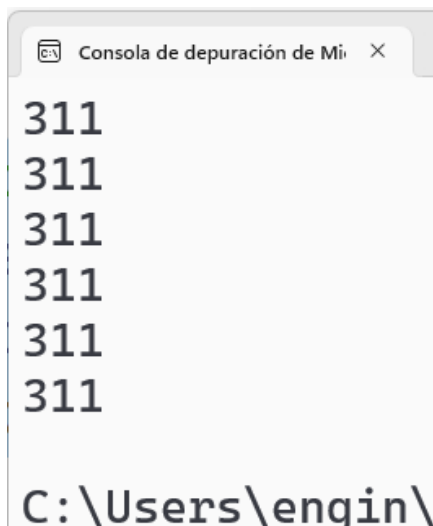


Ilustración 31: No funciona

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);
            }
        }
    }
}
```

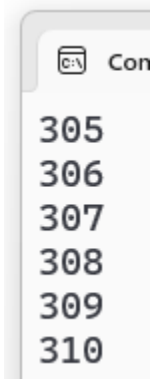


Ilustración 32: Ahora si funciona

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);
            }
        }
    }
}
```

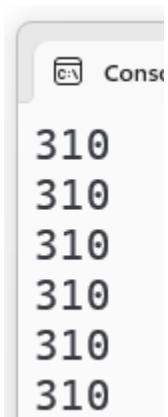


Ilustración 33: No funciona

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);
        }
    }
}
```

```
/* 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);
```



(variable local) `int ValorExterno`

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

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");
            IEnumerable<int> Datos = Lista.Where(x => x > 5);
            foreach (int UnValor in Datos) Console.Write(UnValor + ", ");
        }
    }
}
```

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

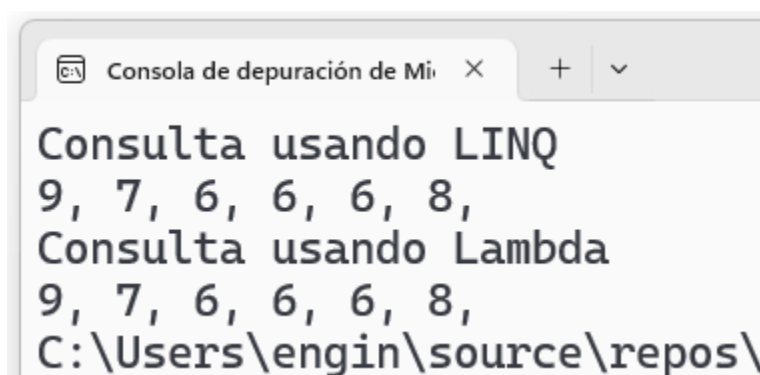


Ilustración 35: Comparativa LINQ y Lambda

Expresiones Lambda: Promedio

F/036.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);  
        }  
    }  
}
```

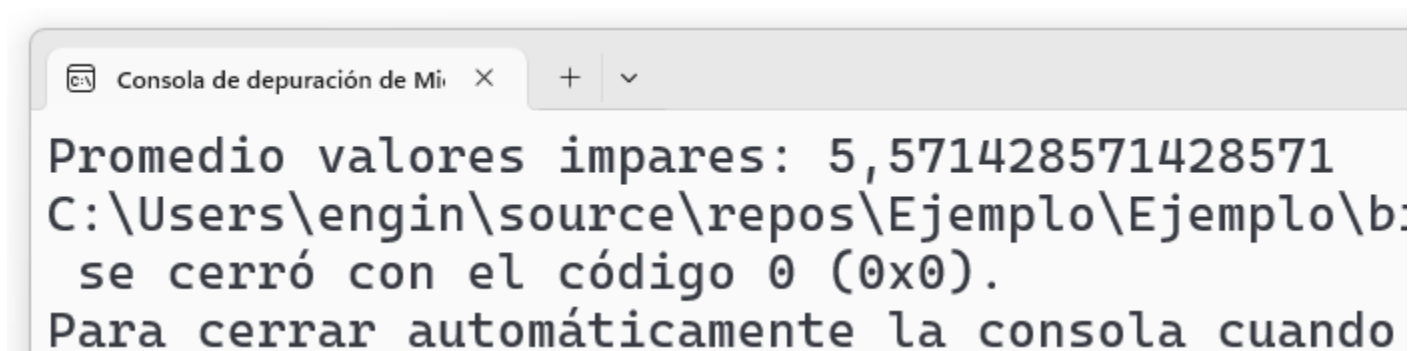


Ilustración 36: Promedio

```
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);
            }
        }
    }
}
```

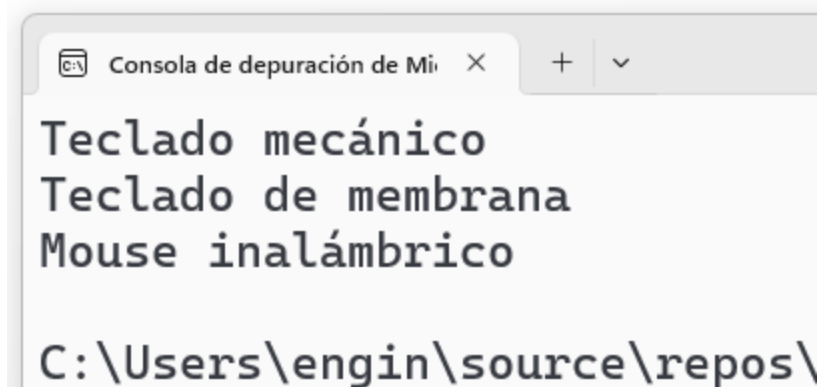



Ilustración 37: JSON y LINQ