Programación Funcional

Ejercicio 1

Crea una aplicación que a partir de una Lista de enteros, te muestre los múltiplos de un número introducido por teclado que existan en la lista, usando **funciones-λ** y el operador ?:

Ejercicio 2

Crea una aplicación que sirva para buscar coincidencias en una lista de cadenas.

Para ello, asignaremos a un delegado genérico una una **función-λ** que reciba una lista y una cadena y sobre la lista con el método **FindAll** busque la cadena.

Ten en cuenta que el método **FindAll** necesitará un predicado para el cual utilizaremos otra **función-λ** para formarlo.

Nota: Puede serte de utilidad la operación **Contains** sobre cadenas.

Por último, muestra con un **foreach** la lista resultante que devuelve el delegado.

Ejercicio 3

Vamos a moficar el ejercicio de la biblioteca del bloque anterior para aplicar funciones lambda, tipando las mismas a través de los delegados genéricos predefinidos en .NET Action<..>, Predicate<..>, Func<...>, etc.

Los métodos a modificar se indicarán a continuación y en ellos deberás completar lo inficado entre corchetes.

```
public Libro BuscaPorISBN(String isbn){
    [Tipo] TieneISBN = [funcion-λ];
    return Libros.Find(TieneISBN);
}
public int CuentaLibrosConNumeroDePaginasMenorA(int num){
    [Tipo] TienepaPaginasMenorANum = [funcion-λ];
    return Libros.FindAll(TienepaPaginasMenorANum).Count;
}
public void EliminaPorAutor(String autor){
    [Tipo] EsDeAutor = [funcion-λ];
    Libros.RemoveAll(EsDeAutor);
}
public bool EstaPrestado(String isbn){
    [Tipo] ContieneCandena = [funcion-λ];
    return (LibrosPrestados().ToList().Find(ContieneCandena) != null);
}
public IEnumerable<string> LibrosPrestados(){
     // Incluye aquí el código donde se lee el StreamReader con los libros
     // prestados que se guardarón con el ToString de los objetos anónimos
     // préstamo. Además, para generar la secuencia utiliza la producción
     // perezosa yield return…

}
```

Ejercicio 4

Vamos ha realizar una serie de operaciones funcionales **usando funciones-λ** con el patrón **MAP – FILTER – REDUCE** (*Select* – *Where* – *Aggregate* en C#)

Partiendo de la siguiente lista de números reales:

```
List<double> reales = new List<double> {
     0.5, 1.6, 2.8, 3.9, 4.1, 5.2, 6.3, 7.4, 8.1, 9.2
};
```

Vamos a realizar las siguientes operaciones:

1. Mostrar la lista usando el método **ForEach(Action<T> action)** de lista. Pasando a la función-λ action, una clausura de la variable string texto, en la que iremos componiendo su contenido separado por un espacio en blanco.
2. Cuenta aquellos elementos cuya parte decimal es menor que 0.5
   - **Map**: Paso del valor real a su parte decimal. Ej: 2.8 → 0.8
   - **Filter**: Filtro aquellas partes decimales que cumplen el predicado: d < 0.5
   - **Reduce**: Contar los elementos en la secuencia resultante.
3. Calcular la suma de todos los valores de la secuencia cuya parte entera sea múltiplo de 3.
   - **Map**: Mapea el valor real a un objeto anónimo con la parte entera y el propio valor real de la secuencia. Ej: 2.8 → new { e = 2, r = 2.8 }
   - **Filter**: Filtro aquellas partes enteras que cumplen el predicado: **o.e** % 3 == 0
   - **Reduce**: Suma todos los **o.r** de la secuencia resultante.
4. Calcular el máximo valor de la secuencia cuya parte decimal es mayor que 0.5
   - **Map**: Mapea el valor real a un objeto anónimo con la parte decimal y el propio valor real de la secuencia. Ej: 2.8 → new { d = 0.8, r = 2.8 }
   - **Filter**: Filtro aquellas partes decimales que cumplen el predicado: **o.d** > 0.5
   - **Reduce**: Me quedo con el máximo de todos los **o.r** de la secuencia resultante.
5. **Ampliación**: Muestra aquellos elementos de la secuencia cuya parte entera es un valor primo. Nota: Seguramente has de hacer más de un Filter e incluso otro Filter dentro de uno de ellos. Además, para crear un predicado que me diga si un número primo de forma funcional (pero poco eficiente) puedes hacer lo siguiente …
   1. Generar una secuencia entre 2 y la parte entera menos uno con **Enumerable.Range()**
   2. Filtrar aquellos valores divisibles por la parte entera.
   3. Preguntar si la secuencia resultante tiene 0 elementos.

Una posible salida del programa debería ser...

```
Elementos: 0,5 1,6 2,8 3,9 4,1 5,2 6,3 7,4 8,1 9,2
Número elementos con parte decimal < 0,5 = 6
Suma elementos con parte entera múltiplo de 3 = 19,4
Máximo cuya parte decimal > 0,5 = 3,9
Elementos parte entera es primo: 2,8 3,9 5,2 7,4

Presione una tecla para continuar . . .
```

Ejercicio 5 Ampliación

A partir de la solución en el ejercicio 4. Rehaz los métodos ...

- public bool **EstaPrestado**(String isbn)
- public Libro **BuscaPorISBN**(String isbn)
- public int **CuentaLibrosConNumeroDePaginasMenorA**(int num)
- public void **EliminaPorAutor**(String autor)

Utilizando consultas de LINQ.

Recursividad

Ejercicio 1

Define una función recursiva que reciba como parámetro una cadena y devuelva su inversa.

```
static string Invierte(string cadena) {...}
```

Ejercicio 2

Escriba una función recursiva para calcular el término n-ésimo de la secuencia: 1, 3, 4, 7, 11, 18, 29, 47, …, $Término_n$ donde $Término_n = Término_{n-1} + Término_{n-2}$

```
static int TerminoSucesion(int n) {...}
```

Ejercicio 3

Escribe una función recursiva que reciba **una cadena** con un valor numérico natural en base 10 y un valor enumerado con la base a la que deseamos pasar (ver ejemplo) y devuelva **otra cadena** con la representación de su valor en la base especificada.

No puedes usar ninguna función de utilidad de C# que lo haga y para hacerlo busca en Google cómo convertir de base decimal a otras bases.

Usa el código de ejemplo para probarlo.

```
enum Base { Binario=2, Octal=8, Hexadecimal=16 }

static string CambioBase(string numeroBase10, Base b) {…}

static void Main()
{
    Console.Write("Introduce un valor decimal positivo: ");
    string _decimal = Console.ReadLine();

    foreach (var b in (Base[])Enum.GetValues(typeof(Base)))
        Console.WriteLine($"En base {b} es: {CambioBase(_decimal, b)}");
    Console.ReadLine();
}
```

Ejercicio 4

Completa el código de la función **local void VerArbol(int nivel, int idPadre)** para que mediante una llamada recursiva y la clausura del parámetro directorios muestre la jerarquía por consola de la siguiente forma ...

```
Marketing
   |__Manuel
   |__Rosa
        |__Empresas
             |__Alicante
        |__Bocetos
Informática
   |__Ana
   |__Jose
   |__Pedro
        |__Claves
```

Código a completar...

```
class Principal {
    class Directorio{
        public int Id { get; private set; }
        public string Nombre { get; private set; }
        public int IdPadre { get; private set; }

        public Directorio(int id, string nombre, int idPadre) {
            Id = id;
            Nombre = nombre;
            IdPadre = idPadre;
        }
    }
    static void VerArbol(Directorio[] directorios) {
        void VerArbol(int nivel, int idPadre) {
           // A COMPLETAR
        }
        const int NIVEL_INICIAL = 0;
        const int ID_RAIZ = 0;
        VerArbol(NIVEL_INICIAL, ID_RAIZ);
    }
    static void Main(){
        Directorio[] directorios = new Directorio[] {
            new Directorio(3, "Manuel", 1),
            new Directorio(4, "Rosa", 1),
            new Directorio(5, "Ana", 2),
            new Directorio(1, "Marketing", 0),
            new Directorio(2, "Informática", 0),
            new Directorio(8, "Empresas", 4),
            new Directorio(9, "Bocetos", 4),
            new Directorio(10, "Claves", 7),
            new Directorio(11, "Alicante", 8),
            new Directorio(6, "Jose", 2),
            new Directorio(7, "Pedro", 2)
        };
        VerArbol(directorios);
    }
}
```

Ejercicio 5

Escriba una función recursiva para calcular el **máximo común divisor** de **dos números enteros** dados aplicando las propiedades recurrentes.

- Si a > b, entoces m.c.d(a, b) = m.c.d(a - b , b)
- Si a < b, entoces m.c.d(a, b) = m.c.d(a , b - a)
- Si a = b, entoces m.c.d(a, b) = m.c.d(b , a) = a = b

Ejercicio 6

Crea con una **función-λ** que genere los **N** primeros términos de la sucesión de **Fibonnaci**.

Solución iterativa (Fuente Wikipedia):

```
funcion fib(n)
     a ← 0
     b ← 1
     Para k desde 0 hasta n hacer
          b ← b + a
          a ← b − a
     devuelve a
```

Realiza la misma operación pero ahora usando recursividad con **funciones-λ**

Solución recursiva (Fuente Wikipedia):

```
funcion fib(n)
     si n < 2 entonces
          devuelve n
     en otro caso
          devuelve fib(n - 1) + fib(n - 2)
```