Tecnología y paradigmas de programación. Laboratorio 7. Grupos 1, 2 y 3.

1. Examinar `functional.continuations.lazy`. Crear un proyecto con una clase y una aplicación de consola reproduciendo la misma estructura de métodos. La clase puede llamarse PerfectSquares. Implementar:
   - `bool IsPerfectSquare(int n)`, devuelve `true` si existe x entero tal que x*x==n. 0 y 1 son cuadrados perfectos. En realidad casi lo único que hay que cambiar es este método.
   - `EagerPerfectSquares`, `LazyPerfectSquaresGenerator`, `LazyPerfectSquares`, análogos a los métodos correspondientes del código de ejemplo, trasladados al caso de los cuadrados perfectos, básicamente sustituir la llamada a `IsPrime` por `IsPerfectSquare`. Para que el cero se contabilice habrá que tocar alguna instrucción de alguno de los métodos.
   - Aplicación de consola de prueba, partir de la aplicación de consola del código de ejemplo para medir tiempos. Probar *desde* 10 y un *total* de 100 números. Observar si hay mejoras en la eficiencia. Para entender lo que está pasando se puede usar la siguiente idea: añadir un parámetro adicional a `IsPerfectSquare`, de tipo char. Usar ese parámetro para saber desde donde se ha llamado a esta función pasando 'L' para lazy, 'E' para eager. Observar cuantos caracteres aparecen en cada caso y cuando.

2. Examinar la solución `higher.order`. Incorporar `ForEach.cs` a la solución de la práctica de modo que esta sea autocontenida. Añadir los siguientes métodos extensores a `ForEach.cs` (alternativamente, algunos podrían implementarse sobrecargando el propio `ForEach`, pensar cuales).
   - `ForEachOdd`, recibe una acción, la aplica a los elementos de la colección en las posiciones impares.
   - `ForEachNth`, recibe una acción, la aplica a los elementos que están en la posición múltiplo de n, un parámetro adicional.
   - `ForEachPred`, recibe una acción y un predicado. Aplica la acción a los elementos de la colección que cumplen el predicado. Este se puede implementar invocando en el propio método que se pide el `ForEach` de ejemplo, con una acción adecuada como parámetro. Es una alternativa interesante.

   Probar los métodos con una aplicación de consola, la colección puede ser un vector de enteros aleatorios, la acción mostrar por la consola, el predicado ser par, por ejemplo.
   - Ejercicio adicional con estos métodos extensores. ¿Cómo se podría inicializar un vector de datos con números aleatorios usando `ForEach`? Calcular cuantas veces se repite cada entero en uno de los vectores usando también `ForEach`. Emplear la típica idea consistente en partir de un vector de contadores (`vContador`) todo a cero y recorrer el vector de datos (`vEnteros`): para cada posición `i` hacer `vContador[i]++`.

3. Ejercicios iniciales sobre LINQ. Las soluciones de estos ejercicios pueden escribirse en el mismo fuente que el ejercicio anterior (y aprovechando usar los métodos extensores). Crear un par de arrays de enteros aleatorios de tamaño y rango moderado para poder comprobar los resultados de los siguientes ejercicios que involucran el uso de `Where`, `Select`, `Aggregate` o combinaciones de varias de estas.
    - Aunque está predefinido, usando Aggregate calcular el máximo de uno de los arrays. Usar adecuadamente el valor inicial para Aggregate.
    - Hallar la diferencia entre dos arrays de enteros asimilándolos a conjuntos: la diferencia entre x e y, sería como el conjunto de elementos de x que no están en y. No hace falta tratar las repeticiones.
    - Eliminar las repeticiones de un array de enteros. Ej: de {4,3,2,4,2,1,9} se seleccionan {4,3,2,1,9}, en un nuevo objeto. Partir del código de ejemplo en donde se devuelve la lista de los elementos que se repiten en un array de enteros: `higher.order/filter/Program.cs`.
    - Calcular la varianza de un array de enteros, no hace falta escribirlo todo en una línea lógica (pero se puede). La varianza (en esta asignatura) la vamos a definir como la media de los cuadrados menos el cuadrado de la media. Las medias es necesario calcularlas como `double`.
      Si se quiere comprobar el resultado de la función usando `var` en R, es necesario multiplicar el resultado obtenido (en R) por el número de datos menos uno y dividir por el número de datos.
    - Encadenando llamadas a `Where`, `Select` y `Aggregate`, calcular la suma de los cuadrados de los elementos de valor par de uno de los arrays.
    - Encadenando llamadas a `Where`, `Select` y `Aggregate`, calcular la suma de los cuadrados de valor impar de los elementos de uno de los arrays.
    - Usando `Aggregate` repetir el ejercicio en el que se cuenta cuantas veces se repite cada valor de un vector de enteros entre 0 y 10.