Tecnología y paradigmas de programación. Laboratorio 6. Grupos 1, 2 y 3.

(El código de ejemplo está en las soluciones functional/closures y functional/continuations)

1. Escribir el método `static Predicate<int> ContieneDivisores (IEnumerable<int> e)` usando currificación. La invocación de este método devuelve un predicado que es cierto si el entero que se pasa como parámetro (al predicado que devuelve) es divisible entre alguno de los elementos de la lista de enteros que se pasa a `ContieneDivisores`. Ejemplo, si `lista` contiene {2,3,5} la llamada `ContieneDivisores(lista)` devuelve un predicado que sería cierto para 10 pero falso para 11. De otro modo, la invocación del predicado devuelto con 10 como parámetro: `ContieneDivisores(lista)(10),` es `true` , pero `ContieneDivisores(lista)(11)` es `false`. El ejemplo de referencia es `CurriedContains` el proyecto `closures.currying`. En el caso del presente ejercicio, el método no es genérico. Sería interesante usar este método revisitando el ejercicio de los números primos de la sesión anterior de prácticas.

2. Usando una clausula definir la estructura de control repeat until del lenguaje de programación Pascal, de modo que la llamada `RepeatUntilLoop(condicion, cuerpo);` repite `cuerpo` hasta que `condicion` sea cierta, como mínimo el cuerpo se ejecuta una vez. Tomar como referencia el ejemplo análogo para el bucle `while`, en los ejemplos de teoría (`closures.closures`). Probar mostrando los enteros de 0 a 9, ambos incluidos.

3. Implementar una clausula que devuelva en la primera invocación el segundo término de la sucesión de Fibonacci, en la segunda el tercero y así sucesivamente, es decir: 1,1,2,3,5,8... Tomar como referencia el ejemplo `ReturnCounter()` visto en las transparencias de teoría similar a este que he reescrito de una forma más extensa aquí.

```
static Func<int> ReturnCounter() {
   int counter = 0;
   return () => {
      counter=counter+1;
      return counter;
      };
}
```

4. Escribir los generadores `InfinitePrime()` y `FinitePrime()`, devolviendo la sucesión de los números primos. Son análogos a los ejemplos correspondientes de teoría, `InfiniteFibonacci()` y `FiniteFibonacci()` en el proyecto `continuations.generators`. Usar el primero para mostrar el primer número primo mayor que 1000. Usar el segundo para mostrar los 20 primeros primos.

5. Usando memorización implementar la clase `MemoizedPrime`, con un método que devuelva el primo en la posición n. Al igual que el ejemplo `MemoizedFibonacci`, (`continuations.memoization`) tendrá un atributo de tipo `IDictionary` para almacenar los los valores ya calculados. Puede usarse una función privada `bool isPrime(int n)` para facilitar la escritura del método `int Primes(int)` (análogo al método `int Fibonacci(int)` en el mencionado ejemplo).
   Lo interesante sería que, a su vez, el método `Primes` usase la estrategia vista en la anterior clase de prácticas: un número es primo si ninguno de los primos menores que él es su divisor, recuérdese que los primos ya obtenidos están almacenados en el atributo de tipo `IDictionary` que se puede recorrer con un foreach. Y rizando el rizo, se puede usar `ContieneDivisores` para implementar esta idea.

NOTA: la tarea autónoma consiste en entregar estos mismos ejercicios.