Tecnologías y Paradigmas de Programación. Actividad presencial del Laboratorio 6.

(El código fuente relacionado se puede encontrar en functional/closures y functional/continuations)

1. Curificar este método

```csharp

static bool ContainsDivisor(IEnumerable<int> nums, int n)

{

foreach(var e in nums)

{

if (n % e == 0)

return true;

}

return false;

}

```

para implementar `static Predicate<int> ContainsDivisor(IEnumerable<int> e)`. El método solicitado devuelve un predicado que, al llamarlo, devuelve verdadero si el parámetro n es divisible por al menos uno de los enteros en el parámetro IEnumerable. Por ejemplo, si una lista contiene {2,3,5}, la llamada `ContainsDivisor(lista)` devuelve un predicado que es verdadero para 10 (10 es divisible por 2 o 5) pero falso para 11 (no es divisible por 2, 3 o 5). Verifica `CurriedContains` en `closures.currying`. No uses genéricos en este ejercicio.

2. Usa un cierre para definir la estructura de control "repeat until" del lenguaje de programación Pascal:

`RepeatUntilLoop(condición, cuerpo);` repite el cuerpo hasta que la condición sea verdadera, el cuerpo se ejecuta al menos una vez (es decir, primero se ejecuta el cuerpo y luego se prueba la condición). El ejemplo de referencia es el `WhileLoop`, en `closures.closures`. Úsalo para mostrar los enteros del 0 al 9, ambos incluidos.

3. Reescribe el siguiente código usando el cierre del bucle `while` encontrado en `closures.closures` en lugar del `while` de C#. La variable de columna se define antes de los bucles para facilitar las cosas.

```csharp

int[,] a = new int[3,4] {

{0, 1, 2, 3} ,

{4, 5, 6, 7} ,

{8, 9, 10, 11} ,

};

int row = 0;

int column = 0;

while (row < 3)

{

while (column < 4)

{

Console.Write("{0} ", a[row,column]);

column++;

}

Console.WriteLine();

column = 0;

row++;

}

```

4. Implementa un cierre que devuelva el siguiente término de la secuencia de Fibonacci cada vez que se llame, es decir: 1, 1, 2, 3, 5, 8... El ejemplo de referencia es `ReturnCounter()` encontrado en las diapositivas, aquí reescrito detalladamente:

```csharp

static Func<int> ReturnCounter() {

int contador = 0;

return () => {

contador = contador + 1;

return contador;

};

}

```

5. Implementa generadores `InfinitePrime()` y `FinitePrime()`, que devuelven la secuencia de números primos de manera similar a `InfiniteFibonacci()` y `FiniteFibonacci()` en `continuations.generators`. Utiliza el primero para mostrar el primer primo mayor que 100, hay métodos de extensión útiles disponibles. Encuentra los métodos adecuados para mostrar los 10 primos siguientes después de los primeros 10 primos con el mismo generador. Haz una llamada al segundo generador para mostrar los primeros 20 números primos.

6. Utilizando la memorización, implementa la clase `MemoizedIsPrime`. Esta clase tiene un método público `IsPrime` que devuelve true si el parámetro int es un número primo, false en caso contrario. Al igual que en `MemoizedFibonacci` (continuations.memoization), esta clase tendrá un atributo `IDictionary` para almacenar las llamadas previamente calculadas a ese método como pares de clave (int) y valor (bool). De esta manera, las llamadas posteriores al método mencionado con el mismo argumento se recuperan de la caché en lugar de calcularse desde cero.

7. Estudia cuidadosamente los ejemplos de memorización proporcionados que utilizan un cierre en lugar de una clase e implementa nuevamente el algoritmo anterior.

Carga la actividad obligatoria antes del próximo Laboratorio.