**LENGUAJES DE PROGRAMACIÓN**

Semestre Agosto-Diciembre 2017

**TAREA 5**

**Forma de trabajo**: En equipo de 2 integrantes **Forma de entrega**: Como asignación de Blackboard **Fecha de entrega**: lunes 23 de octubre de 2017

***Programación en Haskell***

En esta tarea pondrán en práctica sus conocimientos sobre programación en Haskell. Resuelvan cada uno de los siguientes problemas. Para cada problema, entreguen una impresión de su codificación y de las pantallas en donde se muestren las pruebas solicitadas. También deberán subir el código de sus funciones a Blackboard, bajo el nombre **M\_Tarea5.hs** donde deberán sustituir la M del nombre, por sus matrículas.

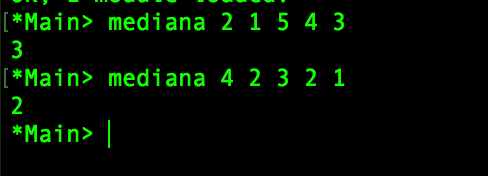
# PROBLEMAS: Programación básica y recursiva SIN LISTAS (20 puntos)

1. Implementar la función **mediana** que obtenga la mediana estadística de un grupo de 5 argumentos. La mediana representa el valor de la variable de posición central en un conjunto de datos ordenados.

Probar con:

> mediana 2 1 5 4 3 **=> 3**

> mediana 4 2 3 2 1 **=> 2**

****

1. Implementar la función recursiva **tabla** que reciba un entero positivo indicando el número de renglones, otro indicando el número de columnas y un tercero indicando el valor de inicio, para que a partir de ellos “despliegue” una tabla de esas dimensiones con elementos de una secuencia incremental a partir del tercer argumento.

Probar con:

> tabla 1 1 4

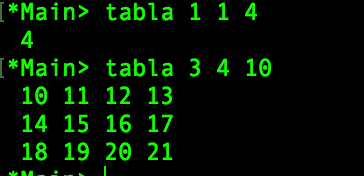
# => 4

> tabla 3 4 10

# => 10 11 12 13

**14 15 16 17**

**18 19 20 21**

****

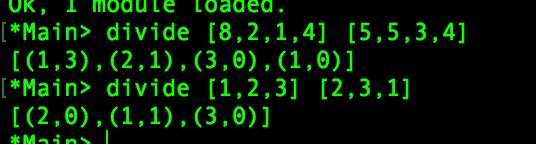
**PROBLEMAS: Programación con Listas (30 puntos)**

1. Implementar la función RECURSIVA **divide** que reciba 2 listas de enteros positivos para regresar una lista de pares que contengan el cociente y el residuo de dividir, elemento por elemento, el mayor por el menor correspondiente.

Probar con:

> divide [8,2,1,4] [5,5,3,4] **=> [(1,3),(2,1),(3,0),(1,0)]**

> divide [1,2,3] [2,3,1] **=> [(2,0),(1,1),(3,0)]**

****

1. Implementar la función RECURSIVA **intercala** que genere una lista de listas insertando un valor en todas las posiciones de una lista plana.

Probar con:

> intercala 0 [1] **=> [[0,1],[1,0]]**

> intercala 0 [1,2] **=> [[0,1,2],[1,0,2],[1,2,0]]**

1. Implementar la función RECURSIVA **permutación** que obtenga la lista de todas las permutaciones generadas a partir de una lista. El orden de los elementos es irrelevante, pero deben de aparecer todos. **Tip**: puedes aprovechar la función **intercala**.

Probar con:

> permutacion [] **=> []**

> permutacion [1] **=> [[1]]**

> permutacion [1,2] **=> [[1,2],[2,1]]**

> permutacion [1,2,3]

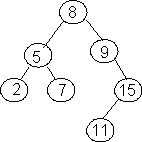
# => [[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]

**PROBLEMAS: Estructuras de Datos (20 puntos)**

Un Árbol Binario (AB) puede ser representado en Haskell, por medio de la siguiente declaración:

data AB t = A (AB t) t (AB t) | V deriving Show

Por ejemplo, si se tiene el siguiente AB:



ab = A (A (A V 2 V)

5

(A V 7 V))

8 (A V

9

(A (A V 11 V) 15

V))

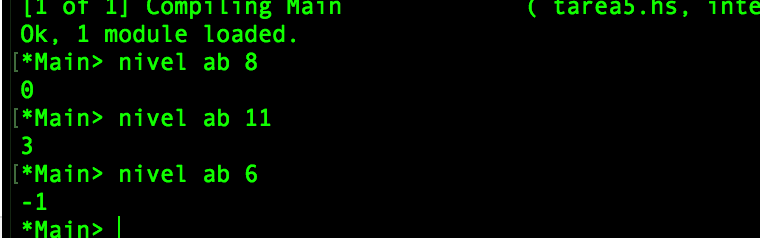
1. Implementar la función recursiva **nivel** que dado un AB y un valor, regrese el nivel del AB donde se encuentra ese valor, o -1 en caso de que no se encuentre. La raíz se encuentra en el nivel 0.

Probar con:

> nivel ab 8 **=> 0**

> nivel ab 11 **=> 3**

> nivel ab 6 **=> -1**

****

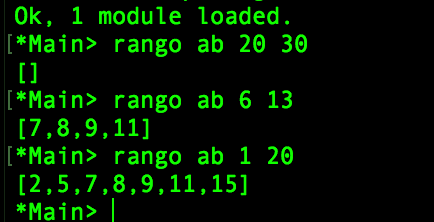
1. Implementar la función recursiva **rango** que dado un árbol binario de búsqueda y dos valores que describen un rango, regrese una lista ordenada que contenga todos los valores en el rango que se encuentren en el árbol

Probar con:

> rango ab 20 30 **=> []**

> rango ab 6 13 **=> [7 8 9 11]**

> rango ab 1 20 **=> [2 5 7 8 9 11 15]**

****

# PROBLEMAS: Funciones de Orden Superior y otras facilidades *(30 puntos)*

- Pueden utilizar FOS, comprensión de listas, funciones auxiliares, etc., pero ninguna debe recurrir a recursión directa o indirecta (por ejemplo, que A llame a B y que B llame a A).

1. Implementar la función NO RECURSIVA **valmat** cree una matriz de NxM con un valor. Probar con:

> valmat 1 2 3 **=> [[3,3]]**

> valmat 2 3 0 **=> [[0,0,0],[0,0,0]]**

1. Implementar la función NO RECURSIVA **chess** que cree una matriz cuadrada de tamaño NxN donde las posiciones con suma de coordenadas par sean 1’s y las posiciones con suma de coordenadas impar sean 0’s.

Probar con:

> chess 2 **=> [[1,0],[0,1]]**

> chess 3 **=> [[1,0,1],[0,1,0],[1,0,1]]**

# > chess 4 => [[1,0,1,0],[0,1,0,1],[1,0,1,0],[0,1,0,1]]

1. Implementar la función no recursiva **reducir** que a partir de una lista genere una lista de listas donde se vayan eliminando sus elementos hasta que se vacíe.

Probar con:

> reducir [] **=> [[]]**

> reducir [1] **=> [[1],[]]**

> reducir [1,2] **=> [[1,2],[2],[]]**

