Nombre y apellido: Losiggio Ignacio Esteban L.U. o D.N.I.: 39773452
Carrera: CS. de la Computación Número de órden: 19 Cant. de hojas: 2

Departamento de Computación FCEyN – UBA

1	2	3	4	5	TOTAL
B	B	B	B	B	A

Taller de Álgebra I - Parcial

corregié Jésica

SEGUNDO CUATRIMESTRE 2017 – TURNO MIÉRCOLES PM

18 de octubre de 2017

Aclaraciones

- El parcial se aprueba con tres ejercicios bien resueltos.
- Programe todas las funciones en lenguaje Haskell. El código debe ser autocontenido. Si utiliza funciones que no existen en Haskell, debe programarlas.
- Incluya la signatura de todas las funciones que escriba.
- No está permitido: alterar los tipos de datos presentados en el enunciado utilizar técnicas no vistas en clase para resolver los ejercicios.

Ejercicio 1

Implementar la función `todoMenor :: (Float, Float, Float) -> (Float, Float, Float) -> Bool` que dados dos vectores $x, y \in \mathbb{R}^3$ decida si todos los elementos de x son menores a todos los de y.

Por ejemplo:

```
todoMenor (3,-1,2) (5,10,0)  ⇝ False
todoMenor (4,-1,7) (5,21,5)  ⇝ True
todoMenor (2,1,31) (40,61,15) ⇝ False
```

Ejercicio 2

Implementar una función `sumaFibonacci :: Integer -> Integer` que para cada $n \geq 1$ calcule $\sum_{j=0}^{n} f_n$, donde f_n es n–ésimo término de la sucesión de Fibonacci.

Por ejemplo:

```
sumaFibonacci 2 ⇝ 1+1+2 ⇝ 4
sumaFibonacci 4 ⇝ 1+1+2+3+5 ⇝ 12
```

Ejercicio 3

Asumiendo que disponemos de la función `esPrimo :: Integer -> Bool` que decide si un entero es primo o no, implementar una función `esBSuave :: Integer -> Float -> Bool` que, dados un entero positivo n y un real positivo B, decida si todos los primos que dividen a n son menores a B.

Por ejemplo:

```
esBSuave 1 90  ⇝ True      (pues no hay primos que dividan a 1)
esBSuave 81 4  ⇝ True
esBSuave 21 6  ⇝ False
```

Ejercicio 4

Programar la función `congruenciasMod3:: [Integer] -> (Integer, Integer, Integer)` que dada una lista (finita) de números enteros xs devuelve una terna de enteros cuyo i-ésimo elemento, entendiendo al primero como el 0-ésimo, es la cantidad de elementos de xs que son congruentes a i módulo 3.

Por ejemplo:

```
congruenciasMod3 [-9,-1,5,0,8,2,1,3] ⇝ (3,1,4)
```

Ejercicio 5

Implementar una función `esSumaMod7DeDos :: Integer -> [Integer] -> Bool` que, dados un entero k y una lista (finita) de números enteros xs, decida si k es igual, *módulo 7*, a la suma de dos elementos de la lista xs.

Por ejemplo:

```
esSumaMod7DeDos 1 [1,-2,3,5,2] ⇝ True   (pues 1 ≡ 3 + 5 mód 7)
esSumaMod7DeDos 2 [1,2,4]      ⇝ False  (no admitimos sumar al 1 consigo mismo)
esSumaMod7DeDos 5 [6,-3,6,2]   ⇝ True   (pues 5 ≡ 6 + 6 mód 7; sí admitimos sumar dos apariciones de
un mismo número en lugares distintos de la lista)
```

Fecha 18/10/2017 Hoja 1 Ignacio Losigsio

Ejercicio 1

```
todoMenor :: (float, float, float) -> (float, float, float) -> Bool
todoMenor (x,y,z) (i,j,k) = x<i && x<j && x<k
                          && y<i && y<j && y<k
                          && z<i && z<j && z<k
```

Ejercicio 2

```
fib :: Integer -> Integer
fib 0 = 1
fib 1 = 1
fib n = (fib (n-1)) + (fib (n-2))

sumafibonacci :: Integer -> Integer
sumafibonacci 1 = (fib 0) + (fib 1)
sumafibonacci n = (fib n) + (sumafibonacci (n-1))
```

Ejercicio 3

podés usar fromInteger

```
afloat :: Integer -> float
afloat 0 = 0
afloat n | n>0  = 1 + (afloat (n-1))
         | otherwise = -1 + (afloat (n+1))

esPrimo :: Integer -> Bool
```

función dada

positivos ↙ mayores a n ↓ divisores de m ↓

```
divisoresPrimosMayoresA :: Integer -> Integer -> [Integer] ✓
divisoresPrimosMayoresA n m | n >= m = [] ⊗1
                            | (esPrimo n) && ((mod m n) == 0) = n : (divisoresPrimosMayoresA (n+1) m) ✓
                            | otherwise = (divisoresPrimosMayoresA (n+1) m) ✓
```

(divisores Primos Mayores A (n+1) m) ✓

⊗1 esto no funciona si m es primo porque cuando quiere verificar si m es divisor de m (o sea n = m) entra por la primer guarda y devuelve [] se arregla viendo ">" en lugar de ">=" porque hace que se vea la 2da condición

```
todosFloatMenor :: [Integer] -> Float -> Bool ✓
todosFloatMenor [] _ = True ✓
todosFloatMenor (x:xs) f = ((toFloat x) < f) && (todosFloatMenor xs f) ✓

esBSuave :: Integer -> float -> Bool
esBSuave n B = todosFloatMenor (divisoresPrimosMayoresA 1 n) B ✓
```

Ejercicio 4

```
CongruenciasMod3ConParciales :: [Integer] -> (Integer, Integer, Integer) -> (Integer, Integer, Integer) ✓
CongruenciasMod3ConParciales [] resultados = resultados ✓
CongruenciasMod3ConParciales (x:xs) (a,b,c) | (mod x 3) == 0 = CongruenciasMod3ConParciales xs (a+1, b, c) ✓
                                            | (mod x 3) == 1 = CongruenciasMod3ConParciales xs (a, b+1, c) ✓
                                            | (mod x 3) == 2 = CongruenciasMod3ConParciales xs (a, b, c+1) ✓
```

↑ Vi xs antes de la terna, me olvidé de ponerlo

Fecha 18/10/2017 Hoja 2 Ignacio Losigsio

```
congruenciasMod3 :: [Integer] -> (Integer, Integer, Integer)
congruenciasMod3 lista = congruenciasMod3conParciales lista (0,0,0)
```

Ejercicio 5

```
esSumaMod7De :: Integer -> Integer -> [Integer] -> Bool

esSumaMod7De a b [] = false
esSumaMod7De a b (x:xs) = ((mod a 7) == (mod (b+x) 7))
                          || (esSumaMod7De a b xs)

esSumaMod7DeDos :: Integer -> [Integer] -> Bool

esSumaMod7DeDos k (x:xs) = (esSumaMod7De k x xs)
                           || (esSumaMod7DeDos k xs)
esSumaMod7DeDos k _ = false
```

otra forma:

```
esSumaMod7DeDos k [] = False
```