Estructuras Discretas 2023-1
Práctica 2: ¿Haaaskell? Sí.

Lourdes del Carmen González Huesca Juan Alfonso Garduño Solís
María Fernanda Mendoza Castillo

Fecha de entrega: Sábado 24 de septiembre de 2022

Resuelve cada uno de los siguientes ejercicios y coloca tu código en **un solo archivo** *.hs* respetando los nombres indicados. No olvides revisar los lineamientos de entrega para no recibir penalizaciones.

1. ¿Por qué? porque sí.

Define cada una de las siguientes funciones **recursivamente**.

Sobre números... *(2 puntos)*

1) `potencia b p`, que recibe un número `b` y lo eleva a la `p`.
   Por ejemplo:

```
potencia 2 2 = 4
potencia 3 3 = 27
potencia 7 7 = 823543
```

2) `suma_pares n`, que recibe un número par `n` y suma **todos** los números pares anteriores a `n`, incluyendo a `n`.
   Por ejemplo:

```
suma_pares 10 = 30
suma_pares 20 = 110
suma_pares 30 = 240
```

3) *El n-ésimo número triangular se obtiene de sumar n número consecutivos a partir del 1.*
   Escribe la función `triangular n`, que recibe un entero regresa el n-ésimo número triangular, osease: $\sum_{i=1}^{n} i$
   Por ejemplo:

```
triangular 5 = 15
triangular 10 = 55
triangular 100 = 5050
```

4) `fibo n`, que recibe un entero `n` y devuelve el número correspondiente al lugar `n` de la sucesión de Fibonacci.
   Por ejemplo:

```
fibo 4 = 3
fibo 9 = 34
fibo 12 =144
```

   Los primeros 16 números de la sucesión de fibonacci son: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610.

Sobre listas... *(2 puntos)*

1) `ultimo lst`, que recibe una lista y responde con el último elemento de la lista.
   Por ejemplo.

```
ultimo [1,2,3] = 3
ultimo [True,True,True] = True
ultimo ['a','e','i','o','u'] = 'u'
```

2) `reversa lst`, que recibe una lista y devuelve la misma lista al revés.
   Por ejemplo.

```
reversa [1,2,3] = [3,2,1]
reversa [True,True,True] = [True,True,True]
reversa ['a','e','i','o','u'] = ['u','o','i','e','a']
```

3) `pertenece elm lst`, que recibe una lista `lst` de tipo `[a]` y `elm` un elemento de tipo `a` y responde si `elm` pertenece a `lst`.
   Por ejemplo.

```
elemento 1 [1,2,3] = True
elemento False [True,True,True] = False
elemento 'r' ['a','e','i','o','u'] = False
```

2. Pensar correcto es lo que hago...

Utiliza tu ingenio y lo que has aprendido sobre haskell para implementar la función que resuelve los siguientes problemas.

¿Perfecto, abundante o deficiente? *(1 punto)*

Nicómaco de Gerasa (**no** el hijo o padre de Aristoteles) fué un matématico que dió la siguiente clasificación:

> *Sea **n** un número entero positivo y **s** su suma alícuota, **n** es abundante si s>n, deficiente si n<s y perfecto si n=s.*

Por otra parte la suma alícuota de un número es la suma de todos sus factores, por ejemplo, la suma alócuota de 10 es 8 porque sus factores son 1, 2 y 5, entonces 10 es un número deficiente mientras que 6 es un número perfecto ya que sus factores son 1, 2 y 3.
Define una función `nicomaco n` con firma

```
nicomaco :: Int -> Categoria
```

donde categoría lo definimos como el tipo

```
data Categoria = Perfecto | Deficiente | Abundante
```

Luhn *(1 punto)*

El algoritmo de Luhn, creado por el científico Hans Peter Luhn, sirve para validar números de identificación, por ejemplo el de las tarjetas bancarias. El algoritmo requiere considerar a cada dígito como un número independiente y consiste de los siguientes pasos:

1. Multiplicar por dos un número sí y otro no comenzando de izquierda a derecha.
2. Restar 9 a cada número mayor a 9 para dejar números de una sola cifra.
3. Sumar todos los números.
4. Si el resultado es divisible por 10, es un número válido.

Por ejemplo, si tenemos el número 3379513561108795, debemos considerar cada dígito como un número independiente, algo así:

| 3 | 3 | 7 | 9 | 5 | 1 | 3 | 5 | 6 | 1 | 1 | 0 | 8 | 7 | 9 | 5 |

Y despúes del primer paso obtenemos este número:

| 6 | 3 | 14 | 9 | 10 | 1 | 6 | 5 | 12 | 1 | 2 | 0 | 16 | 7 | 18 | 5 |

Aplicando el tercer paso:

| 6 | 3 | 5 | 9 | 1 | 1 | 6 | 5 | 3 | 1 | 2 | 0 | 7 | 7 | 9 | 5 |

El resultado de sumar todos los números es 70 que es divisible por 10, por lo tanto es un número válido.
Implementa la función `luhn 1st` con firma

```
luhn :: [Int] -> Bool
```

que corrobore si un número es válido o no.

Conjetura de Collatz. *(1 punto)*

*Sea **n** un número entero positivo, si se le aplican los dos siguientes pasos repetidamente, eventualmente llegará a 1:*

*Si **n** es par divídelo entre dos, en otro caso multiplicalo por 3 y sumale 1.*

La anterior es una observación que el matemático Lothar Collatz presentó en el año de 1937 y es fácil corroborar que los primeros números naturales cumplen con la regla, pero... ¿**Todos** los números enteros positivos cumplen esto?, bueno pues esta pregunta aunque en principio parece fácil, a día de hoy no tiene respuesta y es conocida como la conjetura de Collatz.
Veamos el ejemplo para el número 12:

Paso	resultado
0	12
1	6
2	3
3	10
4	5
5	16
6	8
7	4
8	2
9	1

Después de 9 pasos el 12 se convirtió en 1.
Este es uno de los problemas sin respuesta más famosos de las matemáticas y al parecer va a seguir así por un buen rato.
Para este ejercicio debes escribir dos funciones:

1) `pasosCollatz n` que recibe un número entero positivo y responde con el número de pasos que se deben dar para convertir a n en 1.
   `pasosCollatz 12` debe responder `9`.

2) `listaCollartz n` que recibe un número entero positivo y responde con una lista con las transformaciones que va sufriendo n hasta convertirse en 1.
   `listaCollartz 12` debe responder `[12,6,3,10,5,16,8,4,2,1]`.

3. Expresiones aritméticas.

Considera el siguiente tipo de dato:

```
data EA = N Int | Positivo EA | Negativo EA | Suma EA EA
                | Resta EA EA | Mult EA EA | Div EA EA | Mod EA EA | Pot EA EA
```

Define las siguientes funciones que crean la Expresion Aritmética (`EA`) según sea el caso, respeta la firma *(1 punto)*

1) `creaSumaEA :: Int -> Int -> EA`
2) `creaRestaEA :: Int -> Int -> EA`
3) `creaMultEA :: Int -> Int -> EA`
4) `creaDivEA :: Int -> Int -> EA`
5) `creaModEA :: Int -> Int -> EA`
6) `creaPotEA :: Int -> Int -> EA`

Por ejemplo:

```
creaSumaEA 5 -10 = Suma (N 5) (Negativo (N (-10)))
creaPotEA 12 12 = Pot (N 12) (N 12)
```

A continuación elimina el `deriving Show` de la definición de `EA` y escribe una instancia de la clase `Show` para `EA` *(1 punto)* de manera que los resultados para los ejemplos anteriores ahora sean:

```
creaSumaEA 5 -10 = 5 + (-10)
creaModEA 12 12 = 12 ^ 12
```

Por último implementa la evaluación de las relaciones mayor y menor que para los constructores *unarios* de `EA` *(1 punto)*, el nombre y firma para cada una es

1) `menorque :: EA -> EA -> Bool`
2) `mayorque :: EA -> EA -> Bool`

Notas.

- Las instrucciones para entregar las prácticas están en la página del curso, si aún no sabes cuál es haz click aquí.
- Todo lo necesario para resolver esta práctica se revisará en las siguientes clases, lleva dudas a la clase del laboratorio, esta práctica ya no es de chocolate.
- Se responden dudas en el correo pero de preferencia enviame mensaje por Telegram. Puedes acceder a mi contacto haciendo click aqui.

No dudes en contactarme ante cualquier duda, aunque no sea de la materia.