Lógica Computacional

Práctica 1 - Fundamentos de Haskell

Semestre 2017-2

Fecha de inicio: 1 de febrero de 2017

Fecha de término: 15 de febrero de 2017

Instrucciones

- Completar de manera clara y ordenada las funciones del archivo `practica1.hs`.
- Para tener derecho a calificación, la práctica debe ejecutarse sin errores ni advertencias. No está permitido utilizar primitivas de Haskell que resuelvan directamente los ejercicios, ni modificar la firma de ninguna función.
- La entrega es por **equipos de 3 a 4 integrantes**. Seguir los lineamientos especificados en: `http://sites.ciencias.unam.mx/logica-computacional-2017-2/laboratorio/lineamientos`.

Ejercicios

1. **Derivadas**

   Completar la función `deriva :: Int -> Int -> Int -> Int` para que encuentre la derivada $f'(v)$ de la ecuación $f(x) = ax^2+bx+c$. El primer argumento de la función corresponderá al valor de a, el segundo argumento al valor de b, el tercero al valor de c y el cuarto al valor de v, que es el valor de evaluación. Ejemplos

   ```
   > deriva 1 7 2 9
   25
   > deriva 10 3 8 2
   43
   > deriva 11 10 10 11
   252
   ```

2. **Cilindros**

   Completar la función `areaCilindro :: Float -> Float -> Float` para que pueda calcular el área y otra para el volumen `volumenCilindro :: Float -> Float -> Float` de un cilindro dado el diámetro y la altura como primer y segundo parámetro respectivamente. Fórmulas:

   $$Área = 2\pi r\,(r + h)$$

   $$Volumen = \pi r^2 h$$

   donde r y h son el radio y la altura respectivamente. Ejemplos:

```
> areaCilindro 10.0 20.0
785.37
> areaCilindro 76.0 20.0
13847.73
> volumenCilindro 40.0 125.66
157904.356
> volumenCilindro 82.0 100.0
528102.96
```

3. **Números y condiciones**

   Completar la función `aplicaOperacion :: Char -> Int -> Int -> Int` que recibe tres parámetros, el primero indica la operación que se va a realizar con los otros dos parámetros, las posibles operaciones son:

   `'s'` = devuelve el segundo parámetro
   `'t'` = devuelve el tercer parámetro
   `'a'` = suma
   `'r'` = resta
   `'p'` = multiplicación
   `'d'` = división entera
   `'e'` = potencia (el segundo parámetro elevado al tercero)

   Ejemplos:

```
> aplicaOperacion 's' 1 7
1
> aplicaOperacion 't' 2 9
9
> aplicaOperacion 'a' 1 8
9
> aplicaOperacion 'r' 3 5
-2
> aplicaOperacion 'p' 4 0
0
> aplicaOperacion 'd' 5 5
1
> aplicaOperacion 'e' 0 2
0
```

4. **Aproximación raíz cuadrada**

   Completar la función recursiva `raizEntera :: Int -> Int` para que calcule una aproximación con un número entero a la raíz cuadrada.

   Ejemplos:

```
> raizEntera 64
8
> raizEntera 63
7
> raizEntera 10
3
```

5. **Suma naturales**

   Completar la función `sumaNat :: Int -> Int` para que devuelva la suma de los primeros n naturales. Ejemplos:

```
> sumaNat 5
15
> sumaNat 123
7626
> sumaNat 1729
1495585
```

6. **Longitud números enteros**

   Completar la función recursiva `longitud :: Int -> Int` para que devuelva la longitud de un número entero. No es válido hacer conversiones a cadena o usar funciones que resuelvan directamente el ejercicios. Ejemplos:

```
> longitud 1
1
> longitud 17
2
> longitud 1729
4
```

7. **Funciones de orden superior**

   Definir las siguientes funciones:

   a) Completar la función `tribonaccies :: Int -> [Int]` para que regrese una lista con los n primeros números de tribonacci iniciando con 0, 0, 1. Usar `map` para implementar esta función. Ejemplos:

```
> tribonaccies 1
[0]
> tribonaccies 7
[0,0,1,1,2,4,7]
> tribonaccies 9
[0,0,1,1,2,4,7,13,24]
```

   b) Completar la función `elimDup :: [a] -> [a]` para que dada una lista elimina los elementos duplicados **adyacentes** de la misma, dejando únicamente una aparición de cada elemento. Usar `foldr` para implementar esta función.

   Ejemplos:

```
> elimDup [1,1,2,2,3,3,1,1,1,2,3]
[1,2,3,1,2,3]
> elimDup ‘‘bccaaaabb’’
‘‘bcab’’
> elimDup [1,7,2,9]
[1,7,2,9]
```

c) Completar la función `maximal :: (a -> a -> a) -> [a] -> a` para que dada una función de comparación y una lista como parámetros, devuelva el elemento maximal de la lista para esa función de comparación. Usar `foldl` para implementar esta función.

Ejemplos:

```
> maximal max [1,7,2,9]
9
> maximal min [1,7,2,9]
1
```

8. **Operaciones sobre listas**

Redefinir o definir cada una de las siguientes operaciones sobre listas. No está permitido usar funciones de Haskell que resuelvan directamente los ejercicios. Apoyarse de la técnica de cazamiento de patrones de ser necesario.

a) Redefinir la función `reverse` completando la función `reversa :: [a] -> [a]` para que regrese la reversa de una lista.

b) Redefinir la función `filter` completando la función `filtra :: (a -> Bool) -> [a] -> [a]` que devuelve una lista con los elementos que cumplen con el predicado recibido como parámetro.

c) Completar la función `unicaVez :: [a] -> [a]` para que tome una lista como parámetro y regrese otra lista con los elementos que aparecen una única vez en la original.

d) Completar la función `apariciones :: [a] -> [(Int,a)]` que recibe una lista y regresa una lista de pares `(k, x)`, donde `k` es el número de apariciones consecutivas de `x` en la lista recibida.

e) Completar función `empareja :: [a] -> [(a,a)]` para que dada una lista `l` de la forma `[a0,a1,a2 ... am,an,ao,ap]` devuelva una lista e pares cuyos elementos son `(a0,ap) (a1,ao) (a2 an)`. Se debe asegurar que la lista recibida siempre siempre sea de longitud par.

9. **Listas por comprensión**

Reescribir las siguientes listas como listas por comprensión.

- `[0, 1, 3, 7, 15, 31, 63]`
- `[(3,4), (7,8), (11,12), (15,16), ...]`