Programación Declarativa 2020-2
Facultad de Ciencias UNAM
Tarea de Reposición

Favio E. Miranda Perea Javier Enríquez Mendoza

Fecha de entrega: 4 de mayo de 2020

Funciones de Orden Superior (Tarea 2)

1. Define en Haskell las siguientes funciones haciendo uso de las funciones de orden superior `map` y `filter`:

   (a) `todos :: (a -> Bool)-> [a] -> Bool` que decide si todos los elementos de una lista cumplen un predicado.

   (b) `alguno :: (a -> Bool)-> [a] -> Bool` que decide si al menos un elemento de una lista cumple un predicado.

   (c) `toma :: (a -> Bool)-> [a] -> [a]` selecciona elementos de una lista mientas cumplan un predicado (equivalente a `takeWhile` del preludio).

   (d) `deja :: (a -> Bool)-> [a] -> [a]` elimina elementos de una lista mientas cumplan el predicado.

2. Define la función `altMap :: (a -> b)-> (a -> b)-> [a] -> [b]` que aplica alternadamente las funciones que recibe como argumentos a los elementos de la lista, en otras palabras a los elementos en posiciones pares les aplica la primera función mientras que a los elementos en posiciones impares les aplica la segunda. Por ejemplo

   ```
   > altMap (+10) (+100) [0,1,2,3,4]
   [10,101,12,103,14]
   ```

3. Utiliza la función `altMap` definida en el inciso anterior para definir el algoritmo de Luhn que se utiliza para verificar la validez de los números de tarjetas bancarias. El algoritmo está definido como sigue:

   - Se considera cada dígito como un número independiente.
   - Comenzando de izquierda a derecha se multiplican por dos un número si y un número no.
   - Se resta 9 a cada número que sea mayor a 9, para dejar números de una sola cifra.
   - Se suman todos los números.
   - Si el resultado es divisible por 10, es un número válido.

   Por ejemplo, si tenemos el siguiente número:

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

Después del segundo paso del algoritmo resulta:

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 dígitos es 70 que es divisible por 10 por lo que es un número válido.

La función tiene le tipo `luhn :: [Int] -> Bool`

Programación Origami (Tarea 3)

Las funciones de esta sección deben definirse usando los operadores de plegado `foldr` o `foldl`.

1. Define la función `factorion :: Int -> Int`, que calcula el factorión de un número. El factorión se define como la suma de los factoriales de cada dígito del número. Por ejemplo:

$$\texttt{factorion}\ 145 = 1! + 4! + 5! = 145$$

2. Define la función `iflip :: Int -> Int` que toma un número natural n y regresa el número construido los dígitos de n en orden inverso. Por ejemplo:

$$\texttt{iflip}\ 1720 = 271$$

3. Definir la función `binarios :: [Int] -> [Int]` que recibe una lista de números y regresa una lista de números binarios asociados a los números originales.

$$\texttt{binarios}\ [1..4] = [0, 1, 10, 11, 100]$$

4. Definir la función `triangulares :: [Int] -> [Int]` que recibe una lista de números y regresa una nueva lista que contiene únicamente aquellos que son triangulares. Un número triangular es aquel que puede recomponerse en la forma de un triángulo equilátero.

$$\texttt{triangulares}\ [1..6] = [1, 3, 6]$$

Entrega

- Se pueden obtener hasta 4 puntos por tarea, los cuales se sumaran a la calificación obtenida en la tarea correspondiente.
- La tarea puede entregarse de forma individual o en parejas.
- Se tomará en cuenta tanto el estilo de programación como la complejidad de las soluciones para la calificación de la sección práctica.

- La tarea se entrega a través de Slack:
  - Todos los archivos deben tener al principio como comentario los nombres de los alumnos.
  - Si la tarea se realizo en parejas crear un canal privado con el ayudante y los miembros del equipo para la entrega.
  - Si se realizo de forma individual enviarla en un mensaje directo al ayudante.
  - **No** deben comprimir los archivos, en caso de tener mas de uno enviarlos por separado.