Programación Funcional -

Práctico de Lenguaje Haskell

Usando el código del **Apéndice A**, que se encuentra al final de este documento,, generar el archivo ***PracHaskell.hs*** en el folder **PIII\PF_Haskell\1_PracLab**, usando un editor adecuado para la carga de código. Este archivo compila correctamente en el intérprete de Haskell.

Resolver los ejercicios que se solicitan a continuación. A partir del ejercicio 4, las funciones desarrolladas deben agregarse al archivo ***PracHaskell.hs***

EJERCITACIÓN PRELIMINAR

Ejercicio 1: podemos utilizar el interprete de Haskell (ghci) para trabajar de manera interactiva y familiarizarnos con el lenguaje antes de comenzar a escribir y compilar código. El objetivo de esta ejercitación preliminar es comenzar a conocer algunas funciones predefinidas sobre tipos de datos tales como listas, tuplas, strings.

Entrar en el interprete de Haskell y para las funciones listadas abajo

a) pedir información de cada una de ellas (usar el comando :i)
b) a partir de la información obtenida, evaluar expresiones donde dichas funciones se invoquen con diferentes argumentos.
c) Decir que hace cada función y describir su dominio:
   - con palabras.
   - usando notación de conjuntos: (Ejemplo: dom(x)={x | x =....∧..})
   - usando notación de tipos de Haskell.

1.1. funciones y/o operaciones para la manipulación de **listas**:
Modulo: Data.List
https://hackage.haskell.org/package/base-4.14.0.0/docs/Data-List.html

head, last, tail, init, null, length, reverse, take, drop, maximum, minimum, sum, concat, product, elem, replicate
: , ++, !!

1.2. funciones definidas para **tuplas:**
(*Ver Nota) Modulo extra: Data.Tuple (https://hackage.haskell.org/package/base-4.14.0.0/docs/Data-Tuple.html)

fst, snd, swap*, zip, unzip, zipWith.

1.3. funciones definidas para **strings:**

*Modulo extra:Data.String (http://hackage.haskell.org/package/base-4.14.0.0/docs/Data-String.html)
lines*, words*, unlines*, unwords*.

* **Nota**: Estas funciones no se encuentran en el módulo inicialmente cargado por el interprete de Haskell (Prelude). Es necesario importar el módulo donde están definidas. Para hacer esto colocar:

```
Prelude> import Modulo
```

Ejemplo, para cargar modulo de Tuplas:

```
Prelude> import Data.Tuple
```

1.4. funciones predefinidas: **const, id, flip, curry, uncurry**

Se proponen algunas expresiones a evaluar para entender las funciones mencionadas:

const	const 3 'a'	map (const 'a') [1,2,3]	
flip	flip (-) 16 3	flip (:) [1,2,3] 4	flip (++) [1] [2,3]
uncurry	uncurry (+) (2,3)	uncurry (:) (4, [1,2,3])	uncurry (++) ([4], [2,3])
curry	curry fst 2 3	curry (uncurry (+)) 2 3	

Ejercicio 2: ídem ejercicio 1, pero con las funciones de orden superior (combinadores) listadas a continuación. Se proponen algunos argumentos para usar en la invocación de las funciones.

función	**Argumentos**		
map	(+1) [1,2,3,4]	(+1) [ ]	(uncurry (+)) [(2,4),(5,8)]
filter	(>3) [2,2,3,5,6]	(=='a') ['s','a','v']	(==1) [ ]
any	(>3) [2,3,5,6]	(>13) [2,2,3,5,6]	(>3) [ ]
all	(>3) [2,3,5,6]	(>13) [2,2,3,5,6]	(>3) [ ]
concatMap	(:[]) [2,3,4]	(tail) [[1,2],[3],[5,4]]	(tail) [ ]

Ejercicio 3: ídem ejercicio 1, pero con las funciones de plegado definidas para listas:

función	**Argumentos**			
foldr	(+) 0 [3,10,2]	(*) 1 [3,3,3]	(++) [] [[2],[3]]	(+) 0 [ ]
foldr1	(+) [3,10,2]	(*) [3,3,3]	(++) [[2],[3]]	(+) [ ]
foldl	(+) 0 [3,10,2]	(*) 1 [3,3,3]	(++) [] [[2],[3]]	(+) 0 [ ]
foldl1	(+) 0 [3,10,2]	(*) 1 [3,3,3]	(++) [] [[2],[3]]	(+) [ ]

USO DE COMBINADORES BASICOS

Ejercicio 4: Definir funciones en Haskell que resuelvan los siguientes problemas.

1. Retornar la longitud de una lista sin usar la función primitiva length.
2. Dada una lista y un elemento, contar cuantas veces se encuentra dicho elemento en la lista.

3. Dada una lista y un elemento, retornar un valor booleano que indique si se encuentra el elemento en la lista.
4. Dada una lista de entrada, generar otra de doble longitud, donde cada elemento figure dos veces. Ud. elige el orden que tendrá su solución.
5. Dadas dos secuencias, retorne un 1 si la primera tiene mayor cantidad de elementos que la segunda, un en caso 2 contrario y un 0 si tienen igual cantidad de elementos.
6. Tomando como entrada una lista de átomos numéricos, retornar el producto de sus componentes. Si la lista esta vacía, debe retornar 0 (cero).
7. Tomando como entrada una lista de dos componentes [a, b], retornar el producto de a * b por sumas sucesivas
8. Dada una lista, retornar el reverso de la misma, sin usar la función predefinida reverse
9. Tomando como entrada una lista de listas, devolver otra lista donde cada lista interior esté invertida
   (Ejemplo:Entrada--> [[1,2,3],[4,5],[2]] Salida --> [[3,2,1],[5,4],[2]].)

DESARROLLO DE FUNCIONES RECURSIVAS

Ejercicio 5: Definir funciones recursivas que resuelvan los siguientes problemas:

5.1. Resolver los enunciados del ejercicio 4.

5.2. Para el caso del ej. 4.4, analizar las diferencias en el código a desarrollar si en el enunciado se pidiera un resultado particular en el orden en que deben entregarse los elementos en el resultado.

Ejemplo:

Entrada	**Salida**
> f1 [1,2,3]	> [1,2,3,1,2,3]
> f1 [1,2,3]	> [1,2,3,3,2,1]
> f1 [1,2,3]	> [1,1,2,2,3,3]

5.3. Tome como argumento un numero natural n y retorne una lista con los n primeros números naturales pares.

Ejemplo:

Entrada	**Salida**
> f 3	> [2,4,6]
> f 7	> [2,4,6,8,10,12,14]

5.4. Tome como argumentos 2 números naturales **x** y **n** y retorne una lista con los **n** primeros múltiplos de **x**.

Ejemplo:

Entrada	**Salida**
> f 2 4	> [2,4,6,8]
> f 10 3	> [10,20,30]

DESARROLLO DE FUNCIONES RECURSIVAS CON ACUMULADORES

Ejercicio 6: Definir funciones ***recursivas con acumuladores*** en Haskell que resuelvan los enunciados de los ejercicios 4 y 5.

USO DE PATRONES DE RECURSION PREDEFINIDOS

Ejercicio 7: Definir funciones en Haskell que resuelvan los enunciados de los ejercicios 4 y 5 usando las funciones de primer orden: ***foldl, foldr, foldr1, foldl1.*** Usar la que considere mas conveniente, o usar más de una si son igualmente convenientes. Justifique en cada caso su elección.

Apéndice A

```
-----------------------------------------------
-- Module      : PracHaskG
-- Developer   :
-- Maintainer  :
-- Stability   :  experimental
-- Portability :  experimental
--
-- Practico Haskell
-----------------------------------------------
-- | Este modulo contiene las definiciones de las funciones
solicitadas en el practico Haskell

module PracHaskG  (
  -- *** Ejercicio 4 - Uso de combinadores básicos sobre
funciones primitivas
   f4longi,f4longi', f4longi'', f4cuentaE -- Completar
  -- *** Ejercicio 5 — Funciones definidas recursivamente
  -- Completar (f6genPares,…)
  -- *** Ejercicio 6 - Funciones recursivas con
acumuladores
  -- *** Ejercicio 7 - Uso de patrones de recursión
predefinidos
  -- (foldr: catamorfismos sobre estructuras para
operadores asociativos a derecha)
  -- (foldl: catamorfismos sobre estructuras para
operadores asociativos a izquierda)
  ) where

-- | Calcula la longitud de una lista (f41)
f4longi :: [a] -> Integer
f4longi l= sum (map (const 1) l)


-- | f4longi usando composicion (f41)
f4longi' :: [a] -> Integer
f4longi' l= (sum . (map  (const 1))) l

-- | f4longi pointfree (f41)
f4longi'' :: [a] -> Integer
f4longi'' = (sum . (map  (const 1)))

-- | Cuenta cantidad de elementos iguales al 1er. arg. en
una lista (f42)
f4cuentaE ::  (Eq a) => a -> [a] -> Integer
f4cuentaE e l = f4longi (filter (==e) l)
```