Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3661 - Taller de Lenguajes de Programación I
Enero-Marzo 2015

Quiz Haskell

(20 puntos)

Nombre:

Carnet:

Atención: en todas sus respuestas debe utilizar *exclusivamente* funciones disponibles en el Preludio Haskell. Si Ud. no sabe si una función está o no en el Preludio Haskell, puede preguntarlo *públicamente* para recibir un **si** o **no** como respuesta y **nada** más.

1. **(5 puntos)** Infiera el tipo de datos más general posible para cada una de las siguientes expresiones:

   a) `294 / 7`

   ```
   Fractional a => a
   ```

   b) `length "Foo"`

   ```
   Int
   ```

   c) `map not . map even`

   ```
   Integral a => [a] -> [Bool]
   ```

   d) `((True,42),pred)`

   ```
   (Enum a, Num t) => ((Bool, t), a -> a)
   ```

   e) `y f = f (y f)`

   ```
   (a -> a) -> a
   ```

   f) `uncurry . curry`

   ```
   ((a, b) -> c) -> (a, b) -> c
   ```

   g) `[id,pred,fromInteger,signum]`

   ```
   [Integer -> Integer]
   ```

   h) `map (map succ)`

   ```
   Enum b => [[b]] -> [[b]]
   ```

   i) `map putStr [ w | w <- words "May The Force be with you" ]`

   ```
   [IO ()]
   ```

   j) `map (+)`

   ```
   Num a => [a] -> [a -> a]
   ```

2. **(5 puntos)** Evalúe cada una de las siguientes expresiones:

   a) `take 5 [2,7..]`

   ```
   [2,7,12,17,22]
   ```

   b) `zip "WAT" (repeat '?')`

   ```
   [('W','?'),('A','?'),('T','?')]
   ```

   c) `foldr (:) [1,2,3] [4,5,6]`

   ```
   [4,5,6,1,2,3]
   ```

d) `foldl (flip (:)) [1,2,3] [4,5,6]`

`[6,5,4,1,2,3]`

e) `zipWith map [id,pred,fromInteger,signum] (repeat [1..3])`

`[[1,2,3],[0,1,2],[1,2,3],[1,1,1]]`

3. **(5 puntos)** Escriba la función `partition :: (a -> Bool) -> [a] -> ([a], [a])`, tal que aplique un predicado a cada elemento de una lista, separando aquellos que cumplen el predicado, de aquellos que no lo cumplen, *preservando* el orden relativo de los elementos. Por ejemplo

```
> partition even [1,4,2,2,3,6,7,4,2,6]
([4,2,2,6,4,2,6],[1,3,7])
```

Nota: para obtener los cinco (5) puntos *debe* escribir la función usando funciones de orden superior. Si la implanta usando recursión de cola, solamente obtendrá cuatro (4) puntos y si la implanta con recursión explícita, solamente obtendrá tres (3) puntos.

- Solución usando funciones de orden superior. En este caso es conveniente usar un `foldr` para que pueda operar sobre listas finitas e infinitas

```
partition :: (a -> Bool) -> [a] -> ([a],[a])
partition p xs = foldr (select p) ([],[]) xs
  where select p x (ts,fs) | p x = (x:ts,fs)
                           | otherwise = (ts, x:fs)
```

- Solución empleando recursión de cola. En este caso, sólo funcionará con listas finitas y es ineficiente por el uso de concatenaciones repetidas.

```
partition :: (a -> Bool) -> [a] -> ([a],[a])
partition p l = go ([],[]) l
  where go a       []     = a
        go (ts,fs) (x:xs) = if p x then go (ts ++ [x], fs) xs
```

- Solución empleando recursión explícita. En este caso funciona con listas finitas e infinitas.

```
partition :: (a -> Bool) -> [a] -> ([a],[a])
partition p []     = ([],[])
partition p (x:xs) = if p x then (x : ts, fs)
                            else (ts, x : fs)
                       where (ts,fs) = partition p xs
```

4. **(5 puntos)** Escriba la función `pascal :: [[Integer]]` que genere las infinitas filas del Triángulo de Pascal, esto es

```
> take 5 pascal
[[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
```

Si se tiene una fila cualquiera y se agrega un cero al principio, para luego tomar otra copia de la fila y agregar un cero al final, sumar elemento con elemento producirá la siguiente fila. Iterar la aplicación de esa función, conduce a una solución "obvia", pero un tanto ineficiente por el uso de la concatenación al final.

```
pascal = iterate (\row -> zipWith (+) (0:row) (row ++ [0])) [1]
```

Una solución no tan obvia (y que no voy a explicar) puede construirse sin usar concatenaciones

```
pascal = iterate f [1]
      where f r@(c:cs)        = 1:z (+) r cs
            z g xs []         = xs
            z g (x:xs) (y:ys) = g x y:z g xs ys
```