Lógica Computacional 2015-2
Práctica 0

Favio E. Miranda Perea
José Manuel Reyes Snyder
C. Moisés Vázquez Reyes

28 de enero de 2015
Facultad de Ciencias UNAM

Esta práctica tiene como finalidad introducirlos a la programación funcional, el lenguaje a usar es Haskell. Todas las soluciones a entregar deben ser recursivas, deben tener comentarios, indicando brevemente qué hace. En Haskell se ponen comentarios de linea comenzando con `--`.

1. Ejercicios:

Considera el siguiente tipo de dato para representar números naturales:

```
data Nat = Cero | Suc Nat deriving Show
```

1. `suma :: Nat->Nat->Nat`

   Dados dos números naturales, nos devuelve su suma.

   Ejemplos:

   - `*Main>suma Cero (Suc Cero)`
     `Suc Cero`
   - `*Main>suma (Suc Cero) (Suc $ Suc Cero)`
     `Suc (Suc (Suc Cero))`

2. `prod :: Nat->Nat->Nat`

   Dados dos números naturales, nos devuelve su producto.

   Ejemplos:

- `*Main>prod Cero (Suc Cero)`
  `Cero`
- `*Main>prod (Suc $ Suc Cero) (Suc $ Suc Cero)`
  `Suc (Suc (Suc (Suc Cero)))`

3. `mayorQue :: Nat->Nat->Bool`

   Dados dos números naturales, nos dice si el primero es mayor que el segundo.

   Ejemplos:

   - `*Main>mayorQue Cero (Suc Cero)`
     `False`
   - `*Main>mayorQue (Suc Cero) Cero`
     `True`

4. `menorQue :: Nat->Nat->Bool`

   Dados dos números naturales, nos dice si el primero es menor que el segundo.

   Ejemplos:

   - `*Main>menorQue Cero (Suc Cero)`
     `True`
   - `*Main>menorQue (Suc Cero) Cero`
     `False`

5. `igual :: Nat->Nat->Bool`

   Dados dos números naturales, nos dice si son iguales.

   Ejemplos:

   - `*Main>igual Cero (Suc Cero)`
     `False`
   - `*Main>igual (Suc Cero) (Suc Cero)`
     `True`

6. `power :: Int -> Int -> Int`

   Dados x, y enteros, la función debe calcular x^y

   Ejemplos:

- `*Main>power 2 3`
  `8`
- `*Main>power 10 0`
  `1`

7. `power2 :: Int -> Int -> Int`

   Implementa la potencia de enteros bajo éste esquema:

$$n^k = \begin{cases} (n^2)^{k/2} & \text{si } k \text{ es par} \\ n * (n^{k-1}) & \text{si } k \text{ es impar} \end{cases}$$

   Utiliza la función `div :: Int->Int->Int` para hacer la división de enteros

8. `reversa :: [a] -> [a]`

   Toma una lista y nos devuelve su reversa.

   Ejemplos:

   - `*Main>reversa [2,3,5,1]`
     $[1, 5, 3, 2]$
   - `*Main>reversa [1,4,2]`
     $[2, 4, 1]$

9. `sumal :: [Int] -> Int`

   Toma una lista de números enteros y nos devuelve la suma de sus elementos.

   Ejemplos:

   - `*Main>sumal [2,3,-5,1]`
     `1`
   - `*Main>sumal [1,4,2]`
     `7`

10. `toma :: Int -> [a] -> [a]`

    Toma los primeros n elementos de una lista, $n \geq 0$

    Ejemplos:

    - `*Main>toma 2 [2,3,5,1]`
      $[2, 3]$

- `*Main>toma 10 [1,4,2]`
  $[1, 4, 2]$

11. `tira :: Int -> [a] -> [a]`

    Tira los primeros n elementos de una lista, $n \geq 0$

    Ejemplos:

    - `*Main>tira 2 [2,3,5,1]`
      $[5, 1]$
    - `*Main>tira 10 [1,4,2]`
      []

12. `cuantas :: Eq a => a -> [a] -> Int`

    Toma un elemento x y una lista ℓ, nos dice cuántas veces aparece x en ℓ

    Ejemplos:

    - `*Main>cuantas 1 [1,2,3,1,4]`
      2
    - `*Main>cuantas 7 [3,4,2,1]`
      0

13. `frec :: Eq a => [a] -> [(a, Int)]`

    Dada una lista ℓ, nos devuelve la siguiente lista:

    $$\{(x, y) | x \in \ell,\ y = \text{el número de veces que aparece } x \text{ en } \ell\}$$

    Ejemplos:

    - `*Main>frec [1,2,3,1,4]`
      $[(2, 1), (3, 1), (1, 2), (4, 1)]$
    - `*Main>frec [11,8,3,1,4,4]`
      $[(11, 1), (8, 1), (3, 1), (1, 1), (4, 2)]$

14. `unaVez :: Eq a => [a] -> [a]`

    Dada una lista ℓ, nos devuelve a los elementos que aparecen sólo una vez en ℓ

    Ejemplos:

- `*Main>unaVez [1,2,3,1,4]`
  $[2, 3, 4]$
- `*Main>unaVez [11,8,3,1,4,4]`
  $[11, 8, 3, 1]$

15. **(EXTRA: 2 puntos)** `primosHasta :: Int ->[Int]`

    Dada un entero n, nos regresa una lista con todos los números primos entre 2 y n

    Ejemplos:

    - `*Main>primosHasta 5`
      $[2, 3]$
    - `*Main>primosHasta 11`
      $[2, 3, 5, 7, 11]$

2. Hints:

- Aunque ya lo saben, Internet es su mejor amigo.
- La función `elem` de Haskell es muy útil, nos dice cuándo un elemento pertenece a una lista. Por ejemplo `elem 1 [2,1] = True`, `elem 1 [] = False`
- El operador `(!!)` nos regresa un elemento en la posición n de una lista. Por ejemplo: `[1,2,3] !! 2 = 3`, `"Holai! 3 = 'a'`. Al igual que en Java los índices comienzan en 0.
- Éste es de los mejores sitios para aprender: *http://aprendehaskell.es/*