

Prácticas: Paradigmas de Lenguajes de Programación

Zamboni, Gianfranco

1 de febrero de 2018

Índice

1. Práctica 0	1
1.1. Ejercicio 2	2
1.2. Ejercicio 3	2
1.3. Ejercicio 4	2
1.4. Ejercicio 5	3

1. Práctica 0

Ejercicio 1

```
null :: Foldable t => t a -> Bool
-- Indica si una estructura está vacía. El tipo a debe ser de la
  ↳ clase Foldable, esto es, son tipos a los que se les puede aplicar
  ↳ la función foldr. La notación "t a" indica que es un tipo
  ↳ paramétrico, es decir, un tipo t que usa a otro tipo a, por
  ↳ ejemplo, si le pasamos a la función una lista de enteros,
  ↳ entonces a = Int y t = [Int]

head :: [a] -> a
-- Devuelve el primer elemento de una lista.

tail :: [a] -> [a]
-- Devuelve los últimos elementos de una lista (todos los elementos,
  ↳ salvo el primero).

init :: [a] -> [a]
-- Devuelve los primeros elementos de una lista (todos los elementos
  ↳ salvo el último).

last :: [a] -> a
-- Devuelve el último elemento de una lista.

take :: Int -> [a] -> [a]
-- Devuelve los primeros n elementos de una lista

drop :: Int -> [a] -> [a]
-- Devuelve los últimos n elementos de una lista

(++ ) :: [a] -> [a] -> [a]
-- Concatena dos listas

concat :: Foldable t => t [a] -> [a]
-- Concatena todas las listas de un contenedor de listas que soporte
  ↳ la operación foldr.

(!!) :: [a] -> Int -> a
-- Dado una lista L y un entero N, devuelve el elemento de L que se
  ↳ encuentra en la N-ésima posición. La numeración comienza desde 0.

elem :: (Eq a, Foldable t) => a -> t a -> Bool
-- Dada una estructura T que soporta la operación foldr y que
  ↳ almacene elementos del tipo a que puedan ser comparados por medio
  ↳ de la igualdad y dado un elemento A de ese tipo, indica si A
  ↳ aparecen en T.
```

1.1. Ejercicio 2

```
valorAbsoluto :: Float -> Float
valorAbsoluto x | x < 0      = -x
                | otherwise = x

bisiesto :: Int -> Bool
bisiesto x = (x `mod` 4) == 0

factorial :: Int -> Int
factorial 1 = 1
factorial x = x * factorial (x-1)

cantDivisoresPrimos :: Int -> Int
cantDivisoresPrimos x =
  length (filter (\x -> length (divisores x) == 2) (divisores x))
  where divisores x = [ y | y <- [1..x], x `mod` y == 0 ];
```

1.2. Ejercicio 3

```
inverso :: Float -> Maybe Float
inverso 0 = Nothing
inverso x = Just (1/x)

aEntero :: Either Int Bool -> Int
aEntero (Left x) = x
aEntero (Right x) | x == True = 1
                  | otherwise = 0
```

1.3. Ejercicio 4

```
limpiar :: String -> String -> String
limpiar xs ys = [ y | y <- ys, not(elem y xs) ]

difPromedio :: [Float] -> [Float]
difPromedio xs = map (\y -> y - promedio xs) xs
  where promedio xs = (sum xs) / (genericLength xs)

todosIguales :: [Int] -> Bool
todosIguales xs = foldr (\y rec -> ((length xs == 1) || (y == (head
  ↪ xs)))) && rec) True xs
```

1.4. Ejercicio 5

```
data AB a = Nil | Bin (AB a) a (AB a)

vacioAB :: AB a -> Bool
vacioAB Nil = True
vacioAB (Bin _ _ _) = False

negacionAB :: AB Bool -> AB Bool
negacionAB Nil = Nil
negacionAB (Bin l x r) = Bin (negacionAB l) (not x) (negacionAB r)

productoAB :: AB Int -> Int
productoAB Nil = 1
productoAB (Bin l x r) = x * (productoAB l) * (productoAB r)
```