Informática de 1º de Matemáticas (Grupo 1) (12–Febrero–2010)

Ejercicio 1 (2.5 puntos) Definir la función

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

tal que `(diferencias xs)` es la lista de las diferencias entre los elementos consecutivos de `xs`. Por ejemplo,

```
diferencias [5,3,8,7] ==> [2,-5,1]
```

Escribir una definición por recursión (`diferenciasR`) y otra por comprensión (`diferenciasC`).

Solución: La definición por recursión es

```
diferenciasR :: Num a => [a] -> [a]
diferenciasR []         = []
diferenciasR [_]        = []
diferenciasR (x1:x2:xs) = (x1-x2) : diferenciasR (x2:xs)
```

La definición por recursión puede simplificarse

```
diferenciasR' :: Num a => [a] -> [a]
diferenciasR' (x1:x2:xs) = (x1-x2) : diferenciasR' (x2:xs)
diferenciasR' _          = []
```

La definición por comprensión es

```
diferenciasC :: Num a => [a] -> [a]
diferenciasC xs = [a-b | (a,b) <- zip xs (tail xs)]
```

Ejercicio 2 (2.5 puntos) Definir la función

```
producto :: [[a]] -> [[a]]
```

tal que `(producto xss)` es el producto cartesiano de los conjuntos `xss`. Por ejemplo,

```
*Main> producto [[1,3],[2,5]]
[[1,2],[1,5],[3,2],[3,5]]
*Main> producto [[1,3],[2,5],[6,4]]
[[1,2,6],[1,2,4],[1,5,6],[1,5,4],[3,2,6],[3,2,4],[3,5,6],[3,5,4]]
*Main> producto [[1,3,5],[2,4]]
[[1,2],[1,4],[3,2],[3,4],[5,2],[5,4]]
*Main> producto []
[[]]
```

Solución:

```
producto :: [[a]] -> [[a]]
producto []       = [[]]
producto (xs:xss) = [x:ys | x <- xs, ys <- producto xss]
```

Ejercicio 3 (2.5 puntos) Definir el predicado

```
comprueba :: [[Int]] -> Bool
```

tal que tal que (`comprueba xss`) se verifica si cada elemento de la lista de listas xss contiene algún número par. Por ejemplo,

```
comprueba [[1,2],[3,4,5],[8]]  ==>  True
comprueba [[1,2],[3,5]]        ==>  False
```

Solución: Una definición por recursión es

```
compruebaR :: [[Int]] -> Bool
compruebaR [] = True
compruebaR (xs:xss) = tienePar xs && compruebaR xss
```

donde (`tienePar xs`) se verifica si `xs` contiene algún número par. Por ejemplo,

```
tienePar [3,4,5]  =>  True
tienePar [3,7,5]  =>  False
```

```
tienePar  :: [Int] -> Bool
tienePar []     = False
tienePar (x:xs) = even x || tienePar xs
```

Una definición por comprensión es

```
comprueba:: [[Int]] -> Bool
comprueba xss = and [or [even x | x <- xs] | xs <- xss]
```

Ejercicio 4 (2.5 puntos) Definir la función

```
pertenece :: Ord a => a -> [a] -> Bool
```

tal que (`pertenece x ys`) se verifica si `x` pertenece a la lista ordenada creciente, finita o infinita, `ys`. Por ejemplo,

```
pertenece 22 [1,3,22,34]  ==>  True
pertenece 22 [1,3,34]     ==>  False
pertenece 23 [1,3..]      ==>  True
pertenece 22 [1,3..]      ==>  False
```

Solución: Una definición por recursión es

```
pertenece :: Ord a => a -> [a] -> Bool
pertenece _ []                 = False
pertenece x (y:ys) | x >  y    = pertenece x ys
                   | x == y    = True
                   | otherwise = False
```

Otra definición sin recursión es

```
pertenece' :: Ord a => a -> [a] -> Bool
pertenece' x ys = elem x (takeWhile (<= x) ys)
```