



Práctica Haskell

1. Definir las siguientes funciones en forma recursiva:

- a) *borrarUltimo* que dada una lista borra el último elemento de la lista. No utilizar *reverse*, ni *tail*.
- b) *collect* :: $[(k, v)] \rightarrow$ toma una lista de pares (clave, valor) y asocia cada clave única con todos los valores con los que estaba apareada originalmente. Por ejemplo: *collect*
- c) *serie* que se comporta de la siguiente manera: *serie* [1, 2, 3] = [[], [1], [1, 2], [1, 2, 3]] Dar su tipo más general.
- d) *paresIguales* :: $\text{Int} \rightarrow \text{Int} \rightarrow \text{Int} \rightarrow \text{Int} \rightarrow \text{Bool}$ toma 4 números enteros y retorna *True* si de dos en dos son iguales (en cualquier orden), en los demás casos retorna *False*. Por ejemplo:
paresIguales 3 1 1 2 = *False* *paresIguales* 3 1 3 1 = *True* *paresIguales* 3 3 1 1 = *True*
paresIguales 3 1 1 3 = *True*
- e) *isosceles* :: $\text{Int} \rightarrow \text{Int} \rightarrow \text{Int} \rightarrow \text{Bool}$ que dadas la longitud de los lados de un triángulo nos dice si es un triángulo isósceles.
- f) *ror* que dada una lista *xs* y un entero *n*, tal que $n \leq \text{length } xs$, rota los primeros *n* elementos de *xs* a la derecha: *ror* 3 [1, 2, 3, 4, 5] = [4, 5, 1, 2, 3]. Definir una versión recursiva de *ror*, sin usar *drop*, *take* ni *tail*.
- g) *upto* :: $\text{Int} \rightarrow \text{Int} \rightarrow [\text{Int}]$ que dado dos números enteros *n* y *m* devuelve la lista [n, n + 1, n + 2, ..., m] en caso que $n \leq m$ y la lista [] en otro caso. No usar listas por comprensión.
- h) *eco* que devuelve la cadena obtenida a partir de la cadena *xs* repitiendo cada elemento tantas veces como indica su posición. No usar listas por comprensión.
Por ejemplo: *eco* "hola" = "hoo111aaaa"

2. Definir usando listas por comprensión las funciones:

- a) *cambios* :: $[a] \rightarrow [\text{Int}]$, que dada una lista, devuelve la lista de los índices en que la lista cambia. Es decir, dada la lista *s* retorna la lista con los *i* tal que $s_i \neq s_{i+1}$
cambios [1, 1, 1, 3, 3, 1, 1] = [2, 4]
- b) *oblongoNumber* :: $[\text{Int}]$ que genera la lista de los números oblongos. Un número es oblongo si es el producto de dos naturales consecutivos. Por ejemplo, los números [2, 6, 12, 20, ...]
- c) *abundantes* :: $[\text{Integer}]$ que es la lista de todos los números abundantes. Un número natural *n* se denomina abundante si es menor que la suma de sus divisores propios. Por ejemplo, 12 y 30 son abundantes pero 5 y 28 no lo son. Por ejemplo *abundates* = [12, 18, 20, 24, 30, 36, ...]
- d) *eco* que devuelve la cadena obtenida a partir de la cadena *xs* repitiendo cada elemento tantas veces como indica su posición. No usar listas por comprensión. Por ejemplo: *eco* "hola" = "hoo111aaaa"
- e) *euler* :: $\text{Int} \rightarrow \text{Int}$ tal que *euler* *n* es la suma de todos los múltiplos de 3 ó 5 menores que *n*. Por ejemplo, *euler* 10 = 23. Puedes usar sin definir la función *sum* que suma los elementos de una lista.

- f) $expandir :: [Int] \rightarrow [Int]$ que reemplace en una lista de números positivos cada número n por n copias de sí mismo:

Ejemplo: $expandir [3, 4, 2] = [3, 3, 3, 4, 4, 4, 2, 2]$

3. Dar dos ejemplos de funciones que tengan los siguientes tipos:

- a) $(Int \rightarrow Int) \rightarrow (Bool \rightarrow Bool)$
- b) $Bool \rightarrow (Int \rightarrow Bool)$
- c) $Char \rightarrow Char$
- d) $Int \rightarrow (Int \rightarrow Bool) \rightarrow [Int]$
- e) $[a] \rightarrow (a \rightarrow [b]) \rightarrow [b]$
- f) $[[a]] \rightarrow (a \rightarrow Bool) \rightarrow [a]$
- g) $(a, b, c) \rightarrow Bool$
- h) $(a, b, c) \rightarrow Int \rightarrow c$
- i) $(a, a, a) \rightarrow Int \rightarrow a$

4. Dar el tipo de la siguiente funciones o expresiones:

- a) $foo1\ p = \text{if } p \text{ then } (p \wedge) \text{ else } (p \wedge)$
- b) $foo2\ x\ y\ z = x\ (y\ z)$
- c) $foo3\ x\ y\ z = x\ y\ z$
- d) $foo4\ x\ y\ z = x\ y : z$
- e) $foo5\ x\ y\ z = x : y\ z$
- f) $foo6\ x\ y\ z = x \mathrel{++} y\ z$
- g) $foo7\ a\ b = \text{if } b\ a \text{ then } head\ a \text{ else } []$
- h) $foo8\ a\ b = \text{if } b\ a \text{ then } a \text{ else } []$
- i) $foo9\ a\ b = \text{if } b\ a \text{ then } head\ (:a) \text{ else } (:[])$

5. Definir las siguientes funciones usando foldr:

- a) $map :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$ que dada una función y una lista, aplica la función a cada elemento de la lista.
- b) $filter :: (a \rightarrow Bool) \rightarrow [a] \rightarrow [a]$, que dado un predicado y una lista xs, devuelve una lista con los elementos de xs que satisfacen el predicado.
- c) $unzip :: [(a, b)] \rightarrow ([a], [b])$, que dada una lista de tuplas xs retorna una tupla de listas donde cada una corresponde a los primeros y segundos elementos de los pares respectivamente.

Ej. $unzip [('a', 1), ('z', 7), ('h', 9)] = ("azh", [1, 7, 9])$

- d) $pair2List :: (a, [b]) \rightarrow [(a, b)]$ que dado un par formado por un valor x y una lista xs convierta a la lista xs en una lista de pares, formada con los elementos de xs y x .

Ej. $pair2List\ (x, [y1, y2, y3]) = [(x, y1), (x, y2), (x, y3)]$

- e) $maxSec :: [(Int, Int)] \rightarrow (Int, Int)$, que dada una lista de pares de naturales que represente a una lista de segmentos de la recta, calcule el segmento más largo de la misma.

Ej. $maxSec\ [(1, 2), (0, 7), (4, 6)] = (0, 7)$

Puede definir una función auxiliar $maxL :: (Int, Int) \rightarrow (Int, Int) \rightarrow (Int, Int)$, que dados dos pares de naturales que representan a dos segmentos de la recta, devuelva el segmento cuya longitud sea máxima.

Ej. $maxL\ (1, 2)\ (0, 7) = (0, 7)$.