



Nombre y Apellido:

Legajo:

Examen Final

1. Dada una secuencia con pares de la forma (Artículo, Precio) ordenada cronológicamente, se desea saber cuál fue el artículo que más cantidad de aumentos tuvo a lo largo del tiempo. Suponiendo que se definió un tipo de datos *Art* para representar los distintos artículos, los cuales pueden compararse, definir una función *aumentos* : $Seq (Art, Int) \rightarrow (Art, Int)$, que resuelva el problema calculando el artículo que más cantidad de aumentos tuvo y la cantidad de aumentos.

Definir la función con profundidad de orden $O(\lg n)$, donde n es el largo de la secuencia.

Por ejemplo,

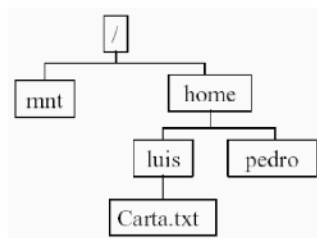
```
| aumentos⟨(a, 5), (b, 4), (a, 4), (a, 30), (b, 7), (b, 10)⟩ = (b, 2) |  
| aumentos⟨(a, 5), (a, 6), (b, 1), (a, 3), (a, 4), (b, 2)⟩ = (a, 2) |
```

2. Se utilizará el siguiente tipos de datos para representar un árbol de directorios en un sistema de archivos.

```
type Name = String  
data DirTree a = Dir Name [DirTree a] -- nombre de un directorio y su contenido  
                | File Name a         -- nombre de un archivo y contenido
```

donde la raíz de un árbol de directorios debe ser "/" y no puede haber dos directorios o archivos con el mismo nombre en el mismo path.

Por ejemplo, para representar el árbol de directorio:



definimos dirtree:

```
dirtree = Dir "/" [Dir "home" [Dir "luis" [File "Carta.txt" "xxx"], Dir "Pedro" []], Dir "mnt" []]
```

Además se utilizará el tipo *Path* para representar una ruta absoluta:

```
type Path = [String]
```

Por ejemplo el path `"/home/pepe"` se representa con la lista `["home", "pepe"]`, mientras que la raíz `"/"` con `[]`. Definir en Haskell las siguientes funciones para manejar esta estructura de directorios.

a) `names :: [DirTree a] → [Name]`, dada una lista de árboles de directorios devuelve una lista con los nombres correspondientes a archivos o directorios que están en la raíces de los árboles.

Por ejemplo,

```
names [Dir "home" [Dir "luis" [File "Carta.txt" "xxx"], Dir "Pedro" []], Dir "mnt" []] = ["home", "mnt"]
```

b) Para crear un directorio vacío en un path, siempre que exista el path y no haya un archivo o directorio con el mismo nombre, se definió la siguiente función:

```
mkDir :: Path → Name → DirTree a → DirTree a  
mkDir p n d = mkdir ("/" : p) n d
```

Completar con la definición de `mkdir`.

c) Para listar los archivos y directorios dentro de una ruta dada se definió:

```
ls :: Path → DirTree a → [Name]
ls p t = ls ("/" : p) t
```

Completar la definición de ls definiendo ls.

Por ejemplo,

```
ls [] dirtree = ["home", "mnt"]
ls ["home"] dirtree = ["luis", "pedro"]
```

3. (Sólo libres) Dadas las siguientes definiciones:

```
elem x [] = False
elem x (y : ys) = x == y ∨ elem x ys
concatMap f = concat ∘ map f
allNames (Dir n xs) = n : concatMap allNames xs
allNames (File n x) = [n]
isName y (Dir n xs) = y == n ∨ or (map (isName y) xs)
isName y (File n x) = y == n
```

Probar por inducción estructural sobre t que:

$$\text{elem } n (\text{allNames } t) = \text{isName } t$$

para cualquier $n :: \text{String}$.

Ayuda: Puede usar la siguiente propiedad $\text{elem } x (\text{concat } xss) = \text{or } (\text{map } (\text{elem } x) xss)$, sin demostrarla.