Examen PF. Febrero 2006. Sistemas y Gestión. UCM

Nota: En todos los ejercicios, cada vez que definas una función deberás proporcionar también la declaración de su tipo.

1. [**0.5 ptos**] Para cada una de las siguientes funciones indica cuál es el tipo más general que puede asignársele.

```
f [] = []
f (x:xs) = x : f xs

g xs
  | xs == [] = []
  | otherwise = head xs : g (tail xs)

h xs = if xs==[] then [] else xs

i xs = xs
```

2. [**0.5 ptos**] Escribe una función `testea` de orden superior y de 3 parámetros (`f`, `g` y `xs`) que devuelva True si y sólo si el resultado que devuelven las dos funciones `f` y `g` para la lista de pares de parámetros de entrada `xs` es siempre el mismo. Ahora bien, las funciones `f` y `g` serán funciones que esperen dos parámetros por separado. Por ejemplo, en el siguiente código:

```
suma x y = x + y

adicion x y
 | y /= 0 = x + y
 | otherwise = -x

resultado1 = testea suma adicion [(1,3),(2,4),(-1,-7),(0,0),(0,-3)]
resultado2 = testea suma adicion [(1,3),(2,4),(-1,0),(0,0),(0,-3)]
```

   debería devolverse `True` en `resultado1` y `False` en `resultado2`.

3. [**0.5 ptos**] Escribe una función recursiva que dada una lista de entrada, devuelva la longitud del segmento ordenado (de menor a mayor) más largo que exista dentro de dicha lista. Por ejemplo, para `[1,3,2,4,4,4,8,7,9,15]` debe devolver `5`, pues el segmento ordenado más largo que tiene es `[2,4,4,4,8]`. **Restricción:** Si la función tiene una longitud n entonces el coste de la función que hagas debe tener a lo sumo coste $O(n)$.

4. [**2 ptos**] Una variante de los árboles binarios de búsqueda es aquella en la que los árboles pueden ser vacíos o estar formados por un nodo interno que además de tener un elemento y dos subárboles hijos tiene un valor entero adicional. En este tipo de árboles, todos los valores que se encuentran en el subárbol izquierdo de un nodo son estrictamente menores que el elemento del nodo, mientras que todos los que se encuentran en el subárbol derecho son estrictamente mayores que el elemento del nodo. Así pues, un mismo valor no puede aparecer en varias posiciones distintas del árbol. Es por ello que el dato extra que se encuentra en cada nodo es un entero que indica cuántas repeticiones deberían existir en el árbol del elemento que está en el nodo.

   Proporciona un nuevo tipo de datos adecuado para manejar este tipo de árboles, e implementa operaciones para crear un nuevo árbol vacío, para comprobar si un árbol está vacío o no, para añadir un elemento a un árbol y para eliminar un elemento de un árbol. Escribe también una función que realice un recorrido en inorden del árbol, pero con la particularidad de que la función debe tener un parámetro booleano adicional que determine si en el recorrido deben mostrarse todas las repeticiones de los elementos o sólo la primera aparición.

   Escribe una función **no recursiva** `ordenaSinRepeticiones` que basándose en las funciones anteriores ordene una lista de entrada pero eliminando todas las repeticiones que en ella hubiera. Por ejemplo, `ordenaSinRepeticiones [3,2,3,7,9,1]` debe devolver `[1,2,3,7,9]`

5. [**1.5 ptos**] En el fichero `patata.txt` tenemos almacenada una lista de números enteros, a razón de un entero en cada línea del fichero. Escribe una función que a partir de dicho fichero genere un segundo fichero `ordenados.txt` en el que estén esos mismos números pero ordenados de menor a mayor. Por ejemplo, si el contenido del fichero `patata.txt` fuera

```
10
2
8
1
```

   entonces en `ordenado.txt` tendría que escribirse

```
1
2
8
10
```

Repite el ejercicio anterior, pero suponiendo que en el fichero de partida no sólo había enteros, sino que cada línea contenía un entero, un nombre y un DNI. Así, para el fichero de entrada

```
10 Juan Gómez 22148312P
2 María García 18413038F
8 Ana Álvarez 89243131S
1 Pablo Sánchez 84716419T
```

debería devolverse

```
1 Pablo Sánchez 84716419T
2 María García 18413038F
8 Ana Álvarez 89243131S
10 Juan Gómez 22148312P
```

6. [**1 pto**] Escribe una función `sufijos` que dada una lista de entrada devuelva la lista de todos sus sufijos. Por ejemplo, para `[3,4,1]` debería devolver `[[3,4,1],[4,1],[1],[]]`.

   Escribe otra función `prefijos` que dada una lista de entrada devuelva la lista de todos sus prefijos. Por ejemplo, para `[3,4,1]` debería devolver `[[],[3],[3,4],[3,4,1]]`.

   A partir de las dos funciones anteriores, define una tercera función `segmentos` que devuelva la lista de todos sus segmentos, incluyendo todos sus segmentos vacíos. Por ejemplo, para `[3,4,1]` debería devolver la lista `[[],[3],[],[3,4],[4],[],[3,4,1],[4,1],[1],[]]`. **Restricción:** La definición de la función `segmentos` debe ocupar a lo sumo una línea.

   Redefine la función `segmentos` para que sólo devuelva la lista de segmentos no vacíos.

7. [**1 pto**] Dadas dos matrices bidimensionales, ambas con el mismo número de filas y columnas, decimos que la primera es *totalmente mayor* que la segunda si el menor de los elementos de la primera matriz es mayor o igual que el mayor de los elementos de la segunda. Por otra parte, decimos que la primera matriz es *continuamente mayor* que la segunda si para cada posible posición de las matrices se cumple que el valor que hay en esa posición en la primera matriz es mayor o igual que el valor que hay en esa posición en la segunda matriz.

   Escribe una función `totalMayor` y otra `continuaMayor` que implementen las ideas anteriores. **Restricción:** No podrás realizar ninguna definición recursiva.

8. [**1 pto**] Explica qué hace la siguiente función `scanl`:

```
scanl :: (a -> b -> a) -> a -> [b] -> [a]
scanl f acum [] = [acum]
scanl f acum (x:xs) = acum : scanl f (f acum x) xs
```

   Recomendación: Intenta ver qué calcularía para `scanl (+) 0 [1..4]`, para `scanl (*) 1 [1..4]` o incluso para `scanl (*) 1 [1..]`.

   A partir de la función anterior, define una función no recursiva `pots` que dado un número devuelva la lista infinita de todas sus potencias. Por ejemplo, `pots 2` debe devolver la lista infinita `[1,2,4,8,16,32,...]`.

   Define también un valor `factoriales` que contenga la lista infinita de todos los factoriales de menor a mayor. Es decir, `factoriales` contiene la lista `[1,1,2,6,24,120,...]`

   A partir de `factoriales` y `pots`, define una función `fvsp` que dado un valor `n` devuelva el menor i tal que el factorial de i sea estrictamente mayor que $\mathtt{n}^i$. Por ejemplo, `fvsp 2` debe devolver 4, pues la lista de factoriales es

   `[1,1,2,6,` **`24`**`,120,720,...]`

   mientras que `pots 2` devuelve

   `[1,2,4,8,` **`16`**`,32,64,...]`