

EXAMEN - 31 de Enero de 2007

1. (2 puntos) Definir una función que dados una lista xs y un elemento z , y recorriendo la lista xs una sola vez, obtenga el par formado por

- la lista de que resulta de quitar en xs todas las apariciones de z y
- el número de apariciones de z en xs .

¿Cual es el tipo de la función definida?

2. (1 punto) Sea la siguiente definición de función:

```
mifun s = foldr op 0 s
      where op x r = head x + r
```

- a) ¿Cual es el tipo de datos de `mifun` ?
- b) ¿Cual es el resultado de evaluar `mifun [1,2,3]`?
- c) ¿Cuál de las definiciones siguientes es una definición alternativa equivalente?

(Def 1) `mifun s = sum (map head s)`

(Def 2) `mifun s = map op s`
 `where op x r = head x + r`

(Def 3) `mifun [] = 0`
 `mifun (x:xs) = x + mifun xs`

3. (2 puntos) a) Generalizar la función:

```
iSort [] = []
iSort (x:xs) = insertar x (iSort xs)
      where insertar z [] = [z]
            insertar z (x:xs)
              | x < z = x : (insertar z xs)
              | otherwise = z : x : xs
```

para obtener una función de orden superior `iSortGen` que tome como argumento (además de la lista) un predicado binario que represente el orden con respecto al cual ordenará. En particular, debe cumplirse que `iSort = iSortGen (<)`

También debe ocurrir que:

```
iSortGen (>) [2,3,1,5,3] => [5,3,3,2,1]
iSortGen (<) ["casa", "perro", "coche"] =>
      ["casa", "coche", "perro"]
iSortGen (==) [2,3,1,5,3] => [2,3,1,5,3]
iSortGen (==) ["casa", "perro", "coche"] =>
      ["casa", "perro", "coche"]
```

Además, se pide:

b) Decir cuál es el tipo de `iSort` y el de `iSortGen`.

c) ¿Cual sería el resultado de

```
iSortGen (>) [(1,2,3), (1,2,5), (3,4,4)]?
```

- d) Definir usando `iSortGen` una función `OrdTrip` que ordene listas de triples por orden decreciente de su tercera componente. Por ejemplo:
- ```
OrdTrip [(1,2,3),(1,2,5),(3,4,4)] =>[(1,2,5),(3,4,4),(1,2,3)]
```
- ¿Cuál es el tipo de `f`? (Recuerda las restricciones de clase)

4. (2 puntos) Sean los siguientes tipos de datos:

```
type Almacen = [Pack]
data Pack = PA (Ref,Marca,Consola,Juegos,Complementos,Precio)
type Juegos = [String]
type Complementos = [String]
data Marca = Nintendo | Sony | MicroSoft
 deriving (Enum, Eq, Show)
type Consola = String
type Ref = Integer
type Precio = Float
```

El siguiente es un ejemplo de almacén con 7 productos diferentes (de p1 a p7):

```
bd :: Almacen
bd = [p1,p2,p3,p4,p5,p6,p7]
p1,p2,p3,p4,p5,p6,p7 :: Pack
p1 = PA (1111, Nintendo, "DS Lite", ["Big Brain Academy"],
 [], 159.00)
p2 = PA (1116, Sony, "PSP", ["WRC", "Medieval Resurrection"],
 ["Estuche", "Tarjeta de memoria", "Funda",
 "Auriculares"], 154.90)
p3 = PA (2254, Nintendo, "Wii", ["Sports"], ["Mando control
 remoto", "Numchaku"], 279.90)
p4 = PA (1113, Nintendo, "GameBoy Advance", [], ["Bolsa",
 "Auriculares", "Adaptador USB"], 114.90)
p5 = PA (1112, Nintendo, "DS Lite", ["Eragon"], ["Bolsa", "2
 stylus"], 155.90)
p6 = PA (2002, MicroSoft, "XBOX 360 PRO", ["Gears of war",
 "FIFA 06 Copa mundial"], ["Kit carga"], 419.00)
p7 = PA (2002, MicroSoft, "XBOX 360 PRO", [], ["Disco duro",
 "2 mandos inalámbricos","Mando DVD",
 "Auriculares", "Micrófono"], 419.00)
```

Utilizando al menos una de las funciones de orden superior sobre listas (`map`, `foldr`, `foldl`, etc), definir una función:

```
complDisp :: Almacen -> [String]
```

que obtenga la lista, sin repeticiones, de todos los complementos disponibles que hay en un almacén, por ejemplo:

```
Main> complDisp bd
["Estuche","Tarjeta de memoria","Funda","Auriculares","Mando
control remoto","Numchaku","Bolsa","Adaptador USB","2
stylus","Kit carga","Disco duro","2 mandos
inalámbricos","Mando DVD","Micrófono"]
```