

EXAMEN - 19 de Junio de 2007

1. (2 puntos) Supongamos ya definida

```
class Show a => MiClase a where
    dimension :: a -> Int
```

es decir una clase "MiClase a" que

- es subclase de la clase Show, es decir que todo tipo a que es instancia de MiClase cuenta con la función (sobrecargada) `show :: a -> String`
- tiene un método que calcula la dimension (o tamaño) de los objetos de los tipos que sean instancia.

Se pide:

- a) Definir una instancia "MiClase Int" donde la dimensión de cualquier entero sea 1.
- b) Definir una instancia "MiClase [a]" de modo que la dimensión de una lista de elementos de tipo a sea la longitud de la lista.
- c) Definir otra instancia "MiClase [a]" de modo que la dimensión de una lista de elementos de tipo a sea la suma de las dimensiones de sus elementos utilizando las funciones `map` y `foldr`.

Indicación: Tener cuidado con las restricciones de clase al definir cada una de las instancias.

2. (1.5 puntos) Sea la siguiente función

```
f h p xs ys = [h x y | x<-xs , p x, y <-ys]
```

- a) ¿Cual es el tipo de f?
- b) ¿Cuál es el resultado de evaluar la expresión `f (+) even [1..4][10..14]`?
- c) Escribe otra definición equivalente de f que no utilice listas ZF/por compresión.

3. (2.5 puntos) Sean los siguientes tipos de datos para representar proposiciones lógicas que pueden ser átomos ('p', 'q', etc) o proposiciones compuestas construidas en base a los conectivos de: negación (No, de aridad 1 o unario) y los conectivos binarios de conjunción, disyunción, implicación y doble-implicación.

```
data Prop =      Atomo Char
                | ConU ConUnario Prop
                | ConD ConBinario Prop Prop

data ConUnario = No

data ConBinario = Conj | Disy | Imp | DImp
```

Un ejemplo de proposición es

```
prop1 = ConU No (ConB Disy (Atomo 'p')
                        (ConB Conj (Atomo 'q') (ConU No (Atomo 'r'))))
```

como representación de la proposición (en Haskell) `"not (p || (q && not r))"`

- a) Definir una función de tipo "fold" o plegado que sirva para plegar proposiciones.
Indicación: Debe tomar como argumentos (además de la proposición a plegar) tres funciones una por cada constructor del tipo Prop. ¿Cuál es su tipo?
- b) Utilizando la función anterior definir una función que sirva para evaluar una proposición en una sustitución dada. Una sustitución es una lista de pares (c, b) donde c es de tipo Char y b es un booleano (True o False) que da valores a los átomos. Un ejemplo se sustitución es:


```
subs1 = [('p', False), ('q', False), ('r', True)]
```
- c) Explicar cual debe ser el resultado de "evaluar `prop1 subs1`" y como se obtiene.

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
    inalambricos","Mando DVD", "Auriculares", "Microfono"], 419.00)
```

Definir una función:

```
losMasEco :: Almacen -> [Pack]
```

que obtiene la lista de los "pack" más económicos, uno por cada marca.

5. (2 puntos) Considérese la siguiente lista infinita

```
[ (0,1), (1,1), (3,2), (6,6), (10,24), ...
```

que proviene de

```
[ (0, 1),
  (0+1, 1*1),
  (0+1+2, 1*1*2),
  (0+1+2+3, 1*1*2*3), ...
```

Define dicha lista infinita con una estructura cíclica.