LP Haskell Abril 2021 soluciones

Lenguajes de programación (Universidad Rey Juan Carlos)

Universidad Rey Juan Carlos

Lenguajes de Programación - Prueba Programación Funcional
07 de abril de 2021

APELLIDOS:	***NOMBRE:***
TITULACIÓN:	***DNI:***

(La duración del examen es de **1 hora y 30'**, pasados los primeros 30 minutos se recogerá el test.)

Ejercicio 1 (3 Puntos)
Las preguntas acertadas sumarán **1** punto y las falladas restarán **0,25**. Las preguntas no contestadas ni suman ni restan. Hay una única respuesta correcta por pregunta. Escriba las respuestas en la siguiente tabla (únicamente se corregirá el test si las respuestas están en la tabla).

Respuestas:

1	2	3	4	5	6	7	8	9	10

1. ¿Cuál de las siguientes expresiones **no** es una lista válida en Haskell?

   a) `[('a',7),('2',6)]`
   b) `[[3+7,5 `mod`2,9],[3/8]]`
   c) `[-2 == 3,x < y,odd 5]`
   d) `["casa"]`

2. Dadas los siguientes ejemplos de aplicación de la función `take` de Haskell, ¿cuál sería la declaración correcta para esta función?

```
take 3 [3,5,7,9,4]  ==  [3,5,7]

take 2 ['l','o','l','a']  ==  "lo"

take 2 "lola" ==  "lo"
```

   a) `take :: a -> [a] -> [a]`
   b) `take :: Int -> [Int] -> [Int]`
   c) `take :: Int -> [a] -> [a]`
   d) Todas las respuestas anteriores son correctas.

3. ¿Cuál es el resultado de evaluar la siguiente expresión?

```
> foldr (\a b-> if even a then a+2 else b-1) 2 [5,9,10,3]
```

   a) 2
   b) 3
   c) 5
   d) 10

APELLIDOS:	*NOMBRE:*
TITULACIÓN:	*DNI:*

4. Entre las bases de la programación funcional está…

   a) El orden de evaluación normal.
   b) El uso de instrucciones iterativas.
   c) La ausencia de efectos laterales.
   d) El concepto de posición de memoria.

5. Dada la siguiente función en haskell

```
misterio :: Eq a => [a] -> [(a, Int)]
misterio [] = []
misterio l@(x:xs) = (x, freq): misterio l_sin_x where
       freq = length (filter (== x) l)
       l_sin_x = filter (/= x) xs
```

   ¿Cuál sería la salida de la siguiente aplicación de la función?

```
misterio ['a', 'g', '1', 'a', 'h', 'v']
```

   a) `[(2,2),(1,2),(3,2),(5,1),(4,1),(6,1),(0,1)]`
   b) `[('a',2),('g',1),('1',1),('h',1),('v',1)]`
   c) `[('a',2),('b',1),('1',1),('y',1),('v',1)]`
   d) Ninguna de las anteriores

6. Si la función `h` tiene esta declaración `h::[(a,b)]->[b]` y se aplica como `h[('1','2')]` ¿A cuál de estos patrones se ajustaría?

   a) `h [] =…`
   b) `h (x:y:xs)=…`
   c) `h [x,y] =…`
   d) Todas las respuestas anteriores son incorrectas.

7. Dada la siguiente función

```
funcion :: (Int, [Int]) -> [Int]
funcion (x, []) = if x < 2 then [x,2] else [2,x]
funcion (x, y:ys) = if x < y then x:y:ys else y:funcion(x,ys)
```

   ¿Qué ocurre si se hace la siguiente aplicación de función?

```
funcion (4,[2])
```

   a) Devuelve una lista vacía.
   b) Daría error por no estar definido el patrón para lista con un elemento.
   c) Daría como resultado una lista de números enteros.
   d) La función no está bien definida y no compila.

APELLIDOS:	*NOMBRE:*
TITULACIÓN:	*DNI:*

8. ¿Qué expresión con notación Lambda daría como resultado la lista `[1,6,0,9]`?

   a) `(\l1 l2 -> l1 ++ l2) [1,6] [0,9]`
   b) `(\l2 -> l2 ++ [0,8]) [1,6]`
   c) `(\l1 l2 -> [head l1]++[head l2]) [1] [6,0,9]`
   d) Todas las anteriores respuestas son correctas.

9. Respecto a las listas en Haskell:

   a) Pueden contener elementos de diferentes tipos.
   b) Sus elementos pueden ser el resultado de expresiones.
   c) El conjunto de elementos que contienen es siempre finito.
   d) El constructor (:) puede utilizarse para añadir un elemento al final de la lista.

10. Si una función `g` recibe una lista polimórfica de elementos: ¿Cuál de los siguientes patrones es el adecuado para el caso en el que la lista tenga un solo elemento?

    a) `g l =…`
    b) `g x:[]=…`
    c) `g(x:xs)=…`
    d) Todas las anteriores respuestas son correctas.

APELLIDOS:	*NOMBRE:*
TITULACIÓN:	*DNI:*

(Cada ejercicio se debe responder en el espacio habilitado para ello. Se valorará positivamente la claridad y extensibilidad del código)

Ejercicio 2 (2 Puntos)
Dados los siguientes tipos de datos:

```
data Categoria = ATP1000 | ATP500 | ATP250 | GrandSlam deriving Show
type Nombre = String
data Torneo = Tor Nombre Categoria
data Temporada = Temp [Torneo]
```

Y las siguientes funciones:

```
openAustralia :: Torneo
openAustralia = Tor "Open de Australia" GrandSlam

indianWells :: Torneo
indianWells = Tor "Indian Wells" ATP1000

mutuaMadridOpen :: Torneo
mutuaMadridOpen = Tor "Mutua Madrid Open" ATP1000

wimbledon :: Torneo
wimbledon = Tor "Wimbledon" GrandSlam

temporada2013 :: Temporada
temporada2013 = Temp [openAustralia, indianWells, mutuaMadridOpen, wimbledon]
```

Se pide implementar una función **utilizando funciones de plegado** que dada una temporada sea capaz de mostrar el listado de torneos de `GrandSlam` que tiene esa temporada. Un ejemplo de aplicación de la función junto con la salida que debe obtener es el siguiente:

```
torneosPorCategoria temporada2013
[Torneo: Wimbledon - Categoria: GrandSlam
,Torneo: Open de Australia - Categoria: GrandSlam
]
```

Se deberá implementar todo lo que considere necesario para que la salida sea en el mismo formato que la que se muestra en el ejemplo.

Solución:

```
getNombre:: Torneo -> String
getNombre (Tor n _) = n

getCategoria:: Torneo -> Categoria
getCategoria (Tor _ c) = c
```

APELLIDOS:	*NOMBRE:*
TITULACIÓN:	*DNI:*

```
instance Show Torneo where
      show torneo = "Torneo: "++getNombre (torneo) ++ " - Categoria: " ++
show(getCategoria(torneo)) ++"\n"


-- listado de torneos de una categoría determinada de GrandSlam
-- con funciones de plegado

torneosPorCategoria :: Temporada -> [Torneo]
torneosPorCategoria (Temp lista) = foldr (\t ac -> if (getCategoria(t)==GrandSlam) then
ac++[t] else ac) [] lista
```

Ejercicio 3 (1 Punto)

Dada la siguiente definición de datos que representa una lista de elementos

```
data List a = Cons a (List a) | Nil
```

a) **(0.25 puntos)** ¿Cómo se representaría con este tipo de datos la lista `[1, 2, 3]`?

Solución:

```
Cons 1 (Cons 2 (Cons 3 Nil))
```

b) **(0.75 puntos)** Implementa una función para calcular la longitud de una lista representada con el tipo de datos `List`

Solución:

```
longitud :: List a -> Integer
longitud Nil = 0
longitud (Cons a lista) = 1 + longitud lista
```

Ejercicio 4 (1 Punto)

La Agencia de Seguridad Aérea desea crear un programa en Haskell que les permita gestionar sucesos de seguridad aérea. Los sucesos se clasifican en dos categorías de acuerdo con su gravedad, accidentes e incidentes, cuya diferencia estriba en que en los accidentes se producen víctimas mortales, mientras que en los incidentes no. Además de esta diferencia, en ambos casos interesa almacenar la siguiente información:

- Aeropuerto: el nombre del aeropuerto donde se produjo el suceso.
- Heridos leves: el número de heridos que no requieren hospitalización.

APELLIDOS:	*NOMBRE:*
TITULACIÓN:	*DNI:*

- Heridos graves: el número de heridos que requieren hospitalización.

Se pide

a) (**0.5 Puntos**) Definir el tipo de dato adecuado para gestionar sucesos.

Solución:

```
data Suceso = Accidente String Int Int Int | Incidente String Int Int
```

b) (**0.5 Puntos**) Implementar todo lo necesario para poder comparar dos sucesos con la función ==. Dos sucesos serán iguales si son del mismo tipo (accidente o incidente) y si todos sus campos son iguales.

Solución:

```
instance Eq Suceso where
    Accidente a m l g == Accidente a' m' l' g' = a==a' && m==m' && l==l' && g==g'
    Incidente a l g == Incidente a' l' g' = a==a' && l==l' && g==g'
    _ == _ = False
```

Ejercicio 5 (3 Puntos)

Dado el siguiente tipo de datos recursivo que representa árboles binarios

```
data Arbol a = AV | Rama (Arbol a) a (Arbol a)
```

Se pide implementar una función polimórfica en Haskell que reciba un árbol binario y devuelva dos listas: una con los nodos sin repetir y otra con los nodos que están repetidos. En ambas listas habrá que incluir tanto los nodos hoja (nodos sin descendientes) como los nodos interiores (nodos que no son hojas). Ejemplos de aplicación de la función podrían ser:

```
> separarNodos (Rama (Rama (Rama AV 12 AV) 49 (Rama (Rama AV 23 AV) 12 (Rama AV 13 AV)))
13 (Rama AV 10 AV))
([49,23,10],[12,13])
> separarNodos (Rama (Rama (Rama (Rama AV 'D' AV) '1' (Rama AV 'R' AV)) 'D' (Rama (Rama
AV 'f' AV) '7' (Rama AV '1' AV))) 'R' (Rama AV 'h' AV))
("f7h","D1R")
```

Solución:

```
arbolAlista :: Arbol a -> [a]
arbolAlista AV = []
arbolAlista (Rama i n d) = arbolAlista i ++ [n] ++ arbolAlista d

contarRec :: Eq a => [a] -> [(a, Int)]
contarRec [] = []
contarRec l@(x:xs) = (x, freq): contarRec l_sin_x where
```

APELLIDOS:	*NOMBRE:*
TITULACIÓN:	*DNI:*

```
    freq = length (filter (== x) l)
    l_sin_x = filter (/= x) xs

partirLista :: Eq a => [a] -> ([a], [a])
partirLista [] = ([], [])
partirLista xs = (unicos, repetidos) where
 unicos = [x | (x, y) <- contarRec xs, y == 1]
 repetidos = [x| (x, y) <- contarRec xs, y > 1]

separarNodos :: Eq a => Arbol a -> ([a], [a])
separarNodos = partirLista.arbolAlista
```