Parcial 1 - Algoritmos I Taller: Tema C

Ejercicio 1

En las siguientes preguntas marque la respuesta correcta.

a) Si tengo una función con la siguiente declaración de tipos de la función g
`g :: [(a,b)] -> (a -> c, b -> d) -> [(c,d)]` puedo decir que:
   1) Es un función polimórfica Ad hoc
   2) Es una función recursiva
   3) Es una función polimórfica Paramétrica
   4) Es un constructor
   5) Ninguna de las anteriores.

b) Si tengo una función con la siguiente declaración

```
g :: a -> b -> (a -> b) -> Bool
g x y g = g x == y
```

   puedo decir que:
   1) Es un constructor
   2) La declaración de tipos está mal, porque debería incluir una clase de tipos.
   3) Es un función polimórfica Ad hoc
   4) La definición de la función es incorrecta, debería utilizar pattern matching.
   5) Ninguna de las anteriores.

¿Cuál es el propósito de la función valorODefault y cómo maneja el valor Nothing?

c) Dada la siguiente declaración de función en Haskell:

```
valorODefault :: Int -> Maybe Int -> Int
valorODefault i Nothing = i
valorODefault _ (Just x) = x
```

¿Cuál es el propósito de la función `valorODefault` y cómo maneja el valor `Nothing`?

   1) La función valorODefault devuelve `x` para cualquier Just o `i` en cualquier otro caso.
   2) La función valorODefault lanza un error cuando se le pasa Nothing.
   3) La función valorODefault siempre devuelve `i` independientemente de su entrada.
   4) La función valorODefault solo funciona con valores Nothing y no con Just.

d) Dada la siguiente declaración de tipo en Haskell:
```
data PodraSer a = NoEs | QuizaSi | Definitivamente a
```

Puedo afirmar que :

1) Ese tipo está mal definido, debería haber utilizado el comando type.
2) No se puede definir un tipo de esa manera.
3) El tipo `PodraSer a` tiene tres constructores, dos de los cuales tienen tipo `PodraSer a`.
4) El tipo `PodraSer a` tiene tres constructores, tres de los cuales tienen tipo `PodraSer a`.
5) El tipo está mal definido porque todos los constructores no toman parámetros.
6) Ninguna de las anteriores.

Ejercicio 2

Se va a representar el stock de un vivero de plantas, usando tipos en Haskell. Los productos que tenemos en cuenta son: `Plantas, BolsaSemillas, Macetas`. La idea es poder detallar para cada tipo de producto junto con sus características más importantes. En tal sentido identificamos las siguientes características de cada uno de los productos a tener en cuenta:

`Planta`

- `TipoPlanta`, que es un tipo enumerado con las siguientes opciones: `PlenoSol`, `MediaSombra`, `Floral, Frutal.`
- `Ubicacion`, que es un tipo enumerado con las siguientes opciones: `Exterior`, `Interior.`
- `Agua,` que es un tipo enumerado representando que tan resistente al exceso de agua es: `MuyResistente, Resistente, NadaResistente.`
- `Precio`, que es un sinónimo de `Int` indicando el precio.

`BolsaSemilla`

- `TipoSemilla`, que es un tipo enumerado con las siguientes opciones: `Hortalizas, Aromaticas, Cesped.`
- Peso, que es un sinónimo de `Float` indicando el peso total de la bolsa.
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Maceta`

- TipoMaceta, que es un tipo enumerado con las siguientes opciones: `Terracota`, `FibroCemento, Plastico`.
- `Forma`, que es un tipo enumerado con las siguientes opciones: `Cuadrada`, `Redonda`.
- `Precio`, que es un sinónimo de `Int` indicando el precio

Para ello:

a) Definir el tipo `ProductoDeVivero` que consta de los constructores `Planta`, `Semilla` y `Maceta`, constructores con parámetros descriptos arriba (Se deben definir también los tipos enumerados `TipoPlanta, Ubicacion, Agua, TipoSemilla, Forma,` TipoMaceta`)`. **Los tipos** `ProductoDeVivero, TipoPlanta, TipoMacera y Ubicacion` **<u>no deben</u> estar en la clase** **`Eq`**, ni en la clase **`Ord`**.

b) Definir la función `cuantasPlantas` de la siguiente manera:

```
cuantasPlantas :: [ProductoDeVivero] -> TipoPlanta -> Int
```

que dada una lista de `ProductoDeVivero pv` y un valor `tp` de `TipoPlanta`, me devuelve un entero indicando la cantidad de plantas que hay en `pv` de tipo `tp`.
NOTA: Dejar como comentario un ejemplo donde hayas probado la función `cuantasPlantas` con una lista con al menos 3 `ProductoDeVivero`.

c) Definir igualdad para el tipo de `ProductoDeVivero`: de tal manera que, dos valores de tipo `Planta` son iguales sólo si tienen el mismo **tipo de planta** y la misma **ubicacion**, dos `BolsaSemilla` son iguales solo si tienen el mismo **peso** y el mismo **precio**, mientras que dos `Maceta` son iguales si tienen el mismo **tipo de maceta**. Como es de suponer las `Planta`, `BolsaSemillas` y `Maceta` son distintos entre sí.
NOTA: Dejar como comentario en el código dos ejemplos en los que probaste la igualdad.

Ejercicio 3

Quedaste manija con el vivero y compraste una planta bien joven para disfrutar con tus amigos. Ahora queres tener registro de su progreso de manera de poder determinar si necesita ser cambiada de maceta, es decir transplantarla de su actual maceta a una mas grande.

a) Definir un tipo recursivo `RegistroPlanta`, que permite guardar datos sobre la evolucion de la planta. El tipo `RegistroPlanta`, tendrá dos constructores:

   1) `DatoPlanta`, que tiene 6 parámetros:

   - `EstadoHojas` (tipo enumerado con el `estado general de las hojas`: Muchas`Quemadas`, PocasQuemadas, `Saludables`)
   - `Int`, altura de la planta en centimetros
   - `Int`, cantidad total de hojas
   - `Int`, cantidad de hojas florecidas
   - `Int`, edad en meses de la planta
   - `RegistroPlanta,` recursión con el resto de las notas.

   2) `NoDato`, que es un constructor sin parámetros, similar al de la lista vacía, para indicar ningún registro sobre alguna planta fue ingresado.

La condición para poder transplantar cualquier planta se describen a continuación, utilizando el registro correspondiente:

- Si el estado de las hojas es `PocasQuemadas` o `Saludables`, su altura supera los 40 centímetros.
- Si el estado de las hojas es MuchasQuemadas y la cantidad total de hojas es menor a 10.

b) Programar la función `trasplantar`, que toma como primer parámetro registros de una planta del tipo **`RegistroPlanta,`** y como segundo parámetro la edad mínima de la planta de tipo **`Int`** y retorna un valor de tipo **`Bool,`** indicando si la planta con al menos esa edad **deberia ser trasplantada o no**.

```
trasplantar :: RegistroPlanta -> Int -> Bool
```

NOTA: Dejar como comentario un ejemplo donde hayas probado `trasplantar` con un parámetro de tipo **`RegistroPlanta`** que tenga al menos 3 datos sobre alguna planta.

c) Programar la función `devolverEstado` con la siguiente declaración:

```
devolverEstado :: RegistroPlanta -> Int -> Maybe EstadoHojas
```

que dado un registro sobre una planta y una mínima cantidad de meses, retorna el primer estado de las hojas que tenia la planta al tener al menos esa cantidad mínima de meses, es decir Just **eh**, o Nothing en caso la planta no tenga esa edad.