Parcial 1 - Algoritmos I Taller: Tema G

Ejercicio 1:

En las siguientes preguntas marque la respuesta correcta.

a) Si tengo una función con la siguiente declaración de tipos de la función `f`

`f :: (a -> Bool) -> [a] -> [a]` puedo decir que:

1. Es una función recursiva.
2. Es un función polimórfica Ad hoc.
3. Es una función polimórfica Paramétrica.
4. Es un constructor.
5. Ninguna de las anteriores.

b) Si tengo una función con la siguiente declaración

```
f :: (Eq a, Ord a) => a -> a -> a
f x y | x<y = x
      | otherwise = y
```

puedo decir que:

1. La declaración de tipos está bien.
2. Es una función polimórfica paramétrica.
3. La definición de la función es incorrecta, debería utilizar pattern matching.
4. Es un constructor.
5. Ninguna de las anteriores.

c) Dada la siguiente declaración función en Haskell:

```
decrementar :: Maybe Int -> Maybe Int
decrementar Nothing = Nothing
decrementar (Just x) = Just (x - 1)
```

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

1) La función `decrementar` lanza un error cuando se le pasa Nothing.
2) La función `decrementar` decrementa el valor dentro de un Just en 1 y devuelve Nothing si se le pasa Nothing.
3) La función `decrementar` siempre devuelve Just 1 independientemente de su entrada.
4) La función `decrementar` solo funciona con valores Nothing y no con Just.
5) Ninguna de las anteriores.

d) Dada la siguiente declaración de tipo en Haskell:

```
data NuevoTipo a = Nada | Algo a
```

Puedo afirmar que :

1) Ese tipo está mal definido, debería haber utilizado el comando `type`.
2) El tipo está mal definido porque ambos constructores no toman parámetros.
3) No se puede definir un tipo de esa manera.
4) El tipo `NuevoTipo` tiene dos constructores, uno sin parámetros y el otro constructor con un parámetro.
5) Ninguna de las anteriores.

Ejercicio 2

Se va a representar el stock de Artículos en una Librería, usando tipos en Haskell. Los artículos que tenemos en cuenta son: `Libros, Agendas, Cuadernos`. La idea es poder detallar para cada tipo de material, las características más importantes. En tal sentido identificamos las siguientes características de cada uno de los materiales a tener en cuenta:

`Libro`

- `Categoria`, que es un tipo enumerado con las siguientes opciones: `Literatura`, `Infantiles`, Autoayuda, `Comics`
- `Editorial`, que es un tipo enumerado con las siguientes opciones: Altea, `Minotauro`, `Panini`
- `Titulo`, que es un sinónimo de `String` indicando el título del libro
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Agenda`

- `Marca`, que es un tipo enumerado con las siguientes opciones: `Monoblock`, `Papikra`.
- AnioAgenda, que es un sinónimo de `Int` indicando el año de la agenda
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Cuaderno`

- `Marca`, que es un tipo enumerado con las siguientes opciones: `Monoblock`, `Papikra` .
- `Precio`, que es un sinónimo de `Int` indicando el precio

Para ello:

a) Definir el tipo `ArtículosLibreria` que consta de los constructores `Libro`, `Agenda` y `Cuaderno`, constructores con parámetros descritos arriba (Se deben definir también los tipos enumerados `Categoria, Editorial , Marca, AnioAgenda)`. **Los tipos** `ArticulosLibreria, Editorial y Marca` **<u>no deben</u> estar en la clase** **`Eq`** , ni en la clase **`Ord`**.

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

```
cuantosLibros :: [ArticulosLibreria] -> Categoria -> Int
```

que dada una lista de `ArticulosLibreria` **`ls`** y un valor **`c`** de `Categoria de Libro`, me devuelve un entero indicando la cantidad de libros que hay en **`ls`** con la categoria **c**.
NOTA: Dejar como comentario dos ejemplo donde hayas probado la función `cuantosLibros` con una lista con al menos 3 `ArticulosLibreria`.

c) Definir igualdad para el tipo de `ArticulosLibreria` : de tal manera que, dos artículos de tipo `Libro` son iguales sólo si tienen la misma **Editorial** y el mismo **Título**, dos

`Agendas` son iguales solo si tienen la misma **marca**, el mismo **año** y el mismo **precio**, mientras que dos cuadernos son iguales si tiene la misma **marca** y el mismo **precio**. Como es de suponer los Libros, Agendas y Cuadernos son distintos entre sí.
NOTA: Dejar como comentario en el código dos ejemplos en los que probaste la igualdad.

Ejercicio 3

Queremos hacer un programa, para que las profesoras de una academia de Inglés puedan saber si sus alumnos de un nivel pueden pasar al siguiente o no.

a) Definir un tipo recursivo `NotasDeIngles`, que permite guardar las notas que tuvo cada estudiante de un nivel en el período. El tipo `NotasDeIngles`, tendrá dos constructores:

1) `EvolucionDelEstudiante`, que tiene 4 parámetros:

- `String`, para el nombre y apellido del alumno
- `Nivel` (Tipo Enumerado con el Nivel actual que está cursando: One, Two, Three)
- `Int` (con la nota del primer parcial, entre 1 y 10)
- `Int` (con la nota del segundo parcial, entre 1 y 10)
- `Int` (con la nota del final 1 a 10,)
- `NotasDeIngles,` recursión con el resto de las notas.

2) `NoHayMasEstudiantes`, que es un constructor sin parámetros, similar al de la lista vacía, para indicar que se terminaron las notas.

La condición para poder obtener el siguiente nivel se describen a continuación según las notas obtenidas en las diferentes instancias:

- Si el estudiante está en Nivel One o Two, debe sacar más de 7 en alguno de los parciales, y haber tenido en el final al menos un 6.
- Si el estudiante está en el Nivel Three debe tener al menos un 7 en los dos parciales, y al menos un 8 en el final.

b) Programar la función `pasaDeNivel`, que toma como primer parámetro `notas` del tipo **`NotasDeIngles,`** y como segundo parámetro el **`nombre`** del estudiante de tipo **`String`** y retorna un valor de tipo **`Bool,`** indicando si el estudiante con ese `nombre` es **pasa de nivel o no**.

```
pasaDeNivel :: NotasDeIngles -> String -> Bool
```

NOTA: Dejar como comentario un ejemplo donde hayas probado `pasaDeNivel` con un parámetro de tipo **`NotasDeIngles`** que tenga al menos 3 estudiantes.

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

```
devolverNivel :: NotasDeIngles -> String -> Maybe Nivel
```

que toma una variable **`notas`** de tipo `NotasDeIngles`, y como segundo argumento un `nombre`, que identifica al estudiante, y en caso que este esté en **`notas`** (en un nivel **n**), retorna **Just `n`** y **`Nothing`** en caso contrario.

NOTA: Dejar como comentario un ejemplo donde hayas probado la función.