Parcial 1 - Algoritmos I Taller: Tema E

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] -> (a -> b) -> Bool` puedo decir que:
   1) Es una función polimórfica Paramétrica
   2) Es una función polimórfica Ad hoc
   3) Es una función recursiva
   4) Es un constructor
   5) Ninguna de las anteriores.

b) Si tengo una función con la siguiente declaración

```
f :: a -> a -> a
f x y | x<y = x
      | otherwise = y
```

puedo decir que:
   1) Es una función polimórfica Ad hoc
   2) La declaración de tipos está mal, porque debería incluir una clase de tipos.
   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 de función en Haskell:

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

¿Cuál es el propósito de la función `incrementar` y cómo maneja el valor `Nothing`?
   1) La función incrementar lanza un error cuando se le pasa Nothing.
   2) La función incrementar siempre devuelve Just 1 independientemente de su entrada.
   3) La función incrementar solo funciona con valores Nothing y no con Just.
   4) La función incrementar incrementa el valor dentro de un Just en 1 y devuelve Nothing si se le pasa Nothing.
   5) ninguna de las anteriores.

d) Dada la siguiente declaración de tipo en Haskell:
`data Quizas a = Nada | Algo a`

Puedo afirmar que :

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

Ejercicio 2

Se va a representar el stock de un corralón de materiales de construcción, usando tipos en Haskell. Los materiales que tenemos en cuenta son: `Ladrillos, Viguetas, Cemento`. 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:

`Ladrillo`

- `TipoLadrillo`, que es un tipo enumerado con las siguientes opciones: `Ceramico`, `Hormigon`, `Tradicional`
- `UsoDeLadrillo`, que es un tipo enumerado con las siguientes opciones: `Pared`, `Techo`
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Vigueta`

- `Largo`, que es un sinónimo de `Float` indicando el largo de la vigueta
- `MaterialViga`, que es un tipo enumerado con las siguientes opciones: `CementoHierro` , `Madera` .
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Cemento`

- `MarcaCemento`, que es un tipo enumerado con las siguientes opciones: `Minetti`, `LomaNegra` .
- `Precio`, que es un sinónimo de `Int` indicando el precio

Para ello:

a) Definir el tipo `MaterialesConstruccion` que consta de los constructores `Ladrillo`, `Vigueta` y `Cemento`, constructores con parámetros descriptos arriba (Se deben definir también los tipos enumerados `TipoLadrillo, UsoDeLadrillo, MaterialViga, MarcaCemento`). **Los tipos** `MaterialesConstruccion` **y** `MaterialViga` **<u>no deben</u> estar en la clase** **`Eq`**, ni en la clase **`Ord`**.

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

```
cuantasViguetas :: [MaterialesConstruccion] -> MaterialViga -> Int
```

que dada una lista de `MaterialesConstruccion lm` y un valor `x` de `MaterialViga`, me devuelve un entero indicando la cantidad de viguetas que hay en `lm` con el material `x`.
NOTA: Dejar como comentario un ejemplo donde hayas probado la función `cuantasViguetas` con una lista con al menos 3 `MaterialesConstruccion`.

c) Definir igualdad para el tipo de `MaterialesConstruccion`: de tal manera que, dos valores de tipo `Ladrillo` son iguales sólo si tienen el mismo **tipo de ladrillo** y el mismo **uso de ladrillo**, dos `Vigueta` son iguales solo si tienen el mismo **largo** y el mismo

precio, mientras que dos Cemento son iguales si tienen la misma **marca** . Como es de suponer los `Ladrillo`, `Vigueta` y `Cemento` 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 el dueño de un dojo de karate pueda saber si sus alumnos de un curso pueden pasar al siguiente cinturón o no.

a) Definir un tipo recursivo `NotasDelDOJO`, que permite guardar las notas que tuvo cada alumno de un cinturón en el año. El tipo `NotasDelDOJO`, tendrá dos constructores:

   1) `EvolucionDelAlumno`, que tiene 6 parámetros:

   - `String`, para el nombre y apellido del alumno
   - `Color` (tipo enumerado con el `Color` del cinturon actual, `Blanco`, `Amarillo`, `Verde`)
   - `Int` (con la nota del primer kumite, entre 1 y 10)
   - `Int` (con la nota del segundo kumite, entre 1 y 10)
   - `Int` (con la nota del kata 1 a 10,)
   - `NotasDelDOJO,` recursión con el resto de las notas.

   2) `NoHayMasAlumnos`, 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 cinturón se describen a continuación, según las notas obtenidas:

- Si el alumno tiene cinturón `Blanco` o `Amarillo`, debe sacar más de 7 en alguno de los kumite, y haber tendido en el kata al menos un 6.
- Si el alumno tiene cinturón verde, debe tener al menos un 7 en cada kumite, y al menos un 8 en el kata.

b) Programar la función `pasaDeCinturon`, que toma como primer parámetro `notas` del tipo **`NotasDelDOJO,`** y como segundo parámetro el `nombre` del alumno de tipo **`String`** y retorna un valor de tipo **`Bool,`** indicando si el alumno con `nombre` es **pasa de cinturón o no**.

```
pasaDeCinturon :: NotasDelDOJO -> String -> Bool
```

NOTA: Dejar como comentario un ejemplo donde hayas probado `pasaDeCinturon` con un parámetro de tipo **`NotasDelDojo`** que tenga al menos 3 alumnos.

c) Programar la función `devolverColorK2` con la siguiente declaración:

```
devolverColorK2 :: NotasDelDOJO -> String -> Maybe Color
```

que toma una variable `notas` de tipo `NotasDelDOJO`, y como segundo argumento un `nombre`, que identifica el alumno, y en caso que el alumno esté en `notas` (con un cinturón de color **c**), retorna Just c y `Nothing` en caso contrario.

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