Parcial 1 - Algoritmos I Taller: Tema F

Ejercicio 1

En las siguientes preguntas marque la respuesta correcta:

a. Si tengo una función definida de la siguiente manera
```
f :: [a] -> a
```
puedo decir que:
   1. Es una función que usa alto orden
   2. Es un función polimórfica Ad hoc
   3. Es la función identidad
   4. Es una función polimórfica paramétrica
   5. Es una función recursiva.

b. Si tengo una función con la siguiente declaración:
```
f :: Integral a => a -> a -> Maybe a
f x 0 = Nothing
f x y = Just (mod x y)
```
puedo decir que:
   1. Es un función polimórfica paramétrica
   2. La declaración de tipos está mal, porque debería incluir la clase `Num`
   3. La definición de la función es incorrecta, debería utilizar `case`.
   4. Es un constructor
   5. Ninguna de las anteriores

c. En Haskell la expresión
```
(+1)
```
   1. Es una expresión inválida
   2. Es una función polimórfica adhoc
   3. Es un constructor del tipo `Int`
   4. Es equivalente a 1
   5. Ninguna de las anteriores

d. En la definición

```
data T a = A | B a | C
```

Según la definición `T` es
   1. Una función que depende de `a`
   2. Es un tipo recursivo porque el tipo `a` aparece tanto del lado izquierdo como en el lado derecho de la definición
   3. Es un tipo enumerado
   4. Es un tipo algebráico polimórfico
   5. Es un sinónimo de tipo

Ejercicio 2

Se va a representar un catálogo de componentes de PC, se debe tener en cuenta que será solo con fines didácticos por lo que no se incluyen todos los necesarios para armar una computadora funcional. Los componentes que tendremos en cuenta son el ***microprocesador***, la ***memoria RAM*** y la ***placa madre*** (o *motherboard*). Cada uno de estos componentes tienen distintas características que se tienen en consideración:

<u>De los microprocesadores</u>

- El fabricante o **marca** del microprocesador se representa con el tipo `MarcaMicro` que tiene dos constructores sin parámetros: `Intel` y `AMD`
- La **cantidad de núcleos** que tiene un microprocesador se representa con el tipo `Nucleos` que consta de constructores sin parámetros que son `DualCore`, `QuadCore`, `HexaCore` y `OctaCore`
- Para representar la velocidad o **frecuencia** de un microprocesador se usa el tipo `MaxFrec` que debe ser un sinónimo del tipo `Float`

<u>De la memoria RAM</u>

- La **marca** de la memoria se representa con el tipo `MarcaRAM` que tiene tres constructores sin parámetros que son `Kingston`, `Markvision` y `Patriot`
- La **cantidad de memoria** o capacidad se representa con el tipo `Gigas` que debe ser un sinónimo del tipo `Int`

<u>De las placas madres</u>

- La marca de la placa madre se representa con el tipo `MarcaPMadre` tres constructores sin parámetros `Asus`, `Asrock` y `MSI`
- El **chipset**, que es el conjunto de chips característico de una placa madre. Se representa con el tipo `Chipset` que tiene cuatro constructores sin parámetros cuatro que son `A630`, `B650`, `B660` y `B760`

a) Definir el tipo `ComponentePC` que debe tener tres constructores `Micro`, `RAM` y `PlacaMadre` que están asociados a los componentes microprocesador, memoria RAM y placa madre respectivamente. Los constructores deben tener los parámetros como se describen arriba (también se deben definir los tipos `MarcaMicro`, `Nucleos`, `MaxFrec`, `MarcaRAM`, `Gigas`, `MarcaPMadre` y `Chipset`). **No se debe usar** `deriving Eq` **ni** `deriving Ord` **para ninguno de los tipos.** Se puede sin embargo incluir los tipos en la clase `Show`

b) Definir de manera recursiva función `cuantosMicros`

```
cuantosMicros :: [ComponentePC] -> MarcaMicro -> Int
```

que dada una lista de componentes de PC `cs` y una marca de microprocesador `m` devuelve cuantos microprocesadores de la marca `m` hay en `cs`.

NOTA: Dejar como comentario un ejemplo donde se haya probado la función `cuantosMicros` con una lista con al menos 3 elementos donde dos de ellos sean microprocesadores de distintas marcas de `ComponentePC`.

c) Definir la igualdad para el tipo `ComponentePC` de manera tal que dos microprocesadores se consideran equivalentes si tienen la misma cantidad de nucleos; dos memorias RAM se consideran iguales si tienen la misma cantidad de gigas; dos placas madres se consideran iguales si tienen el mismo chipset. Como es de suponer `Micro`, `RAM` y `PlacaMadre` son distintos entre sí.

NOTA: Dejar como comentario en el código dos ejemplos en los que se prueba la igualdad.

Ejercicio 3

Se va a representar una lista de reproducción (parecido a las utilizadas en plataformas como Spotify, Youtube, etc) para canciones. Para ello

a) Definir un tipo recursivo `PlayList`, que permite guardar las canciones que se van reproduciendo. que tuvo cada alumno de un cinturón en el año. El tipo tendrá dos constructores:
   1) `Tema`, que tiene cinco parámetros con los siguientes tipos:
      - `Titulo`, que es el nombre de la canción y debe ser sinónimo de `String`
      - `Rank` es el ranking (pensarlo como cantidad de estrellas) el cual debe ser sinónimo de `Int`
      - `Estado`, que es un tipo enumerado con constructores `Reproducido` y `SinReproducir`
      - `Duracion`, que es la cantidad de segundos que dura el tema y debe ser un sinónimo de `Int`
      - `PlayList`, recursión con el resto de las canciones
   2) `SinTemas`, que es un constructor sin parámetros, similar al de la lista vacía, para indicar que se terminaron las canciones.

   NOTA: Los tipos nuevos definidos no deben estar en la clase `Eq` ni `Ord`

b) Programar la función `segundos_restantes`

```
segundosRestantes :: PlayList -> Rank -> Duracion
```

   que dada una playlist `pls` y un rank `r` devuelve la cantidad de segundos que suman las canciones de `pls` que aún no han sido reproducidas y que tienen ranking mayor o igual a `r`.

   NOTA: Dejar como comentario un ejemplo donde se haya probado `segundosRestantes` con un parámetro de tipo `PlayList` que tenga al menos 3 canciones.

c) Programar la función

```
estadoDelTema :: Titulo -> PlayList -> Maybe Estado
```

   que dado un título de tema `t` y una playlist `pls` si el tema está en `pls` con estado `e` debe devolver `Just e` y si el tema no está debe devolver `Nothing`

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