Parcial 1 - Algoritmos I Taller: Tema G

Debes entregar el código completo en los campos correspondientes de cada ejercicio del formulario en el que completaste tus datos personales. Este código debe poder ejecutarse en haskell sin errores. Te recomendamos para ello que pruebes con diferentes ejemplos antes de entregar.

Descripción del problema

En las billeteras virtuales normalmente encontramos descuentos y reintegros. Vamos a diseñar un sistema que nos permita tener funciones para ver qué billetera virtual usar para ahorrar más dinero.

Ejercicio 1:

a)

Definir el tipo Ahorro que consta de dos constructores `Descuento` y `Reintegro` con los siguientes parámetros:

- El constructor `Descuento` debe tomar como parámetros el nombre de la billetera (por ejemplo "PersonalPay", "Uala", "BnaMas", "NaranjaX"), la categoría, es decir en qué es el descuento (por ejemplo "Combustible" , "Supermercado", "ComidaRapida"), el porcentaje de descuento, el tope de descuento y la lista de días de la semana que funciona el descuento.
- El constructor `Reintegro` debe tomar como parámetros el nombre de la billetera virtual en que te darán reintegro, la categoría, es decir en qué es el descuento (por ejemplo "Combustible", "Supermercado", "ComidaRapida") y el monto de reintegro.

*Ayuda: para el día de la semana use **data Dia = Lunes | Martes | Miercoles | Jueves | Vienes | Sabado | Domingo deriving (Eq, Show)**. Puede usar la función elem del Preludio para saber si un elemento pertenece a una lista.

b)

A partir del tipo definido en el punto anterior, definir los siguientes ejemplos de Ahorro:

```
naftaPP :: Ahorro
naftaPP = <COMPLETAR>
```

correspondiente al descuento “PersonalPay” en “Combustible”, 25, 2500, días Lunes y Martes.

```
macNX :: Ahorro
macNX = <COMPLETAR>
```

correspondiente al reintegro “NaranjaX”, en concepto “comida rapida”, por $3000

c)

Definir la función `hayDescuentoDia :: Ahorro -> Dia -> Bool` que dado un ahorro y un Dia determinado, devuelve `True` si es el ahorro es un Descuento (notar que no puede ser reintegro) del Dia dado, False en caso contrario.

d)

Definir la función `esDescuentoCombustible :: Ahorro -> Bool` que dado un `Ahorro`, devuelve `True` si el ahorro es un Descuento en Combustible.

e)

Definir la función `mayorDescuentoOReintegro :: [Ahorro] -> Int` que dada una lista de ahorros, devuelve el mayor de los descuentos o reintegros, tomando en el caso de Descuento el tope de descuento. En caso que no haya ahorros devuelve 0.

Ejercicio 2:

Dado el tipo recursivo `ColaAhorro` definido de la siguiente manera

```
data ColaAhorro = NoHayAhorro | AgregarAhorro Ahorro ColaAhorro deriving Show
```

podemos definir los ahorros de María como

```
ahorrosMaria :: ColaAhorro

ahorrosMaria = AgregarAhorro naftaPP (AgregarAhorro macNX NoHayAhorro))
```

Definir la función `ahorrosCombustible :: ColaAhorro -> ColaAhorro`
que dada una cola de ahorros q, devuelve la cola de ahorros que tiene solamente los descuentos o reintegros en combustible (en el mismo orden que aparecen en q).