Parcial 1 - Algoritmos I Taller: Tema D

Ejercicio 1

Vamos a representar vehículos usando tipos definidos en Haskell. Para ello:

a) Definir el tipo `Color` con constructores `Rojo`, `Negro` y `Azul`. Los constructores no toman parámetros. El tipo `Color` **no debe estar en la clase Eq**, pero sí en la clase `Show`. Programar la función

```
mismo_color :: Color -> Color -> Bool
```

que dados dos valores `c1` y `c2` de tipo `Color` devuelve `True` si y sólo `c1` y `c2` son el mismo color (se construyen con el mismo constructor).

b) Definir el tipo `Tipo` que tiene constructores `Auto`, `Moto` y `Camion`. Luego definir el tipo `Vehiculo` que consta de un único constructor `Cons` que toma dos parámetros, el primero de tipo `Tipo` y el segundo del tipo `Color`. Finalmente programar la función

```
pintar_auto :: Vehiculo -> Color -> Maybe Vehiculo
```

que dados un vehículo `v` y un color `c` devuelve `Nothing` si `v` no es un auto. Si `v` es un auto, devuelve un auto de color `c` usado constructor `Just`.

Ejercicio 2

Programar la función

```
solo_de_color :: Color -> [Vehiculo] -> [Tipo]
```

que dado un color `c` y una lista de vehículos `cs` devuelve los tipos de vehículos de `cs` que tienen color `c`.

- Inventar un ejemplo concreto con un lista de vehículos de al menos 3 elementos, ejecutarlo y decirlo como comentario en lo que se suba al parcial.

Ejercicio 3

Vamos a implementar una lista de compras usando tipos en Haskell. Para ello definir el tipo `Precio` como un sinónimo de `Int` y al tipo `Nombre` como sinónimo de `String`. Además definir el tipo `Producto` que tiene un único constructor `Item` que toma dos parámetros, el primero de tipo `Nombre` y el segundo de tipo `Precio`. Por último a partir de los tipos anteriores, definir el tipo recursivo `Compra` cuyos constructores son:

- `AgregarProd`: Toma tres parámetros. El primero de tipo `Producto` (el producto que se agrega a la compra), el segundo de tipo `Int` (la cantidad de productos del mismo tipo que se compra) y el tercero de tipo `Compra` (la lista de compras a la que se agrega el nuevo producto).
- `Nada`: No toma parámetros y representa la compra vacía.

Finalmente programar la función

```
costo :: Compra-> Precio
```

que dado una compra `comp` devuelve la suma de los precios de los productos comprados. Recordar que cada producto se puede comprar n-veces.

Ejercicio 4*

Programar la función

```
arbol_busca :: Arbol (Int, String) -> Int - > Maybe String
```

que dado un árbol `as` y un entero `k` devuelve la cadena que aparece asociada a `k` dentro del árbol `as`, o `Nothing` si no existe esa clave.