Parcial 1 - Algoritmos I Taller: Tema A

Ejercicio 1

Se van a implementar algunos aspectos del juego *Poker* en Haskell.

a) Definir el tipo `Palo` que consta de los constructores `Treboles`, `Corazones`, `Picas`, `Diamantes`. Los constructores no toman parámetros. **El tipo `Palo` no debe estar en la clase `Eq`**. Luego programa la función usando *pattern matching*:

```
mismo_palo :: Palo -> Palo -> Bool
```

que dados dos valores `p1` y `p2` del tipo `Palo` debe devolver `True` cuando `p1` y `p2` son el mismo palo (se construyen con el mismo constructor) y `False` en caso contrario.

Si se usan más de cinco casos, este apartado sumará menos puntaje.

b) Definir el tipo `Naipe` que representa una carta de poker. Tiene constructores:

- Constructor `Numerada`: Toma dos parámetros, el primero de tipo `Numero` y el segundo de tipo `Palo`
- Constructores `Rey`, `Reina`, `Jota`, `As`: Todos son constructores con un sólo parámetro de tipo `Palo`

El tipo `Numero` debe ser un sinónimo del tipo `Int`.

c) Programar la función

```
valor_naipe :: Naipe -> Int
```

teniendo en cuenta que el valor de una carta será:

- Si es una carta numerada : Su valor es el número de la carta.
- Si es el naipe `Jota` : Su valor es 11
- Si es el naipe `Reina` : Su valor es 12
- Si es el naipe `Rey` : Su valor es 13
- Si es el naipe `As` : Su valor es 14

d) Incluir el tipo `Naipe` en la clase `Ord` de manera tal que un naipe se considere mayor que otro si su valor según la función `valor_naipe` es más grande.

Ejercicio 2

a) Programar de manera recursiva la función

```
solo_numeradas :: [Naipe] -> Palo -> [Numero]
```

que dada una lista de cartas `ns` y un palo `p` devuelve una lista con los números de las cartas numeradas (las que no son ases, jotas, reyes ni reinas) de `ns` que son del palo `p`.

b) Escribir una lista de naipes con al menos tres elementos, donde uno de ellos debe ser una figura, y otro debe ser una carta numerada.

c) Escribir el resultado de `solo_numeradas` para la lista del punto b)

Ejercicio 3

Basados en el tipo `ListaAsoc` del *Proyecto 2*, programar la función:

```
la_menores :: ListaAsoc a b -> b -> ListaAsoc a b
```

que dada una lista de asociaciones `la` y un dato `x` devuelve una nueva lista de asociaciones con las asociaciones de `la` cuyos valores son menores que `x`. Completar el tipado de la función para incluir los *type classes* necesarios para programarla.

Ejercicio 4*

a) Programar la función

```
a_esCota_sup :: a -> Arbol a -> Bool
```

que dado un valor `e` de tipo `a` y un árbol `as` indica si `e` es una cota superior de todos los elementos dentro del árbol `as`. Es decir indica si `e` es mayor o igual a todos los elementos del árbol `as`. Completar el tipado de la función para incluir los *type classes* necesarios para programarla

b) Inventar un ejemplo de uso de la función creando un árbol con al menos 3 elementos

c) Escribir el resultado de la función aplicada al ejemplo del inciso b)