Parcial 1 - Algoritmos I Taller: Tema E

Ejercicio 1

Se van a implementar algunos aspectos de las dedicaciones de las Personas en Haskell.

a) Definir el tipo `Dedicacion` que consta de los constructores: `Simple`, `Semi`, `Full`, `Investigador`. Los constructores no toman parámetros. **El tipo `Dedicacion` no debe estar en la clase `Eq`**. Luego programa la función usando *pattern matching*:

```
misma_dedicacion :: Dedicacion -> Dedicacion -> Bool
```

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

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

b) Programar la función

```
horas_trabajo :: Dedicacion -> Cantidad
```

- Si es Simple devuelve 10
- Si es Semi devuelve 20
- Full devuelve 50
- Investigador devuelve 60

Cantidad debe ser definida como sinónimo de Int

c) Agregar al tipo Persona la Dedicacion a los constructores Decane, Docente y NoDocente.

d) Incluir el tipo `Dedicacion` en la clase `Ord` de manera tal que un dedicacion se considere mayor que otra si su valor según la función `horas_trabajo` es más grande.

Ejercicio 2

a) Programar de manera recursiva la función

```
solo_dedicacion :: [Persona] -> Dedicacion -> [Persona]
```

que dada una lista de personas `ps` y un dedicacion `r` devuelve una lista con las personas de `ps` que tienen dedicación `r`.

b) Escribir una lista de personas con al menos tres elementos, donde al menos uno de ellos debe tener dedicación, y otro no debe tener dedicación.

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

Ejercicio 3

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

```
la_mismo_valor :: ListaAsoc a b -> b -> ListaAsoc a b
```

que dada una lista de asociaciones `la,` devuelve una nueva lista de asociaciones con las asociaciones de `la` cuyos valores son iguales al segundo parámetro. Completar el tipado de la función para incluir los *type clases* necesarios para programarla.

Ejercicio 4*

a) Programar la función

```
dar_subarbol :: a -> Arbol a -> Arbol a
```

que dado un valor `e` de tipo `a` y un árbol `as` devuelve el subarbol que tiene como raiz a e o Hoja en caso de no existir. 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)