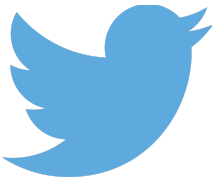

Pipper++

Las redes sociales hoy en día están tan de moda que se generan una cada día. En el día de hoy nos han pedido a nosotros hacer una más para agregar a la lista.
Nos dicen que esta red social va a permitir que un usuario pueda seguir personas, lo cual modelaremos de la siguiente forma:

```
Data Usuario = Usuario{
      nombre :: Nombre,
      seguidos :: [Nombre]
}
usuarios = [ Usuario "@marsupialRengo" ["@don_churrasco", "@dodainOk"], Usuario
"@don_churrasco" [], Usuario "@lapipi"  ["@dodainOk"], Usuario "@dodainOk" ["@lapipi",
"@marsupialRengo"] ]
```

Cada usuario puede realizar Pips. Un Pip es un mensaje que podrán leer sus seguidores, marcarlos como favoritos o Repipearlos. Repipear un Pip es hacer que ese Pip ahora sea un mensaje tuyo. Un mensaje tiene la siguiente estructura:

```
data Mensaje = Mensaje {
      usuario :: Nombre,
      Texto :: Texto,
      Repips :: Cantidad,
      Favs :: Cantidad }

type Nombre = String
type Texto = String
type Cantidad = Int

mensajes = [ Mensaje "@dodainOk" "Las personas que viajan en tren se quejan de llenos" 20
100, Mensaje "@lapipi" "En mi mundo todos son un pony" 1 0, Mensaje "@lapipi" "y comen
arcoiris y su popó son mariposas" 0 0, Mensaje "@dodainOk" "No hay problema en cometer
errores, el secreto es no pasarlos a producción" 1 3, ... ]

quickSortBy f []     = []
quickSortBy f (x:xs) =
      sublistaQueCumple ((< f x).f)  ++  [x]  ++  sublistaQueCumple ((>= f x).f)
      where sublistaQueCumple condicion = quickSortBy f . filter condicion $ xs
```

Ayuda: Definir la función find, para que funcione de la siguiente manera.

```
> find even [1..]
2
```

1. Queremos saber, dados dos nombres de usuarios, si el primero sigue al segundo (según la lista *usuarios*).

```
>sigueA "@dodainOk" "@lapipi"
True    (Porque @dodainOk tiene entre la lista de siguiendo a @lapipi)
> sigueA "@don_churrasco" "@marsupialRengo"
False    (Porque @don_churrasco no sigue a nadie)
```

2. Hacer la función `seguidores` que, dado un nombre de usuario y una lista de usuarios, devuelve los nombres de sus seguidores (según la lista *usuarios*, nuevamente).

```
> seguidores "@don_churrasco"
[Usuario {nombre = "@marsupialRengo", seguidos = ["@don_churrasco","@dodainOk"]}]
```

3. Definir `trencitoDeUsuarios`, que se cumple para una lista de nombres de usuarios en la cual cada uno sigue al siguiente.

```
> trencitoDeUsuarios ["@marsupialRengo", "@don_churrasco", "@lapipi", "@dodainOk" ]
False
```

(@marsupialRengo (el primero) sigue a @don_churrasco (el segundo), pero este último ya no sigue a @lapipi (la tercera) y por lo tanto no es un trencito)

4.

a. Definir una funcion favear un mensaje que permita agregar un favorito

```
> favear (Mensaje "@dodainOk" "Las personas que viajan en tren se quejan de llenos"
20 100)
Mensaje "@dodainOk" "Las personas que viajan en tren se quejan de llenos" 20 101
```

b. Definir una función `repipear` un mensaje por parte de un nombre de usuario, que devuelve el nuevo mensaje, con el mismo texto que el original excepto que se agrega el nombre del usuario y ": " al principio, con 0 repips y 0 favoritos".

```
> repipear (Mensaje "@dodainOk" "Las personas que viajan en tren se quejan de
llenos" 20 100) "@don_churrasco"
Mensaje "@don_churrasco" "@dodainOk: Las personas que viajan en tren se quejan de
llenos" 0 0
```

c. Definir `repipearTodos` que, dada una lista de mensajes y un nombre de usuario, devuelva una lista con todos los mensajes originales más los repipeados por el usuario, con la salvedad que los "originales" tienen que tener un "repip" más en la lista resultante.

5. Hacer la función `usuariosTeSigoSiMeSeguis` que devuelven los nombres de los usuarios que siguen a todos los que los siguen.

```
> usuariosTeSigoSiMeSeguis usuarios
["@marsupialRengo","@lapipi","@dodainOk"]
```

6.

Nota: No se pueden definir nuevas funciones auxiliares en <u>todo</u> el punto 6, ni usar definiciones locales o expresiones lambda excepto en el 6.2.c. Sí se pueden usar las funciones mencionadas en este parcial o creadas para puntos anteriores.

1. Hacer una función `losMasAlgo`, que dada una lista y una función de valoración devuelve los 3 elementos que mayor valor le dan a la función.

2. Usar `losMasAlgo` para definir las siguientes funciones:

a. Hacer la función `losMasRepipeados`, que dada una lista de pips devuelve los 3 pips con más repips.

b. Hacer la función `losMasGrandesPips`, que dada una lista de pips devuelve los 3 pips con más caracteres.

c. Hacer la función `losMasPipeadores`, que dada una lista de usuarios devuelve los 3 usuarios con más pips.

7. Indicar el tipo de `funcionLoca`.

```
funcionLoca k p r = map p . filter ((> r).k)
```