Hoja de Ejercicios 4

Listado de ejercicios para poner en práctica los conocimientos adquiridos sobre definición de tipos sinónimos y nuevos tipos, tipos recursivos y tipos recursivos polimórficos. Y también sobre el manejo de clases de tipos en Haskell.

Ejercicios:

a) Se quiere ordenar los elementos de una lista (cuyos elementos son comparables) mediante el algoritmo del quicksort.

b) Se pide implementar una función que dada un número (de cualquier tipo que soporte la operación de división) y una lista de números del mismo tipo, divida a ese número por cada uno de los elementos contenidos en la lista y devuelva una lista con el resultado.
Ejemplos de aplicación de la función son:

```
> divisiones 5 [1,2,3]
[Just 5,Just 2,Just 1]

> divisiones 5 [1,2,3,0,9,10]
[Just 5,Just 2,Just 1,Nothing,Just 0,Just 0]
```

c) Dado un nuevo tipo de datos para representar un árbol binario de cualquier tipo, definido como sigue:

```
data Arbol a = AV | Rama (Arbol a) a (Arbol a)
```

Se pide definir una función que visualice el árbol por pantalla de una determinada forma: separando cada hijo izquierdo y derecho por "|", la raíz entre guiones y cada nivel diferente del árbol por "( )". Ejemplos de aplicación de la función sería los siguientes:

```
> mostrarArbol (Rama (Rama (Rama AV 60 AV) 8 AV) 5 (Rama AV 4 AV))
"((60)|-8-|())|-5-|(4)"

> mostrarArbol (Rama AV 5 (Rama AV 4 AV))
"()|-5-|(4)"
```

¿Sería equivalente a declarar el nuevo tipo de datos `Arbol` como una instancia de la clase `Show`?

```
data Arbol a = AV | Rama (Arbol a) a (Arbol a) deriving Show
```

d) Dado el siguiente tipo de datos que representa un árbol binario:

```
data Arbol a = AV | Rama (Arbol a) a (Arbol a) deriving Show
```

Se pide definir una función que calcule el espejo de un árbol.

Ejemplos de aplicación de la función serían:

```
> espejo (Rama (Rama (Rama AV 60 AV) 8 AV) 5 (Rama AV 4 AV))
Rama (Rama AV 4 AV) 5 (Rama AV 8 (Rama AV 60 AV))

> espejo (Rama AV 5 (Rama AV 4 AV))
Rama (Rama AV 4 AV) 5 AV
```

e) Se quiere poder mostrar por pantalla los datos de los estudiantes matriculados en una universidad que pertenezcan a alguna de las asociaciones de ésta (culturales, deportivas, de representación estudiantil, etc.). Para ello se deberán crear nuevos tipos de datos que representen:
   - Estudiante, de cada uno se debe disponer del nombre y titulación
   - Titulación, que pueden ser tres: Grado II, Grado II_ADE, Grado ADE
   - Lista de estudiantes matriculados
   - Lista de estudiantes que pertenecen a asociaciones

   Un ejemplo de aplicación de la función que se pide podría ser:

```
> mostrarAlumnosAsociaciones(listaMatriculados,listaAsociaciones)
"(Carlos Calle,GradoADE_II)(Irene Plaza,GradoADE)"
```

   Donde Carlos Calle e Irene Plaza son los únicos estudiantes matriculados que pertenecen a algún tipo de asociación en la universidad.

f) Se quiere poder representar una fecha de la siguiente forma: dd/mm/aaaa, para ello se deberá crear un nuevo tipo de datos en Haskell. Por ejemplo, si se crea un nuevo tipo de datos cuyo constructor de datos es Fecha, en el intérprete al poner fechas concretas nos devolvería la representación de la fecha que hayamos definido:

```
> Fecha 10 10 2013                > Fecha 24 12 2012
10/10/2013                        24/12/2012
```

g) Teniendo en cuenta el nuevo tipo de datos Fecha definido anteriormente, se pide una función que sea capaz de comparar dos fechas. Ejemplos de aplicación de la función serían:

```
> mismaFecha (Fecha 10 10 2013) (Fecha 10 10 2013)
True

> mismaFecha (Fecha 10 11 2013) (Fecha 10 10 2013)
False
```

h) Teniendo en cuenta la definición de la función qs del apartado (b) de este listado de ejercicios, se pide ordenar una lista de fechas mediante quicksort. Ejemplos de aplicación de la función serían:

```
> qs [(Fecha 10 10 2013), (Fecha 24 12 2012), (Fecha 10 09 2013), (Fecha
12 12 2013)]
[24/12/2012,10/9/2013,10/10/2013,12/12/2013]
```

i) Se pide crear una nueva clase de tipos, llamada `Coleccion`, para representar colecciones de datos de cualquier tipo, donde los tipos pertenecientes a esta clase tendrán el siguiente comportamiento:

```
esVacia: función para saber si la colección está vacía.
insertar: insertará un nuevo elemento en la colección.
primero: devolverá el primer elemento de la colección.
eliminar: eliminará un elemento de la colección.
size: devolverá el número de elementos de la colección.
```

Algunas de las funciones anteriores variarán su implementación en función del tipo de colección particular que sea instancia de la clase `Coleccion`. Por ello, se pide crear dos instancias diferentes de esta clase para los dos nuevos tipos de datos que se presentan a continuación:

```
data Pila a = Pil [a] deriving Show
data Cola a = Col [a] deriving Show
```

El primero de ellos representa una estructura de datos LIFO con elementos de tipo `a`. El segundo representa una estructura de datos FIFO de elementos de tipo `a`.

Ejemplos de aplicación de las funciones para ambos tipos de datos serían:

```
> insertar 10 (Col [1,2,3,4])
Col [1,2,3,4,10]

> insertar 10 (Pil [1,2,3,4])
Pil [1,2,3,4,10]

> primero (Col [1,2,3,4,10])
1

> primero (Pil [1,2,3,4,10])
10

> eliminar (Col [1,2,3,4,10])
Col [2,3,4,10]

> eliminar (Pil [1,2,3,4,10])
Pil [1,2,3,4]
```