Ejercicios Prácticos de Programación Declarativa

Sesión de laboratorio 4

Curso 2019/20

- Realizad los ejercicios de la Sesión 4 (al menos los 3 primeros). Subid un fichero .hs al Campus Virtual antes de que acabe la clase. Es suficiente con que los suba uno si lo hacéis entre dos.
- No olvidéis incluir comentarios y poner vuestros nombres en las primeras líneas del fichero.

1. Definir un tipo `Nat` para representar números naturales con la aritmética de Peano. Es decir, toda expresión de tipo `Nat` será `Cero` o el sucesor de un elemento de `Nat` (expresión de la forma `Suc e` con `e:: Nat`. Declarar el tipo como instancia de `Eq` y `Ord` usando `deriving`.
   - Definir operadores infijos para calcular la suma y el producto de elementos de `Nat`.
   - Definir una función `natToInt` que convierta una expresión de tipo `Nat` en su equivalente en el tipo `Int`.
   - Utilizar `natToInt` para declarar `Nat` como instancia de `Show`. *(Para declarar un tipo* `T` *como instancia de Show basta con definir la función* `show::T->String`*, no hace falta definir las otras funciones de la clase Show).*
2. Definir un tipo para representar números complejos y declararlo como instancia de las clases `Eq`, `Num`, y `Show` usando `deriving` solo cuando sea conveniente. Por ejemplo, `show` del término Haskell que represente al complejo $2 + 3i$ será el string `''2+3i''` (análogamente `''2-3i''` para $2 - 3i$). Tendrás que redefinir los métodos de `Num` para expresar las operaciones aritméticas entre números complejos, rdefine al menos `(+)`, `(-)` y `(*)`.
3. Definir una clase de tipos `Medible` que disponga de un método `tamanyo::a ->Int` que se pueda aplicar a cada tipo `a` de dicha clase. Declara algunos tipos como instancia de la clase `Medible`, por ejemplo `Bool`, `[a]`, `(a,b)`, definiendo la función `tamayo` para cada uno de ellos.
4. Definir un tipo enumerado `Direccion` con cuatro valores que representen movimientos (*arriba, abajo, izquierda, derecha*) por una cuadrícula en el plano con coordenadas enteras. Convertirlo en instancia de `Eq, Ord, Show` usando `deriving`. Definir una función `destino` que al aplicarse a un punto del plano y una lista de movimientos, devuelva el punto final al que se llega.

   Opcional: Definir una función `trayectoria` que al aplicarse a un punto del plano y una lista de movimientos, devuelva la lista de puntos por los que se pasa al aplicar `movs` a `punto`.
5. Definir un tipo de datos polimórfico para representar árboles generales, en los que cada nodo tiene una información y n hijos ($n \geq 0$, y puede variar con cada nodo). No se consideran árboles vacíos.
   - Programar las siguientes funciones:
     - `listaHojas t`, que obtiene la lista de las informaciones de todas las hojas del árbol `t`.

    - `listaNodos t`, que obtiene la lista de las informaciones de todos los nodos del árbol `t`.
    - `repMax t`, que devuelve el árbol resultante de poner como información de todos los nodos del árbol `t` la información más grande que aparece en `t`.
- Declarar **explícitamente** el tipo de los árboles como instancia de la clase `Ord` (usando `instance`), de manera que el orden definido sea el mismo que resultaría de usar `deriving Ord`.
- Declarar el tipo de los árboles como instancia de la clase `Show`, de manera que la vista en pantalla de un árbol sea visualmente más atractiva que lo que nos da el poner simplemente `deriving Show`.