

Relación de ejercicios nº 2

1.- Dado el siguiente tipo de árboles binarios:

```
data Arbus a = Vac | Nodo (Arbus a) a (Arbus a)
```

- Definir una función de tipo “fold” para el tipo Arbus a
- Usando la función del apartado anterior, definir una función que, dados un predicado y un árbol, devuelve el número de nodos que verifican el predicado.

2.- Sea el tipo type ArPares a = Arbus (a, a). Definir una función que, dado un árbol del tipo ArPares a y empleando la función del ejercicio 1.-b) determine el número de nodos (x,y) tales que x es menor que y.

3.- Definimos el siguiente tipo de expresiones aritméticas

```
data Expr = Lit Int |  
          Expr :+: Expr |  
          Expr :-: Expr |  
          Expr *: Expr |  
          Expr :/: Expr
```

Definir una función `foldExpr` para, en función de ella, definir funciones que permitan:

- evaluar una expresión
- calcular el número de operadores de una expresión.

4.- Repetir el ejercicio anterior pero cambiando el tipo de datos por

```
data Expr = Lit Int |  
          Op Ops Expr Expr  
data Ops = Mas | Sub | Mul | Div
```

Añadir después el operador `Mod` ¿Qué cambios hay que hacer?

5.- Intentando usar la concatenación como constructora, definimos el siguiente tipo de datos :

```
data CLista a = Nil  
              | Unit a  
              | Conc (CLista a) (CLista a)
```

La intención es que `Nil` represente a `[]`, `Unit x` represente `[x]` y `Conc xs ys` represente `xs++ys`. Sin embargo, hay expresiones diferentes que representan al mismo objeto, e.g.

```
Conc xs (Conc ys zs) == Conc (Conc xs ys) zs
Conc zs Nil == zs
```

Para conseguir dichas igualdades, definir el tipo algebraico `CLista a` como una instancia apropiada de la clase **Eq**.

Después definir una función que obtenga el primer elemento de una lista de tipo `CLista a`.

6 - Sean el siguiente tipo de árboles n-arios:

```
data Arbn a = Nodo a [Arbn a]
```

y la siguiente función de plegado

```
foldt :: (a -> [b] -> b) -> Arbn a -> b
foldt f (Nodo x ts) = f x (map (foldt f) ts)
```

- a) Definir una función `g` de tal manera que `foldt g arbol` obtenga la rama más larga de `arbol`.
- b) Usando `foldt` definir funciones que, dado un árbol n-ario, calculen:
 - el número de nodos
 - profundidad
 - recorrido en preorden

7 - Generalizar el tipo `Maybe a` para permitir mensajes en el caso de `Nothing`, esto es definir un tipo nuevo:

```
data Error a = Err String | OK a
```

Definir la función que generalize a la predefinida `maybe`.