FUNDAMENTOS DE LA COMPUTACIÓN
LABORATORIO 1
SEPTIEMBRE 2017

0. Preludio

Recuerde incluir la siguiente línea como primera del programa:

```
{-#LANGUAGE GADTs, EmptyDataDecls, EmptyCase #-}
```

Luego, se define el módulo donde se va a trabajar y se importa el laboratorio anterior:

```
module Lab2 where
import Prelude (Show)
import Lab1
```

1. Tipos Finitos

Ej.1. El siguiente tipo inductivo representa las cargas eléctricas:
`data Carga where {Positivo::Carga ; Negativo::Carga } deriving Show`
Definir la función `opuesto::Carga -> Carga` que retorne la carga opuesta a una carga dada, utilizando la expresión `case` correspondiente al nuevo tipo.

Ej.2. El siguiente tipo inductivo representa los días de la semana:
`data Dia where {Lu::Dia;Ma::Dia;Mi::Dia;Ju::Dia;Vi::Dia;Sa::Dia;Do::Dia} deriving Show`

(a) Definir la función `siguiente::Dia -> Dia` que retorne el siguiente día de la semana de uno dado, utilizando la expresión `case` correspondiente al nuevo tipo.

(b) Definir la función `laborable::Dia -> Bool` que retorna `True` para los días laborables y `False` para los no laborables.

Ej.3. Se quiere jugar al juego de Piedra, Papel y Tijera. Éste es un juego de manos en el cual existen tres elementos: la **piedra** que vence a la tijera rompiéndola, la **tijera** que vencen al papel cortándolo, y el **papel** que vence a la piedra envolviéndola.
Se pide:

(a) Definir en Haskell un tipo finito `PPT` que represente los tres elementos del juego.
(b) Definir la función `gana::PPT-> PPT-> PPT` que recibe dos elementos y retorna el que gana. En caso de empate, deberá devolver el elemento que lo causó.

2. Estructuras abstractas (Clases)

Las clases `Eq` y `Ord` están definidas como sigue:

```
class Eq a where
  { (==),(/=)::a->a->Bool ;
    (/=) = \x y -> not (x==y)  }


class Eq a => Ord a  where
 { (<=),(<),(>=),(>)::a->a->Bool ;
```

```
  (>) =  \x y -> not (x <= y)  ;
  (>=) = \x y -> (x > y) || (x == y);
  (<) = flip (>)}
```

Ej.4. Definir las clases `Eq` y `Ord` en Haskell.

Ej.5. Definir las instancias de `Eq` y `Ord` para `Bool` (para la última consideraremos que `False` es menor que `True`).

```
instance Eq Bool where
  (==) = ...

instance Ord Bool where
  (<=) =
```

Ej.6. Definir las instancias de `Eq` y `Ord` para el tipo `PPT` de modo tal que la relación `>` refleje quién gana entre dos expresiones del tipo.

```
instance Eq PPT where
  (==) = ...

instance Ord PPT where
  (<=) = ...
```

Ej.7. Definir las funciones `minimo::Ord a => a -> a -> a` y `maximo::Ord a => a -> a -> a` usando las funciones de la instancia de `Ord` para `a`.

Ej.8. Definir `min3::Ord a => a -> a -> a -> a` que calcula el mínimo de tres elementos de un tipo ordenado.