Programación declarativa **(30 de noviembre de 2010)**

Apellidos:

Nombre:

Grupo 1A

Una asociación de datos y enteros se dice fallida si a un mismo dato se le asocia un número y su opuesto.

Ejercicio 1 Define el predicado

```
esFallida ::  Eq a => [(a, Int)] -> Bool
```

que determine si una asociación es fallida. Por ejemplo:

```
esFallida [('a', 1), ('b',(-1)), ('c', 0)] ==> False
esFallida [('a', 1), ('a',(-2)), ('c', 0)] ==> False
esFallida [('a', 1), ('c', 0), ('a',(-1))] ==> True
```

Sintaxis de la lógica clausal:

- Un átomo es una variable proposicional (utilizaremos los caracteres para denotar las variables proposicionales).

  ```
  type Atomo = Char
  ```

- Un literal es un átomo o la negación de un átomo.

  ```
  data Literal = Pos Atomo | Neg Atomo deriving (Eq,Ord,Show)
  ```

- Una cláusula es un conjunto finito de literales.

  ```
  type Clausula = [Literal]
  ```

Por ejemplo, la cláusula `{¬ p, q, p}` vendrá representada por:

```
cl1 ::  Clausula
cl1 = [Neg 'p', Pos 'q', Pos 'p']
```

Ejercicio 2 Define la funcion

```
variables ::  Clausula -> [Atomo]
```

tal que `(variables c)` es el conjunto de variables que aparecen en la cláusula `c`. Por ejemplo:

```
variables cl1 = ['p', 'q']
```

Ejercicio 3 Define la función

```
resolvente :: Literal -> Clausula -> Clausula -> Clausula
```

tal que `(resolvente l c1 c2)` devuelve la resolvente de `c1` y `c2` respecto del literal `l` (`l` es un literal de `c1` y su complementario es un literal de `c2`).

```
resolvente (Pos 'p') [Pos 'p',Neg 'q'] [Neg 'p', Pos 'r',
Neg 'q'] ==> [Neg 'q',Pos 'r']
resolvente (Neg 'q') [Pos 'p',Neg 'q'] [Neg 'p', Pos 'q',
Neg 'r'] ==> [Pos 'p',Neg 'p',Neg 'r']
```