FUNDAMENTOS DE LA COMPUTACIÓN
LABORATORIO 1 - BOOL

1. Para empezar

Incluir las siguientes líneas como primeras del programa:

```
{-#LANGUAGE GADTs, EmptyDataDecls, EmptyCase #-}
{-# OPTIONS_GHC -fno-warn-tabs #-}
```

La primera permite realizar definiciones de tipos con el formato visto en calse. La segunda es para poder usar caracteres de tabulación "tab" en el programa sin que el compilador genere *warnings*.

2. Módulos

Lo que uno escribe en Haskell es en general un **módulo.** La siguiente línea será la que le da nombre al módulo. El nombre del módulo debe comenzar con mayúscula:

```
module Lab1 where
```

Finalmente, la siguiente línea permitirá importar del Preludio (la biblioteca estándar de Haskell) sólo la definición de la clase Show, que permite mostrar los resultados en pantalla. El resto de las funciones del Preludio no se importará para permitirnos efectuar definiciones que no se contradigan con las definidas en el Preludio de Haskell:

```
import Prelude (Show)
```

3. Funciones

Dado que para programar es preferente usar caracteres ASCII, Haskell no hace uso del carácter λ; en su lugar usa la barra inversa \.

4. Los Booleanos

Para definir en Haskell el tipo de los Booleanos se utilizará la siguiente notación:

```
data Bool where {False::Bool ; True::Bool}
  deriving Show
```

La segunda línea (`deriving Show`) sirve sólo para que Haskell pueda mostrar los valores de `Bool` por pantalla.
Es importante ponerla con espacios adelante, ya que es la forma de indicarle al compilador que la misma forma parte de la definición del tipo.

Ejercicios

Ej.1. Programar `not` y ponerla prueba en todos los casos posibles. ¿Cuántos son?

Ej.2. Programar en Haskell los siguientes conectivos Booleanos y verificar en cada caso la tabla de verdad del conectivo, probándolo con todos los valores posibles (son 4 combinaciones en cada caso):

```
(||) :: Bool -> Bool -> Bool
(||) = ....

(&&) :: Bool -> Bool -> Bool
(&&) = ....

xor :: Bool -> Bool -> Bool
xor = ...

(>>) :: Bool -> Bool -> Bool
(>>) = ...
```

Ej.3. La equivalencia lógica ($\Leftrightarrow$) es la igualdad Booleana, y se define como el operador `==` en Haskell.

(a) Definir `(==) :: Bool -> Bool -> Bool` usando case.
(b) Dar otra definición de esta función usando otras funciones ya definidas y sin usar `case` (va a ser necesario darle otro nombre a la nueva función, por ejmplo `===`).
(c) Definir la desigualdad booleana `(/=) :: Bool -> Bool -> Bool`.
Compararla con el conectivo `xor`, la disyunción excluyente. ¿Qué conclusión se puede sacar?.
(d) Defina el orden entre booleanos como una función $(\leq)$ `::Bool->Bool->Bool`, de modo tal que `False` sea menor que `True`.

Ej.4. Programar las siguientes funciones booleanas. En todos los casos dar dos definiciones, una usando `case` y otra usando los conectivos definidos en los ejercicios anteriores (será necesario ponerle nombres diferentes a las funciones cuando se definan por segunda vez):

(a) `unanimidad :: Bool -> Bool -> Bool -> Bool`, la cual recibe tres booleanos y devuelve verdadero cuando los tres booleanos recibidos sean iguales.

(b) `mayoria :: Bool -> Bool -> Bool -> Bool`, la cual recibe tres booleanos y devuelve el resultado de la mayoría de ellos.

(c) `impar :: Bool -> Bool -> Bool -> Bool` tal que `impar b1 b2 b3` es `True` cuando una cantidad impar de sus argumentos es `True`.

Ej.5.

(a) Redefinir los conectivos `(&&)` y `(>>)` sin usar case, y utilizando sólo `not` y `(||)`.
Será necesario ponerle un nombre nuevo a las funciones para que no se choquen con los anteriores.

(b) Redefinir `xor` utilizando `not`, `(||)` y `(&&)`.

Ej.6.

(a) Defina la función `(@@) :: Bool -> Bool -> Bool`, que es `True` cuando por lo menos uno de sus argumentos es `False`.

(b) Defina la función `(#) :: Bool -> Bool -> Bool`, que es `True` cuando exactamente uno de sus argumentos es `False`.

(c) Defina la función `tri :: Bool -> Bool -> Bool -> Bool`, tal que `tri b1 b2 b3` devuelve `True` si hay más argumentos `False` que `True`.