**FUNDAMENTOS DE LA COMPUTACIÓN**
**LABORATORIO 2**
**23 DE ABRIL DE 2014**

**0. Preludio.** Recordar comenzar el programa Haskell con la declaración del nombre del módulo, que debe ser el mismo que el del archivo (.hs). Conviene importar el laboratorio anterior, donde están definidas algunas funciones booleanas.

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

**1. Los naturales.** Declaramos los naturales. Usamos la letra `O` en lugar del "cero" como constructor inicial, para evitar conflicto con el 0 de Haskell:

```
data N = O | S N
  deriving (Show)
```

Nótese que no es necesario declarar el tipo completo de cada constructor, como hacemos en la teoría. La razón es que que Haskell deduce (porque no hay otra posibilidad) que el tipo final de cada constructor debe ser `N`. Específicamente entonces, en el caso del constructor inicial `O` no se declara ningún tipo, y en el caso del constructor funcional `S` sólo se declara el tipo de su argumento (que es `N`). El comando `:t` del intérprete Hugs puede usarse para verificar el tipo de cualquier expresión.

**?1.** Comprobar los tipos de los constructores de `N`

Programemos ahora la función `pred`:

```
pred :: N -> N
pred = \n-> case n of {
                O -> O ;
                S x -> x
                }
```

**?2.** Programar el predicado *par*:

```
par :: N -> Bool
par = \n-> case n of {
                O -> .... ;
                S x -> .....
                }
```

**?3.** Programar los operadores aritméticos `(+)` y `(*)`

**?4.** Programar la resta natural (que da `O` si el minuendo es menor que el sustraendo).

**?5.** Programar las funciones `fact` (factorial) y `pot` (potencia)

**?6.** Programar la función `sumi::N -> N`, tal que (sumi n) calcule recursivamente la suma de los primeros `n` naturales.

**2. Estructuras abstractas de datos.** A continuación se presenta la definición de la estructura abstracta `Eq` vista en clase, en el formato definido por Haskell:

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

Para dar la definir la instancia correspondiente de `Eq` para `N` se debe esribir:

```
instance Eq N where {
         (==) = ....
         }
```

**?7.** Completar la instanciación de Eq en `N`.[1]

La estructura abstracta `Ord` es una extensión de `Eq`. Esto se escribe en Haskell como sigue:

```
class  (Eq a) => Ord a  where {
                        ....
                        }
```

**?8.** Completar la declaración de Ord.

**?9.** Definir la instancia de `Ord` para `N`.

**?10.** Programar las funciones `max` (máximo de dos números) y `min` (mínimo de dos números)

**?11.** Programar las funciones `max3` y `min3` que calculan el máximo y mínimo de tres números respectivamente

**3. Más funciones sobre naturales.** Ahora nos dirigimos a definir una función general que nos permitirá definir algunas de las funciones ya definidas de manera más abstracta. La función `times` recibe como paparámetros un natural `n` y una función `f` y devuelve una función que aplica `n` veces `f` a su argumento. Es decir `times n f` $= \lambda x. \underbrace{f(f(f...(f}_{\text{n veces}} x)))$

**?12.** Programar la función `times`.

**?13.** Redefinir la función (+) usando `times`. Qué función se debe iterar para definir la suma?

**?14.** Redefinir las funciones (*) y `pot` usando `times`.

[1]Puede que la definición de `(==)` y `(/=)` de Bool que hicimos en el Lab1 de problemas. En cuyo caso, podemos cambiar la línea `import Lab1` por `import Lab1 hiding ((==),(/=))`.