FUNDAMENTOS DE LA COMPUTACIÓN
RECURSIÓN PRIMITIVA EN $\mathbb{N}$
SETIEMBRE DE 2017

0. Preludio.

Recuerden que para definir los tipos introducidos en el curso, la siguiente línea debe ser la primera del programa:

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

Luego se debe 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 fue definido el tipo `Bool` y las funciones booleanas.

```
module Lab3 where
import Prelude (Show)
import Lab1
import Lab2
```

1. Los números naturales. Para iniciar este laboratorio será necesario en primera instancia declarar nuestra representación de los números naturales. Debemos usar la letra `O` (letra "o" mayúscula) en lugar del número "cero" como constructor inicial, para evitar conflicto con el `0` (cero) de Haskell:

```
data N where {O::N ;  S::N -> N}
 deriving Show
```

Ahora que tenemos definido el tipo de datos `N`, podemos comenzar a definir funciones sobre él; por ejemplo la función `pos::N->Bool`, que recibe un natural y decide si éste es positivo:

```
pos::N -> Bool
pos = \n-> case n  of {O -> False;
                       S x -> True}
```

Conviene definir algunos naturales para poder probar las funciones:

```
uno::N
uno = S O

dos::N
dos = S uno

tres::N
tres = S dos
```

...

2. Programar en Haskell las siguientes funciones:

(1) `pred::N->N` que recibe un natural calcula su predecesor (sin funciones auxiliares).

(2) `par::N->Bool` que recibe un natural y decide si es par o no.

(3) `impar::N->Bool` que recibe un natural y decide si es impar o no. Dar dos definiciones de esta función, una usando `case` y otra usando alguna función definida anteriormente.

(4) `doble::N->N` que recibe un natural calcula su doble (sin funciones auxiliares).

(5) `triple::N->N` que recibe un natural calcula su triple (sin funciones auxiliares).

(6) `existe::N->(N->Bool)->Bool`, que recibe un predicado `p` y un natural `n`, y devuelve `True` si hay algún número menor o igual a `n` para el cual `p` es verdadero.

(7) `todos::N ->(N->Bool)->Bool`, que recibe un predicado `p` y un natural `n`, y devuelve `True` si `p` es verdadero para todos los números menores o iguales a `n`.

(8) `contar::N->(N->Bool)->N`, un predicado `p` y un natural `n`, y computa la cantidad de números menores o iguales que `n` para los cuales `p` es verdadero.
Probarla con algunos ejemplos: `contar cuatro par`; `contar cinco pos`.

(9) Operaciones básicas de naturales: suma, producto y potencia: `(+)::N->N->N`, `(*)::N->N->N` y `(^)::N->N->N`. Probarlas con algunos ejemplos.

(10) `fact::N->N`, que recibe un natural `n` y calcula su factorial.

(11) `sumi::N->N`, que recibe un natural `n` y calcula la sumatoria de todos los naturales menores o iguales que `n` ($\sum_{i=0}^{n} i$).

(12) `sumdobles::N->N`, que recibe un natural `n` y calcula la sumatoria de los dobles de todos los naturales menores o iguales que `n` ($\sum_{i=0}^{n} 2i$).

(13) `sumfi::(N->N)->N->N`, que recibe un natural `n` y una función `f` y computa la sumatoria de `(f i)` para `i = 0,...n` ($\sum_{i=0}^{n}(f\ i)$).

(14) `sumpares::N->N`, que recibe un natural `n` y computa la sumatoria de todos los naturales menores o iguales que `n` que son pares.

(15) `sumpi::(N->Bool)->N->N`, que recibe un natural `n` y un predicado `p` y computa la sumatoria de todos los naturales menores o iguales que `n` para los cuales `p` es verdadero.

3. Estructuras abstractas (Clases)

(16) Definir la instancia de `Eq` para `N`.

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

(17) Definir la instancia de `Ord` para `N`.

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