Examen de *Programación Declarativa*
del 22 de enero de 2010
Grupo 2

Andrés Cordón Franco

Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 4 de febrero de 2010

Ejercicio 1.– Usando listas por comprensión, definimos en Haskell la función:

```
diagonal xs = or [x == y | (x,y) <- xs]
```

1. Determina el tipo inferido por Haskell para la función `diagonal`.
2. Escribe una definición *recursiva* de la función `diagonal`.

Solución:

1. `diagonal :: Eq a => [(a,a)] -> Bool`
2. ```
   diagonal [] = False
   diagonal ((x,y):xs) | x == y    = True
                       | otherwise = diagonal xs
   ```

Ejercicio 2.– En el juego $J(a,b)$ se dispone de $a+b$ cartas numeradas del 0 al $a+b-1$ y se reparten a cartas al jugador 1 y b cartas al jugador 2. Por ejemplo:

```
data Mano = M [Int] [Int] deriving Show
m1 = M [0,2] [1,3,4]     -- es un reparto de J(2,3)
m2 = M [3,4,5,6] [0,1,2] -- es un reparto de J(4,3)
```

Define en Haskell la función `generaManos :: Int -> Int -> [Mano]` tal que `(generaManos a b)` devuelve todos los posibles repartos del juego $J(a,b)$. Por ejemplo:

```
Main> generaManos 2 3
  [M [0,1] [2,3,4], M [0,2] [1,3,4], M [0,3] [1,2,4],
   M [0,4] [1,2,3], M [1,2] [0,3,4], M [1,3] [0,2,4],
   M [1,4] [0,2,3], M [2,3] [0,1,4], M [2,4] [0,1,3],
   M [3,4] [0,1,2]]
```

Solución:

```
import Data.List

generaAux :: Int -> Int -> Int -> [[Int]]
generaAux 0 _ _ = [[]]
generaAux (n+1) min max = [x:xs | x <- [min..max],
                                  xs <- generaAux n (x+1) max]

generaMano :: Int -> Int -> [Mano]
generaMano a b = [M xs (f xs) | xs <- generaAux a 0 (a+b-1)]
             where f ys = [0..(a+b-1)] \\ ys
```

```

Ejercicio 3.– Escribe un programa Haskell interactivo

```
cuentaCifras :: IO ()
```

que tenga el siguiente comportamiento:

```
Main> cuentaCifras
Escribe un numero: 5467
El número 5467 tiene 4 cifras.
Main>
```

Solución:

```
cuentaCifras :: IO ()
cuentaCifras = do putStr "Escribe un numero: "
                  xs <- getLine
                  putStrLn ("El numero " ++ xs ++ " tiene "
                            ++show (length xs) ++ " cifras.")
```

Ejercicio 4.– Se considera el programa lógico:

```
a(X) :- b(X),no(c(X)).   %R1
b(0).                    %R2
b(1).                    %R3
b(2).                    %R4
c(1).                    %R5
no(P) :- P,!,fail.       %R6
no(P).                   %R7
```

Construye el árbol de resolución para el programa anterior y la pregunta:

```
?- a(X).
```

Solución:

Similar al árbol de deducción de `?- aprobado(X).` en la transparencia 35 del tema 15 de los apuntes de clase.
