Examen Programación Declarativa Febrero 2007

Resuelto por Zarovich

```
# let x f = f,f;;
val x : 'a -> 'a * 'a = <fun>

# let a::b = [x 1; x 2] in (a,b);;
-  : (int * int) * (int * int) list = ((1, 1), [(2, 2)])

# let doble x y = x (x y);;
val doble : ('a -> 'a) -> 'a -> 'a = <fun>

# let f = doble (function x -> x * x);;
val f : int -> int = <fun>

# let x = f 2 in x + 1;;
-  : int = 17

# let h f = function x -> let c::_ = f x in c;;
val h : ('a -> 'b list) -> 'a -> 'b = <fun>

# let s = h List.tl in s [1;2;3];;
-  : int = 2

# let s l = h List.tl l;;
val s : 'a list -> 'a = <fun>

# let rec num x = function [] -> 0
       |h::t -> (if x = h then 1 else 0) + num x t;;
val num : 'a -> 'a list -> int = <fun>

# num "hola";;
-  : string list -> int = <fun>

# let rec pre l s = match (l,s) with
       ([],_) -> false
      | (_,[]) -> true
      | (h1::t1, h2::t2) -> h1 = h2 && pre t1 t2;;
val pre : 'a list -> 'a list -> bool = <fun>

# let l = ['1';'2';'3'] in
       pre l ['1';'2'], pre l (List.tl l);;
-  : bool * bool = (true, false)
```

2. Definir una función suma que sume los elementos de dos listas respectivamente. Si una de las listas es mas corta que la otra, se rellenará con ceros. Es decir [2;5;4;9] y [2;4;3] debería de dar la lista... [4;9;7;9]. Si se define con recursividad terminal será un punto mas.

Sin recursividad terminal:

```
let rec suma l1 l2 = match (l1, l2) with
      ([], []) -> []
      | (h::t, []) -> h::(suma t [])
      | ([], h::t) -> h::(suma [] t)
      | (h1::t1, h2::t2) -> (h1 + h2)::(suma t1 t2);;
```

Con recursividad terminal

```
let suma l1 l2 = let rec aux l1 l2 res =
      match (l1, l2) with
            ([],[]) -> res
            | (h::t, []) -> aux t [] (h::res)
            | ([], h::t) -> aux [] t (h::res)
            | (h1::t1, h2::t2) -> aux t1 t2 ((h1+h2)::res)
      in aux (List.rev l1) (List.rev l2) [];;
```

3. Para los tipos de datos siguientes:

```
      type 'a a2 = A0 of 'a
            |AIz of 'a * 'a a2
            |ADc of 'a * 'a a2
            |A2 of 'a * 'a a2 * 'a a2;;


      type 'a abin = V | N of 'a * 'a abin * 'a abin;;
```

Definir la función a2_of_abin, que a partir de un abin generará un a2 y la función abin_of_a2 que hará lo contrario.

```
let rec abin_of_a2 = function
      A0 a -> N(a,V,V)
      | AIz (a,i) -> N (a,abin_of_a2 i, V)
      | ADc (a,d) -> N (a,V,abin_of_a2 d)
      | A2 (a,i,d) -> N(a,abin_of_a2 i,abin_of_a2 d);;

let rec a2_of_abin = function
      N(a,V,V) -> A0 a
      |N(a,V,d) -> ADc (a, a2_of_abin d)
      |N(a,i,V) -> AIz (a, a2_of_abin i)
      |N(a,i,d) -> A2 (a,a2_of_abin i, a2_of_abin d);;
```