Exámenes 2014, preguntas y respuestas

Paradigmas de Programación (Universidade da Coruña)

Examen Paradigmas de la Programación Junio 2014

```
# 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
   ([],[]) -> []
 | (l,[])  -> l
 | ([],l)  -> l
 | (h1::t1,h2::t2) -> (h1+h2):: suma t1 t2;;
```

Con recursividad terminal

```
let suma l1 l2 =
  let rec aux l1 l2 r = match (l1,l2) with
    ([],[]) -> r
  | (l,[])  -> r@l
  | ([],l)  -> r@l
  | (h1::t1,h2::t2) -> aux t1 t2 (r@[h1+h2])
  in aux l1 l2 [];;
```

3. Para los tipos de datos siguientes:

```
type 'a a2 = AO 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 =
   AO(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
   V -> raise (Failure "a2_of_abin")
 | N(a,V,V) -> AO(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);;
```

PARADIGMAS DE LA PROGRAMACIÓN

30 DE ENERO DE 2014

PROGRAMACIÓN FUNCIONAL

NOMBRE: _____

1. (2,5 puntos) Escriba el resultado de la compilación y ejecución de las siguientes frases, con tipos y valores, como lo indicaría el "toplevel" de ocaml:

```
let no f x = not (f x);;
```
```
val no: ('a → bool) → 'a → bool = <fun>
```

```
let par x = x mod 2 = 0 in no par;;
```
```
-: int → bool = <fun>
```

```
let rec rep n f x = if n > 0 then rep (n-1) f (f x) else x;;
```
```
val rep: (int → ('a → 'a) → 'a) → 'a = <fun>
```

```
rep 3 (function x -> x * x) 2, rep 4 (function x -> 2 * x) 1;;
```
```
-: (int * int) = (256,16)
```

```
(let par x y = function z -> x z, y z in par ((+) 2) ((/) 2)) 3;;
```
```
-: (int * int) = (5,0)
```

2. (2 puntos) Dada la siguiente definición del tipo de dato *'a arbol*, que sirve para representar cierto tipo de árboles binarios

```
type 'a arbol = Vacio | Nodo of ('a * 'a arbol * 'a arbol);;
```

a. **defina** una función ***cont:'a -> 'a arbol -> int***, que devuelva el número de nodos de un árbol que están etiquetados con un valor determinado.

```
let rec cont x = function
    Vacio      -> 0
 | Nodo(r,i,d) -> if r = x then 1 + (cont x i) + (cont x d)
                           else (cont x i) + (cont x d);;
```

b. defina una función ***subst: 'a -> 'a -> 'a arbol -> 'a arbol***, de forma que ***subst x y*** sea una función que, al aplicarla a un árbol, devuelve un arbol igual al original salvo los nodos que tuviesen valor ***x***, que tendrán valor ***y***.

```
let rec subst x y = function
    Vacio      -> Vacio
 | Nodo(r,i,d) -> if r = x then Nodo(y,subst x y i, subst x y d)
                           else Nodo(r,subst x y i, subst x y d);;
```

3. (1 punto) **Defina** una función ***l_ordenada: ('a -> 'a -> bool) -> 'a list -> bool***, de forma que si ***f*** es una relación de orden en el tipo ***'a*** (esto es, una función que dice si dos elementos de este tipo están ordenados), ***l_ordenada f*** sea la función que dice si una lista está ordenada según el orden ***f***.

```
let rec l_ordenada f = function
   []          -> true
 | h::[]       -> true
 | h1::h2::t   -> if f h1 h2 then l_ordenada f (h2::t)
                             else false;;
```

4. (1 punto) **Defina utilizando exclusivamente recursividad terminal** una función ***l_max: 'a list -> 'a*** que devuelva de cada lista el mayor de sus elementos.

```
let l_max = function
   []  -> raise (Failure  "max"
)
| h::t ->  let rec aux m = function
               []  -> m
            | h::t -> if h>m then aux h t
                            else aux m t
           in aux h t;;
```