Exámenes 2012, preguntas y respuestas

Paradigmas de Programación (Universidade da Coruña)

PARADIGMAS DE LA PROGRAMACIÓN
11 de diciembre de 2012

NOMBRE:_____DNI:_____

Programación funcional

1. (2.5 ptos.) Las siguientes frases corresponden al código introducido en una sesión de trabajo con el toplevel de ocaml. Indique, después de cada una de ellas, cuál sería la respuesta que daría el compilador.

```
# let x y = y + 1,y - 1;;
```

```
val x: int → (int * int) = <fun>
```

```
# let y = x 0;;
```

```
val y: (int * int) = (1,-1)
```

```
# let x,y = y and y = 0;;
```

```
ERROR
```

```
# let rec fold2 op1 op2 e = function
      h::t -> op1 h (fold2 op2 op1 e t)
    | []   -> e;;
```

```
val fold2: ('a → 'b → 'b) → ('a → 'b → 'b) → 'b → 'a list → 'b = <fun>
```

```
# fold2 (+) ( * ) 0 [1;2;3;4], fold2 (+) (-) 1 [1;2;3;4];;
```

```
-: (int * int) = (7,-3)
```

2. (1.5 ptos.) Indique el tipo de las funciones definidas e continuación y redefínalas en modo "curry" de la forma más breve que pueda.

```
# let op = function p -> fst p (snd p);;
```

```
val op: (('a → 'b) * 'a) → 'b = <fun>
```

```
let op f x = f x;;
```

```
val op : ('a -> 'b) -> 'a -> 'b = <fun>
```

```
# let div (x,y) = (/) x y;;
```

```
val div: (int * int) → int = <fun>
```

```
let div x y = x / y;;
```

```
val div: int → int → int = <fun>
```

```
# let max (p,ord) =  if ord p then snd p else fst p;;
```

```
val max: (('a * 'a) * (('a * 'a) → bool)) →'a = <fun>
```

```
let max p ord = if ord p then snd p else fst p;;
```

```
val max : 'a * 'a -> ('a * 'a -> bool) -> 'a = <fun>
```

3. (1.5 ptos.) Un valor de tipo (float * float) list puede representar un camino poligonal en el plano (sería la lista ordenada de las coordenadas de los vértices del camino). Defina una función **distancia**: (float * float) list -> float que calcule la longitud de cada uno de estos caminos. Recuerde que la distancia entre dos puntos, ($x1$, $y1$) y ($x2$, $y2$), viene dada por la fórmula $\sqrt{(x2-x1)^2+(y2-y1)^2}$

```
let longitud (x1,y1) (x2,y2) =
  sqrt( ((x2 -. x1) ** 2.0) +. ((y2 -. y1) ** 2.0) );;

let distancia l =
  let rec aux r = function
    p1::p2::t -> aux (r +. longitud p1 p2) (p2::t)
   | _ -> r
 in aux 0. (l @ [List.hd l]);;
```

PARADIGMAS DE LA PROGRAMACIÓN

13 DE SEPTIEMBRE DE 2012

NOMBRE: _____

1. (6 puntos) Escriba el resultado de la compilacion y ejecucion de las siguientes frases, con tipos y valores, como lo indicaría el toplevel de ocaml:

```
let x, y = 2, 5;;
```
```
val x: int 2
val y: int 5
```

```
let f y = x + y;;
```
```
val f: int → int = <fun>
```

```
f (let f = function x -> x * x in f x);;
```
```
-: int = 6
```

```
let v x f = f x;;
```
```
val v: 'a → ('a → 'b) → 'b = <fun>
```

```
let cero x = v 0 x and dos x = v 2 x;;
```
```
val cero: (int → 'a) → 'a = <fun>
val dos: (int → 'a) → 'a = <fun>
```

```
let g = cero (-) in g 1;;
```
```
-: int = -1
```

```
let h = dos (+);;
```
```
val h: int → int = <fun>
```

```
dos h;;
```
```
-: int = 4
```

```
g 0, h 0;;
```
```
ERROR
```

```
let doble = let vacia = "" in function
        vacia -> vacia
      | s -> s ^ s;;
```
```
val doble: string → string = <fun>
```

```
doble "hola";;
```
```
-: string = “hola”
```

```
let rec ap = function
     [] -> 0 | h::t -> h (ap t);;
```
```
val ap: (int → int) list → int = <fun>
```

2. (2 puntos) Realice una nueva definición para la función f , de forma que sólo se utilice recursividad terminal.

```
let rec f = function
        [] -> 0
    | h::t -> h + 2 * f t;;
```

```
let f l =
  let rec aux r = function
     []   -> r
  | h::t -> aux (h + 2*r) t
 in aux 0 (List.rev l);;
```

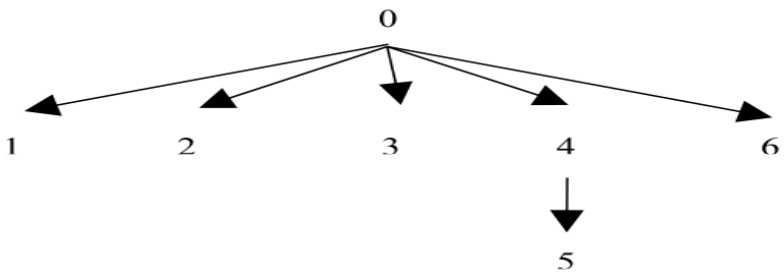
3. (2 puntos) Considere la siguiente definición en ocaml para el tipo de dato 'a arbolgen (que sirve para representar árboles con nodos etiquetados con valores de tipo 'a, en los que de cada nodo puede colgar cualquier número de ramas)

```
type 'a arbolgen = Nodo of 'a * 'a arbolgen list;;
```

de modo que, por ejemplo, el valor ag1 definido por

```
let ag1 = Nodo (0, [Nodo (1, []); Nodo (2, []);
                    Nodo (3, []); Nodo (4, [Nodo (5, [])]);
                    Nodo (6, [])]);;
```

corresponderia arbol



Defina una funcion nnodos : 'a arbolgen -> int que devuelva el número de nodos de un árbol, de forma que, por ejemplo, nnodos ag1 = 7

```
let rec nnodos = function
  Nodo(r,[])   -> 1
| Nodo(r,h::t) -> nnodos h + nnodos (Nodo(r,t));;
```

PARADIGMAS DE LA PROGRAMACIÓN

14 DE FEBRERO DE 2012

NOMBRE:_____

1. (5.5 puntos) Escriba el resultado de la compilacion y ejecucion de las siguientes frases, con tipos
y valores, como lo indicaría el compilador de ocaml:

```
let f, x = (+), 0;;
```
```
val f: int → int → int = <fun>
val x: int 0
```

```
f x;;
```
```
-: int → int = <fun>
```

```
let y = x + 1, x - 1;;
```
```
val y: (int * int) = (1,-1)
```

```
let a, b = y in if a > b then b else a;;
```
```
-: int = -1
```

```
let z = let a, b = y in let z = a - b in z * z;;
```
```
val z: int = 4
```

```
(function x -> x) (function s -> s ^ s);;
```
```
-: string → string = <fun>
```

```
let rec itera op = function [] -> () | h::t -> op h; itera op t;;
```
```
val itera: ('a → 'b) → 'a list → unit = <fun>
```

```
let rec clist n x = if n < 1 then [] else x :: clist (n-1) x;;
```
```
val clist: int → 'a → 'a list = <fun>
```

```
clist 3 true;;
```
```
-: bool list = [true;true;true]
```

```
let rec powerr n op x = if n <= 1 then x else powerr (n / 2) op (op x
x);;
```
```
val powerr: int → ('a → 'a → 'a) → 'a → 'a = <fun>
```

```
let g = powerr 5 (+) in g 3;;
```
```
-: int = 12
```

2.(1.5 puntos) Considere la siguiente definicion en ocaml:

```
let rec comp l x = match l with
                     [] -> x
                   | h::t -> h (comp t x);;
```

¿Cuál es el tipo de la funcion comp?

```
val comp: ('a → 'a) list → 'a → 'a = <fun>
```

Realice una nueva definición para comp de forma que sólo se utilice recursividad terminal.

```
let comp l x =
     let rec aux r = function
         []   -> r
       | h::t -> aux (h r) t
    in aux x (List.rev l);;
```

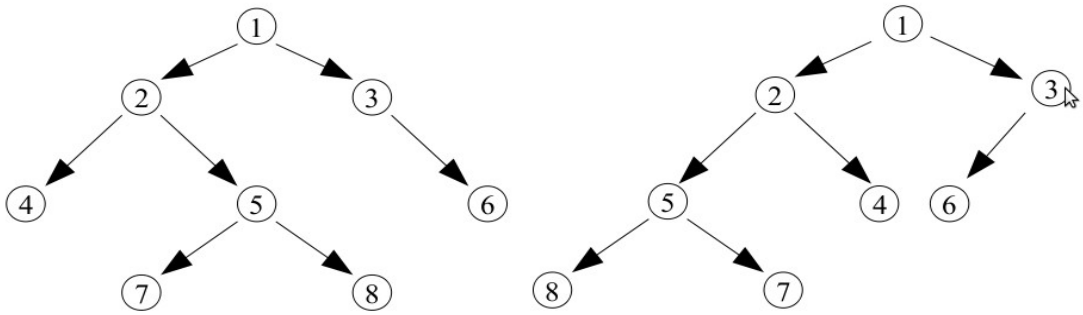
3.(1.5 puntos) La función max_en está definida de forma que el valor de max_en x l corresponde a la función de la lista l que alcanza el mayor valor en el punto x. Redefínala para optimizar su eficiencia.

```
let rec max_en x = function
      [] -> raise (Failure "max_en")
    | [f] -> f
    | f::l -> if f x > (max_en x l) x then f
                  else max_en x l;;
```

```
let max_en x = function
    []  -> raise (Failure "max_en")
 | h::t ->  let rec aux fmax vmax = function
               []  -> fmax
            | h::t -> let hx = h x in
                              if hx > vmax then aux h hx t
                                           else aux fmax vmax t
           in aux h (h x) t;;
```

4. (1.5 puntos) Diremos que un árbol binario es un “giro” de otro árbol binario si el primero puede obtenerse del segundo intercambiando las ramas de cualesquiera de sus nodos.

Así, por ejemplo, los dos árboles siguientes son el uno “giro” del otro, pues el segundo se puede obtener a partir del primero intercambiando las ramas de los nodos “2”, “3”, y “5”.



Utilizando, para representar los árboles binarios, el tipo de dato 'a bintree definido a continuación, implemente en ocaml una función giro: 'a bintree -> 'a bintree -> bool que indique si dos árboles son el uno giro del otro.

```
type 'a bintree = Empty | Node of 'a bintree * 'a * 'a bintree;;
```

```
let rec giro a1 a2 = match (a1,a2) with
    (Empty,Empty) -> true
  | (Node(i1,r1,d1),Node(i2,r2,d2)) -> (r1 = r2) &&
                        ((giro i1 i2 && giro d1 d2) ||
                         (giro i1 d2 && giro d1 i2))
  | _ -> false;;
```