Programación Declarativa
2008-2009
Práctica 8

1. Escriba el resultado de la compilación y ejecución de las siguientes frases, con tipos y valores, como lo indicaría el compilador interactivo de *ocaml*:

```
let f = (function p -> fst p :: snd p);;
```

```
f ('a',['e';'i']);;
```

```
let rec zip f = function ([], l) -> l | (l, []) -> l |
(h1::t1, h2::t2) -> f h1 h2::zip f (t1, t2);;
```

```
zip (+), zip (@);;
```

```
(function x -> function y -> x @ y ) [false] ;;
```

```
let rec fold f e = function
[] -> e
| h::t -> fold f (f e h) t;;
```

```
let rec cut l n = match (n,l) with
(0,_) -> l
|(n, h::t) -> cut t (n-1);;
```

```
cut [2;3;4;5] 2;;
```

```
fold cut ['a'; 'b'; 'c'; 'd'; 'e'; 'f'; 'g'; 'h'; 'i'; 'j']
[2;1;3];;
```

```
let x = 1 + let x = 2 in x + 1;;
```

```
(let x = x + 1 in x + 1) + x , x;;
```

```
let x, y = let x, y= x + x, x * x in y, x;;
```

2. Realice una nueva definición para la función ***comp***, de forma que sólo se utilice recursividad terminal.

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

3. Considere la siguiente definición en *ocaml* para el tipo de dato ‘*a tree* (que sirve para representar cierto tipo de árboles binarios con nodos etiquetados con valores de tipo ‘*a*):

```
type 'a tree = Leaf of 'a | Node of 'a tree * 'a * 'a tree;;
```

Leaf x represetaría un árbol con un único nodo (etiquetado con el valor *x*) y *Node (ri, r, rd)* representaría un árbol con rama izquierda *ri*, rama derecha *rd* y raíz etiquetada con el valor *r*.

Diremos que un árbol es *montículo* si el valor de cada nodo es mayor o igual que el de todos los nodos que “cuelgan” de él. Defina una función ***montículo****: 'a tree -> bool*, que indique si un árbol es montículo o no.