PARADIGMAS DE PROGRAMACIÓN

APELLIDOS:_____NOMBRE: _____

1. *(4 puntos)* Escriba el resultado de la compilación y ejecución de las siguientes frases, con tipos y valores y distinguiendo claramente expresiones de definiciones, como lo haría el compilador interactivo de *ocaml*:

```
let lista = [(0,1); (2,3)];;
```

*val lista : (int * int) list = [(0, 1); (2, 3)]*

```
let delist = (function x -> List.hd x, List.tl x);;
```

*val delist: 'a list -> 'a * 'a list = <fun>*

```
let a, b = delist lista;;
```

*val a: int * int = (0, 1)*
*val b: (int * int) list = [(2, 3)]*

```
let x, y = List.hd b;;
```

val x: int = 2
val y: int = 3

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

*- : int * int = (6, -4)*

```
(let x = x * x in 2 * x) + x + y;;
```

- : int = 13

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

val iter : ('a -> 'b) -> 'a list -> unit = <fun>

```
let pr_names l = iter (fun o -> print_endline o#name) l;;
```

val pr_names : < name : string; .. > list -> unit = <fun>

```
let l = let o1 = object method name = "O1"
                     end
        in o1::[];;
```

val l : < name : string > list = [<obj>]

```
let o2 = object method name = "O2"
                method number = 2
         end
in o2::l;;
```

No compila por error de tipo: l es de tipo < name: string > list, pero ahí se necesita una expresión de tipo < name: string; number: int > list

2. *(2 puntos)* Defina una función ***minl: 'a list -> 'a*** de modo que ***minl l*** sea el mínimo de la lista ***l*** segun la relación de orden ***(<=)***. **La definición debe ser recursiva terminal.**

```
let minl = function
    [] -> raise (Invalid_argument "minl")
  | h::t -> List.fold_left min h t;;
```

3. *(2 puntos)* Considere la siguiente definición en ocaml para el tipo de dato ***treeSk,*** que sirve para representar estructuras con formas de árbol ("esqueletos" de árbol, sin valores asociados a los nodos):

```
type treeSk = Tree of treeSk list
```

De este modo, *t1* = *Tree [ ]* representaría el árbol con un sólo nodo, *t2* = *Tree [t1; t1]* representaría un árbol con *2* ramas y *3* nodos, y *t3* = *Tree [t1; t1; t2]* representaría un árbol con *3* ramas y *6* nodos.

Defina una función ***nnodos : treeSk -> int*** que devuelva, para cada uno de estos árboles, su número de nodos.

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

4. *(2 puntos)* Considere la función ord definida a continuación de modo imperativo:

```
let ord l =
    if l = [] then true else
    let l = ref l in
        while List.tl !l <> [] &&
              (List.hd !l) <= (List.hd (List.tl !l))
        do l := List.tl !l done;
    List.length !l = 1;;
```

¿Cuál es el tipo de ord?

'a list -> bool

Redefina ***ord*** de modo funcional (sin usar bucles ni variables). En esta definición dé preferencia al pattern-matching sobre el if_then_else.

```
let rec ord = function
   [] | _::[] -> true
 | h1::h2::t -> h1 <= h2 && ord (h2::t);;
```

¿Es esta nueva definición recursiva terminal? (justifique muy brevemente su respuesta).

Sí es recursiva terminal, ya que h1 <= h2 && ord (h2::t) se evalúa igual que if h1 <= h2 then ord (h2::t) else false
y ahí se ve claramente que, en el caso recursivo, la aplicación recursiva de la función ord es la última operación que se realiza.