Programación Declarativa
2008-2009
Práctica 3

1. Intente deducir "a mano" cuál será el resultado de la compilación y ejecución de las siguientes expresiones. Escriba el resultado que espera y compruebe luego la respuesta real del compilador al introducir esas frases. Tome buena nota de los tipos de dato que vayan apareciendo, así como de todas las respuestas inesperadas y trate de explicar su razón. Si se produjera algún error, lea atentamente el mensaje del compilador y tome nota de la causa.

```
[1];;

[3;2;1];;

[1;"dos"];;

[1;2.0];;

2 :: [3;4];;

List.length [3;3;3;3];;

List.nth [5;4;3;2;1] 2;;

List.nth [5;4;3;2;1] 1 + List.nth [5;4] 0;;

[1;1+1;1+1+1];;

[1; 2] @ [3; 4];;

List.hd [1; 2; 3];;

List.tl [1; 2; 3];;

List.hd [0];;

List.tl [0];;

['a'; 'e'; 'i'; 'o'; 'u'];;

List.rev ['a'; 'b'; 'c'];;

[];;

"hola" :: [];;

[1,2,3];;

[1; "dos"; 3];;

0 :: [1; 2; 3];;
```

```
[];;

3 :: [];;

2 :: (3 :: []);;

1 :: (2 :: (3 :: []));;

(1 :: 2 :: 3 ::[]) ;;

List.length ([1; 2] @ [3; 4]);;

List.append ['a'; 'e'; 'i'] (List.tl ['o'; 'u']);;

List.hd [];;

List.tl [];;

function x -> [x];;

(function x -> [x]) 0;;

function x -> List.hd x, List.tl x;;

(function x -> List.hd x, List.tl x) [3;2;1];;

(function x -> List.hd x, List.tl x) [[]];;

function x -> [x; [x]];;
```

2. Las definiciones de funciones de la forma

```
let <name> = function <x> -> <exp>
```

pueden abreviarse de la forma

```
let <name> <x> = <exp>
```

Así, por ejemplo, en vez de

```
let doble = function x -> 2 * x
```

podemos escriblir

```
let doble x = 2 * x
```

Utilice esta forma abreviada para reescribir las definiciones de las funciones **f**, **dup** y

absolut que aparecían en el ejercicio 3 de la 2ª práctica.

3. En ocaml pueden hacerse definiciones recursivas si añadimos la palabra reservada "rec" justo después de "let":

```
let rec fact n =
    if n <= 1 then 1 else n * fact (n-1);;

let rec fib n =
    if n <= 1 then n
    else fib (n-1) + fib (n-2);;

let rec sumalista l =
    if l = [] then 0
    else List.hd l + sumalista (List.tl l);;
```

Defina una función **`nprimeros: int -> int list`** que devuelva, para cada número *n*, la lista de los primeros *n* números naturales. Así, por ejemplo, *nprimeros 4* deberá ser la lista *[1; 2; 3; 4]*.

Defina una función **`suma_hasta: int -> int`** tal que *suma_hasta n* sea la suma de los *n* primeros enteros positivos.

Defina una función **`from_to: int -> int -> int list`,** de forma que *from_to m n* sea la lista ascendente de los enteros mayores o iguales que *m* y menores o iguales que *n.* Así, por ejemplo, *from_to 3 7* sería la lista *[3; 4; 5; 6; 7].*

Defina las funciónes **`lmax:'a list -> 'a`** y **`lmin:'a list ->'a`** que devuelvan respectivamente el mayor y el menor valor de los que aparecen en una lista.

Defina una función **`rango:'a list ->'a *'a`** tal que si **`rango l = (a,b)`** entonces **`lmin l = a`** y **`lmax l = b`**.

Defina una función **`ordenada: 'a list -> bool`** que indique si una lista tiene sus elementos ordenados según el orden *(<=)*. Así, por ejemplo, *ordenada [2; 5; 5; 7]* valdría *true*, pero *ordenada ['a'; 'u'; 'i']* valdría *false.*