

# Paradigmas de Programación

## Práctica 7

### Nota Importante:

Realice las implementaciones de esta práctica en los ficheros `mylist2.ml`, y opcionalmente `mylist3.ml`.

Cuando se solicite la entrega de esta práctica, cada alumno deberá enviar únicamente estos ficheros.

Sea muy cuidadoso a la hora de crear los ficheros, y **respete los nombres indicados**.

Además, **estos ficheros deben compilar sin errores** con las siguientes órdenes:

```
ocamlc -c mylist2.ml
ocamlc -c mylist3.ml
```

### Ejercicios:

1. Algunas de las definiciones originales del módulo `List` de la librería estándar de OCaml (`hd`, `tl`, `nth`, `init`, `map2`, `find`, `combine`) consideran casos de error de ejecución que llevan asociadas unas determinadas excepciones.

Y por otra parte, algunas de las definiciones originales del módulo `List` (`length`, `compare_lengths`, `nth`, `rev`, `init`, `rev_append`, `rev_map`, `fold_left`, `find`, `for_all`, `exists`, `mem`, `filter`, `find_all`, `partition`) son recursivas terminales.

Copie el contenido del fichero `mylist.ml` elaborado durante la realización de la práctica 6 a otro fichero con nombre `mylist2.ml`.

Perfeccione, cuando sea preciso, las definiciones contenidas en el fichero `mylist2.ml`, de forma que el comportamiento de cada función sea exactamente el mismo que el de la correspondiente función en el módulo `List`. Es decir:

- En aquellas definiciones en las que hay casos de error de ejecución, las excepciones asociadas deben ser exactamente las mismas que las correspondientes en el módulo `List`.
- Y deben ser recursivas terminales todas aquellas funciones que lo son en el módulo `List`.

Además, en el caso de la función `compare_lengths`, debe cumplirse lo que dice el manual: el cálculo debe terminar después de recorrer la lista más corta.

2. (Ejercicio opcional) Implemente en otro fichero `mylist3.ml` lo siguiente:

- Una función `remove`: `'a -> 'a list -> 'a list`, que “elimine la primera aparición, si la hay, de un valor en una lista”. Así, por ejemplo, `remove 3 [2; 6; 3; 4; 3]` debería ser la lista `[2; 6; 4; 3]` y `remove 3 [1; 2; 4]` debería ser `[1; 2; 4]`.
- Una función `remove_all`: `'a -> 'a list -> 'a list`, que “elimine todas las apariciones de un valor en una lista”. Así, por ejemplo, `remove_all 3 [2; 6; 3; 4; 3]` debería ser la lista `[2; 6; 4]`.
- Una función `ldif`: `'a list -> 'a list -> 'a list`, de forma que `ldif l1 l2` elimine de `l1` todas las apariciones de todos aquellos valores que aparezcan en `l2`. Así, por ejemplo, `ldif [1;2;3;2;4] [2;3;3;5]` debería ser la lista `[1;4]`.

- Una función `lprod`: `'a list -> 'b list -> ('a * 'b) list`, de forma que `lprod l1 l2` calcule el “producto cartesiano” de `l1` y `l2`. Así, por ejemplo, `lprod [1;3;1;2] ['a';'b']` debería ser la lista `[(1,'a'); (1,'b'); (3,'a'); (3,'b'); (1,'a'); (1,'b'); (2,'a'); (2,'b')]`.
- Una función `divide`: `'a list -> 'a list * 'a list`, de forma que `divide l` devuelva un par de listas `(l1,l2)`, donde `l1` contiene los elementos de `l` que ocupan posición impar y `l2` los que ocupan posición par. En ambos casos, los elementos de `l1` y `l2` deben mantener el mismo orden relativo que tienen en la lista original `l`. Así, por ejemplo, `divide ['a';'e';'i';'o';'u']` debería ser el par `(['a';'i';'u'], ['e';'o'])`.