# Récursion, évaluation, filtrage, listes

## Récursion

**Exercice 1.** Traduire la fonction C suivante en CAML (fonctionnel) :

```
int sommeborne(int n, int f(int))
{
  int x = 0, i=0;

  while ( i <= n)
    x += f(i++);

  return x;
}
```

**Exercice 2.** Définir une fonction CAML `iter` dont le type est

```
iter : int -> ('a -> 'a) -> 'a -> 'a
```

et dont la sémantique est $iter\ n\ f\ x = f^n(x)$. Généraliser cette fonction à une fonction `fold` dont le type est

```
fold : int -> (int -> 'a -> 'a) -> 'a -> 'a
```

et dont la sémantique est $fold\ n\ f\ x = f(n, \ldots f(1, f(0, x))$. Utiliser la fonction `fold` pour résoudre l'exercice précédent.

**Exercice 3.** Le schéma de récursion primitive explique qu'on peux définir une fonction $f$ à partir des fonctions $g$ et $h$ de la façon suivante :

$$f(0, x) = g(x)$$
$$f(n+1, x) = h(n, x, f(n, x))\,.$$

A quel ensemble appartient ici $x$ ? Donner les "types" de $g$ et $h$.
Écrire une fonction CAML `prim_rec` qui prend $g$ et $h$ en paramètre, et retourne $f$ définie comme ci-dessus.

**Exercice 4.** La suite de Fibonacci peut âtre définie par une fonction CAML comme il suit :

```
let rec fibo n =
  if n = 0 then 0
  else
    if n = 1 then 1
    else fibo (n-2) + fibo ( n-1);;
```

- Estimer la complexité de cette fonction.
- Proposer une autre définition de la même fonction, qui est bien plus efficace. Suggestion : utiliser les produits Cartésiens.

## Stratégies d'évaluation

**Exercice 5.** Que se passe-t'il si dans un langage fonctionnel dont la stratégie d'évaluation est par valeur, on ne possède ni des constructeurs comme `if ... then ... else ...` ni des constructeurs de filtrage comme `function` et `match ... with` ?

**Exercice 6.** Considérer le code suivant :

```
(fun x -> fun y -> (x + y))(1 + 2)(fibo 0);;
(fun x -> x + x) 1 + 2;;
let f1 = function f2 -> (function x -> f2 x);;
let g = function x -> x + 1;;
f1 g 2;;
```

Donner les étapes de la réduction par valeur de ces expressions. Réduire les mêmes expressions en utilisant la stratégie par nom.

## Filtrage

**Exercice 7.** Modifier les deux définitions de la suite de Fibonacci en utilisant le filtrage par `function` et `match ... with`.

## Listes

**Exercice 8.**

**algox.ml**

```
 1 : let rec length = function
 2 :     [] -> 0
 3 :   | x::xs -> 1 + length(xs);;
 4 :
 5 : let rec split liste n =
 6 :   if n <= 0 then ([],liste )
 7 :   else
 8 :     match liste with
 9 :         [] -> ([],[])
10 :       | x::xs ->
11 :           let (l1,l2) = split xs (n-1) in
12 :             (x::l1,l2) ;;
13 :
14 : let rec merge liste1 liste2 lesseq =
15 :   match  (liste1,liste2) with
16 :       ([],_) -> liste2
17 :     | (_,[]) -> liste1
18 :     | (x::xs,y::ys) ->
19 :         if lesseq x y then
20 :           x::(merge xs liste2 lesseq)
21 :         else
22 :           y::(merge liste1 ys lesseq) ;;
23 :
24 : let rec algox lesseq = function
25 :     [] -> []
26 :   | [x] -> [x]
27 :   | liste  ->
28 :       let
29 :         moitie = length liste / 2
30 :       in
31 :       let
32 :         (l1,l2) = split liste moitie
33 :       in
34 :         merge ( algox lesseq l1) ( algox lesseq l2) lesseq ;;
```

Expliquer ce que fait le programme `algox.ml`

**Exercice 9.** Proposer un fonction CAML de type `int -> int`, qui étant donné un entier $n \geq 0$, calcule le $n$-nième nombre premier. Suggestion : on pourra construire d'abord une fonction `int -> int list` qui construit la liste des premiers $n$-nombres premiers.