

# Corrigé du TD1

Ci-dessous un corrigé Cam1 des premiers exercices du TD1. Le but est de vous montrer que tous les programmes étaient réalisables de manière claire et concise (regarder en particulier le programme de multiplication de deux polynômes) sans utiliser aucune référence.

## Exercice 0 : Quelques procédures récursives

- *Suite de Fibonacci*. Pas de difficulté.

```
let rec fibo n = match n with
| 0 -> 1
| 1 -> 1
| n -> fibo (n-1) + fibo (n-2);;
```

- *Décomposition en facteurs premiers*. L'algorithme consiste à tester la divisibilité de  $n$  par tous les entiers supérieurs ou égaux à 2, dans l'ordre croissant. Dès que l'on trouve un entier  $p$  tel que  $p|n$ , on rajoute  $p$  dans la liste des diviseurs de  $n$ , et on poursuit avec  $n/p$ . On est certain que chacun des entiers ainsi obtenus est premier, car si  $p|n$  avec  $p = a \cdot b$ ,  $a < p$  alors  $a|n$ , ce qui est absurde

```
let dec_fac_prem n =
  let aux n p l = match n with
  | 1 -> l
  | n when n mod p = 0 -> aux (n/p) p (p::l)
  | _ -> aux n (p+1) l
  in aux n 2 [];
```

- *Longueur d'une suite*.

```
let rec longueur l = match l with
| [] -> 0
| x::reste -> 1+ longueur reste;;
```

- *Fonction miroir*. On introduit une fonction auxiliaire nous permettant de faire apparaître une nouvelle liste en argument, et on fait successivement passer tous les éléments de la première liste à la deuxième. Lorsque la première liste est vide, on renvoie la deuxième.

```
let miroir l =
  let rec aux l1 l2 = match l1 with
  | [] -> l2
  | x::r -> aux r (x::l2)
  in aux l1 [];
```

## Exercice 1 : Exponentiation rapide

```
let rec expo_rap x e = match e with
| 0 -> 1
| e when e mod 2 = 0 -> let y = expo_rap x (e/2) in y*y
| _ -> let y = expo_rap x (e/2) in x*y*y;;
```

### Exercice 2 : Polynômes

```
let rec degre p = match p with
| [] -> -1
|x :: reste -> 1+(degre reste) ;;

let rec horner p x = match p with
| [] -> 0
| y :: reste -> y+x*(horner reste x) ;;

let rec addition p1 p2 = match p1, p2 with
| [], p2 -> p2
| p1, [] -> p1
|x :: p, y :: q -> (x+y) :: (addition p q) ;;
```

Pour la multiplication, on commence par définir un programme qui effectue la multiplication d'un polynôme  $p$  par une constante  $c$ , puis on définit la multiplication de deux polynômes comme la somme d'une suite de multiplications d'un polynôme par un monôme.

```
let rec mul_const p c = match p with
| [] -> []
|x :: reste -> (c*x) :: (mul_const reste c) ;;

let rec multiplication p1 p2 = match p1 with
| [] -> []
|x :: reste -> addition (mul_const p2 x) (0 :: (multiplication reste p2)) ;;
```

### Exercice 3 : Tri fusion

Le premier programme coupe la liste en deux en mettant ses éléments successivement dans la première puis dans la deuxième liste; le deuxième fusionne deux listes et le dernier réalise le tri fusion.

```
let rec coupe l = match l with
| [] -> [], []
|[x] -> [x], []
|x :: y :: reste -> let (l1, l2) = coupe reste in x :: l1, y :: l2 ;;

let rec merge l1 l2 = match l1, l2 with
| [], l2 -> l2
| l1, [] -> l1
|x :: r1, y :: r2 when x < y -> x :: (merge r1 l2)
|x :: r1, y :: r2 -> y :: (merge l1 r2) ;;

let rec tri_fusion l = match l with
| [] -> []
|[x] -> [x]
| _ -> let (l1, l2) = coupe l in merge (tri_fusion l1) (tri_fusion l2) ;;
```

### Exercice 4 : Algorithme d'Euclide

On montre plutôt le lemme suivant, dont se déduit directement le résultat demandé dans l'énoncé :

**Lemme 0.1.** *Si  $a > b \geq 0$  et l'algorithme d'Euclide appelé sur  $a$  et  $b$  fait  $k \geq 1$  appels récursifs, alors  $a \geq F_{k+2}$  et  $b \geq F_{k+1}$ .*

*Démonstration.* Par récurrence sur  $k$ .

-  $k = 1$  Alors  $b \geq 1 = F_2$ , et comme  $a > b$ ,  $a \geq 2 = F_3$ .

- Supposons le lemme vrai pour  $k-1$ . Comme  $k > 0$ ,  $b > 0$ , et le programme effectue un appel récursif avec les arguments  $b$  et  $a \bmod b$  (c'est le principe de l'algorithme d'Euclide...); qui fera  $k-1$  itérations. Par hypothèse de récurrence (on peut l'appliquer car on a bien  $b > (a \bmod b)$ ),  $b \geq F_{k+1}$  et  $(a \bmod b) \geq F_k$ . D'où :

$$b + (a \bmod b) = b + (a - \lfloor a/b \rfloor b) \leq a$$

car  $a > b > 0 \implies \lfloor a/b \rfloor \geq 1$ . Donc,

$$a \geq b + (a \bmod b) \geq F_{k+1} + F_k = F_{k+2}.$$

□

L'algorithme permettant de calculer les coefficients de Bézout est écrit ci-dessous. Il peut paraître un peu obtus mais est très élégant, et je vous laisse prouver sa correction...

`bezout a b` renvoie le triplet  $d, x, y$  tel que  $d = a*x + b*y$ .

```
let rec bezout a b =
  if b=0 then (a,1,0) else
  let (d,x,y)=(bezout b (a mod b)) in (d,y,x-(a/b)*y);;
```