Algorithmique - Programmation 1

Cours 6

Université Henri Poincaré

CESS Epinal

Automne 2008

Plan

Rappels : les listes

- Suite d'**éléments de même type**
- Construction de listes :
  - opérateur `::`
    `'a -> 'a list -> 'a list = <fun>`
  - opérateur `@` (concaténation)
    `'a list -> 'a list -> 'a list = <fun>`
- Exemples :

```
# 1.2 ::  1.5 ::  1.8 ::  [];;
```

Rappels : les listes

- Suite d'**éléments de même type**
- Construction de listes :
  - opérateur `::`
    `'a -> 'a list -> 'a list = <fun>`
  - opérateur `@` (concaténation)
    `'a list -> 'a list -> 'a list = <fun>`
- Exemples :

```
# 1.2 :: 1.5 :: 1.8 :: [];;
- : float list = [1.2; 1.5; 1.8]
```

Rappels : les listes

- Suite d'**éléments de même type**
- Construction de listes :
  - opérateur `::`
    `'a -> 'a list -> 'a list = <fun>`
  - opérateur `@` (concaténation)
    `'a list -> 'a list -> 'a list = <fun>`
- Exemples :

```
# 1.2 :: 1.5 :: 1.8 :: [];;
- : float list = [1.2; 1.5; 1.8]

# [1.2] @ [1.5; 1.8];;
```

Rappels : les listes

- Suite d'**éléments de même type**
- Construction de listes :
  - opérateur `::`
    `'a -> 'a list -> 'a list = <fun>`
  - opérateur `@` (concaténation)
    `'a list -> 'a list -> 'a list = <fun>`
- Exemples :

```
# 1.2 ::  1.5 ::  1.8 ::  [];;
- :  float list = [1.2; 1.5; 1.8]

# [1.2] @ [1.5; 1.8];;
- :  float list = [1.2; 1.5; 1.8]
```

Rappels : les listes

- Suite d'**éléments de même type**
- Construction de listes :
  - opérateur `::`
    `'a -> 'a list -> 'a list = <fun>`
  - opérateur `@` (concaténation)
    `'a list -> 'a list -> 'a list = <fun>`
- Exemples :

```
# 1.2 ::  1.5 ::  1.8 ::  [];;
- :  float list = [1.2; 1.5; 1.8]

# [1.2] @ [1.5; 1.8];;
- :  float list = [1.2; 1.5; 1.8]

# [];;
```

Rappels : les listes

- Suite d'**éléments de même type**
- Construction de listes :
  - opérateur ::
    `'a -> 'a list -> 'a list = <fun>`
  - opérateur @ (concaténation)
    `'a list -> 'a list -> 'a list = <fun>`
- Exemples :

```
# 1.2 ::  1.5 ::  1.8 ::  [];;
- :  float list = [1.2; 1.5; 1.8]

# [1.2] @ [1.5; 1.8];;
- :  float list = [1.2; 1.5; 1.8]

# [];;
- :  'a list = []
```

Rappels : les listes

- Opérations sur les listes :

  1. **opérations prédéfinies** (`List.length`, `List.hd`, `List.tl`, `List.fold_left`, *etc*)

  2. **traitement récursif** (parcours de la liste), exemple :

```
# let rec inc = function l -> match l with
                    []    -> []
                  | t::r -> (t+1) :: (inc r);;
val inc : int list -> int list = <fun>
```

Rappels : les listes

- Notion d'**accumulateur** : utilisation d'un paramètre de type liste dans le but d'accumuler des résultats intermédiaires au cours d'un parcours récursif
- Exemple : renversement d'une liste

```
# let rec renverse l accu = match l with
        []    -> accu
      | t::r -> renverse r (t::accu);;
val renverse : 'a list -> 'a list -> 'a list = <fun>
```

- Question : que vaut `renverse [1;2;3;4;5];;` ?
- Exercice : écrire une fonction `separe` qui, à partir d'une liste d'entier, retourne une liste où apparaissent d'abord les entiers pairs, puis les impairs

Plan

Utilisation des listes

- Modélisation de données homogènes en nombre quelconque (mais fini)
- Opérations sur ces données (application de fonctions, comparaisons, tri, *etc*)
- Exemples d'utilisation des listes :
  - modélisation de polynômes
  - modélisation d'un annuaire
  - modélisation du calcul matriciel

Plan

Modélisation de polynômes

- Polynôme de degré n :

$$P(x) = a_0 + a_1 x^1 + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n$$

- Un polynôme peut être défini par les coefficients associés à chaque degré, exemple :

Modélisation de polynômes

- Polynôme de degré n :

$$P(x) = a_0 + a_1 x^1 + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n$$

- Un polynôme peut être défini par les coefficients associés à chaque degré, exemple :

  a) Représentation "pleine"

  $P_1(X) = 2 + 3x^2 \leftrightarrow$ `# let p1 = [2; 0; 3];;`

Modélisation de polynômes

- Polynôme de degré n :

$$P(x) = a_0 + a_1 x^1 + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n$$

- Un polynôme peut être défini par les coefficients associés à chaque degré, exemple :

  a) Représentation "pleine"

  $P_1(X) = 2 + 3x^2 \leftrightarrow$ `# let p1 = [2; 0; 3];;`

  b) Représentation "creuse"

  $P_1(X) = 2 + 3x^2 \leftrightarrow$ `# let p1 = [(2,0); (3,2)];;`

Modélisation de polynômes (suite)

- Comment implanter la somme de polynômes, pour chacune de ces représentations ?

Modélisation de polynômes (suite)

- Comment implanter la somme de polynômes, pour chacune de ces représentations ?
- Réponse (représentation “pleine”) :

Modélisation de polynômes (suite)

- Comment implanter la somme de polynômes, pour chacune de ces représentations ?
- Réponse (représentation "pleine") :

```
let rec somme p1 p2 = match (p1, p2) with
        [],[] -> []
      | _ ,[] -> failwith "pb de representations"
      | [],_ -> failwith "pb de representations"
      | t1::r1,t2::r2 -> (t1+t2)::(somme r1 r2);;
```

Modélisation de polynômes (suite)

- Comment implanter la somme de polynômes, pour chacune de ces représentations ?
- Réponse (représentation "pleine") :

```
let rec somme p1 p2 = match (p1, p2) with
        [],[] -> []
      | _ ,[] -> failwith "pb de representations"
      | [],_ -> failwith "pb de representations"
      | t1::r1,t2::r2 -> (t1+t2)::(somme r1 r2);;

val somme :  int list -> int list -> int list =
<fun>
```

Modélisation de polynômes (suite)

- Réponse (représentation “creuse”) :

Modélisation de polynômes (suite)

- Réponse (représentation “creuse”) :

```
let rec somme p1 p2 = match p1,p2 with
      [],[] -> []
    | _, [] -> p1
    | [], _ -> p2
    | (a1,b1)::r1,(a2,b2)::r2 -> if (b1=b2)
        then (a1+a2, b1)::(somme r1 r2)
        else if (b1<b2)
              then (a1, b1) ::  (somme r1 p2)
              else (a2, b2) ::  (somme p1 r2);;
```

Modélisation de polynômes (suite)

- Réponse (représentation “creuse”) :

```
let rec somme p1 p2 = match p1,p2 with
      [],[] -> []
    | _, [] -> p1
    | [], _ -> p2
    | (a1,b1)::r1,(a2,b2)::r2 -> if (b1=b2)
        then (a1+a2, b1)::(somme r1 r2)
        else if (b1<b2)
              then (a1, b1) ::  (somme r1 p2)
              else (a2, b2) ::  (somme p1 r2);;
val somme :  (int * 'a) list -> (int * 'a) list ->
(int * 'a) list = <fun>
```

Modélisation de polynômes (suite)

- Qu'en est-il du calcul du produit de polynômes ?
- Proposez une implantation du produit pour la représentation "creuse"

Modélisation de polynômes (suite)

- Qu'en est-il du calcul du produit de polynômes ?
- Proposez une implantation du produit pour la représentation "creuse"
- Réponse :

Modélisation de polynômes (suite)

- Qu'en est-il du calcul du produit de polynômes ?
- Proposez une implantation du produit pour la représentation "creuse"
- Réponse :

  1. calcul du produit par un monôme :

Modélisation de polynômes (suite)

- Qu'en est-il du calcul du produit de polynômes ?
- Proposez une implantation du produit pour la représentation "creuse"
- Réponse :

  1. calcul du produit par un monôme :

```
# let produit_m (a,b) l =
      List.map (function x,y -> a*x,b+y) l;;
```

Modélisation de polynômes (suite)

- Qu'en est-il du calcul du produit de polynômes ?
- Proposez une implantation du produit pour la représentation "creuse"
- Réponse :

1. calcul du produit par un monôme :

```
# let produit_m (a,b) l =
      List.map (function x,y -> a*x,b+y) l;;

val produit_m :  int * int -> (int * int) list
-> (int * int) list = <fun>
```

Modélisation de polynômes (suite)

- Qu'en est-il du calcul du produit de polynômes ?
- Proposez une implantation du produit pour la représentation "creuse"
- Réponse :

1. calcul du produit par un monôme :

```
# let produit_m (a,b) l =
      List.map (function x,y -> a*x,b+y) l;;

val produit_m :  int * int -> (int * int) list
-> (int * int) list = <fun>
```

2. calcul du produit de polynômes (version incomplète) :

Modélisation de polynômes (suite)

- Qu'en est-il du calcul du produit de polynômes ?
- Proposez une implantation du produit pour la représentation "creuse"
- Réponse :

1. calcul du produit par un monôme :

```
# let produit_m (a,b) l =
      List.map (function x,y -> a*x,b+y) l;;

val produit_m :  int * int -> (int * int) list
-> (int * int) list = <fun>
```

2. calcul du produit de polynômes (version incomplète) :

```
# let produit1 l1 l2 =
      (List.flatten (List.map (function x ->
produit_m x l2) l1));;
```

Modélisation de polynômes (suite)

- Calcul du produit de polynômes (version complète) :

```
# let produit l1 l2 =
      reduce (List.flatten (List.map (function x ->
produit_m x l2) l1));;
```

Modélisation de polynômes (suite)

- Calcul du produit de polynômes (version complète) :

```
# let produit l1 l2 =
      reduce (List.flatten (List.map (function x ->
produit_m x l2) l1));;

val produit :  (int * int) list -> (int * int) list
-> (int * int) list = <fun>
```

Modélisation de polynômes (suite)

- Calcul du produit de polynômes (version complète) :

```
# let produit l1 l2 =
      reduce (List.flatten (List.map (function x ->
produit_m x l2) l1));;

val produit :  (int * int) list -> (int * int) list
-> (int * int) list = <fun>

# let rec reduce l = let lprim =
      List.sort (function (a,b) -> function (c,d) ->
      compare b d) l) in
```

Modélisation de polynômes (suite)

- Calcul du produit de polynômes (version complète) :

```
# let produit l1 l2 =
      reduce (List.flatten (List.map (function x ->
produit_m x l2) l1));;

val produit :  (int * int) list -> (int * int) list
-> (int * int) list = <fun>

# let rec reduce l = let lprim =
      List.sort (function (a,b) -> function (c,d) ->
      compare b d) l) in match lprim with
        [] -> []
      | (a,b)::[] -> [(a,b)]
      | (a,b)::(c,d)::r -> if (b=d) then
              (reduce ((a+c,b)::r))
        else (a,b)::(reduce ((c,d)::r)) ;;
```

Modélisation de polynômes (suite)

- Exemple d'appel de la fonction `produit` :

```
# produit [(2,0); (3,1); (4,2)] [(2,1); (3,2)];;

- : (int * int) list =
    [(4, 1); (12, 2); (17, 3); (12, 4)]
```

Plan

Modélisation d'un annuaire

- Représentation (simplifiée) :

  $\longrightarrow$ liste de chaînes de caractères

- Exemple :

```
# let annuaire = ["Durand-0383123457";
      "Wagner-0383123458"; "Dupont-0383123456"];;

val annuaire : string list =
  ["Durand-0383123457"; "Wagner-0383123458";
   "Dupont-0383123456"]
```

Modélisation d'un annuaire (suite)

- Question : comment trier un annuaire ?
- Proposition : **tri par recherche du minimum**

  1. parcours de la liste pour récupérer le minimum
  2. mise en tête du minimum
  3. appel récursif sur le reste de la liste (*i.e.* sans le minimum)

Modélisation d'un annuaire

- Comment écrire une fonction qui retourne (i) le minimum de la liste et (ii) le reste de la liste ?
- Réponse :

Modélisation d'un annuaire

- Comment écrire une fonction qui retourne (i) le minimum de la liste et (ii) le reste de la liste ?
- Réponse :

```
# let rec rechercheMin l m lprim = match l with
```

Modélisation d'un annuaire

- Comment écrire une fonction qui retourne (i) le minimum de la liste et (ii) le reste de la liste ?
- Réponse :

```
# let rec rechercheMin l m lprim = match l with
  [] -> m,lprim
```

Modélisation d'un annuaire

- Comment écrire une fonction qui retourne (i) le minimum de la liste et (ii) le reste de la liste ?
- Réponse :

```
# let rec rechercheMin l m lprim = match l with
   [] -> m,lprim
| t::r -> if (m=[]) then (rechercheMin r [t] lprim)
```

Modélisation d'un annuaire

- Comment écrire une fonction qui retourne (i) le minimum de la liste et (ii) le reste de la liste ?
- Réponse :

```
# let rec rechercheMin l m lprim = match l with
   [] -> m,lprim
| t::r -> if (m=[]) then (rechercheMin r [t] lprim)
    else let oldmin = (List.hd m) in
```

Modélisation d'un annuaire

- Comment écrire une fonction qui retourne (i) le minimum de la liste et (ii) le reste de la liste ?
- Réponse :

```
# let rec rechercheMin l m lprim = match l with
   [] -> m,lprim
| t::r -> if (m=[]) then (rechercheMin r [t] lprim)
    else let oldmin = (List.hd m) in
     if (t < oldmin)
        then (rechercheMin r [t] (lprim @ [oldmin]))
```

Modélisation d'un annuaire

- Comment écrire une fonction qui retourne (i) le minimum de la liste et (ii) le reste de la liste ?
- Réponse :

```
# let rec rechercheMin l m lprim = match l with
   [] -> m,lprim
| t::r -> if (m=[]) then (rechercheMin r [t] lprim)
    else let oldmin = (List.hd m) in
     if (t < oldmin)
        then (rechercheMin r [t] (lprim @ [oldmin]))
        else (rechercheMin r m (lprim @ [t])) ;;
```

Modélisation d'un annuaire

- Comment écrire une fonction qui retourne (i) le minimum de la liste et (ii) le reste de la liste ?
- Réponse :

```
# let rec rechercheMin l m lprim = match l with
   [] -> m,lprim
| t::r -> if (m=[]) then (rechercheMin r [t] lprim)
    else let oldmin = (List.hd m) in
     if (t < oldmin)
        then (rechercheMin r [t] (lprim @ [oldmin]))
        else (rechercheMin r m (lprim @ [t])) ;;

val rechercheMin :  'a list -> 'a list -> 'a list
-> 'a list * 'a list = <fun>
```

Modélisation d'un annuaire (suite)

- Comment écrire un fonction récursive qui tri un annuaire en plaçant systématiquement le plus petit élément en tête de liste ?
- Réponse :

Modélisation d'un annuaire (suite)

- Comment écrire un fonction récursive qui tri un annuaire en plaçant systématiquement le plus petit élément en tête de liste ?
- Réponse :

```
# let rec tri l = match l with
```

Modélisation d'un annuaire (suite)

- Comment écrire un fonction récursive qui tri un annuaire en plaçant systématiquement le plus petit élément en tête de liste ?
- Réponse :

```
# let rec tri l = match l with
    [] -> []
```

Modélisation d'un annuaire (suite)

- Comment écrire un fonction récursive qui tri un annuaire en plaçant systématiquement le plus petit élément en tête de liste ?
- Réponse :

```
# let rec tri l = match l with
    [] -> []
  | a::[] -> [a]
```

Modélisation d'un annuaire (suite)

- Comment écrire un fonction récursive qui tri un annuaire en plaçant systématiquement le plus petit élément en tête de liste ?
- Réponse :

```
# let rec tri l = match l with
    [] -> []
  | a::[] -> [a]
  | _ -> let min,reste = (rechercheMin l [] []) in
```

Modélisation d'un annuaire (suite)

- Comment écrire un fonction récursive qui tri un annuaire en plaçant systématiquement le plus petit élément en tête de liste ?
- Réponse :

```
# let rec tri l = match l with
    [] -> []
  | a::[] -> [a]
  | _ -> let min,reste = (rechercheMin l [] []) in
    (List.hd min)::(tri reste);;
```

Modélisation d'un annuaire (suite)

- Comment écrire un fonction récursive qui tri un annuaire en plaçant systématiquement le plus petit élément en tête de liste ?
- Réponse :

```
# let rec tri l = match l with
    [] -> []
  | a::[] -> [a]
  | _ -> let min,reste = (rechercheMin l [] []) in
    (List.hd min)::(tri reste);;

val tri :  'a list -> 'a list = <fun>
```

Modélisation d'un annuaire (suite)

- Appel de la fonction de tri :

```
# let annuaire = ["Durand-0383123457";
      "Wagner-0383123458"; "Dupont-0383123456"];;

val annuaire : string list =
  ["Durand-0383123457"; "Wagner-0383123458";
   "Dupont-0383123456"]

# tri annuaire;;

- : string list =
["Dupont-0383123456"; "Durand-0383123457";
 "Wagner-0383123458"]
```

Plan

Modélisation du calcul matriciel

- Modélisation d'une matrice sous forme de liste de listes
- Exemple :

$$m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
# let m = [ [1;2;3]; [4;5;6]; [7;8;9] ];;
val m : int list list = [[1; 2; 3]; [4; 5; 6]; [7; 8; 9]]
```

Modélisation du calcul matriciel

- Question 1 : comment implanter la multiplication de matrices ?

Modélisation du calcul matriciel

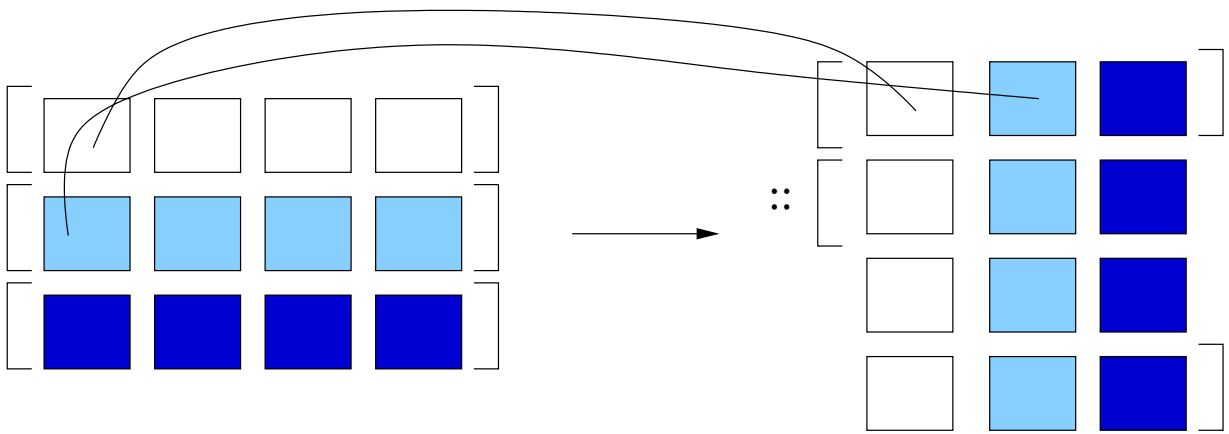
- Question 1 : comment implanter la multiplication de matrices ?
- Réponse : utiliser la **transposée** pour pouvoir faire la multiplication ligne à ligne

Modélisation du calcul matriciel

- Question 1 : comment implanter la multiplication de matrices ?
- Réponse : utiliser la **transposée** pour pouvoir faire la multiplication ligne à ligne
- Question 2 : comment implanter la fonction de transposition ?

Modélisation du calcul matriciel

- Question 1 : comment implanter la multiplication de matrices ?
- Réponse : utiliser la **transposée** pour pouvoir faire la multiplication ligne à ligne
- Question 2 : comment implanter la fonction de transposition ?



Modélisation du calcul matriciel (suite)

- Idée : utiliser la fonction `List.map`

Modélisation du calcul matriciel (suite)

- Idée : utiliser la fonction `List.map`

```
# let rec transpose mat = match mat with
```

Modélisation du calcul matriciel (suite)

- Idée : utiliser la fonction `List.map`

```
# let rec transpose mat = match mat with
    [] -> []
```

Modélisation du calcul matriciel (suite)

- Idée : utiliser la fonction `List.map`

```
# let rec transpose mat = match mat with
    [] -> []
  | [] ::_ -> []
```

Modélisation du calcul matriciel (suite)

- Idée : utiliser la fonction `List.map`

```
# let rec transpose mat = match mat with
    [] -> []
  | [] ::_ -> []
  | _ ::_ -> (List.map List.hd mat) ::
```

Modélisation du calcul matriciel (suite)

- Idée : utiliser la fonction `List.map`

```
# let rec transpose mat = match mat with
    [] -> []
  | [] ::_ -> []
  | _ ::_ -> (List.map List.hd mat) ::
      (transpose (List.map List.tl mat)) ;;
```

Modélisation du calcul matriciel (suite)

- Idée : utiliser la fonction `List.map`

```
# let rec transpose mat = match mat with
    [] -> []
  | [] ::_ -> []
  | _ ::_ -> (List.map List.hd mat) ::
      (transpose (List.map List.tl mat)) ;;

val transpose :  'a list list -> 'a list list =
<fun>
```

Modélisation du calcul matriciel (suite)

- Idée : utiliser la fonction `List.map`

```
# let rec transpose mat = match mat with
    [] -> []
  | [] ::_ -> []
  | _ ::_ -> (List.map List.hd mat) ::
      (transpose (List.map List.tl mat)) ;;

val transpose :  'a list list -> 'a list list =
<fun>
```

- Transposition de m ?

```
# transpose m;;
- :  int list list = [[1; 4; 7]; [2; 5; 8]; [3; 6;
9]]
```

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
```

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
    [],[] -> 0
```

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
    [],[] -> 0
  | _,[] -> failwith "erreur au produit scalaire"
  | [],_ -> failwith "erreur au produit scalaire"
```

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
    [],[] -> 0
  | _,[] -> failwith "erreur au produit scalaire"
  | [],_ -> failwith "erreur au produit scalaire"
  | t1::r1,t2::r2 -> (t1*t2) + (pscal r1 r2);;
```

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
    [],[] -> 0
  | _,[] -> failwith "erreur au produit scalaire"
  | [],_ -> failwith "erreur au produit scalaire"
  | t1::r1,t2::r2 -> (t1*t2) + (pscal r1 r2);;

val pscal :  int list -> int list -> int = <fun>
```

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
    [],[] -> 0
  | _,[] -> failwith "erreur au produit scalaire"
  | [],_ -> failwith "erreur au produit scalaire"
  | t1::r1,t2::r2 -> (t1*t2) + (pscal r1 r2);;

val pscal :  int list -> int list -> int = <fun>
```

- Implantation du produit ligne à ligne

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
    [],[] -> 0
  | _,[] -> failwith "erreur au produit scalaire"
  | [],_ -> failwith "erreur au produit scalaire"
  | t1::r1,t2::r2 -> (t1*t2) + (pscal r1 r2);;

val pscal :  int list -> int list -> int = <fun>
```

- Implantation du produit ligne à ligne

```
# let rec prodligne m1 m2 = match m1 with
```

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
    [],[] -> 0
  | _,[] -> failwith "erreur au produit scalaire"
  | [],_ -> failwith "erreur au produit scalaire"
  | t1::r1,t2::r2 -> (t1*t2) + (pscal r1 r2);;

val pscal :  int list -> int list -> int = <fun>
```

- Implantation du produit ligne à ligne

```
# let rec prodligne m1 m2 = match m1 with
  [] -> []
```

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
    [],[] -> 0
  | _,[] -> failwith "erreur au produit scalaire"
  | [],_ -> failwith "erreur au produit scalaire"
  | t1::r1,t2::r2 -> (t1*t2) + (pscal r1 r2);;

val pscal :  int list -> int list -> int = <fun>
```

- Implantation du produit ligne à ligne

```
# let rec prodligne m1 m2 = match m1 with
  [] -> []
| t::r -> (List.map (function x -> pscal t x) m2)
      ::  (prodligne r m2);;
```

Modélisation du calcul matriciel (suite)

- Implantation du produit scalaire (**une** ligne avec **une** ligne)

```
# let pscal l1 l2 = match l1, l2 with
    [],[] -> 0
  | _,[] -> failwith "erreur au produit scalaire"
  | [],_ -> failwith "erreur au produit scalaire"
  | t1::r1,t2::r2 -> (t1*t2) + (pscal r1 r2);;

val pscal :  int list -> int list -> int = <fun>
```

- Implantation du produit ligne à ligne

```
# let rec prodligne m1 m2 = match m1 with
  [] -> []
| t::r -> (List.map (function x -> pscal t x) m2)
     ::  (prodligne r m2);;

val prodligne :  int list list -> int list list ->
int list list = <fun>
```

Modélisation du calcul matriciel (suite)

- Finalement : produit matriciel

Modélisation du calcul matriciel (suite)

- Finalement : produit matriciel

```
# let multmat m1 m2 = prodligne m1 (transpose m2);;
```

Modélisation du calcul matriciel (suite)

- Finalement : produit matriciel

```
# let multmat m1 m2 = prodligne m1 (transpose m2);;
val multmat :  int list list -> int list list ->
int list list = <fun>
```

Modélisation du calcul matriciel (suite)

- Finalement : produit matriciel

```
# let multmat m1 m2 = prodligne m1 (transpose m2);;
val multmat :  int list list -> int list list ->
int list list = <fun>
```

- Exemple :

Modélisation du calcul matriciel (suite)

- Finalement : produit matriciel

```
# let multmat m1 m2 = prodligne m1 (transpose m2);;
val multmat :  int list list -> int list list ->
int list list = <fun>
```

- Exemple :

```
# let m1,m2 = [[6;5];[0;8]],[[1;2];[3;4]];;
```

Modélisation du calcul matriciel (suite)

- Finalement : produit matriciel

```
# let multmat m1 m2 = prodligne m1 (transpose m2);;

val multmat :  int list list -> int list list ->
int list list = <fun>
```

- Exemple :

```
# let m1,m2 = [[6;5];[0;8]],[[1;2];[3;4]];;

val m1 :  int list list = [[6; 5]; [0; 8]]
val m2 :  int list list = [[1; 2]; [3; 4]]
```

Modélisation du calcul matriciel (suite)

- Finalement : produit matriciel

```
# let multmat m1 m2 = prodligne m1 (transpose m2);;

val multmat :  int list list -> int list list ->
int list list = <fun>
```

- Exemple :

```
# let m1,m2 = [[6;5];[0;8]],[[1;2];[3;4]];;

val m1 :  int list list = [[6; 5]; [0; 8]]
val m2 :  int list list = [[1; 2]; [3; 4]]

# multmat m1 m2;;
```

Modélisation du calcul matriciel (suite)

- Finalement : produit matriciel

```
# let multmat m1 m2 = prodligne m1 (transpose m2);;

val multmat :  int list list -> int list list ->
int list list = <fun>
```

- Exemple :

```
# let m1,m2 = [[6;5];[0;8]],[[1;2];[3;4]];;

val m1 :  int list list = [[6; 5]; [0; 8]]
val m2 :  int list list = [[1; 2]; [3; 4]]

# multmat m1 m2;;
- :  int list list = [[21; 32]; [24; 32]]
```