Les vecteurs

Mardi 16 Octobre 2012

Option Informatique

Ecole Alsacienne

Plan

1. Points-virgules et portée des variables
2. Qu'est-ce qu'un vecteur ?
3. Les vecteurs en Caml
4. Premières fonctions
5. Diviser pour régner
6. Exercices

Points-virgules et portée des variables

Points-virgules : la règle générale

- La règle générale :

  A la fin de chaque instruction, il faut un point-virgule

- Il existe néanmoins plusieurs exceptions :
  - `do … done` (boucles `for` et boucles `while`)
  - `if … then … else` (conditions)
  - `let … in` (déclarations internes)
  - `;;` (fin de phrase)

POINTS-VIRGULES : L'EXCEPTION `do` ... `done`

- L'exception `do` ... `done` :

  Il n'y a pas de point-virgule avant le mot clef `done`

- Boucle `for` :

```
for i = 1 to 10
do
     print_int i;
     print_newline()
done;
```

POINTS-VIRGULES : L'EXCEPTION `do` ... `done`

- L'exception `do` ... `done` :

  Il n'y a pas de point-virgule avant le mot clef `done`

- Boucle while :

```
let i = ref 0 in
while (!i < 11)
do
     print_int i;
     print_newline();
     i := !i + 1
done;
```

POINTS-VIRGULES : L'EXCEPTION `IF … THEN … ELSE`

- L'exception `if … then … else` :

Il n'y a pas de point-virgule avant les mots clefs `then` et `else`

- Construction simple `if … then … else` :

```
if (age<18)
then
      "mineur"
else
      "majeur";
```

POINTS-VIRGULES : L'EXCEPTION `IF ... THEN ... ELSE`

- L'exception `if ... then ... else` :

Il n'y a pas de point-virgule avant les mots clefs `then` et `else`

- Construction sans le `else` :

```
if (age<18)
then
      print_string "mineur";
```

- Attention, sans le `else`, le `then` doit être de type `unit`

```
if (age<18)
then
      "mineur";
```

POINTS-VIRGULES : L'EXCEPTION `IF ... THEN ... ELSE`

- L'exception `if ... then ... else` :

Il n'y a pas de point-virgule avant les mots clefs `then` et `else`

- Que fait ce code ?

```
if (age<18)
then
     print_string "mineur";
     print_newline()
else
     print_string "majeur";
     print_newline();
```

POINTS-VIRGULES : L'EXCEPTION `IF ... THEN ... ELSE`

- L'exception `if ... then ... else` :

  Il n'y a pas de point-virgule avant les mots clefs `then` et `else`

- Deux nouveaux mots clefs : `begin` et `ends`

```
If (age<18)
then
        begin
        print_string "mineur";
        print_newline()
        end
else
        begin
        print_string "majeur";
        print_newline()
        end;
```

- L'exception `end` :

  Il n'y a pas de point-virgule avant le mot clef `end`

POINTS-VIRGULES : L'EXCEPTION `LET` … `IN`

- L'exception `let … in` :

  Il n'y a pas de point-virgule après le mot clef `in`

- Premiers exemples :

```
let x = 2 in
x * 2;

let x = (factorielle 5) in
print_int x;
```

POINTS-VIRGULES : L'EXCEPTION `LET` ... `IN`

- L'exception `let ... in` :

  Il n'y a pas de point-virgule après le mot clef `in`

- Cas d'une fonction :

```
let affiche_et_teste_si_majeur age =
      print_int age;
      (age > 17)
      in

affiche_et_teste_si_majeur 17;
```

Portée des variables

- Comment sont évaluées les lignes de code suivantes ?

```
let x = 3;;

x ;;

let y = 2 in
y * 3;;

y ;;

x;;
```

PORTÉE DES VARIABLES

- Comment sont évaluées les lignes de code suivantes ?

```
let affiche_carre x =
      let carre = x*x in
      print_int carre;;

affiche_carre 4;;

carre;;
```

Portée des variables

- Les éléments définis grâce à un `let … in` sont définis jusqu'à prochain double points-virgules

- Ces doubles points-virgules marquent « la fin de la phrase », c'est-à-dire l'endroit où l'interpréteur va s'arrêter

- Exemple

```
let affiche_carre x =
    let carre = x*x in
    print_int carre;;

affiche_carre 4;;
```

Qu'est-ce qu'un vecteur ?

Plusieurs appellations

Trois façons de parler de la même chose :

- Vecteur
- Tableau (unidimensionnel)
- Liste indexée

Un meuble à tiroir

Qu'est-ce qu'un vecteur ?

Un vecteur est un ensemble de taille fixe contenant des données de même type indexées par des entiers.

- Un ensemble de taille fixe
- Des données de même type
- Indexée par des entiers : $[\![0 ; n-1]\!] = \{0 , 1 , \dots , n-1\}$

Remarque : une chaîne de caractère est en quelque sorte un vecteur composé de caractères

B	o	n	j	o	u	r
0	1	2	3	4	5	6

FORMELLEMENT

- Un **type de données abstrait** est défini par :
  - Un ensemble de données
  - La liste des opérations qui permettent de manipuler ces données

- L'implémentation d'une structure de données dans un langage de programmation est appelée une **structure de données**

Mise en situation

- Que pouvez-vous faire face à cette structure ?
- Que ne pouvez-vous pas faire face à cette structure ?

Un meuble a une taille fixe

- La taille d'un vecteur est **fixée lors de la création** de ce vecteur (et ne peut être modifiée par la suite)

```
v = CreerVecteur(taille, valeur)
```

- Si le vecteur est de taille *n*, il est impossible d'y mettre plus de *n* éléments (si vous essayez, Caml vous le reprochera)

- Cette **taille** peut être **connue** à tout moment **en temps constant** (instantanément)

```
n = Taille (v)
```

Ouvrir les tiroirs

- On peut **accéder** au contenu de n'importe quelle case en temps constant.
  - En pseudo code : `v.(i)` ou `v[i]`

- On peut modifier le contenu de n'importe quelle case en temps constant
  - En pseudo code : `v.(i) <- val` ou `v[i] <- val`

- Attention, l'indexation commence à 0.

B	o	n	j	o	u	r
0	1	2	3	4	5	6

Les vecteurs en Caml

Le module **Array**

- Caml fonctionne avec un système de **modules** : lorsque vous travaillez sur un domaine précis, vous pouvez charger les fonctions correspondantes.

- Avantages :
  - Inutile de charger toutes les fonctionnalités tout le temps
  - Les fonctions les plus courantes sont déjà implémentées (et bien implémentées)

- Pour les vecteurs, il existe le module **Array** :

```
open Array;;
```

DÉFINIR UN VECTEUR

- `[| ... |]` : Case par case :
  - Syntaxe : `let nom = [| e0 ; e1 ; … ; en |] ;;`
  - Exemple : `let v1 = [| 3 ; 1 ; 6 |] ;;`

- `Array.make` : par sa taille et sa valeur de base
  - Syntaxe : `let nom = Array.make longueur valeur ;;`
  - Exemple : `let v2 = Array.make 4 true ;;`

```
val v2 : bool array = [|true; true; true; true|]
```

- `Array.init` : par sa taille et une fonction :
  - Syntaxe : `let nom = Array.init taille fonction ;;`
  - Exemple : `let v3 = Array.init 4 (function x->5*x);;`

```
val v3 : int array = [|0; 5; 10; 15|]
```

FONCTIONS DE BASE

- Longueur du vecteur :

```
let n = Array.length(v);;
```

- Accéder à un élément : `v.(i)`

- Modifier un élément : `v.(i) <- val`

- Que va répondre Caml ?

```
let v = [| 2 ; 4; 6 |];;
v.(3);;
    Exception: Invalid_argument "index out of bounds".
```

Fonctions avancées

- `Array.concat` : concaténer deux vecteurs
- `Array.sub` : récupérer une sous-partie du vecteur
- `Array.copy` : créer une copie du vecteur
- `Array.iter` : appliquer une fonction à tous les éléments du vecteur
- `Array.map` : créer un vecteur `[|f e0; … ; f en|]` à partir d'un vecteur `[| e0; … ; en |]`

Premières fonctions

TROUVER LE MAXIMUM

- **Exercice** : Comment trouver le maximum d'un vecteur ?

- **Solution** :

```
maxVect(v) =
    max_courant <- v.(0)
    n <- taille(v)
    Pour i allant de 1 à (n-1)
          Faire
                Si (v.(i)>max_courant)
                      Alors
                            max_courant = v.(i)
                Fin si
          Fin faire
    Renvoyer max_courant
```

TROUVER L'INDICE DU MAXIMUM

- **Exercice** : Comment trouver l'indice correspondant à ce maximum ?

- **Solution** :

```
maxVect2(v) =
    max_courant <- v.(0)
    i_max_courant <- 0
    n <- taille(v)
    Pour i allant de 1 à (n-1)
          Faire
                Si (v.(i)>max_courant)
                      Alors
                            max_courant = v.(i)
                            i_max_courant <- i
                Fin si
          Fin faire
    Renvoyer i_max_courant
```

Recherche d'un élément – Version basique

- **Exercice** : Comment savoir si l'élément `x` apparait dans le vecteur `v` ?

- **Solution** :

```
rechercheVect(v,x) =
    il_est_la = faux
    n <- taille(v)
    Pour i allant de 0 à (n-1)
          Faire
                Si (v.(i)=x)
                      Alors
                            il_est_la = vrai
                Fin si
          Fin faire
    Renvoyer il_est_la
```

Recherche d'un élément – Un peu plus malin

- **Exercice** : Comment savoir si l'élément `x` apparait dans le vecteur `v` ? (sans parcourir automatiquement tout le vecteur)

- **Solution** :

```
rechercheVect2(v,x) =
    il_est_la = faux
    n <- taille(v)
    i <- 0
    Tant que ( (il_est_la = faux) && (i<n) )
           Faire
                  Si (v.(i)=x)
                         Alors
                                il_est_la = vrai
                         Sinon
                                i = i+1
                  Fin si
           Fin faire
    Renvoyer il_est_la
```

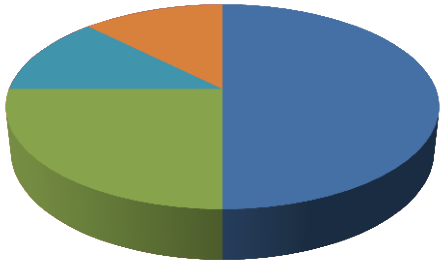
Diviser pour régner

Donnez moi un chiffre

- Donnez moi un chiffre entre 1 et 1 000

- Je vous le retrouve en 10 questions binaires (oui – non)

Comment ça marche ?



- Taille de l'intervalle de recherche :

Questions posées	0	1	2	3	4	5	6	7	8	9	10
Intervalle	1024	512	256	128	64	32	16	8	4	2	1

Diviser pour régner

- Une méthode **Diviser pour régner** (en anglais "divide and conquer") consiste à diviser un problème de grande taille en sous-problèmes analogues.

- Intérêt :
  - On se ramène récursivement à un cas de base
  - Facile à implémenter avec les fonctions récursives

Recherche d'un élément dans un vecteur trié

- **Question** : On suppose désormais que le vecteur `v` est trié par ordre croissant. Comment savoir si l'élément `x` apparait dans ce vecteur ?

- **Idée** : Comme **le vecteur est trié**, on a pour tout $0 \leq m < n$ (n étant la taille du vecteur) :
  - Soit `x=v.(m)` : dans ce cas `x` appartient à `v`
  - Soit `x<v.(m)` : dans ce cas `x` appartient à `v` *ssi* `x` apparait dans `v` avant l'indice `m`
  - Soit `x>v.(m)` : dans ce cas `x` appartient à `v` *ssi* `x` apparait dans `v` après l'indice `m`

Recherche d'un élément dans un vecteur trié

- Approche **"Diviser pour régner"** :
  - Soit on est dans un cas trivial : `x=v.(m)`
  - Soit on se ramène à un problème analogue sur un vecteur de taille plus petite :

- Comment choisir `m` ?
  - On coupe l'intervalle qu'on étudie en deux
  - Plus formellement, si on travaille entre les indices `a` et `b`, on prend `m = (a + b) / 2`

Généraliser pour mieux résoudre

- Un principe assez courant en mathématiques et en informatique : on **généralise le problème pour le résoudre**.

- **Nouvel énoncé** : On suppose désormais que le vecteur `v` est trié par ordre croissant. Comment savoir si l'élément `x` apparait dans ce vecteur **entre les indices a et b**?

IMPLÉMENTATION

- En pseudo-code :

```
Appartient_Aux(x,v,a,b) =
    Si (a>b) (* Borne inférieure > Borne supérieure *)
            Alors Renvoyer faux
    Fin Si

    m <- (a+b)/2 (* milieu du sous-vecteur considéré *)

    Si x=v.(m)
            Alors  (* Cas de base *)
                   Renvoyer vrai
            Sinon  (* Cas récursifs *)
                   Si x<v.(m)
                          Alors Appartient_Aux(x,v,a,m-1)
                          Sinon Appartient_Aux(x,v,m+1,b)
                   Fin si
    Fin si
```

QUAND A-T-ON `a>b` ?

- Le cas `a>b` survient lorsque l'élément recherché `x` n'est pas dans le vecteur `v`

5	7	8	12	17	23	26	27	42	43	49	51	55	68	70	75
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Exemple :
  - On cherche 25
  - On cherche donc :
    - Entre 0 et 15 → `m=7` → `v.(7) = 27` → recherche à gauche
    - Entre 0 et 6 → `m=3` → `v.(3) = 12` → recherche à droite
    - Entre 4 et 6 → `m=5` → `v.(5) = 23` → recherche à droite
    - Entre 6 et 6 → `m=6`→ `v.(6) = 26` → recherche à gauche
    - Entre 7 et 6 → `7>6` → on renvoie faux

ENCAPSULATION

- En pratique, ce qui nous intéresse, c'est de savoir si l'élément `x` est dans le vecteur `v` (en entier)

- On utilise donc notre fonction `Appartient_Aux(x,v,a,b)` avec
  - `a = 0` (première case du vecteur)
  - `b = taille(v)-1` (dernière case du vecteur)

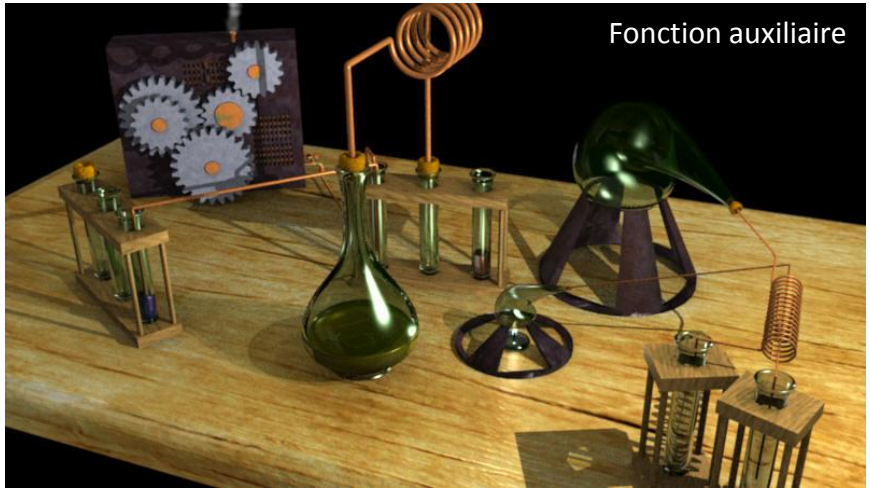
- Plutôt que de devoir taper à chaque fois ces arguments, on **encapsule** notre fonction auxiliaire dans une fonction avec moins d'arguments.

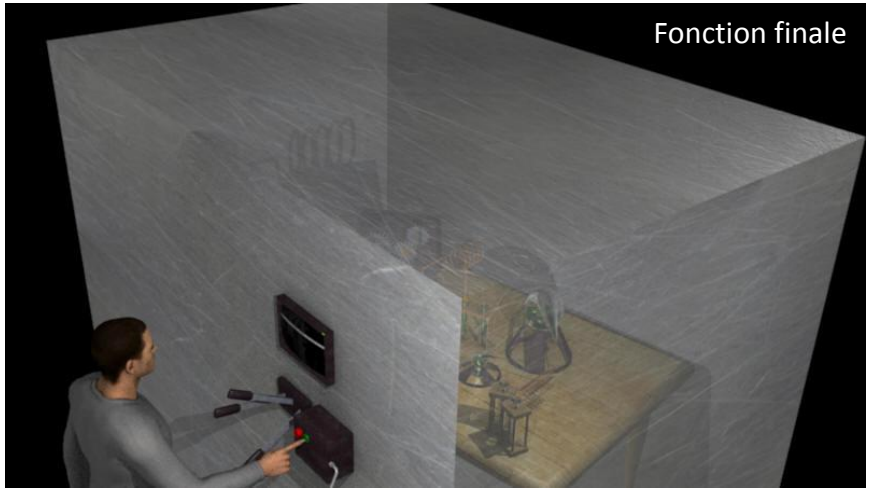
- Utilisation :

```
Appartient(v,x) =
    Appartient_Aux(a,b) = ...



    Appartient_Aux(0, taille(v)-1)
```

ENCAPSULATION

- Avantages :
  - L'utilisateur a une fonction simple à utiliser `Appartient(v,x)`
  - La fonction auxiliaire connait déjà les variables `v` et `x`, donc elle n'a que deux arguments : `Appartient_Aux(a,b)`
- Inconvénient :
  - La fonction auxiliaire n'est pas accessible ailleurs dans le programme





Source : Nab, Site du Zero, Tutoriel C++ sur la POO

En Caml

- Implémentation en Caml :

```
let appartient x v =

let rec aux a b =
      if a > b
            then false (* Cas de base *)
            else
                  let m = (a+b) / 2 in
                  if x = v.(m)
                        then true
                        else if x < v.(m)
                              then aux a (m-1)
                              else aux (m+1) b
in

aux 0 (Array.length v);;
```

Exercices

TRIÉ PAR ORDRE CROISSANT ?

- **Exercice** : Ecrire une fonction qui teste si un vecteur est trié par ordre croissant

- **Solution** :

```
estCroissant(v) =
    pour_l_instant <- vrai;
    n <- taille(v)
    Pour i allant de 0 à (n-2)
          Faire
                Si (v.[i]>v.[i+1])
                      Alors
                            pour_l_instant = faux
                            BREAK
                Fin si
          Fin faire
    Renvoyer pour_l_instant
```

MOYENNE D'UN VECTEUR DE RÉELS ?

- **Exercice** : Ecrire une fonction qui calcule la moyenne d'un vecteur de réels

- **Solution** :

```
moyenneVect(v) =

    somme <- 0;
    n <- taille(v)

    Pour i allant de 0 à (n-1)
          Faire
                somme <- v.[i] + somme
          Fin faire

    Renvoyer (somme/n) (* Attention aux types *)
```

Les petits chaperons rouges

- Le petit chaperon rouge se promène dans la forêt.
- Lorsqu'elle entre sur le terrain de chasse du grand méchant loup, celui-ci la dévore.
- Heureusement, nous avons tout un stock de petits chaperons rouges en réserve
- **Question** : comment déterminer (le plus rapidement possible) où commence le terrain de chasse du Grand Méchant Loup ?

Source : Steedman, Amy. Nursery Tales. Paul Woodroffe, illustrator

LES PETITS CHAPERONS ROUGES

- Plus formellement :
  - Forêt de taille n
  - On cherche l'indice k à partir duquel on entre sur les terres du loup

```
trouverLeLoup(n)=

   pas <- n/2
   endroit_calme <- 0

   Tant que (pas > 0)
   Faire
        EnvoyerChaperon(endroit_calme + pas);

        Si ChaperonEncoreEnVie
           Alors
              endroit_calme <- endroit_calme + pas
        Fin si

        pas <- pas/2

   Fin faire

Renvoyer endroit_calme
```

- Pour approfondir :
  - Avec un seul chaperon rouge (trivial)
  - Avec uniquement deux chaperons rouges (beaucoup moins trivial)

EXPONENTIATION CLASSIQUE

- **Exponentiation** = calcul des puissances d'un nombre (en général entier)

- Notation : $a^n = a \times a \times \cdots \times a$ (n fois)

- **Exercice** : Ecrire une fonction d'exponentiation qui à partir de a et n calcule a^n

- **Solution** :

```
Exponentiation(a,n) =
Si n = 1
   Alors a
   Sinon a * Exponentiation(a, n-1)
```

PEUT-ON FAIRE PLUS RAPIDE ?

- On a suivi sans trop réfléchir la formule :

$$a^n = a \times a \times \cdots \times a \text{ (n fois)}$$

- **Question** : Est-ce qu'on ne peut pas faire plus rapide ?

- **Idée** :

$$x^n = \begin{cases} x & si\ n = 1 \\ (x^2)^{n/2} & si\ n\ est\ pair \\ x \times (x^2)^{(n-1)/2} & si\ n\ est\ impair \end{cases}$$

EXPONENTIATION RAPIDE

- **Exercice** : Ecrire une fonction d'exponentiation rapide basée sur la formule suivante :

- **Solution** :

```
expRapide(x,n) =

    Si n = 1
          Alors Renvoyer x
    Fin si

    Si n est pair
          Alors Renvoyer expRapide(x*x, n/2)
          Sinon Renvoyez x * expRapide(x*x, (n-1)/2)
    Fin si
```

Exponentiations – Comparaison des résultats

- Nombre de multiplications :

N	Exponentiation classique	Exponentiation rapide
10	9	4
50	49	7
100	99	8

- Exemple :

```
er(2,10)                          appel récursif
        = er(2×2,5)               multiplication 1
        = er(4,5)                 appel récursif
        = 4×er(4×4,2)             multiplication 2
        = 4×er(16,2)              appel récursif
        = 4×er(16×16,1)           multiplication 3
        = 4×er(256,1)             cas de base
        = 4×256                   multiplication 4
        = 1024
```

Prochaine séance

Mardi 23 octobre 2011

[TD] Les Vecteurs en pratique