TP2 MPSI: Traits impératifs de Caml Light

Alexis Saurin

5 mars 2003

Introduction

On va étudier dans ce TP les principaux traits impératifs de Caml Light : structures de données (références, tableaux (ou vecteurs), enregistrements modifiables), de contrôles (boucles, séquences, exceptions), et quelques algorithmes basés sur ces éléments du langage (tris de tableaux...).

1 Traits impératifs du langage Caml Light

1.1 L'itération, les boucles

Question 1: Factorielle

Construire une fonction `factorielle` de type `int -> int`, qui calcule la factorielle d'un entier, et cela sans utiliser de définition par récurrence, mais avec une boucle.

Question 2: Nombres premiers

Construire une fonction `est_premier` de type `int -> bool` qui détermine si un nombre est premier ou non. Construire ensuite une fonction `imprime_diviseurs` de type `int -> unit` qui imprime les diviseurs de l'entrée ainsi que le nombre de diviseurs.

On utilisera les fonctions `print_int`, `print_string`, `print_newline` et `print_char`, on pourra se servir des caractères spéciaux '\t' et '\n' pour formater le résultat.

Question 3: Interactivité

Jusque là, tous les programmes que l'on a écrits prenaient un argument, faisaient un calcul, rendaient un résultat, mais il n'y avait pas d'interactivité entre l'utilisateur et le programme en train de s'exécuter. En particulier, une fois le programme en train de s'exécuter, l'utilisateur ne peut plus fournir d'information qui guidera l'exécution du programme. Cela n'est pas un comportement que l'on voudrait : les programmes informatiques doivent la plupart du temps agir et réagir en fonction du comportement de l'environnement (penser à un distributeur de billets qui doit recevoir les instructions tapées sur le clavier ou bien l'écran, à un ascenseur qui doit pouvoir être arrêté si on appuie sur le bouton d'alarme, ou bien à l'interpréteur de Caml Light qui enregistre les programmes que vous tapez sur le clavier...). Pour le moment on ne sait pas écrire ce genre de programmes, c'est le but de cet exercice.

On a vu des fonctions à effets de bord : `print_int`, `print_string`, ... qui étaient de type `int -> unit`, `string -> unit` ... elles recevaient un argument d'un certain type et l'imprimaient sur la sortie standard ; on va maintenant utiliser des fonctions qui d'une certaine manière sont duales des premières : elles auront pour type `unit -> int`, elles liront sur l'entrée standard et fourniront un résultat du type demandé. ces fonctions seront entre autres `read_int` et `read_line`, de types respectifs `unit -> int` et `unit -> string`.

Essayer ces fonctions sur quelques

exemples pour en comprendre le fonctionnement.

Écrire un programme qui demande un nombre à l'utilisateur et qui lui répond si ce nombre est premier ou non, un autre qui lui affiche la liste de tous les diviseurs de ce nombre. On aura des programmes de type `unit -> unit`.

Question 4: c+c-

Connaissez-vous le jeu c+c- ? Cela se joue à deux : l'un des joueurs choisit un nombre tandis que le second essaie de le deviner en faisant des essais. Le premier joueur donne pour seule indication : "c'est plus !" ou bien "c'est moins !" selon la position du nombre proposé par rapport à la solution. La partie est finie lorsque le second joueur a trouvé la solution.

On va programmer ce jeu pour que l'ordinateur joue le rôle du premier joueur.

Voici à quoi devra ressembler une exécution du programme :

```
#cpcm ();;
Taper un nombre : 200
c'est plus!
taper un nombre : 800
c'est moins!
taper un nombre : 600
c'est plus!
taper un nombre : 700
c'est moins!
taper un nombre : 650
c'est moins!
taper un nombre : 625
BRAVO, c'était bien 625

- : unit = ()
```

Dans un premier temps, vous ferez un programme qui joue lorsqu'on lui donne le nombre à deviner en argument, c'est-à-dire un programme `cpcm_aux` de type `int -> unit`.

Ensuite, vous utiliserez la fonction `random__int` de type `int -> int` qui rend un entier tiré "au hasard" entre 0 et son argument, et vous construirez le programme `cpcm` de type `unit -> unit` qui permet de jouer. vous choisirez par exemple de jouer entre 0 et 1000 (mais vous pourriez aussi faire en sorte que l'intervalle dans lequel le nombre est tiré soit demandé par l'utilisateur).

Question bonus:

Sauriez-vous écrire le programme qui échange les rôles : vous choisissez un nombre et l'ordinateur essaie de le deviner selon vos indications ? (sauriez-vous tenir compte du fait que l'utilisateur peut éventuellement tricher ?)

1.2 les tableaux

Question 5: Manipulations de tableaux

Essayez de créer un tableau, d'accéder à un élément du tableau, de modifier un élément, de déterminer la longueur d'un tableau.

Question 6: Application d'une fonction aux éléments d'un tableau

Définir une fonction prenant un tableau d'entier en argument et calculant la somme de tous ses éléments. Faire de même avec la multiplication, la soustraction, la division (pour un tableau d'élements non nuls), la somme des carrés des élements, des cubes, des valeurs absolues...

Plus généralement (et plus rapidement), définir une fonction `applique_a_vect` de type `int -> (int -> int -> int) -> (int vect) -> int` c'est-à-dire une fonction qui prend un élément initiale a_0, une fonction de deux arguments, un talbeau d'entiers, et qui calcule f(f(...f(f(a_0, v.0), v.1),...), v.n).

Question 7: Triangle de Pascal

Écrire un programme `pascal` qui prend un entier n et qui construit les n premières lignes du triangle de Pascal :

```
[|
[|1|];
[|1; 1|];
[|1; 2; 1|];
[|1; 3; 3; 1|];
[|1; 4; 6; 4; 1|];
|]
```

1.3 Enregistrements à champs modifiables (ou mutables)

Certains d'entre vous ont vu à la fin du TP précédent les enregistrements. Il s'agit d'un type de données à part (ou plutôt une famille de types : il y a DES types enregistrements), que l'on doit définir, qui correspond à un n-uplet à champs nommés, c'est-à-dire qu'on accéde aux différents champs, non pas en indiquant la position de l'élément auquel on cherche à accéder, mais en le désignant grâce à un label.

Pour ceux qui n'avaient pas eu le temps d'aborder les dernières questions du précédent TP, les voici :

Question 8: Enregistrements

Il existe d'autres types en Caml Light que ceux déjà rencontrés, et on peut en définir de nouveaux, en particulier une famille de types appellés types enregistrements :

```
type pave =
{L : float; l: float; h: float};;
let p={L=1.5; l=1.; h=3.8};;
p.L;;
```

Écrire un programme calculant le volume d'un parallélépipède rectangle.

Question 9: Nombres complexes

Définir un type pour les nombres complexes et les opérations principales sur les complexes : partie réelle, partie imaginaire, addition, multiplication, conjugaison, module.

On va maintenant parler d'une variante des enregistrements : les enregistrements mutables : quand on considère des données comme une base de données de comptes en banque, on peut avoir certaines données qui sont stables (comme le nom des clients, leur date de naissance) mais on peut aussi vouloir modifier certains de champs (le crédit courant du compte, le montant des intérêts non crédités, l'adresse du client...) Pour cela, on a besoin d'un type de donnée modifiable, qu'on a pas besoin de recréer à chaque fois qu'on veut faire une modification. Les enregistrements mutables vont être adaptés à ce genre d'utilisation : certains champs de l'enregistrements, déclarés comme tels, vont pouvoir être modifiés.

Définition d'un type enregistrement modifiable :

```
type nom_du_type = { ... ;
             mutable label_i : t ;
                       ... }
```

On peut bien sûr mélanger champs modifiables et champs non modifiables.

Exemple :

```
type date =
{jour: int; mois: int; annee: int};;

type client =
    {nom : string;
     prenom : string;
     telephone : int;
     naissance : date;
     mutable prochain_RDV : date;
     mutable credit_compte_courant : float;
     mutable interets_cc : float;
     mutable crédit_livret_A : float;
     mutable interets_livret_A : float;
     mutable commentaire : string};;
```

On peut alors modifier la valeur d'un champ déclaré `mutable` de la manière suivante :

```
dupont.credit_compte_courant <- 0.
```

Question 10: Banque
Écrire des programmes de gestion d'un compte : accès au crédit, modification du crédit, calcul de l'intérêt d'un mois sur l'autre (étant donné un taux d'intérêt mensuel), virement d'un compte sur l'autre, enregistrement d'un rendez-vous, écriture d'un commentaire...

Question 11: Simulation des enregistrements
Simuler les enregistrements mutables en utilisant des champs de type `'a ref` au lieu de `'a`. Implémenter les fonctions de bases de manipulation des enregistrements mutables.

En fait, en Caml Light, c'est plutôt dans l'autre sens que les choses marchent : une référence est un enregistrement à un champ modifiable :

```
type 'a ref = {mutable valeur : 'a};;
```

On peut paramètrer une définition de type par une variable de type...

Vous pouvez redéfinir les fonctions de manipulation des références à partir de cette définition.

1.4 Chaînes

Les chaînes de caractères sont en fait elles aussi des objets à caractère impératif : ce sont en fait des vecteurs un peu spéciaux. En particulier, on peut accèder à un élément particulier de la chaîne, modifier physiquement l'un des éléments...

```
let s="coucou";;
s.[3];;
s.[0]<-`C`;;
s;;
```

2 Algorithmes de tris

Question 8: Tri sélection
Programmer le tri sélection : définir d'abord une fonction `min_vect_i_j` qui prend un vecteur `v` sur un type `'a`, deux entiers `i` et `j` et renvoie l'indice dans `v` d'un élément minimal dans `{ v.(i), v.(i+1), ..., v.(j)}`. Utiliser ensuite cette fonction `tri_selection` qui fait un tri par sélection d'un tableau de type `'a vect`.

Question 9: Tri insertion
Programmer le tri par insertion :

- on définit d'abord une fonction `trouve_position` qui étant donné un vecteur v et un entier j et à supposer que v soit trié jusqu'à l'indice j-1, retourne le plus grand indice $i \in [0, j-1]$ tel que v.(j) soit compris entre v.(i-1) et v.(i) ;
- on définit ensuite une finction `insertion` qui prend un vecteur v, un entier j (indice jusqu'auquel le tableau est trié), et un entier i (position d'insertion de v.(j) et insert v.(j) à la position i en décalant les v.(i) ... v.(j-1) d'un cran vers la droite ;
- on définit enfin la fonction `tri_insertion` qui effectue le tri par insertion.