Informatique Option
Programmation impérative : boucles, références et tableaux

Sujet (l' Victor Nicollet 2006, 2007)

12 mars 2008

1 Références

Une référence est une valeur qui se comporte comme une flèche: elle pointe en permanence vers une autre valeur, et il est possible de la réorienter pour la faire pointer vers une autre valeur du même type.

Le code suivant crée une référence (pointant vers 2), change sa valeur (la réoriente vers 3), et renvoie sa valeur (l'objet vers lequel elle pointe):

```
# let x = ref 2;;
x : int ref = ref 2
# x := 3;;
- : unit = ()
# !x;;
- : int = 2
```

Une référence vers des valeurs de type `'a` aura pour type `'a ref`. Cela permet d'empêcher les erreurs où une valeur d'un type incorrect serait attribuée à une référence.

Question 1 *Que font les programmes suivants ?*

```
let f(x) =
incr x;
!x  in
f(ref 17);;
```

```
let a = ref 3 in
let b = a in
a := 4;
!b;;
```

Question 2 *Écrire les fonctions:*

- `get(x)` *qui prend en argument une référence* x *et renvoie son contenu.*
- `set(x)(v)` *qui prend en argument une référence* x *et la fait pointer vers* v*.*
- `clone(x)` *qui renvoie une référence pointant vers le même objet que* x*.*

Quel est leur type ?

2 Boucles

Il est souvent nécessaire d'effectuer plusieurs fois de suite une même opération (à effet de bord) dans un algorithme. Pour cela, on utilise des boucles: il en existe deux sortes en Caml Light, chacune étant destinée à un certain ensemble de situations.

2.1 Boucle `for`

La boucle `for` permet d'exécuter une série d'opérations un nombre prédéterminé de fois. Par exemple, pour afficher les entiers de *min* à *max* (inclus), on écrit:

```
for entier = min to max do
   print_int entier;
   print_newline ( )
done
```

Une telle boucle s'exécute une fois pour chaque valeur entière entre le minimum et le maximum. Le type d'une boucle est `unit`: il est possible de l'utiliser à gauche d'un point-virgule.

Question 3 *Afficher la table de multiplication (un carré* 10×10*).*

Il est possible d'utiliser une référence pour mémoriser une valeur calculée pendant l'exécution d'une boucle (puisque la boucle peut modifier la référence), et utiliser ensuite la valeur de la référence.

La boucle `for` s'avère très utile lorsqu'on cherche à calculer le n-ème terme d'une suite définie par récurrence, ou pour des opérations comme des sommes ou des produits d'un nombre prédéterminé de termes.

Question 4 *Écrire les fonctions qui calculent le* n*-ème terme des suites d'entiers suivantes:*

$$a_{n+1} = f(n, a_n) \quad a_0 = x$$

$$u_{n+1} = (n+1) \cdot u_n \quad u_0 = 1$$

$$w_{i+2} = w_{i+1} + w_i \quad w_0 = w_1 = 1$$

Où f est une fonction et x une valeur, qui seront des arguments de la fonction `a(f,x)(n)` *calculant le n-ème terme de* a_n.

On remarquera que certaines suites peuvent s'exprimer en fonction d'autres.

2.2 Boucle `while`

La boucle `while` permet d'exécuter une série d'opérations jusqu'à ce qu'une propriété devienne vraie. Par exemple, on peut écrire un programme qui divise un entier par deux jusqu'à ce qu'il soit nul:

```
let entier = ref (read_int ( )) in
while !entier <> 0 do
    print_int !entier; print_newline( );
    entier := !entier / 2
done
```

La boucle `while` est utile lorsqu'on souhaite compter des choses. Pour cela, on utilise en général une référence entière dont la valeur initiale est zéro, et on y ajoute un (avec `incr`) à chaque fois que la boucle est exécutée.

Question 5 *Soit la suite de Syracuse définie par son terme initial* $u_0 > 0$ *et la relation de récurrence:*

$$u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{sinon} \end{cases}$$

Écrire une fonction qui, étant donnée la valeur de u_0*, calcule la plus petite valeur de n pour laquelle* $u_n = 1$.

Plus généralement, on utilise une boucle `while` dès qu'il devient difficile d'exprimer le nombre de répétitions de la boucle avant de la commencer. En particulier, c'est le cas lorsque le nombre de répétitions est une conséquence directe du travail de la boucle.

Même si le nombre maximal d'opérations est connu à l'avance, on peut vouloir (pour des raisons de vitesse) arrêter l'exécution dès que le résultat est obtenu.

Question 6 *Déterminer si un entier* $n \geq 2$ *est premier*[1].

Vérifier qu'il y a 16342 nombres premiers inférieurs à 180000.

3 Tableaux

Les références ne permettent de conserver qu'une quantité fixe de données en mémoire. Lorsqu'on souhaite pouvoir manipuler des quantités plus grandes de données, on utilise des tableaux. Le code suivant crée un tableau contenant les entiers premiers inférieurs à 20:

```
[| 2; 3; 5; 7; 11; 13; 17; 19|];;
```

[1]Indication: n est premier ssi aucun entier $1 < m \leq \sqrt{n}$ ne divise n

Il est également possible de créer un tableau de taille arbitraire avec la fonction `make_vect`. Ci-dessous, un tableau de taille 10 dont tous les éléments sont le mot "Bonjour".

```
let bonjour = make_vect 10 "Bonjour";;
```

On peut enfin accéder, en lecture et en écriture, aux éléments d'un tableau. Dans un tableau de taille `n`, les élements sont numérotés de `0` inclus à `n` exclus. Le code ci-dessous lit donc le troisième élement, puis modifie le septième et affiche le tableau modifié.

```
bonjour.(2);;
bonjour.(6) < -- "Au revoir"; bonjour;;
```

Les tableaux ont chacun leur type. Un tableau d'objets `'a` sera un `'a vect`, et on ne peut y écrire des objets d'un type autre que `'a`. On peut connaître la longueur d'un tableau en lui appliquant la fonction `vect_length`.

Question 7 *Écrire une fonction qui prend en argument deux tableaux a et b de réels et calcule leur produit scalaire, à savoir (avec n la taille du plus petit)* $\sum_{i=0}^{n-1} a_i b_i$.

Lorsque le résultat recherché est un tableau, une boucle `for` est en général un très bon choix si la longueur du tableau est connue à l'avance.

Question 8 *Écrire une fonction qui prend en argument une taille n et un réel X, et qui construit un tableau dont le i-ème élément est* X^{i-1}.

4 Pour finir

4.1 Polynômes

On souhaite travailler sur des polynômes, qu'on choisit de représenter comme des tableaux de coefficients réels. Ainsi:

`[| 1.; -2.; 0.; 3.; 0. |]` représente $3X^3 - 2X + 1$

Plus généralement, si $n =$ `vect_length P` $- 1$,

$$P(X) = \sum_{i=0}^{n} \texttt{P.(i)} X^i$$

Question 9

Écrire une fonction `deg` *calculant le degré d'un polynôme (l'indice de son plus grand coefficient non nul). Pour simplifier, on convient que le degré d'un polynôme nul est* 0.

Écrire une fonction `eval` *qui calcule la valeur d'un polynôme pour une valeur de X donnée. Utiliser les questions 7 et 8.*

Construire le terme $h_n(X)$ *de la suite définie par:* $h_k(X) = a_{n-k} + X h_{k-1}$ *avec* $h_0(X) = a_n$. *Cette suite vérifie:* $h_n(X) = \sum_{i=0}^{n} a_i X^i$. *Utiliser cette propriété pour en déduire une fonction* `eval2` *plus rapide que* `eval`.

4.2 Cours de la bourse

On veut jouer en bourse et gagner de l'argent. On s'intéresse pour cela à une action en particulier. On connaît son prix sur une période donnée, au jour le jour. On souhaite en acheter une unité un jour a (en dépensant le prix qu'elle a atteint), puis la revendre un jour $v \geq a$ (en gagnant le prix de l'action le jour v). Le profit est, bien entendu, la différence entre le prix auquel on l'a vendu et le prix auquel on l'a acheté.

Question 10

Écrire une fonction capable de calculer le jour d'achat et le jour de vente qui permettent le profit maximal.

(Indication 1) En supposant qu'on veuille vendre à une date v donnée, trouver la date a à laquelle il faut acheter pour maximiser le profit.

(Indication 2) En connaissant le jour d'achat a optimal pour un jour de vente v, comment peut-on calculer le jour d'achat optimal pour le jour $v + 1$?

On supposera que les prix sont donnés comme un tableau d'entiers de taille supérieure ou égale à 1, dont le i-ème élément est le prix de l'action pour le jour i.

Créer un tableau de taille 20.000 contenant des entiers aléatoirement choisis et entre 0 et 150.000, et calculer les jours d'achat et de vente, ainsi que le profit obtenu.