ISIMA 1ère année

5 février 2008
Durée : 2 heures
Documents autorisés

PROGRAMMATION FONCTIONNELLE

- *Il est permis d'écrire des fonctions auxiliaires.*
- *Penser à expliquer le principe de vos fonctions lorsque celui-ci ne découle pas immédiatement de la lecture de celles-ci.*

Exercice 1 :

Soient les fonctions Scheme `un` et `deux` ci-dessous :

```
(define un (lambda (A B C)
   (if A
      (C (car L) (un (cdr A) B C))
       B ) ))
(Define deux (lambda (L Z)
   (un L 0 (lambda (X Y) (+ X (* Z Y)))) ))
```

Quelle est la valeur retournée par l'évaluation de l'expression `(deux '( 3 6 9) 5)` ?

Quelle est, en général, la valeur retournée par l'évaluation de l'expression `(deux L x)`, où `L` est une liste de nombres réels et `x` un nombre réel ?

Solution :

La valeur retournée par l'évaluation de l'expression `(deux '( 3 6 9) 5)` est $3 + 6 \times 5 + 9 \times 5^2$.

En général la valeur retournée par l'évaluation de l'expression `(deux L x)`, est la valeur en `x` d'un polynôme dont les coefficients sont donnés dans la liste `L` par degrés croissants. Cette évaluation de la valeur du polynôme en `x` est effectuée suivant un schéma de Horner.

Exercice 2 :

1°) Ecrire une fonction Scheme `ts2` ayant comme argument une liste `L` du type $(x_1, \ldots, x_n)$, et telle que l'évaluation de l'expression `(ts2 L)` retourne la liste des couples (x_i, x_j) avec $i < j$.

Par exemple, l'évaluation de l'expression `(ts2 '(3 4 5))` retourne la liste `((3 4) (3 5) (4 5))` et celle de l'expression `(ts2 '(2 3 4 5))` retourne la liste `((2 3) (2 4) (2 5) (3 4) (3 5) (4 5))`.

Solution :

```
(define ts2 (lambda (L)
   (if L (append (map (lambda (x) (list (car L) x)) (cdr L))
                 (ts2 (cdr L)) )
         () ) ))
```

2°) Ecrire une fonction Scheme `ts3` ayant comme argument une liste `L` du type $(x_1, \ldots, x_n)$, et telle que l'évaluation de l'expression `(ts3 L)` retourne la liste des triplets (x_i, x_j, x_k) avec $i < j < k$.

Par exemple, l'évaluation de l'expression `(ts3 '(2 3 4 5))` retourne la liste

`((2 3 4) (2 3 5) (2 4 5) (3 4 5))`

et celle de l'expression `(ts3 '(1 2 3 4 5))` retourne la liste

```
((1 2 3) (1 2 4) (1 2 5) (1 3 4) (1 3 5) (1 4 5)
         (2 3 4) (2 3 5) (2 4 5) (3 4 5)).
```

Solution :

```
(define ts3 (lambda (L)
   (if L (append (map (lambda (c) (cons (car L) c)) (ts2 (cdr L)))
                 (ts3 (cdr L)) )
         () ) ))
```

3°) Ecrire une fonction Scheme `ts*` ayant comme argument une liste `L` du type $(x_1, \ldots, x_n)$ et un entier `k`, et telle que l'évaluation de l'expression `(ts3 L k)` retourne la liste des `k`-listes $(x_{i_1}, \ldots, x_{i_k})$ avec $i_1 < \ldots < i_k$.

Solution :

```
(define ts* (lambda (L n)
   (if (= n 2) (ts2 L)
      (if L (append
               (map (lambda (n-1-uplet) (cons (car L) n-1-uplet))
                    (ts* (cdr L) (- n 1)) )
               (ts* (cdr L) n) )
            () ) ) ))
```

Exercice 3 : Crible d'Eratosthène

La méthode du crible d'Eratosthène permet de construire la liste des nombres premiers inférieurs ou égaux à un nombre entier *N*. Rappelons qu'un nombre entier est dit premier si et seulement s'il est supérieur ou égal à 2 et s'il n'est divisible que par 1 et par lui-même. Un entier *n* sera dit multiple non trivial de *m* si *n* est multiple de *m* et $n \neq m$.

Cette méthode consiste à supprimer successivement de la liste initiale (2 3 … *N*) tous les nombres pairs, puis tous les nombres multiples non triviaux de 3, de 5, de 7, etc. Par exemple pour $N = 30$, et en partant de la liste des nombres impairs compris entre 3 et 30, les listes obtenues à chaque étape sont données dans le tableau ci-dessous :

étape 0	3	5	7	9	11	13	15	17	19	21	23	25	27	29
étape 1	3	5	7		11	13		17	19		23	25		29
étape 2	3	5	7		11	13		17	19		23			29

Le principe est le suivant : si à une étape on a supprimé de la liste tous les multiples non triviaux d'un élément, 3 par exemple, alors à l'étape suivante, on supprime tous les multiples non triviaux de l'élément suivant (donc 5 avec l'exemple donné) dans la liste restante ; on remarquera que pour obtenir les nombres premiers entre 3 et 30, trois étapes suffisent, parce que comme $\sqrt{30} < 6$, il est certain que tous les multiples des nombres supérieurs ou égaux à 6 ont déjà été éliminés ; on remarquera également que si on devait par exemple trouver tous les nombres premiers entre 3 et 200, alors après avoir éliminé les multiples de 7 de la liste, ce seraient les multiples de 11 que l'on éliminerait, et non ceux de 9, puisque 9 aurait déjà été éliminé, en tant que multiple de 3.

On pourra, pour écrire les fonctions Scheme demandées ci-dessous, utiliser toutes les fonctions prédéfinies du langage Scheme qui seront jugées utiles, comme par exemple :

`integer?`, qui retourne vrai si et seulement si son argument est un nombre entier ;

`odd?`, qui retourne vrai si et seulement si son argument est un nombre entier impair ;

`sqrt`, qui retourne la racine carrée de son argument .

1°) Ecrire une fonction Scheme, `limp3`, ayant comme argument un entier `n` et telle que l'évaluation de l'expression `(limp3 n)` retourne la liste des entiers impairs compris entre 3 et `n`, en un temps de calcul qui soit linéaire en `n`.

Solution :

```
(define limp3 (lambda (n) (limp3a (imp n) ())))
(define imp (lambda (n) (if (odd? n) n (-n 1))))
(define limp3a (lambda (n L)
   (if (>= n 3) (limp3a (- n 2) (cons n L)) L) ))
```

2°) a) Que fait la fonction `f` suivante ?

```
(define f (lambda (L P)
   (append_map (if (P x) () (list x)) L) ))
```

Solution : Cette fonction élimine de la liste `L` les éléments qui satisfont le prédicat `P`.

b) Ecrire, en utilisant impérativement la fonction `f` ci-dessus, une fonction Scheme, `elmul`, ayant comme arguments un entier `k` et une liste d'entiers `L` et telle que l'évaluation de l'expression `(elmul k L)` retourne la liste obtenue à partir de `L` en éliminant tous les multiples de `k`.

Solution :

```
(define elmul (lambda (k L)
   (f L (lambda (x) (integer? (/ x k)))) ))
```

3°) Ecrire une fonction Scheme, `crible`, ayant comme argument un entier `n` et telle que l'évaluation de l'expression `(crible n)` retourne la liste des nombres premiers compris entre 2 et `n`.

Solution :

```
(define crible (lambda (n)
   (cond ((< n 2) ())
         ((= n 2) '(2))
         (else (cons 2 (lprem3 n (limp3 n)))) ) ))
(define lprem3 (lambda (n L)
   (if (> (car L) (sqrt n)) L
      (cons (car L) (lprem3 n (elmul (car L) (cdr L)))) ) ))
```

4°) On remarque que un entier *n* est premier si et seulement si aucun nombre premier entre 1 et la partie entière de $\sqrt{n}$ ne divise *n*.

En déduire, en utilisant impérativement la fonction `f` de la question 2°) a), une fonction Scheme, `premier`, ayant comme argument un entier `n`, et telle que l'évaluation de l'expression `(premier n)` retourne une valeur logique vraie si et seulement si `n` est un nombre premier.

Solution :

D'après la remarque ci-dessus, si on élimine du crible entre 1 et la partie entière de $\sqrt{n}$ les nombres premiers qui ne divise pas *n*, alors si *n* est premier, cette liste doit être vide. Pour effectuer cette élimination, on peut utiliser la fonction `f` qui élimine d'une liste tous les éléments qui satisfont un prédicat passé en argument (ici, ne pas diviser *n*).

```
(define premier (lambda (n)
   (null? (f (crible (sqrt n))
             (lambda (x) (not (integer ? (/ n x)))) )) ))
```