Programmation fonctionnelle

Répétition 3

1 mars 2007

Exercices simples pour s'échauffer

1.♣ (1/03/1998, Q2) Ecrire une fonction `erase-0` prenant comme argument une liste d'entiers et retournant cette liste privée de toutes les occurences du nombre 0.

`(erase-0 '(1 0 2 0 0 1 2 0))` ⇒ `(1 2 1 2)`

2. Définir la fonction `nombre-de` à deux arguments qui renvoie le nombre de fois que l'on a rencontré le premier argument dans la liste en second argument.

3. Définir la fonction `make-list` qui prend comme arguments un nombre `n` et une s-expression quelconque `x` et qui construit une liste composée de `n` fois l'élément `x`.

Correction d'exercices proposés

4. Ecrire une fonction `removeAll` à deux arguments `l` et `n` dont les valeurs sont respectivement une liste et un nombre entier, qui retourne la liste privée de toutes les occurences de `n`. Ainsi on a

`(removeAll '(2 7 1 7 3 1) 1)` ⇒ `(2 7 7 3)`

5.♣ (24/02/1997, Q2) Définir la fonction `suffix` à deux arguments `l1` et `l2` dont les valeurs sont des listes et qui retourne `#t` si `l1` est suffixe de `l2` et `#f` sinon. Ainsi on a :

`(suffix '(1 2) '(3 x 1 2))` ⇒ `#t`
`(suffix '(1 2) '(3 x 2))` ⇒ `#f`

Correction de l'interro du 17 novembre 2005

Question 1. Evaluer les expressions suivantes :

```
(if (> 4 7) (cond) 1)
(cons (list '(a b) '(c)) (append '(a b) '(c)))
(cons (list (append '(1 2) '(3 4)) 5) '(a b))
(append (list '(1 2) '(3 4)) (list a b))
```

```
((lambda (x) 'x) ((lambda (x y) (cons x y)) '(1 2) '(3 4)))
((lambda (first second) (car first)) '(second) 'first)

((lambda (a b c) (a b c))
  (lambda (x y) (+ 3 x y))
  ((lambda (x y) (* x y)) (+ 2 4) 7)
  ((lambda (z) (+ 5 z)) 0))

(define f ((lambda (x y) (x y)) (lambda (x) x) (lambda (y) y)))
(f 0)
```

Question 2. Définir la fonction *alternate* qui prend comme argument une liste et qui renvoie la liste où les éléments de rang pair ont été permutés avec les éléments de rang impair ; c'est-à-dire que l'on permute le premier et le deuxième élément, le troisième et le quatrième, et ainsi de suite.

```
(alternate '(a b c d e f))  ==>  (b a d c f e)
(alternate '(a b c d e f g))  ==>  (b a d c f e g)
```

Question 3. Définir la fonction *f* qui prend comme argument une liste de symboles et qui renvoie la liste où toutes les occurences successives d'un symbole sont remplacées par le symbole suivi du nombre d'occurrences consécutives.

```
(f '(a a a b b c a a a c c))  ==>  (a 3 b 2 c 1 a 3 c 2)
```

Question 4. Spécifier la fonction suivante :

```
(define xxx
  (lambda (s l)
    (cond ((null? l) '())
          ((null? (cdr l)) (if (eq? s (car l))
                               (list '())
                               '()))
          ((eq? (car l) s) (cons (cadr l)
                                 (xxx s (cdr l))))
          (else (xxx s (cdr l))))))
```

Quelle est la valeur de `(xxx 'b '(a b c b d e))` ?
Quelle est la valeur de `(xxx 'b '(a b c b d b))` ?
Quelle est la valeur de `(xxx 'b '(a b b d b e))` ?

Exercices proposés

Etant donné que quasiment personne ne l'avait fait la semaine dernière, le revoilà :

6. Ecrire une fonction `frequency` prenant en argument une liste d'atomes `ls` et renvoyant une table d'apparition de chacun des atomes dans la liste `ls`. Cette table sera représentée par une liste de paires pointées, dont le car est un atome et le cdr la fréquence d'apparition de cet atome. Tous les atomes de `ls` apparaissent une et une seule fois dans la table.

`(frequency '(a b c b a b d a c b b))` $\Rightarrow$ `((a . 3) (b . 5) (c . 2) (d . 1))`

Exercice sur la mémorisation des résultats intermédiaires :

7.* Soit la fonction `f` definie comme suit : si $n < 3$, alors $f(n) = n$. Sinon, $f(n) = f(n-1)+f(n-2)+f(n-3)$. Ecrire un code SCHEME qui implémente efficacement cette fonction.