Programmation fonctionnelle

Répétition 2

22 février 2007

Petit conseil de programmation

Sur les machines du réseau, vous avez sans doute remarqué que les retours en arrière sont foireux. Il y a deux manières de résoudre le problème :

- Travailler dans Emacs. Pour cela il faut faire "ALT-x" puis "run-scheme".
- Travailler dans un fichier quelconque puis le charger à l'aide de `(load "nom_fichier")`

Quelques évaluations pour s'échauffer...

```
(cons '(1 2 3) '(4 5))
(append '(1 2 3) '(4 5))
(list '(1 2 3) '(4 5))
(list? (cons 'a 'b))
(list? (cons 'a (cons 'b '())))
(list? (cons (cons 'b '()) 'a))
(list? (cons (cons 'b '()) (cons 'b '())))
(null? 'a)
(null? (car '(a)))
(null? (cdr '(a)))
(number? 1)
(number? '1)
(number? #t)
(number? 'a)
(number? a)
(number? (+ 3 a))
(boolean? 3)
(boolean? #t)
(boolean? #f)
(boolean? '#t)
(boolean? 'd)
(boolean? d)
(boolean? '())
(null? #t)
(null? '())
(null? '(a b))
```

```
(null? (car '(())))
(null? (car '((()))))
(null? #f)
(symbol? a)
(symbol? 'b)
(symbol? (car '(a b c)))
(symbol? (cons '() '()))
(symbol? #f)
(equal? 'a (car '(a b)))
(equal? '(a b c) '(a b c))
(equal? '(a (b c)) '(a b c))
(equal? (cdr '(a c d)) (cdr '(b c d)))
(equal? '(car '((b) c)) (cdr '(a b)))
(lambda (y x) (cons x y))
((lambda (y x) (cons x y)) '() 'a)
(define id (lambda (x) x))
(id 1)
(id '(1 2 3))
(id id)
((id id) (id id))
(((id id) (id id)) 3)
```

Corrections des exercices plus complexes faits chez vous

1. La variable `phrase` est définie comme suit

```
(define phrase
  '(((e) x (e r (c (i c e) c (o m (p l (i q))) u e)))))
```

Extraire le `m` au moyen de `car`, `cdr`, `cadr`, ...

2. Construire la liste `(s c h e m e)` au seul moyen de la liste `ls` définie comme suit

```
(define ls '(C E H M S))
```

et des fonctions `cons`, `car`, `cdr`, `cadr`, ...

Exercices de compréhension de la structure interne

3. `Représentation interne des listes`
Dessiner l'arbre correspondant à la représentation interne de la liste

(a b c)

4. `Différence entre cons, list et append`
Illustrer graphiquement les opérations suivantes :

- Ajout de l'élement `a` au début de la liste (4 5)
  → (cons 'a '(4 5))

- Création d'une liste à deux éléments, respectivement `a` et la liste (4 5)
  → (list 'a '(4 5))

- Concatenation des listes (1 2 3) et (4 5)
  → (append '(1 2 3) '(4 5))

- Ajout de la liste (1 2 3) au début de la liste (4 5)
  → (cons '(1 2 3) '(4 5))

- Création d'une paire pointée dont les membres sont a et 2
  → (cons 'a 2)

Notions d'efficacité

5. Définir la fonction `length-list` qui renvoie la longueur de la liste passée en argument.

PS : la fonction *length* est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

6. Définir la fonction `reverse-list` qui prend en argument une liste *ls* et renvoie une liste contenant les éléments de *ls* dans l'ordre inverse.

PS : la fonction *reverse* est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

Exercices de base sur les listes

7.♣ (3/3/1999, Q4) Ecrire une fonction `big` qui prend comme arguments un nombre entier `n` et une liste `l` de nombres entiers, telle que `(big n l)` est la liste des éléments de `l` plus grands que `n`.

`(big 5 '(7 -3 6 1 -2 5 6))` ⇒ `(7 6 6)`

8. Définir la fonction `min` qui renvoie le minimum de la liste non vide de nombres donnée en argument.

Exercices proposés

9. 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)`

10.♣ (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`

11. 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))` ⇒ `((a . 3) (b . 5) (c . 2) (d . 1))`