Programmation fonctionnelle

Répétition 2

19 février 2015

Encore quelques évaluations

Exercice 1.

```
(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)
(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)
```

Remarque : La valeur renvoyée par `(define ...)` n'est pas spécifiée.
Remarque : Toute liste est une paire pointée, mais le contraire n'est pas vrai.

Correction exercices proposés

Exercice 2.
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`, ...

Exercice 3.
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`, ...

Exercice 4.
Définir la fonction `min` qui renvoie le minimum de la liste non vide de nombres donnée en argument.

Exercice 5.
Spécifier la fonction suivante :

```
(define xxx
  (lambda (u)
    (if (null? u)
        0
        (if (> (car u) 0)
            (+ (car u) (xxx (cdr u)))
            (xxx (cdr u))))))
```

Compréhension de la structure interne

Exercice 6.
Représentation interne des listes :
Dessiner l'arbre correspondant à la représentation interne de la liste

(a b c)

Exercice 7.
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

Exercice 8.
Définir la fonction `length-list` qui renvoie la longueur de la liste passée en argument.

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

Exercice 9.
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.

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

Exercices proposés

Exercice 10.
Écrire 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)
```

Exercice 11.
Écrire 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 occurrences de `n`.

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

Exercice 12.
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.

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