Ecole **N**ationale des **S**ciences A.U. 11/12
de l'**I**nformatique
***

Correction Programmation Fonctionnelle
Devoir Surveillé

Classe : II2 Filières ISID/ILSI	Date : 13/03/2012
Enseignantes : W. LEJOUAD-CHAARI, A. BEN HASSINE, I. ALAYA	Durée : 2h
Documents : Non	Nombre de pages : 2

Exercice 1 (2.5pts)

Evaluer les expressions suivantes :
1. (**cons** '(1 2) '(4))
2. (**cons** '(1 2) 4)
3. (**car** '(((1) 2) 3 4))
4. (**append** '(1 2) nil '(4))
5. (**list** '(1 2) '(4))
6. (**list** '(1 2) 4)
7. (**list** '(1 2) nil '(4))
8. (**mapcar** (lambda (x) (- 10 x)) '(1 3 5 7 9))
9. (**mapcar** (lambda (x y) (+ x y)) '(1 2 3) '(4 5 6))
10. (**mapcar** (lambda (x) (if (evenp x) 0 x)) '(1 2 3 4 5))

---

1. ((1 2) 4)
2. ((1 2). 4) – paire pointée
3. ((1) 2)
4. (1 2 4)
5. ((1 2) (4))
6. ((1 2) 4)
7. ((1 2) () (4))
8. (9 7 5 3 1)
9. (5 7 9)
10. (1 0 3 0 5)

Exercice 2 (4pts)

Ecrire en Le_Lisp la fonction récursive **strict-doublons** qui prend une liste **l** en argument et retourne une nouvelle liste qui contient tous les éléments qui figurent exactement deux fois dans **l**. Par exemple :

? **(strict-doublons '(1 2 3 1 4 2 5 1 4))**
= (2 4)

```
; Fonction qui retourne le nombre d'occurrences d'un élément dans une liste
(de  nbocc (e l)
    (cond
      ((null  l)   0)
      ((eq  e  (car l))  (+  1  (nbocc   e  (cdr l))))
      (t                 (nbocc e (cdr  l)))))
; Fonction qui supprime toutes les occurrences d'un élément dans une liste
(de omettre (e  l)
    (cond
      ((null  l)  ())
      ((eq  e  (car l))  (omettre  e  (cdr l)))
      (t                 (cons (car l) (omettre  e  (cdr  l))))))

(de  strict-doublons (l)
    (cond
     ((null   l)   ())
     ((>  (nbocc (car  l)  l)  2)  (strict-doublons  (omettre  (car  l)  l)))
     ((=  (nbocc (car  l)  l)  2)  (cons (car l) (strict-doublons (cdr  l))))
     ((strict-doublons  (cdr  l)))))
```

Exercice 3 (5.5pts)

On représente un *polynôme* par une liste de couples (coefficient puissance). Par exemple, le polynôme $1 + x + 3x^2 + 2x^5$ est représenté par la liste ((1 0) (1 1) (3 2) (2 5)).

En utilisant **mapcar et lambda**, écrire une fonction **(derive l)** calculant le polynôme dérivé (on fera attention au cas d'un polynôme constant). Dans l'exemple précédent, le polynôme dérivé est $1 + 6x + 10x^4$.

```
(de derive (l)
    (cdr (mapcar (lambda (x) (list (* (car x) (cadr x)) (1-  (cadr  x)))) l)))

; Une autre version récursive sans Lambda et Mapcar    (non demandée)
; Le calcul nécessaire dans un couple de valeurs
(de  op  (cl)
    (list  (*  (car  cl) (cadr  cl))  (1-  (cadr  cl))))

(de  derive (l)
    (cond
      ((null  l)  ())
      ((=  (cadar  l)  0)   (derive  (cdr l)))
      (t     (cons (op (car l))  (derive  (cdr l))))))
```

Exercice 4 (4pts)

Ecrire en Le_Lisp la fonction **split** qui prend comme arguments une liste **l** et un entier **n** et retourne une liste de deux listes : la première contient les **n** premiers éléments de la liste, et la deuxième, le reste de la liste. Par exemple :

? (split '(1 2 3 4 5 6) 4)
= ((1 2 3 4) (5 6))

```
(de split (l n)
   (cond
        ((null l) nil)
        (list (npremier l n) (dernier l n))
))
; npremier est une fonction qui retourne les n premiers éléments d'une liste
(de npremier (l n)
   (cond
        ((null l) nil)
        ((= n 0) nil)
        (t (cons (car l) (npremier (cdr l) (- n 1))))
; dernier est une fonction qui retourne la liste sans les n premiers éléments
(de dernier (l n)
   (cond
        ((null l) nil)
        ((= n 0) l)
        (t  (dernier (cdr l) (- n 1))))

; une autre version
(de split (l n)
   (cond
     ((null l) nil)                                  ; cas spécifique
     ((> n (length l)) (list l '()))                 ; cas spécifique
     ((= n 0) (list '() l))                          ; condition d'arrêt récursivité
     (t (append (list (cons (car l) (car (split (cdr l) (- n 1))))) (cdr (split (cdr l) (- n 1)))))
                                                     ; appel récursif
))
```

Ou autrement, pour éviter de refaire appel au même appel récursif :

```
(de split (l n)
    (cond
      ((null l) nil)                                 ; cas spécifique
      ((> n (length l)) (list l '()))                ; cas spécifique
      ((= n 0) (list '() l))                         ; condition d'arrêt récursivité
      (t (let ((aux (split (cdr l) (- n 1)))) (append (list (cons (car l) (car aux))) (cdr aux))))
                                                     ; appel récursif
))
```

Exercice 5 (4pts)

Ecrire une fonction **(remove-if pred l)**. Cette fonction reconstruit la liste **l** sans les éléments qui vérifient **pred**.

? **(remove-if (lambda (x) (> x 0)) '(2 -5 6 7 9 12 -9))**
= (-5 -9)

```
(de remove-if (pred  l)
   (cond
      ((null l)  nil)
      ((funcall pred (car l)) (remove-if pred (cdr l)))
      (t  (cons (car l) (remove-if pred (cdr l))))
))
```