Université Pierre et Marie Curie
DEUG SPI seconde année
Option informatique

Année 2001-2002
**Examen juin**
13 juin 2002
2 heures

**Remarques :**

- *Aucun document ni machine électronique n'est permis à l'exception de la carte de référence de Scheme.*
- *Les questions peuvent être résolues de façon indépendante. Il est possible, voire même utile, pour répondre à une question, d'utiliser les fonctions qui sont l'objet des questions précédentes.*
- *La clarté des réponses et la présentation des programmes seront appréciées. Toutes les fonctions apparaissant dans les réponses doivent être accompagnées de leur spécification.*
- *Le barême (total sur 20) n'est donné qu'à titre indicatif.*

**Question 1** (1 point) :
Écrire une spécification de la fonction `premier-dans-reste` qui a comme définition Scheme :

```
(define (premier-dans-reste L)
  (member (car L) (cdr L)))
```

Donner aussi des exemples d'applications de cette fonction (avec les valeurs renvoyées).

**Question 2** (2 points) :
Écrire une définition Scheme de la fonction `existe-doublon?` qui a comme spécification :

`;;;` *existe-doublon? : LISTE[alpha] -> bool*
`;;;` *(existe-doublon? L) renvoie #t ssi il existe deux éléments de la liste "L" qui sont égaux.*
`;;;` *Par exemple, (existe-doublon? '(3 5 6 7 8 6)) renvoie #t (car 6 est un doublon)*
`;;;` *et (existe-doublon? '(3 5 6 7 8)) renvoie #f.*

**Question 3** (1 point) :

Écrire une définition Scheme de la fonction `cons-L` qui a comme spécification :

```
;;; cons-L : alpha * LISTE[LISTE[alpha]] -> LISTE[LISTE[alpha]]
;;; (cons-L e L) renvoie la liste obtenue en ajoutant la liste qui
;;; contient (uniquement) l'élément "e" devant la liste (de listes) "L".
;;; Par exemple, (cons-L 1 '((2 3) (4))) renvoie ((1) (2 3) (4))
```

**Question 4** (3 points) :

Écrire une définition Scheme de la fonction `add-prefixe` qui a comme spécification :

```
;;; add-prefixe : alpha * LISTE[LISTE[alpha]] -> LISTE[LISTE[alpha]]
;;; (add-prefixe p L) renvoie la liste (de listes) obtenue en ajoutant l'élément "p"
;;; devant chaque élément de la liste (de listes) "L".
;;; Par exemple (add-prefixe 3 '((4 5) (6 7 8) ())) renvoie ((3 4 5) (3 6 7 8) (3))
```

La grammaire suivante décrit un nœud (d'un système de fichier par exemple) :

| | | | |
|---|---|---|---|
| *<noeud>* | → | ( *<nom>* *<contenu>* ) | RÉPERTOIRE |
| | | ( *<nom>* *<taille>* ) | FICHIER |

*<contenu>* → *<noeud>*$^\star$

*<nom>* → `un symbole Scheme`

*<taille>* → `un entier naturel`

Un nœud possède un *nom* et il peut être
- un *répertoire* qui a un *contenu* constitué par un nombre quelconque (éventuellement 0) de nœuds
- ou un *fichier* qui a une certaine *taille*

Ainsi, le nœud `(a (b 5) (c (d 7) (e 9)))`, que l'on peut écrire

```
(a (b 5)
   (c (d 7)
      (e 9)))
```

pour être plus lisible, est un répertoire qui a pour nom *a* et qui contient un fichier de nom *b* et de taille *5* et un répertoire de nom *c*, ce dernier contenant deux fichiers de noms respectifs *d* et *e* et de tailles respectives *7* et *9*.

La barrière syntaxique est constituée des quatre fonctions :

```
;;; fichier? : Noeud -> bool
;;; nom-noeud : Noeud -> Symbole
;;; taille-fichier : Fichier -> nat
;;; contenu-repertoire : Repertoire -> LISTE[Noeud]
```

**Question 5** (3 points) :

Donner une implantation de ces quatre fonctions.

**Question 6** (3 points) :

Écrire une définition Scheme de la fonction `taille-noeud` qui a comme spécification :

**;;;** *taille-noeud : Noeud -> nat*
**;;;** *(taille-noeud noeud) renvoie la somme des tailles des fichiers présents dans le noeud donné.*
**;;;** *Par exemple, (taille-noeud '(a (b 5) (c (d 7) (e 9)))) renvoie 21.*

Pour être nommés correctement, les différents nœuds contenus dans un répertoire doivent avoir des noms différents. Par exemple, les noms des différents nœuds contenus dans le répertoire `(a (b ...) (c ...))` sont corrects alors que les noms des différents nœuds contenus dans le répertoire `(a (b ...) (c ...) (b ...))` ne sont pas corrects (car `b` est présent deux fois).

Les noms présents dans un nœud sont valides lorsqu'il en est ainsi pour tous les répertoires présents dans ce nœud.

**Question 7** (3 points) :

Écrire une définition Scheme de la fonction `noms-valide?` qui a comme spécification :

```
;;; noms-valide? : Noeud -> bool
;;; (noms-valide? noeud) renvoie #t ssi il n'existe pas de répertoire présent dans
;;; le noeud donné qui contient deux noeuds ayant le même nom.
```

Exemples :

```
(noms-valide? '(a (b 5)
                  (c (d 7)
                     (b 9))
                  (b 6)))  →  #f

(noms-valide? '(a (b (d 7)
                     (b 9))))  →  #t

(noms-valide? '(a (b (d 7))
                  (c (d 8))))  →  #t
```

**Définition** : une liste $(s_0 \ \dots \ s_i \ s_{i+1} \ \dots \ s_p)$ non vide de symboles est un *chemin* d'un nœud $n$ – et nous dirons que ce chemin aboutit à un nœud de nom $s_p$ – lorsqu'il existe une suite $(n_0, \dots n_i, n_{i+1}, \dots n_p)$ de nœuds telle que :

- $n_0$ est le nœud donné $n$,
- pour tout $i$ strictement inférieur à $p$, $n_i$ est un répertoire qui contient le nœud $n_{i+1}$,
- pour tout $i$ compris entre 0 et $p$ (inclus), $s_i$ est le nom du nœud $n_i$.

**Remarque** : les nœuds $n_0, \dots n_i, n_{i+1}, \dots n_{p-1}$ sont obligatoirement des répertoires, $n_p$ pouvant être un répertoire ou un fichier.

**Question 8** (4 points) :

Écrire une définition Scheme de la fonction `est-chemin?` qui a comme spécification :

```
;;; est-chemin? : LISTE[Symbole]/non vide/ * Noeud -> bool
;;; ERREUR lorsque le chemin donné est vide
;;; HYPOTHESE : "(noms-valide? noeud)" est vrai
;;; (est-chemin? chemin noeud) renvoie #t ssi "chemin" est un chemin du noeud "noeud"
```

Exemples :

```
(let ((le-noeud '(a (b 5)
                    (c (d 7)
                       (b 9))
                    (d 6))))
  (and (est-chemin? '(a c) le-noeud)
       (est-chemin? '(a c b) le-noeud)
       (not (est-chemin? '(a b c) le-noeud))
       (not (est-chemin? '(a e) le-noeud))))  →  #t
```

Utilisez-vous l'hypothèse ? Si oui, en quoi est-ce intéressant ?

## Question facultative (et difficile) :

Nous voudrions enfin savoir si un symbole donné est le nom d'un nœud présent dans un nœud donné. Plus précisément, étant donné un nœud et un symbole, nous voudrions donner la liste des chemins du nœud donné qui aboutissent à des nœuds ayant pour nom le symbole donné. Par exemple, en nommant `cherche` cette fonction :

```
(cherche 'b '(a (b (c 7)
                (b 8))
             (c (d 7)
                (b 9))))  →  ((a b) (a b b) (a c b))
```

**Question 9** (4 points) :

Écrire une spécification et une définition Scheme de la fonction `cherche`.