Université Pierre et Marie Curie — Année 2002-2003
DEUG MIAS première année — **Devoir sur table novembre**
Informatique-1 — 19 novembre 2003
1,5 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.
- La note tiendra compte de l'efficacité de vos programmes.
- Le barème (total sur 26) n'est donné qu'à titre indicatif.

Exercice I (questions de cours)

Question 1 (2 points) :

Écrire une spécification et une définition de la fonction `factorielle` qui, étant donné un entier naturel *n*, retourne la factorielle de *n*.

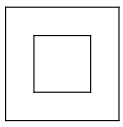
Question 2 (2 points) :

Écrire une spécification et une définition de la fonction `produit-liste` qui, étant donnée une liste *L* de nombres retourne le produit des éléments de *L*. Par convention, le produit des éléments d'une liste vide est égal à 1.

Exercice II (récursion sur les nombres)

On dispose d'une fonction `dessin-carre` de spécification :

```
;;;dessin-carre : Nombre/compris entre 0 et 2/ -> Image
;;;(dessin-carre c) retourne le pourtour d'un carré centré en (0,0) et de
;;;côté c
```

Par exemple : `(dessin-carre 1)` → 

Et on considère la fonction `mystere` définie par :

```
(define (mystere n c)
  (if (= n 0)
      (image-vide)
      (overlay (dessin-carre c)
               (mystere (- n 1) (* c 0.5)))))
```

Question 1 (1 point) :

Donner les applications successives de `mystere` (appels récursifs) qui sont effectuées lors de l'évaluation de `(mystere 3 2)`.

Question 2 (1 point) :

Dessiner le résultat de l'évaluation de `(mystere 3 2)`.

Question 3 (2 points) :

Donner la spécification de la fonction `mystere`.

Exercice III (récursion sur les listes)

Cet exercice est découpé en parties mais il forme un tout ; les titres des différentes parties font référence aux connaissances requises pour traiter les questions.

Partie 1 : exercices simples sur les listes

Question 1 (2 points) :

Écrire une spécification et une définition de la fonction `au-moins-2?` qui, étant donnée une liste L d'éléments, retourne vrai si la liste L a au moins deux éléments et faux sinon. Attention à l'efficacité !

Question 2 (2 points) :

Écrire une spécification et une définition de la fonction `ajout-1-au-car` qui, étant donnée une liste non vide L d'entiers naturels, retourne la liste obtenue en ajoutant 1 au premier élément de la liste L. Par exemple :

`(ajout-1-au-car (list 2 5 1 4))` → `(3 5 1 4)`

Question 3 (2 points) :

Écrire une spécification et une définition de la fonction `initialisation` qui, étant donnés un entier p et une liste L d'entiers naturels, retourne la liste formée de l'entier 1, de l'entier p et de tous les éléments de la liste L. Par exemple :

`(initialisation 5 (list 4 3 7 9))` → `(1 5 4 3 7 9)`

Partie 2 : Récursion sur les nombres

Voici une suite de listes :

la liste 1 est la liste `(1)`
la liste 2 est la liste `(1 1)`
la liste 3 est la liste `(2 1)`
la liste 4 est la liste `(1 2 1 1)`
la liste 5 est la liste `(1 1 1 2 2 1)`
la liste 6 est la liste `(3 1 2 2 1 1)`

D'après vous, quelle pourrait-être la liste 7 qui complèterait cette suite ? Bien souvent, cette question embarrasse davantage les mathématiciens que les petits enfants, car la réponse est très simple : il suffit de savoir compter sur ses doigts. Pour passer d'une liste à la suivante, on remplace toutes les occurrences consécutives d'un élément *x* par le nombre d'occurrences consécutives de *x* suivi de l'élément *x*. Ainsi :

- la première liste est `(1)`, elle est formée de 1 *chiffre 1*, la deuxième liste est donc `(1 1)`
- cette deuxième liste, `(1 1)`, est formée de 2 *chiffres 1*, la troisième liste est donc `(2 1)`
- cette troisième liste, `(2 1)` est formée de 1 *chiffre 2* et 1 *chiffre 1*, la quatrième liste est donc `(1 2 1 1)`
- cette quatrième liste, `(1 2 1 1)`, est formée de 1 *chiffre 1*, 1 *chiffre 2* et 2 *chiffres 1*, la cinquième liste est donc `(1 1 1 2 2 1)`
- cette cinquième liste, `(1 1 1 2 2 1)`, est formée de 3 *chiffres 1*, 2 *chiffres 2* et 1 *chiffre 1*, la sixième liste est donc `(3 1 2 2 1 1)`
- cette sixième liste, `(3 1 2 2 1 1)`, est formée de 1 *chiffre 3*, 1 *chiffre 1*, 2 *chiffres 2* et 2 *chiffres 1*
- la liste suivante sera donc `(1 3 1 1 2 2 2 1)`.

On appellera *liste naïve de rang n* la n-ième liste obtenue par ce procédé. La liste naïve de rang 1 est `(1)`... la liste naïve de rang 6 est `(3 1 2 2 1 1)`.

Question 4 (1 point) :

Quelle est la liste naïve de rang 8 ?

On suppose que l'on dispose d'une fonction `comptage` de spécification :

```
;;;comptage : LISTE[nat]/non vide/ -> LISTE[nat]
;;;(comptage L) retourne la liste obtenue en remplaçant toutes les occurrences
;;;consécutives d'un élément p de la liste L par le nombre d'occurrences
;;;consécutives de l'élément p dans L suivi de l'élément p lui-même.
```

Par exemple :

```
(comptage (list 1 1 1 1)) → (4 1)
(comptage (list 1 1 1 1 2 2 2)) → (4 1 3 2)
(comptage (list 1 1 1 1 2 2 1)) → (4 1 2 2 1 1)
```

Question 5 (3 points) :

Écrire une spécification et une définition de la fonction `liste-naive` qui, étant donné un entier *n* strictement positif, retourne la liste naïve de rang *n*. Par exemple : `(liste-naive 6)` →`(3 1 2 2 1 1)`

Partie 3 : récursion sur les listes

Cette partie a pour objet de définir la fonction `comptage`.

Question 6 (2 points) :

Supposons que *L* soit une liste ayant au moins deux éléments ; appelons *cdrCompte* la liste obtenue en appliquant la fonction `comptage` à la liste `(cdr L)`.

a) Supposons que les deux premiers éléments de la liste *L* ne soient pas égaux ; après avoir illustré votre réponse au moyen d'un exemple, exprimer `(comptage L)` en fonction de *cdrCompte*.

b) Même question lorsque les deux premiers éléments de la liste *L* sont égaux.

Question 7 (3 points) :

Écrire une spécification et une définition de la fonction `comptage`.

Partie 4 : liste de listes

Question 8 (3 points) :

Écrire une spécification et une définition de la fonction `listes-naives` qui, étant donné un entier *n* strictement positif, retourne la liste formée des *n* premières listes naïves. Par exemple :

```
(listes-naives 9) ->
((3 1 1 3 1 2 1 1 1 3 1 2 2 1)
 (1 1 1 3 2 1 3 2 1 1)
 (1 3 1 1 2 2 2 1)
 (3 1 2 2 1 1)
 (1 1 1 2 2 1)
 (1 2 1 1)
 (2 1)
 (1 1)
 (1))
```

Partie 5 : Pour aller plus loin (hors devoir)...

Attention : cette partie est à faire uniquement s'il vous reste du temps. Cette question ne vous rapportera aucun point, vous n'en tirerez que la satisfaction de l'avoir traitée.

Question 9 (0 points) :

Spécifier et définir la fonction `comptage-inverse`, réciproque de la fonction `comptage`, c'est-à-dire la fonction qui, étant donnée une liste *C*, retourne la liste *L* telle que `C=(comptage L)`.