Université Pierre et Marie Curie
DEUG MIAS première année
Informatique-1

Année 2003-2004
Devoir sur table
19 novembre 2003
1 heure 30

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.
- La note tiendra compte de l'efficacité de vos programmes.
- Le barème (total sur 46, la note étant considérée comme une note sur 40) n'est donné qu'à titre indicatif.

Exercice I (question de cours)

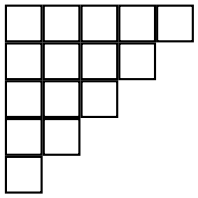
Question 1 (1 point) :

Écrire une spécification et une définition de la fonction `carre` qui étant donné un nombre x rend le carré de x.

Question 2 (2 points) :

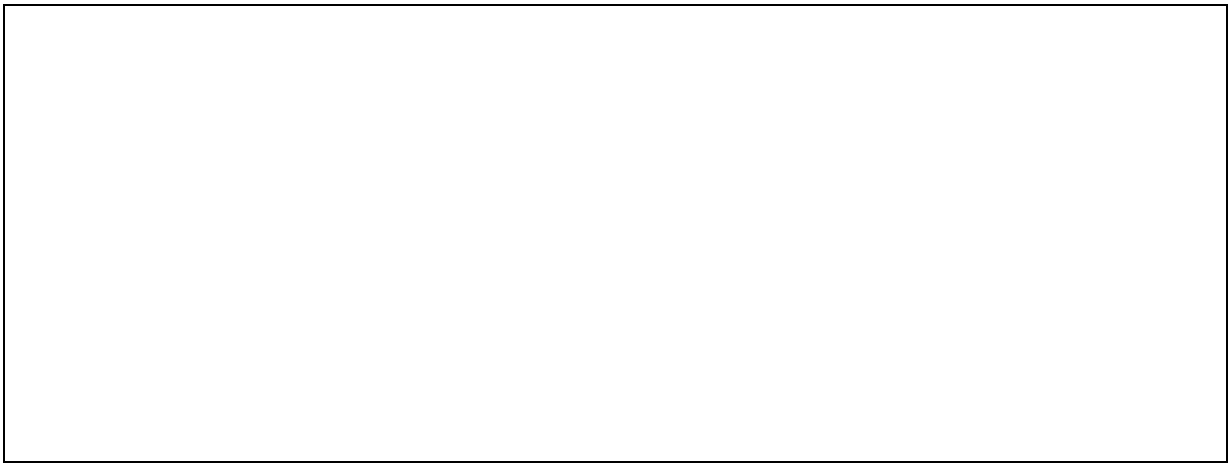
Écrire la signature et une définition de la fonction `somme-carres` qui, étant donnée une liste L de nombres, rend la somme des carrés des éléments de L.

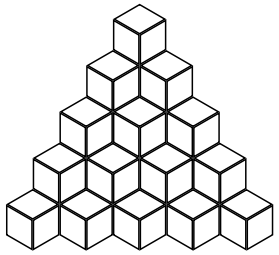
Exercice II (récursion sur les entiers)



Un « triangle de Pascal d'ordre n » est une surface en escalier constituée d'un premier rectangle de longueur n, d'un second rectangle de longueur $n-1$... et d'un dernier rectangle de longueur un, tous ces rectangles étant de largeur un. Par exemple, le dessin ci-contre est le « triangle de Pascal d'ordre 5 » et son aire est 15.
Noter qu'un « triangle de Pascal d'ordre n » est constitué par un rectangle de longueur n et de largeur un – son aire est donc n – (la première ligne dans le dessin ci-contre) et par un « triangle de Pascal d'ordre $n-1$ » (les autres lignes dans le dessin ci-contre).

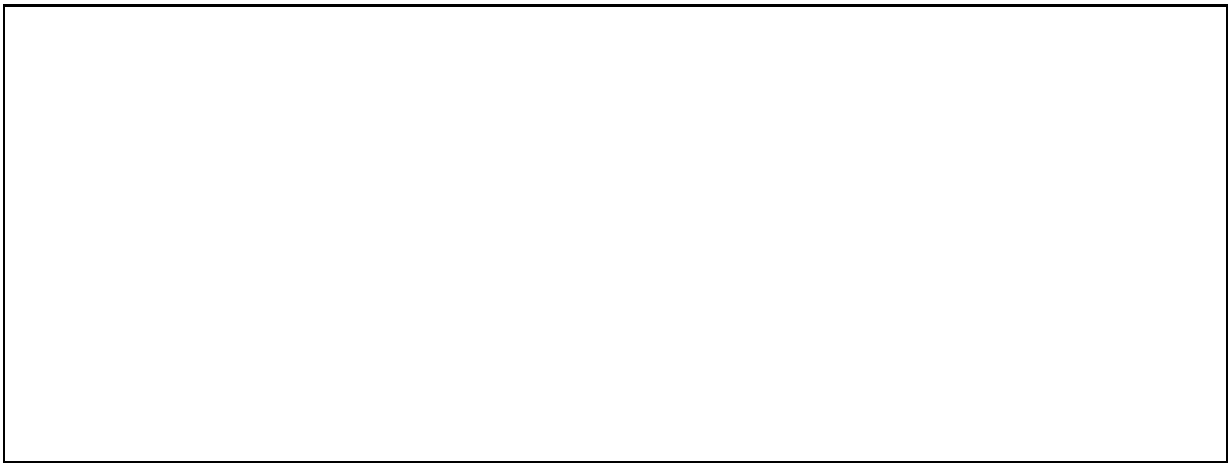
Question 1 (4 points) :
Écrire la signature et une définition récursive de la fonction `aire-Pascal` qui, étant donné un entier strictement positif n rend l'aire de la surface du « triangle de Pascal d'ordre n ».





Une « pyramide de Pascal d'ordre n » est un volume en escalier, chaque marche de l'escalier étant de hauteur un – son volume est donc égal à sa surface –, dont le premier niveau est un « triangle de Pascal d'ordre n », le deuxième niveau est un « triangle de Pascal d'ordre $n-1$ »,... le dernier niveau étant un cube dont les côtés ont une longueur égale à un. Par exemple, le dessin ci-contre représente une « pyramide de Pascal d'ordre 5 » dont le volume est 35.

Question 2 (4 points) :
Écrire la signature et une définition de la fonction `volume-Pascal` qui, étant donné un entier strictement positif n rend le volume de la « pyramide de Pascal d'ordre n ».



Exercice III (récursion sur les entiers)

Question 1 (4 points) :

Écrire la signature et une définition de la fonction `premiers-entiers-decroissante` qui, étant donné un entier naturel *n* rend la liste des entiers compris entre 1 et *n*, les éléments de la liste étant rangés en ordre décroissant. Par exemple :

```
(premiers-entiers-decroissante 7)  → (7 6 5 4 3 2 1)
(premiers-entiers-decroissante 0)  → ()
```

Nous voudrions maintenant écrire, sans utiliser la fonction `premiers-entiers-decroissante` et la fonction `reverse`, une définition de la fonction `premiers-entiers-croissante` qui, étant donné un entier naturel *n*, rend la liste des entiers compris entre 1 et *n*, les éléments de la liste étant rangés en ordre croissant. Par exemple :

```
(premiers-entiers-croissante 7)  → (1 2 3 4 5 6 7)
```

Pour cela, nous définirons la fonction `suite-entiers` dont la spécification est :

*;;; suite-entiers : nat * nat -> LISTE[nat]*
;;; HYPOTHESE : p <= n
;;; (suite-entiers p n) rend la liste des entiers de p à n.

Par exemple :

```
(suite-entiers 7 7)  → (7)
(suite-entiers 3 7)  → (3 4 5 6 7)
```

Question 2 (2 points) :

En utilisant la fonction `suite-entiers`, donner une définition de `premiers-entiers-croissante`.

Question 3 (3 points) :

Donner une définition de la fonction `suite-entiers`.

Exercice IV (question de cours)

Question 1 (4 points) :

Écrire la signature et une définition, **la plus efficace possible**, de la fonction `somme-cumulee` qui, étant donnée une liste L de nombres rend la liste dont le premier élément est égal à la somme des éléments de L, dont le deuxième élément est égal à la somme des éléments de $(cdr\ L)$... dont le dernier élément est égal au dernier élément de L. Par exemple, `(somme-cumulee (list 1 2 3 4))` → `(10 9 7 4)`

Exercice V (exercice simple sur les listes de listes)

Question 1 (1 point) :
Écrire une expression qui a pour valeur la liste `((5 2))`, liste qui a comme unique élément la liste qui a deux éléments, les entiers 5 et 2.

Question 2 (2 points) :
Écrire la signature et une définition de la fonction `prem-prem` qui, étant donnée *LL*, une liste non vide de listes non vides d'éléments rend le premier élément de la première liste de *LL*. Par exemple,

```
(prem-prem (list (list 5 2) (list 3)))  → 5
```

Exercice VI (récursion sur les listes)

Listes décroissantes en triangle

Question 1 (3 points) :
Écrire la signature et une définition de la fonction `liste-triangle-dec` qui, étant donnée une liste *L*, rend la liste (de listes) dont le premier élément est la liste *L* donnée, dont le second élément est la liste *L* donnée privée de son premier élément, dont le troisième élément est la liste *L* donnée privée de ses deux premiers éléments... et dont le dernier élément est la liste, de longueur un, qui comporte le dernier élément de la liste donnée *L*. Par exemple :

```
(liste-triangle-dec (list 1 2 3 4))  → ((1 2 3 4) (2 3 4) (3 4) (4))
```

Listes croissantes en triangle

On voudrait maintenant travailler sur la fonction qui, étant donnée une liste *L*, rend la *liste croissante en triangle de L* c'est-à-dire la liste (de listes) dont le premier élément est la liste qui comporte le dernier élément de la liste donnée *L*, dont le deuxième élément est la liste qui comporte les deux derniers éléments de la liste donnée *L*... et dont le dernier élément est la liste donnée *L*. Par exemple, si *L* est la liste (1 2 3 4), la *liste croissante en triangle de L* est égal à la liste (de listes) ((4) (3 4)(2 3 4)(1 2 3 4))

La fin du problème consiste à écrire, sans utiliser la fonction `reverse`, plusieurs définitions de la fonction `liste-triangle-croi` dont la spécification est :

`;;;` *liste-triangle-croi : LISTE[alpha] -> LISTE[LISTE[alpha]]*
`;;;` *(liste-triangle-croi L) rend la liste croissante en triangle de L*

Première implantation

Question 2 (2 points) :
Écrire la signature et une définition de la fonction `ajout-en-fin` qui, étant donnés un élément *e* et une liste *L*, rend la liste obtenue en ajoutant *e* à la fin de la liste *L*.

Question 3 (3 points) :
En utilisant la fonction `ajout-en-fin`, donner une première définition de la fonction `liste-triangle-croi`.

Deuxième implantation

Étant donnés un entier positif *n* et une liste *L*, la fonction `liste-trapeze1` rend la liste constituée par les *n* derniers éléments de la *liste croissante en triangle de L*. Par exemple,

```
(liste-trapeze1 2 '(1 2 3 4))  → ((2 3 4) (1 2 3 4))
```

Question 4 (3 points) :

Compléter la définition suivante de `liste-trapeze1` :

```
;;; liste-trapeze1 : nat * LISTE[alpha] -> LISTE[LISTE[alpha]]
;;; HYPOTHESE: n est inférieur ou égal à la longueur de L.
;;; (liste-trapeze1 n L) rend la liste constituée par les n derniers éléments
;;; de la liste croissante en triangle de L.
(define (liste-trapeze1 n L)
  (if (= n 1)
      
      (let ((LL1 (liste-trapeze1 (- n 1) L)))
        
      )
  )
)
```

Question 5 (2 points) :

En utilisant la fonction `liste-trapeze1`, donner une deuxième définition de `liste-triangle-croi`.

Troisième implantation

Question 6 (4 points) :

Écrire une définition de la fonction `liste-trapeze2` dont la spécification est :

```
;;; liste-trapeze2 : LISTE[beta] * LISTE[alpha] -> LISTE[LISTE[alpha]]
;;; n étant la longueur de la liste L1, (liste-trapeze2  L1 L) rend
;;; les n+1 derniers éléments de (liste-triangle-croi L)
```

Il est interdit d'utiliser la fonction `liste-trapeze1`.

Question 7 (2 points) :

En utilisant la fonction `liste-trapeze2`, donner une troisième définition de `liste-triangle-croi`.