Devoir sur table – Novembre 2000
MIAS 1ère année – 1er semestre

Aucun document ni machine électronique n'est permis à l'exception de la carte de référence de Scheme.

Répondre sur la feuille même, dans les boîtes appropriées. La taille des boîtes suggère le nombre de lignes de la réponse attendue (utiliser le dos de la feuille précédente si la réponse déborde des boîtes). Le barême (total sur 32) apparaissant dans chaque boîte n'est donné qu'à titre indicatif.

La clarté des réponses et la présentation des programmes seront appréciées. Toutes les fonctions apparaissant dans les réponses doivent être accompagnées de leur spécification. Ne pas désagrafer les feuilles.

Exercice 1

Qu'appelle-t'on une forme spéciale ? Quelles sont les formes spéciales de Scheme que vous connaissez ?

Réponse [4/32]

Une forme spéciale est une expression Scheme ayant une règle d'évaluation spécifique. Les formes spéciales de Scheme telles que recensées dans la carte de référence sont : `cond`, `if`, `and`, `or`, `quote`, `begin`, `let` et `let*`.

Commentaire: On peut tolérer `define` dans cette liste.

Exercice 2

Écrire une expression valant `((1 2)(3))` en utilisant au moins un `cons` et au moins un `list`.

Réponse [4/32]

Voici une solution :

```
(list (list 1 2) (cons 3 (list)))
```

Exercice 3

Écrire une fonction, nommée `somme-carres`, qui prend un entier n et qui calcule la somme des carrés des entiers de l'ensemble $[-n \ldots +n]$? Ainsi :

```
(somme-carres 1) → 2
(somme-carres 2) → 10
```

Section	Groupe	Nom	Prénom

Réponse [6/32]

```
;;; somme-carres: nat -> nat
;;; (somme-carres n) calcule la somme des carrés des entiers de l'ensemble
;;; [-n ... n] (soit deux fois la somme des carrés de [1 ... n]).
(define (somme-carres n)
  (if (> n 0)
      (+ (* 2 n n)                        ;(* (- n) (- n)) = (* n n)
         (somme-carres (- n 1)) )
      0 ) )
```

On peut aussi écrire mais c'est plus lent :

```
(define (somme-carres-bis n)
  (define (sommer i n)
    (if (<= i n)
        (+ (* i i) (sommer (+ i 1) n))
        0 ) )
  (sommer (- n) n) )
```

Exercice 4

Soit une liste non vide d'entiers naturels. Écrire une fonction, nommée `extraction`, qui prenant une telle liste renvoie cette liste privée de ses premiers termes ; le nombre de termes enlevés est précisément la valeur du premier d'entre eux. Ainsi

```
(extraction (list 2 3 4 5)) → (4 5)
(extraction (list 0 3 4 5)) → (0 3 4 5)
```

En revanche, `(extraction (list 9 1))` doit provoquer une erreur.

Réponse [6/32]

```
;;; extraction: LISTE[nat]/non vide/ -> LISTE[nat]
;;; ERREUR lorsque la liste n'est pas assez longue.
;;; (extraction L) renvoie la liste L privée de ses (car L) premiers termes.
(define (extraction L)
  ;; sortir: LISTE[nat] * nat -> LISTE[nat]
  ;; ERREUR lorsque la liste n'est pas assez longue.
  (define (sortir L n)
    (if (= n 0)
        L
        (if (pair? L)
            (sortir (cdr L) (- n 1))
            (error 'extraction "liste insuffisamment longue") ) ) )
  (sortir L (car L)) )
```

Commentaire: On distinguera bien l'emploi de contraintes de types, d'hypothèse et de signalisation d'erreur.

Exercice 5

Écrire une fonction, nommée `paires`, prenant deux listes de même longueur et construisant une liste des couples des termes de rang correspondant des deux listes. Ainsi :

```
(paires (list 1 2 3) (list "un" "deux" "trois")) → ((1 "un") (2 "deux") (3 "trois"))
```


Section	Groupe	Nom	Prénom

Réponse [6/32]

```
;;; paires: LISTE[alpha] * LISTE[beta] -> LISTE[NUPLET[alpha beta]]
;;; HYPOTHÈSE: les listes ont même longueur
;;; (paires (list a1 a2 ... aN) (list b1 b2 ... bN)) rend
;;;  (list (list a1 b1) (list a2 b2) ... (list aN bN))
(define (paires l1 l2)
  (if (pair? l1)
      (if (pair? l2)
          (cons (list (car l1) (car l2))
                (paires (cdr l1) (cdr l2)) )
          (list) )
      (list) ) )
```

Exercice 6

Que vaut l'expression `(mystere (list 1 2 3 4 5))` ? Que vaut l'expression `(mystere (list 1 2 3 4 5 6))` ? Écrire une spécification de la fonction `mystere` que voici :

```
;;; Une fonction bien mystérieuse.
(define (mystere x)
  (if (pair? x)
      (if (pair? (cdr x))
          (cons (list (car x) (cadr x))
                (mystere (cddr x)) )
          (list (list (car x) (car x))) )
      x ) )
```

Réponse [6/32]

```
(mystere (list 1 2 3 4 5)) → ((1 2) (3 4) (5 5))
(mystere (list 1 2 3 4 5 6)) → ((1 2) (3 4) (5 6))
```

Voici une spécification :

```
;;; mystere: LISTE[int] -> LISTE[NUPLET[int int]]
;;; (mystere x) prend une liste de termes et renvoie une nouvelle liste
;;; où ces termes sont regroupés deux par deux (en commençant par la gauche).
;;; Si la liste initiale a une taille impaire, le dernier terme est redoublé
;;; dans le résultat.
```

