Examen – Module d'informatique 1 – Janvier 2002
MIAS 1ère année – 1er semestre

Aucun document ni machine électronique n'est permis à l'exception de la carte de référence de Scheme. Les téléphones doivent être éteints et rangés dans les sacs.

L'examen dure deux heures. Ce sujet comporte 10 pages.

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.

Répondre sur la feuille même, dans les cadres appropriés. La taille des cadres suggère le nombre de lignes de la réponse attendue (utiliser le dos de la feuille précédente si la réponse déborde des cadres). Le barème (total sur 50) apparaissant dans chaque cadre 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

Question 1.1 – Qu'est-ce qu'une barrière d'abstraction ?

Réponse [3/50]

Un ensemble de fonctions permettant — de manipuler des concepts sans exposer leur représentation (si l'on se place comme utilisateur de ces fonctions) — de modifier l'implantation de ces fonctions sans se soucier des utilisations antérieures (si l'on se place comme programmeur de ces fonctions). Parmi ces fonctions, on distingue communément les constructeurs, reconnaisseurs et sélecteurs.

Question 1.2 – Donnez le schéma général de récursion sur les listes.

Réponse [2/50]

```
(define (f L)
  (if (pair? L)
      (composition (car L)
                   (f (cdr L)) )
      (traitement-vide) ) )
```

[Note pédagogique: On peut tout aussi bien admettre `(composition L (f (cdr L)))`. **Fin de note pédagogique]**

Exercice 2

Soit la fonction dite de Syracuse ainsi définie :

$$syracuse(n) = \begin{cases} n/2 & \text{si } n \text{ est pair} \\ 1 + 3n & \text{si } n \text{ est impair} \end{cases}$$

Section	Numéro d'anonymat

Une suite de Syracuse est définie à partir d'un entier naturel $u_0 \neq 0$ par

$$\forall n > 0, u_{n+1} = syracuse(u_n)$$

On admettra que ces suites possèdent la propriété (P) :

$$\forall u_0 \neq 0, \exists n, u_n = 1$$

Par exemple, si $u_0 = 2$ alors $u_1 = 1$ et P est vraie. Si $u_0 = 5$, alors $u_1 = 16, u_2 = 8, u_3 = 4, u_4 = 2, u_5 = 1$. On se propose d'étudier ces suites dans les questions qui suivent.

Question 2.1 – Quels sont les termes de la suite de Syracuse débutant par $u_0 = 3$ (on s'arrêtera au premier 1 rencontré) ?

Réponse [2/50]

Calculons ces termes avec l'une des fonctions ci-dessous corrigées : on a (orbite 3) → (3 10 5 16 8 4 2 1).

Question 2.2 – Écrire une fonction nommée `syracuse` répondant à la définition précédente.

Réponse [3/50]

```
;;; syracuse: nat/>0/ -> nat
;;; (syracuse n) calcule le terme suivant de la suite de Syracuse.
(define (syracuse n)
  (if (even? n)
      (quotient n 2)
      (+ 1 (* 3 n)) ) )
```

Question 2.3 – On nomme « orbite de p » la liste des termes de la suite de Syracuse u_n avec $u_0 = p$ jusqu'à l'occurrence du premier 1. Écrire la fonction nommée `orbite` calculant l'orbite de p. Ainsi :

```
(orbite 3)  → (3 10 5 16 8 4 2 1)
(orbite 13)  → (13 40 20 10 5 16 8 4 2 1)
(orbite 1)  → (1)
```

Réponse [3/50]

```
;;; orbite: nat -> LISTE[nat]
;;; (orbite n) calcule la liste des termes de la suite de Syracuse jusqu'au
;;; premier 1.
(define (orbite n)
  (if (= n 1)
      '(1)
      (cons n (orbite (syracuse n))) ) )
```

Question 2.4 – On nomme « apogée d'une orbite » le plus grand naturel de cette orbite. Écrire la fonction `apogee` prenant un entier p et calculant le plus grand naturel apparaissant dans l'orbite de p. Ainsi

```
(apogee 3)  → 16
(apogee 13)  → 40
```

Section Numéro d'anonymat

Réponse [3/50]

Voici une première solution en une ligne :

```
;;; apogee: nat -> nat
;;; (apogee p) renvoie le plus grand terme de l'orbite de p.
(define (apogee p)
  (reduce max 1 (orbite p)) )
```

et une seconde qui ne fabrique aucune liste intermédiaire :

```
(define (apogee-2 p)
  (if (= p 1)
      1
      (max p (apogee-2 (syracuse p))) ) )
```

Question 2.5 – Écrire la fonction nommée `apogees` prenant deux entiers naturels p et q tels que $q > p > 0$ et calculant la liste des couples (i, apogée(i)) pour i entre p (inclus) et q exclus. Ainsi

`(apogees 3 12)` → `((3 16) (4 4) (5 16) (6 16) (7 52) (8 8) (9 52) (10 16) (11 52))`

Réponse [4/50]

```
;;; apogees: nat * nat -> LISTE[NUPLET[nat * nat]]
;;; (apogees p q) renvoie la liste des couples (j, apogée(orbite j)) pour j
;;; entre p (inclus) et q (exclus).
;;; HYPOTHÈSE: p < q
(define (apogees p q)
  (if (< p q)
      (cons (list p (apogee p))
            (apogees (+ p 1) q) )
      '() ) )
```

Question 2.6 – Soit la fonction `mystere` suivante

```
(define (mystere p)
  (define (c x y)
    (if (< (cadr x) (cadr y))
        y
        x ) )
  (car (reduce c (list 1 1) (apogees 1 p))) )
```

Écrivez sa spécification ainsi que celle de sa fonction interne `c`. Voici quelques exemples d'emploi

`(mystere 5)` → `3`
`(mystere 10)` → `7`
`(mystere 15)` → `7`

On rappelle également la définition de la fonction `reduce` :

```
;;; reduce: (alpha * beta -> beta) * beta * LISTE[alpha] -> LISTE[beta]
;;; (reduce f end '(e1 e2 ... eN)) = (f e1 (f e2 ... (f eN end) ...))
(define (reduce f end list)
  (if (pair? list)
      (f (car list) (reduce f end (cdr list)))
      end ) )
```

Section Numéro d'anonymat

Réponse [5/50]

Voici cette fonction correctement nommée :

```
;;; plus-grand-apogee: nat/>1/ -> nat
;;; (plus-grand-apogee p) renvoie le plus petit entier entre 1 et p de
;;; plus grand apogée.
(define (plus-grand-apogee p)
  ;; compare: NUPLET[nat * nat] * NUPLET[nat * nat] -> NUPLET[nat * nat]
  ;; (compare i+a j+b) ramène celui de ses arguments qui
  ;; correspond au plus grand apogée.
  (define (compare i+a j+b)
    (if (< (cadr i+a) (cadr j+b))
        j+b
        i+a ) )
  (car (reduce compare (list 1 1) (apogees 1 p))) )
```

Question 2.7 – L'orbite de 13 passe par 10, elle a donc comme suffixe l'orbite de 10. Ainsi calculer l'orbite de 13 peut s'arrêter à la rencontre de 10 pour réutiliser l'orbite de 10. Écrire la fonction `orbites` qui prend un nombre p et qui renvoie la liste de toutes les orbites des nombres de l'orbite de p. On essaiera de ne jamais recalculer une orbite déjà calculée. Ainsi

```
(orbites 2)→
 ((2 1) (1))

(orbites 3)→
 ((3 10 5 16 8 4 2 1)
 (10 5 16 8 4 2 1)
 (5 16 8 4 2 1)
 (16 8 4 2 1)
 (8 4 2 1)
 (4 2 1)
 (2 1)
 (1))
```

Section Numéro d'anonymat

Réponse [4/50]

De nombreuses solutions existent ! La suivante est la plus coûteuse de toutes mais c'est la solution la plus courte :

```
;;; Assez coûteuse mais correcte.
(define (orbites1 p)
  (map orbite (orbite p)) )
```

Celle-ci invoque `orbite` une fois de moins :

```
(define (orbites2 p)
  (if (= p 1)
      (list (orbite 1))
      (cons (orbite p)
            (orbites2 (syracuse p)) ) ) )
```

Celle-ci ne calcule l'orbite qu'une seule fois puis renvoie la liste de tous ses suffixes successifs.

```
(define (orbites3 p)
  (define (repete L)
    (if (pair? L)
        (cons L (repete (cdr L)))
        '()))
  (repete (orbite p)))
```

Enfin, on peut entrelacer les deux calculs précédents et construire le résultat en une seule fois et sans invoquer `cdr`.

```
(define (orbites4 p)
  (if (= p 1)
      '((1))
      (let ((resultat (orbites4 (syracuse p))))
        (cons (cons p (car resultat))
              resultat ) ) ) ) )
```

Exercice 3

Dans cet exercice, on s'intéresse aux propriétés électriques de certaines configurations de résistances qui seront assemblées en série ou en parallèle.

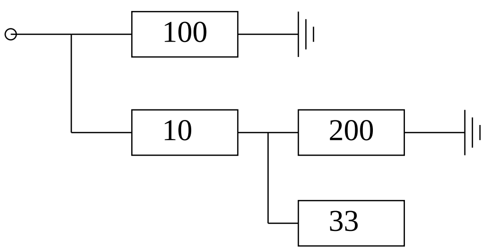
Soit la grammaire M suivante :

```
<circuit>     →  (TERRE)
                 (SERIE <résistance> <circuit> )
                 (PARALLELE <circuit> <circuit> )
                 <résistance>

<résistance>  →  (RESISTANCE <r> )

<r>           →  Nombre strictement positif
```

La grammaire M décrit des circuits électriques sous forme d'arbres. Le paramètre r d'une résistance décrit sa valeur (en Ohm). Par exemple, l'expression suivante obéit à la grammaire M.

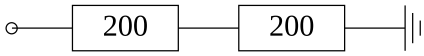
```
(PARALLELE (SERIE (RESISTANCE 100) (TERRE))
           (SERIE (RESISTANCE 10)
                  (PARALLELE (SERIE (RESISTANCE 200) (TERRE))
                             (RESISTANCE 33) ) ) )
```

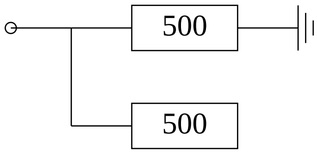
La précédente expression décrit le circuit suivant :



Section Numéro d'anonymat

Question 3.1 – Écrire les S-expressions obéissant à la grammaire M correspondant aux deux circuits suivants :





Réponse [4/50]

```
(SERIE (RESISTANCE 200) (SERIE (RESISTANCE 200) (TERRE)))
(PARALLELE (SERIE (RESISTANCE 500) (TERRE))
           (RESISTANCE 500) )
```

Question 3.2 – On désire manipuler ces circuits à l'aide d'une barrière d'abstraction que voici partiellement. Tout d'abord les quatre reconnaisseurs :

```
;;; serie?: Circuit -> bool
;;; (serie? c) reconnaît les circuits série.
(define (serie? c)
  (and (pair? c)
       (equal? (car c) 'SERIE) ) )

;;; parallele?: Circuit -> bool
;;; (parallele? c) reconnaît les circuits parallèles.
(define (parallele? c)
  (and (pair? c)
       (equal? (car c) 'PARALLELE) ) )

;;; terre?: Circuit -> bool
;;; (terre? c) reconnaît le circuit à la terre.
(define (terre? c)
  (and (pair? c)
       (equal? (car c) 'TERRE) ) )

;;; resistance?: Circuit -> bool
;;; (resistance? c) reconnaît un circuit formé d'une unique résistance.
(define (resistance? c)
  (and (pair? c)
       (equal? (car c) 'RESISTANCE) ) )
```

Voici trois des quatre constructeurs :

```
;;; serie: Resistance * Circuit -> Circuit
;;; (serie resistance circuit) construit un circuit mettant en série une
;;; resistance suivie d'un circuit.
(define (serie resistance circuit)
  (list 'SERIE resistance circuit) )

;;; parallele: Circuit * Circuit -> Circuit
;;; (parallele circuit1 circuit2) construit un circuit mettant en parallèle
;;; deux circuits.
(define (parallele circuit1 circuit2)
  (list 'PARALLELE circuit1 circuit2) )

;;; resistance: int/>0/ -> Resistance
```

Section | Numéro d'anonymat

```
;;; (resistance valeur) construit une résistance de valeur en Ohm.
(define (resistance valeur)
  (list 'RESISTANCE valeur) )
```

Enfin voici deux des cinq accesseurs :

```
;;; parallele-circuit1: Circuit/parallèle/ -> Circuit
;;; (parallele-circuit1 c) rend le premier circuit d'un circuit parallèle.
(define (parallele-circuit1 c)
  (cadr c) )

;;; parallele-circuit2: Circuit/parallèle/ -> Circuit
;;; (parallele-circuit2 c) rend le deuxième circuit d'un circuit parallèle.
(define (parallele-circuit2 c)
  (caddr c) )
```

Compléter la barrière en donnant les définitions et la nature (constructeur, reconnaisseur, accesseur) des fonctions manquantes `terre`, `serie-resistance`, `serie-circuit` et `resistance-valeur`.

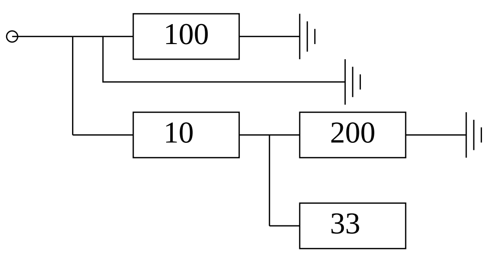
Réponse [3/50]

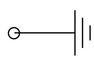
Voici le constructeur `terre` et les trois accesseurs `serie-resistance` et `serie-circuit`.

```
;;; terre: -> Circuit
;;; (terre) construit un circuit vide relié à la terre.
(define (terre)
  '(TERRE) )

;;; serie-resistance: Circuit/série/ -> Resistance
;;; (serie-resistance c) rend la première résistance d'un circuit série.
(define (serie-resistance c)
  (cadr c) )

;;; serie-circuit: Circuit/série/ -> Circuit
;;; (serie-circuit c) rend le deuxième circuit d'un circuit série.
(define (serie-circuit c)
  (caddr c) )

;;; resistance-valeur: Resistance -> int
;;; (resistance-valeur c) renvoie la valeur (en Ohm) d'une résistance.
(define (resistance-valeur c)
  (cadr c) )
```

Question 3.3 – On suppose que l'on alimente ce circuit en imposant une différence de potentiel entre le point haut gauche du circuit (la source) et la terre (représentée par le symbole conventionnel (à droite sur la figure ci-dessous)).

Écrire une fonction nommée `court-circuit?`, prenant un circuit et détectant s'il y a un court-circuit c'est-à-dire un chemin direct (ne passant par aucune résistance) entre la source et la terre. C'est notamment le cas des deux circuits suivants :





Ainsi,

Section	Numéro d'anonymat

```
(court-circuit?
  (parallele
    (parallele
      (serie (resistance 100) (terre))
      (terre))
    (serie (resistance 10)
           (parallele
             (serie (resistance 200) (terre))
             (resistance 33)))))→
 #t
```

Réponse [4/50]

La solution qui suit reprend la structure de la grammaire et ses quatre cas. On pourrait bien sûr regrouper les cas renvoyant Faux ensemble avec une clause `else` mais ici on est sûr de n'avoir rien oublié.

```
;;; court-circuit?: Circuit -> bool
;;; (court-circuit? c) vérifie qu'un circuit est en court-circuit.
(define (court-circuit? c)
  (cond
   ((terre? c)      #t)
   ((resistance? c) #f)
   ((serie? c)      #f)
   ((parallele? c)
    (or (court-circuit? (parallele-circuit1 c))
        (court-circuit? (parallele-circuit2 c)) ) ) ) )
```

Question 3.4 – Les branches du circuit qui ne sont pas finalement connectées à la terre sont inutiles. Écrire un semi-prédicat `elagage` qui prend un circuit et construit un circuit élagué de toute branche inutile. Si le circuit tout entier est inutile, la fonction `elagage` renverra la constante `#f`. Ainsi,

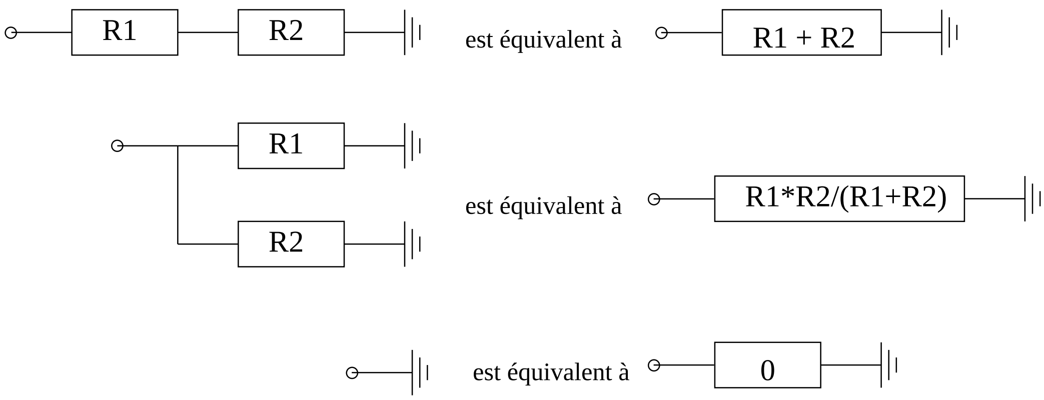
```
(elagage (serie (resistance 100) (terre)))→
 (SERIE (RESISTANCE 100) (TERRE))

(elagage
  (parallele
    (resistance 200)
    (serie (resistance 33) (terre))))→
 (SERIE (RESISTANCE 33) (TERRE))

(elagage (serie (resistance 50) (resistance 33)))→
 #f
```

Section Numéro d'anonymat

Réponse [5/50]

```
;;; elagage: Circuit -> Circuit + #f
;;; (elagage c) élague les branches non conductrices d'un circuit. Renvoie
;;; #f si le circuit ne peut être élagué.
(define (elagage c)
  (cond
   ((terre? c) c)
   ((resistance? c) #f)
   ((serie? c)
    (let ((c2 (elagage (serie-circuit c))))
      (if c2
          (serie (serie-resistance c) c2)
          #f ) ) )
   ((parallele? c)
    (let ((m1 (elagage (parallele-circuit1 c)))
          (m2 (elagage (parallele-circuit2 c))) )
      (if m1
          (if m2
              (parallele m1 m2)
              m1 )
          m2 ) ) ) ) )
```

Question 3.5 – Écrire une fonction nommée `resistance-equivalente` prenant un circuit élagué et calculant la valeur de la résistance équivalente d'un tel circuit. On rappelle les règles de résistances équivalentes :



Ainsi,

```
(resistance-equivalente
  (serie (resistance 100)
         (serie (resistance 200) (terre))))→
 300

(resistance-equivalente
  (parallele
    (serie (resistance 100) (terre))
    (serie (resistance 100) (terre))))→
 50
```

Section   Numéro d'anonymat

Réponse [5/50]

```
;;; resistance-equivalente: Circuit/élagué/ -> int
;;; (resistance-equivalente c) calcule la résistance équivalente d'un circuit.
(define (resistance-equivalente c)
  (cond
   ((terre? c) 0)
   ((serie? c)
    (+ (resistance-valeur (serie-resistance c))
       (resistance-equivalente (serie-circuit c)) ))
   ((parallele? c)
    (let ((r1 (resistance-equivalente (parallele-circuit1 c)))
          (r2 (resistance-equivalente (parallele-circuit2 c))) )
      (/ (* r1 r2) (+ r1 r2)) ) ) ) )
```