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 9 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 ?

[3/50]

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

[2/50]

Section | Numéro d'anonymat

Exercice 2

Soit la fonction dite de Syracuse ainsi définie :

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

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) ?

[2/50]

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

[3/50]

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)
```

[3/50]

Section Numéro d'anonymat

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
```

[3/50]

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))
```

[4/50]

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 ) )
```

[5/50]

Section Numéro d'anonymat

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))
```

[4/50]

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*> `)`

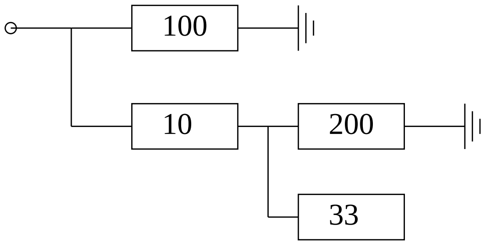
<*r*> → `Nombre strictement positif`

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.

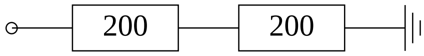
```
(PARALLELE (SERIE (RESISTANCE 100) (TERRE))
           (SERIE (RESISTANCE 10)
                  (PARALLELE (SERIE (RESISTANCE 200) (TERRE))
                             (RESISTANCE 33) ) ) )
```

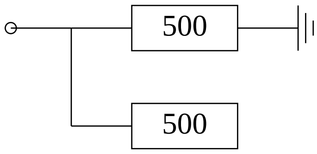
La précédente expression décrit le circuit suivant :

Section Numéro d'anonymat



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





[4/50]

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) ) )
```

Section | Numéro d'anonymat

```
;;;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
;;;(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) )
```

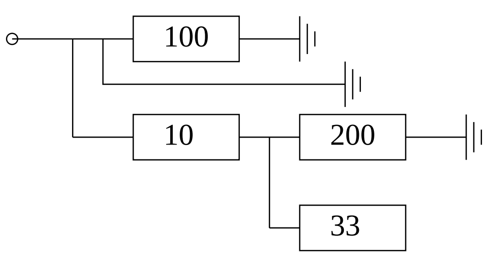
Compléter la barrière en donnant les définitions et la nature (constructeur, reconnaisseur, accesseur) des fonctions manquantes `terre`, `serie-resistance`, `serie-circuit` et `resistance-valeur`.

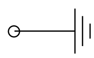
Section Numéro d'anonymat

[3/50]

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,

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

Section Numéro d'anonymat

[4/50]

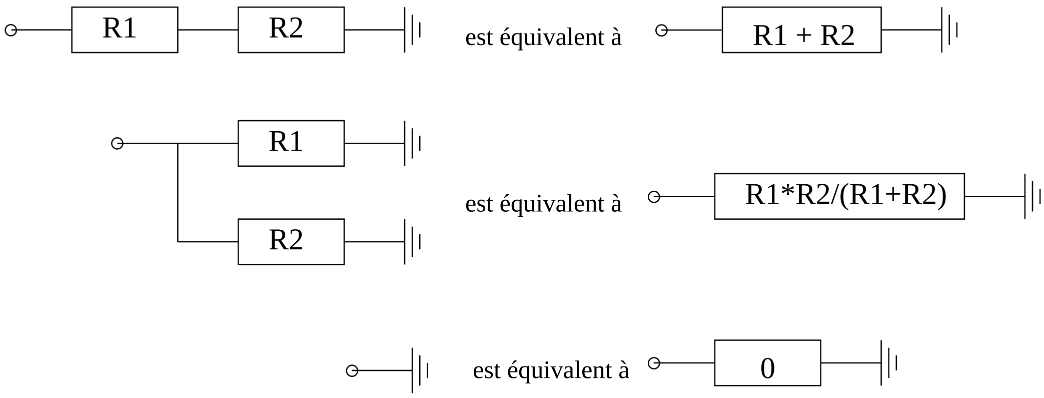
Question 3.4 – Les branches du circuit qui ne sont pas finalement connectées à la terre sont inutiles. Écrire un semi-prédicat `elagage` qui prend un circuit et construit un circuit élagué de toute branche inutile. Si le circuit tout entier est inutile, la fonction `elagage` renverra la constante `#f`.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

[5/50]

Section | Numéro d'anonymat

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
```

[5/50]