TD numéro 1

1 Algorithmes récursifs : un robot dessinateur

On considère un robot fixé sur un tableau en un point O. Ce robot sait exécuter deux primitives :

Procédure **TracerCercle**
argument : r, de type entier
environnement modifié : le tableau
description : le robot trace un cercle de centre O et de rayon r.

Procédure **EffacerCercle**
argument : r, de type entier
environnement modifié : le tableau
description : le robot efface sur un cercle de centre O et de rayon r.

Écriture d'une nouvelle procédure

Écrire une procédure récursive Dessiner(n), où n est un entier strictement positif, qui permet au robot de tracer n cercles concentriques de centre O et de rayons 1, 2, ..., n.

Comprendre un algorithme

On définit la procédure Mystère(n). Simuler l'exécution de Mystère(3) et donner la spécification de la procédure Mystère(n).

```
Procédure Mystère(n)
Début
   Si n = 1 Alors
      TracerCercle(1)
      EffacerCercle(1)
   Sinon
      TracerCercle(n)
      Mystère(n-1)
      EffacerCercle(n)
   FinSi
Fin
```


2 Évaluation d'expressions en Scheme

Donner la valeur retournée par les expressions ci-dessous :

- (+ 2 6)
- (+ (* 2 3 5) (- 6 8))
- '(+ 2 6)
- '(1 2 3)
- (1 2 3)

On suppose que les définitions suivantes ont été faites dans l'ordre donné :

```
(define moineau 5)
(define condor 435)
(define rapace 'condor)
(define oiseau condor)
```

Trouver les résultats des évaluations suivantes:

- baleine
- 'baleine
- moineau
- 'moineau
- rapace
- (+ oiseau moineau)
- (+ rapace condor)
- (+ (eval rapace) condor)

- (and (> 21 45) (= 3 (/ 12 4)))
- (and (> 21 45) (= 3 (/ 12 0)))
- (and (= 3 (/ 12 0)) (> 21 45))
- (boolean? (number? 3))
- (boolean? (number? "abc"))
- Soit expr une expression booléenne, évaluer :
  - (not (or expr true))
  - (not (and (or expr false) (not expr)))

3 Premières fonctions en Scheme

Écrire:

- une fonction qui renvoie le double d'un nombre passé en argument ;
- une fonction qui renvoie la moyenne de deux nombres passés en argument ;
- une fonction qui renvoie le signe d'un nombre ;
- une fonction qui renvoie la mention pour une note donnée ;
- une fonction qui calcule le $n^{ième}$ terme de la suite de Fibonacci ($u_n = u_{n-1} + u_{n-2}$, $u_0 = 1$, $u_1 = 1$)

Donnez la spécification de la fonction mystère ci-dessous :

```
(define mystere
  (lambda (x)
    (= 0 (modulo x 2))))
```

TD numéro 2

1 Fonctions sur les listes en langage algorithmique

On dispose de trois fonctions prédéfinies sur les listes :

- Premier(liste) : retourne le premier élément de la liste
- Reste(liste) : retourne la liste privée du premier élément
- Vide(liste) : retourne VRAI si la liste est vide, FAUX sinon

• Donner une définition récursive de la fonction **Longueur**(liste), qui retourne le nombre d'éléments d'une liste.

• Donner une définition récursive de la fonction **Somme**(liste), qui retourne la somme des éléments d'une liste de nombres.

• Donner une définition récursive de la fonction **Appartient**(élément, liste), qui retourne VRAI si l'élément est dans la liste et FAUX sinon.

• Donner une définition récursive de la fonction **Dernier**(liste), qui retourne le dernier élément d'une liste non vide.

2 Évaluation d'expressions en Scheme

- (car (cdr '(a b c d)))
- (car (cdr '(abc d)))
- (cdr '(a (b c d)))
- (cdr '((a b c d)))
- (cdr (car (cdr '(a (b c) (d e)))))
- (cons '(a b) '(c d))
- (cons 'a (cons 'b '(c d)))
- (cons '(a b) 'c)
- (list? (+ 2 3))
- (list? '(+ 2 3))

Université Lyon1 – Licence Sciences, Technologies, Santé UE LIF3

3 Premières fonctions sur les listes en Scheme

Écrire une fonction:

- qui retourne le second élément d'une liste d'au moins deux éléments ;
- qui retourne vrai si et seulement si une liste n'a qu'un seul élément.

4 Fonctions récursives sur les listes en Scheme

Écrire une fonction :

- qui calcule la longueur d'une liste ;
- qui calcule la somme des éléments d'une liste de nombres ;
- qui prend une liste et un élément et retourne vrai si et seulement si l'élément appartient à la liste ;
- qui retourne le $n^{ième}$ élément d'une liste ;
- qui insère un élément dans une liste après le $i^{ème}$ élément.

Donner la spécification de la fonction mystère ci-dessous :

```
(define mystere
  (lambda (x l)
    (cond ((null? l) 0)
          ((eq? x (car l)) (+ 1 (mystere x (cdr l))))
          (else (mystere x (cdr l))))))
```

TD numéro 3

1. Évaluation d'expressions

Donner la valeur retournée par les expressions ci-dessous :

```
(list 'a (cons '(b c) '(d)))
(append '(a) '(b c) '(d))
(append (list 'a '(b)) '(c))
```

On définit les listes suivantes :

```
(define l1 '(a b c))
(define l2 '(d e))
```

Donner les expressions utilisant l1 et l2 et permettant d'obtenir les résultats suivants :

```
((b c) (d e))
(c e)
(b c d)
```

2. Concaténation de listes

• Écrire une fonction qui concatène deux listes.

`(concatene '(a b c) '(d e)) → (a b c d e)`

• Écrire une fonction qui renverse une liste.

`(renverse '(a b c d)) → (d c b a)`

3. Mémorisation

• Écrire une fonction qui, étant donnés les coefficients d'un trinôme, rend les racines sous forme d'une liste. La liste sera vide si $\Delta<0$, n'aura qu'un élément si $\Delta=0$, et deux éléments sinon.

`(racines 1 2 -3) → (-3 1)`

• Écrire la fonction `som-prod` qui rend la somme et le produit d'une liste de nombres.

`(som-prod '(1 4 2 3)) → (10 24)`

• Écrire la fonction f telle que $f(x) = x+x^2+x^4+x^8+x^{16}$ en utilisant uniquement la fonction `carre`.

• Écrire une fonction qui, étant donnée une liste d'entiers, construit une liste de deux sous-listes : celle contenant les éléments impairs et celle contenant les éléments pairs.

`(parite '(1 2 6 5 7)) → ((1 5 7) (2 6))`

TD numéro 4

1 Tri à bulles

Le tri à bulles est un tri par sélection : on sélectionne l'élément minimum de la liste à trier, on le met en première position, et l'on recommence avec le reste de la liste.

Pour extraire l'élément minimum de la liste, on le fait remonter (comme une bulle) vers le début de la liste par échanges successifs de deux éléments voisins.

• Écrire la fonction `bulle` qui fait remonter le minimum en début de liste.

• Dérouler le tri à bulles sur la liste (7 9 1 6 2 3)

• Écrire la fonction `tri-bulles`, qui trie une liste de nombres par appels successifs à la fonction `bulle`.

• Si l est une liste de longueur n, quel est le nombre de comparaisons effectuées par l'appel :

```
(tri-bulles l)
```

2 Listes ordonnées

• Écrire une fonction qui teste si un nombre appartient à une liste ordonnée de nombres.

```
(appart? 12 '(-1 5 10 18)) → #f
```

3 Calculs en montant ou en descendant

• On veut écrire une fonction qui calcule la somme des chiffres d'un entier positif.

```
(somme-des-chiffres 341) -> 8
```

1. Définir une version récursive de cette fonction de la manière habituelle, en utilisant le résultat de l'appel récursif pour effectuer le calcul en remontant.

2. Définir ensuite une version qui, bien qu'étant récursive, s'inspire de la programmation itérative, en utilisant un paramètre supplémentaire pour effectuer les calculs en descendant.

TD numéro 5

Exercice 1 : récursivité profonde

• Écrire une fonction calculant le nombre total d'éléments d'une liste quelconque.

```
(compter '(1 2 (q w e) (r (e (w t)) 3 (e)) (o))) -> 12
```

• Écrire une fonction qui prend une liste et ajoute 2 à chaque nombre de cette liste en profondeur.

```
(ajoute2 '(2 b (10 a (56 3) 5) 4)) -> (4 b (12 a (58 5) 7) 6)
```

Exercice 2 : le triangle de Pascal

```
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5 10 10  5  1
```

• Écrire une fonction de deux entiers p et n, qui calcule le nombre de combinaisons de p éléments parmi n : $C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$ avec $C_n^0 = 1$ et $C_n^n = 1$

• Écrire une fonction qui construit une liste Li contenant les coefficients des binômes de degré i (la $i^{ème}$ ligne du triangle de Pascal).

• Écrire une fonction qui construit une liste de listes correspondant aux n premières lignes du triangle de Pascal.

Exercice 3 : une autre version du triangle de Pascal

• Écrire une fonction `som2` qui, étant donnée une liste de nombres, rend la liste des sommes des éléments consécutifs de cette liste.

```
(som2 '(1 2 3 4)) -> (3 5 7)
```

• Utiliser la fonction `som2` pour écrire une autre version de la fonction qui construit une liste de listes correspondant aux n premières lignes du triangle de Pascal.

TD numéro 6
Arbres binaires

• Écrire une fonction booléenne qui teste qu'une liste représente bien un arbre binaire.

• Écrire une fonction booléenne qui teste qu'un arbre est une feuille, c'est-à-dire un nœud qui n'a aucun fils.

• Écrire une fonction qui calcule la hauteur d'un arbre, définie ainsi : la hauteur d'un arbre est 1 + le maximum des hauteurs des sous-arbres (gauche, droit), la hauteur d'une feuille étant zéro.

• Écrire une fonction booléenne qui teste si une valeur appartient à un arbre.

• Écrire une fonction qui retourne la liste résultant du parcours infixe d'un arbre : en chaque nœud, on parcourt le fils gauche, puis on note la valeur du nœud, puis on parcourt le fils droit.

TD numéro 7
Arbres binaires de recherche

On considère maintenant des arbres dont les valeurs sont des nombres, et pour lesquels les nœuds sont ordonnés : en tout nœud, le sous-arbre gauche contient des nombres plus petits que celui du nœud, et le sous-arbre droit des nombres plus grands.

• Écrire une fonction booléenne qui teste qu'un arbre binaire est un arbre binaire de recherche.

• Écrire une fonction booléenne qui teste si une valeur appartient à un arbre binaire de recherche.

• Écrire une fonction qui insère un nombre dans un arbre binaire de recherche.

• Écrire une fonction qui trie une liste de nombres en passant par la construction d'un arbre binaire de recherche.

• Donner la spécification de la fonction mystère ci-dessous :

```
(define mystere
  (lambda (a x) ; a ABR, x nb
    (cond ((vide? a)'())
          ((= (valeur a) x) (list (valeur a)))
          ((< x (valeur a)) (cons (valeur a) (mystere (fils-g a) x)))
          (else (cons (valeur a) (mystere (fils-d a) x))))))
```

TD numéro 8
Des fonctions en argument et en résultat

Exercice 1

• Écrire une fonction qui étant donnés une liste plate l (les éléments sont des atomes) et un prédicat unaire p (fonction à un argument rendant un résultat booléen) renvoie la liste des éléments de l qui vérifient p.

```
(verifie '(1 2 7 5 4) even?) → (2 4)
```

Exercice 2

On considère deux fonctions composables unaires f et g de E dans E.

• Écrire la fonction `comp` qui renvoie le résultat de la composition de ces deux fonctions appliquée à un élément. Par exemple si f et g sont deux fonctions de E dans E et x un élément de E, la fonction `comp` renverra l'élément f(g(x)).

```
(comp carre cube 2) → 64
```

• Écrire la fonction `fn` qui, étant donnés une fonction f de E dans E, un entier n et x un élément de E, renvoie l'élément $f^n(x)$, où f^n est la composée n fois de f.

```
(fn carre 3 2) → 256
```

• Que fait la fonction `compf` ainsi définie :

```
(define compf
  (lambda (f g)
    (lambda (x)
      (f (g x)))))
```

• Écrire une version de `compf` utilisant `comp`.

• Écrire une fonction `fnf` qui, étant donnés une fonction f et un entier n, renvoie f^n.

```
(define puissance4 (fnf carre 2))
```

Exercice 3

• En utilisant la fonction abstraite sur les arbres vue en cours, écrire une fonction qui compte le nombre de valeurs paires dans un arbre de nombres.
• En adaptant la spécification de la fonction abstraite sur les arbres vue en cours, écrire une fonction qui multiplie par deux toutes les valeurs d'un arbre de nombres.

Exercice 4

• Définir la spécification de la fonction mystère ci-dessous :

```
(define mystere
  (lambda (a b)
    (if (null? b)
        '()
        (cons (a (car b)) (mystere a (cdr b))))))
```

TD numéro 9
map et apply

Pour chaque fonction, écrire une version de la fonction sans utiliser map, puis une en l'utilisant.

• Écrire une fonction qui, étant donnée une liste d'atomes, construit une liste de singletons :

```
(consing '(a b 1 3 c)) → ((a) (b) (1) (3) (c))
```

• Écrire une fonction qui utilise une fonction de tri passée en argument pour trier les sous-listes d'une liste passée en argument.

```
(trilist tri-bulles '((2 3 1 4) (3 4 2) (5 4 3 7)))
          → ((1 2 3 4) (2 3 4) (3 4 5 7))
```

• Écrire une fonction qui étant donnée une liste de listes, par exemple ((a b c) (d e) (f)), renvoie ((1 a b c) (1 d e) (1 f)).

• Écrire une fonction qui étant données deux listes de symboles construit une liste de couples de la manière suivante :

```
(conscouples '(anne sophie lisa marie) '(yann pierre olivier luc))
→ ((anne yann) (sophie pierre) (lisa olivier) (marie luc))
```

• Écrire une fonction qui calcule le maximum d'une liste de nombres.

• Écrire une fonction qui calcule la profondeur d'une liste.

• Écrire une fonction qui calcule le maximum d'une liste de listes de nombres.

```
(maxlistes '((2 1 3) (5 1 6 4) (4 3 2 1))) → 6
```

• Écrire une fonction qui calcule le maximum d'une liste en profondeur.

```
(maxprof '(1 2 (3 1 (4) 5) (2 3))) → 5
```

• Définir la spécification de la fonction mystère ci-dessous :

```
(define mystere
  (lambda (l) ; liste d'entiers
    (if (null? l)
        '(0 0)
        (let ((r (mystere (cdr l))))
          (if (even? (car l))
              (cons (+ (car l) (car r)) (cdr r))
              (list (car r) (+ (cadr r) (car l))))))))
```

TD numéro 10
Listes d'associations

Un dictionnaire bilingue

On définit un (tout) petit dictionnaire Français-Anglais :

```
(define dico-FA '((chat . cat) (chien . dog) (souris . mouse) (est . is) (noir . black) (le . the) (la . the)))
```

• Définir une fonction qui retourne la liste des mots de la première langue :

```
(mots-langue1 dico-FA) -> (chat chien souris est noir le la)
```

• Définir une fonction qui retourne la liste des mots de la deuxième langue :

```
(mots-langue2 dico-FA) -> (cat dog mouse is black the the)
```

• Définir une fonction qui inverse le dictionnaire :

```
(inverse dico-FA) -> ((cat . chat) (dog . chien) (mouse . souris) (is . est) (black . noir) (the . le) (the . la))
```

• Définir une fonction qui permet la traduction d'une phrase mot à mot grâce au dictionnaire :

```
(traduction dico-FA '(le chien est noir)) -> (the dog is black)

(traduction dico-FA '(le petit chat est noir)) -> (the pas-dans-le-dico cat is black)
```

TD numéros 11 et 12 : logique

Exercice 1 : fonctions booléennes en Scheme

• Soit expr une expression booléenne, évaluer :

- (not (or expr true))
- (not (and (or expr false) (not expr)))

• Définir en Scheme la fonction booléenne `majorité` vue en cours.

Exercice 2 : logique propositionnelle

Modéliser en logique des propositions le problème suivant :

> L'attaque envisagée réussira seulement si l'ennemi est surpris ou si la position est peu défendue. L'ennemi ne sera pas surpris, à moins qu'il ne soit téméraire. Il n'est pas téméraire si la position est peu défendue. Donc l'attaque ne réussira pas.

On utilisera les propositions suivantes :

A : attaque réussie
S : ennemi surpris
P : position peu défendue
T : ennemi téméraire

Quelle est la formule à démontrer ou infirmer pour savoir si le raisonnement est correct ?

Exercice 3 : diagrammes de Venn

Vérifier les lois d'absorption et de distributivité au moyen de diagrammes de Venn.

Exercice 4 : algèbre de Boole

Simplifier les expressions suivantes grâce aux règles de transformation de l'algèbre de Boole :

- $x.(x+y)+y.(x+z)$
- $(a+b+c).(a+\overline{b}+c).(a+\overline{b}+\overline{c})$

Exercice 5 : diagrammes de Quine

• A l'aide d'un diagramme de Quine, simplifier l'expression suivante : $(pq+pr)\supset(q\supset r)$

• A l'aide d'un diagramme de Quine, déterminer si le raisonnement de l'exercice 2 est correct en analysant la formule trouvée.

Exercice 6 : tables de Karnaugh

• Simplifier les expressions suivantes grâce à des tables de Karnaugh :

- xyz+z(xy+xy)
- xy+zxy+xz
- w(z+y)+zw+xyzw+xyz

• Reprendre l'énigme policière du cours, et démontrer la formule trouvée à l'aide d'une table de Karnaugh.

TD révisions 1

• Définir une fonction booléenne qui vérifie que tous les éléments d'une liste d'entiers sont multiples d'un entier passé en paramètre.

```
(multiples? '(5 15 35 20) 5) → #t
(multiples? '(5 15 36 20) 5) → #f
```

• Définir une fonction qui renvoie la somme des entiers pairs inférieurs ou égaux à un entier passé en paramètre.

```
(somme_pairs 11) → 30
```

• Définir une fonction qui, étant donnée une liste, construit la liste de ses nombres entiers associés à leurs carrés.

```
(n-carre '(a b 2 c 3.5 1)) → ((2 4) (1 1))
```

• Définir une fonction qui calcule le maximum d'une liste de nombres ainsi que le nombre d'occurrences de ce maximum (faire une version qui effectue les calculs en montant et une qui les fait en descendant).

```
(max-occ '(3 6 2 3 6 4 1 6)) → (6 3)
```

• Définir une fonction qui répète autant de fois un élément d'une liste que sa position dans la liste.

```
(repepete '(h e l l o )) → (h e e l l l l l l l o o o o o)
```

• Définir une fonction qui traite une liste en profondeur en remplaçant chacun des nombres par son opposé.

```
(oppose_en_profondeur '(2 c (4 -1 e) (f ((((-12)))) g h)))
          → (-2 c (-4 1 e) (f ((((12)))) g h))
```

• Définir une fonction qui calcule la moyenne des nombres d'un arbre de nombre en un seul parcours de l'arbre.

• Définir une fonction qui prend en paramètre un arbre binaire de nombres entiers et qui retourne un arbre où les valeurs sont des couples : la valeur du nœud et un booléen indiquant s'il est pair.

```
(construire-pairs '(2 (3 (4()())(5()()))(1()(6()()))))
→ ((2 #t)((3 #f)((4 #t)()())((5 #f)()()))((1 #f)()((6 #t)()())))
```

TD révisions 2

• On cherche à construire la liste des nombres entiers qui sont égaux à la somme de leurs chiffres mis au cube (par exemple 153) inférieurs à un nombre donné. Définir les fonctions intermédiaires nécessaires à la réalisation de cette fonction et implémenter l'ensemble.

```
(armstrong 1000) → (0 1 153 370 371 407)
```

• Définir une fonction qui compte en profondeur le nombre d'entiers d'une liste quelconque divisibles par un entier donné.

```
(compter-divisibles-prof '(a 3 6 (7 j (t 9)) 4 12) 3) → 4
```

• On considère des arbres où les valeurs des nœuds sont des couples (listes de deux éléments). Le but de l'exercice est de décrypter un mot caché. Il faut parcourir l'arbre suivant un chemin donné et récupérer le 1er ou le 2ème élément de la liste suivant l'indication associée au chemin. Le chemin ainsi que l'élément à choisir sont donnés par une liste de couples. On suppose que le chemin reste dans l'arbre.

```
(motcache '((p i)((e u)((t z)()())((s r)()((z d)()((u o)()()))))))
                  '((1 g)(1 d)(2 d)(2 d)(1 d))) → (p e r d u)
```

• Soit une liste de listes. Définir en utilisant map une fonction qui ajoute au début de chaque sous-liste sa longueur.

```
(longueur_en_tete '((4 z)(3 a &)(1 2 3 4 5 6)))
        → ((2 4 z) (3 3 a &) (6 1 2 3 4 5 6))
```

• Définir une fonction qui calcule la moyenne des valeurs d'un arbre de nombres en faisant un seul parcours de l'arbre.

• En utilisant la fonction abstraite sur les arbres vue en cours, définir une fonction qui enlève 1 à toutes les valeurs impaires d'un arbre.