

IA01 – Rapport TP1

Montée en compétences Lisp

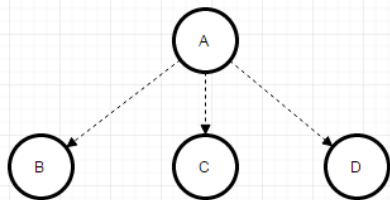
Exercice 1 : Mise en condition

1.

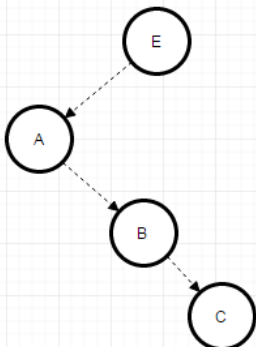
```
(caddr ' (A B C D)) ; D
(cadr (cadr (car ' ((a (b c)) e)))) ; C
(car (car (car ' (((DIEU) ENCORE) UNE)))) ; DIEU
(cadr (car ' (((DIEU) ENCORE) UNE))) ; ENCORE
```

2.

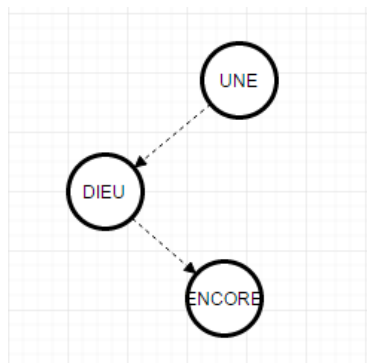
(A B C D) : (père, fils gauche, frère droit, fils droit)



((A (B C)) E) : (fils gauche, fils droit, père)



(((DIEU) ENCORE) UNE) : (fils gauche, fils droit, père)



3.

```
(cadr (cdr (cdr (cdr '(DO RE MI FA SOL LA SI))))); SOL
(CONS (cadr '((a b)(c d))) (caddr '(a (b (c))))) ; ((c d)) : construction de la liste avec l'élément liste (c d)
; caddr de (a (b (c)))=NIL
(cons (cons 'hello nil) '(how are you)) ; ((hello) how are you)
; cons 'hello nil=(hello)
(cons 'je (cons 'je (cons 'je (cons 'balbuter nil)))) ; (je (je (je (balbuter))))
; cons 'balbuter nil=(balbuter)
(cadr (cons 'tiens (cons '(c est simple) ()))) ; cadr de (tiens (c est simple))=(c est simple)
```

4.

double (L) :

renvoie une liste avec chaque élément en double

```
(defun double (L)
  (if (listp L)
      (if (< 0 (length L)) ; si not fin liste
          (append
            (list (car L) (car L)) (double(cdr L))) ; doubler élément puis réitérer avec reste liste
          L)
      L)
)
```

```
Break 2 [3]> (double '(a (b c) d))
(A A (B C) (B C) D D)
Break 2 [3]> (double '(a b c d))
(A A B B C C D D)
```

rang (X L) : Retourne l'index de l'élément X dans L.

Renvoie 0 si X n'appartient pas à L.

```
(defun rang (X L)
  (cond
    ((not (member X L)) 0) ; X n'appartient pas à L => 0
    ((equal X (car L)) 1) ; si fin de liste alors X=nil donc True.
    (t (+ 1 (rang X (cdr L)))) ; sinon appel rang sur élément suivant
  )
)
```

```
Break 2 [3]> (rang 'c '(a b c d e f g))
3
Break 2 [3]> (rang 'z '(a b c d e f g))
0
```

tete (N L) : renvoie les N premiers éléments.

Si N est plus grand que le nombre d'éléments de L alors la liste L est renvoyée.

Si N=0 alors nil est renvoyée.

```
(defun tete (N L)
  (if (<= N (length L)); N<= nb. élément de L
      (if (> N 0); Nème élément pas atteint
          (cons (car L) (tete (- N 1) (cdr L))); then construction tete liste
          L); avec i ème élément
      L); sinon return L
  )
)
```

```
Break 1 [32]> (tete 0 L)
NIL
Break 1 [32]> (tete 3 L)
(A B C)
Break 1 [32]> (length L)
7
Break 1 [32]> (tete 8 L)
(A B C D E F G)
```

`elim (L)` : renvoie une liste sans doublon, *nil* si vide et *t* si *t*.

```
(defun elim(L)
  (cond
    ((and L (listp L)) ; L différent de nil
      (setq a (car L)) ; a=ième élément
      (setq new_list (list a)) ; new_list=liste sans doublon
      (dolist (i (cdr L)) ; boucle sur les [i+1;N] éléments
        (unless (eq a i) ; i est doublon de a ??
          (setq new_list (append new_list (list i)))) ; si non doublon alors ajout i à new_list
      )
      (cons (car new_list) (elim (cdr new_list)))) ; new_list est une liste sans doublon par rapport à l'élément a.
    )
    (t L) ; liste vide alors return la liste. ; construction de new_list avec le reste de la liste => récursion avec l'élément i+1.
  ))
```

```
[14]> (elim '(a b c d))
(A B C D)
[15]> (elim '(a z g a s a z d g z s))
(A Z G S D)
[16]> (elim '(a b a b a b a))
(A B)
[17]> (elim nil)
NIL
[18]> (elim t)
T
```

`monEqual (X Y)` :

```
Break 6 [7]> (eq 'luc 'luc)
T
Break 6 [7]> (equal 'luc 'luc)
T
Break 6 [7]> (eq 'luc 'daniel)
NIL
Break 6 [7]> (equal 'luc 'daniel)
NIL
Break 6 [7]> (eq (car '(do re)) (cadr '(mi do sol)))
T
Break 6 [7]> (equal (car '(do re)) (cadr '(mi do sol)))
T
Break 6 [7]> (eq '(d p f t r) '(d p f t r))
NIL
Break 6 [7]> (equal '(d p f t r) '(d p f t r))
T
```

La différence entre *eq* et *equal* se remarque pour le test de l'égalité de liste.

equal renvoie *t* si liste identique alors que *eq* renvoie *nil*.

Nous pouvons donc construire la fonction *monEqual* de cette manière :

```
(defun monEqual (X Y)
  (if (and (listp X) (listp Y)) ; if X ET Y list
      (equal X Y) ; then appliquer equal
      (eq X Y) ; else appliquer eq
  )
)
```


```
Break 6 [7]> (monequal 'luc 'luc)
T
Break 6 [7]> (monequal 'luc 'daniel)
NIL
Break 6 [7]> (monequal (car '(do re)) (cadr '(mi do sol)))
T
Break 6 [7]> (monequal '(d p f t r) '(d p f t r))
T
```

monEqual combine les retours des fonction *equal* et *eq*. *Equal* est uniquement appelé si X et Y sont des listes.

Exercice 2 : Objets fonctionnels

Fonction très simplement exécutée avec mapcar.

```
(defun list-triple (LL)
  (mapcar #'(lambda (j) (* j 3)) LL))
```



```
[11]> (list-triple '(-5 1 9 23))
(-15 3 27 69)
[12]> (list-triple '(0 2 3 11))
(0 6 9 33)
```

Exercice 3 : a-list

D'abord, nous avons commencé par définir une liste Test qui permettra de tester les différentes fonctions.

```
Ex.3
; Définition d'une a-list pour tester les fonctions
(setq Test '((Moi 21) (MinhTri 21) (Bebe 1)))

(defun my-assoc (cle a-list) ; Retourne la valeur d'une clé dans une liste d'associations
  (if (eq (caar a-list) cle)
      (print (car a-list))
      (if (not (null a-list)) (my-assoc cle (cdr a-list))
          nil))
  )
(my-assoc 'Bebe Test)
```

Le fonctionnement de *my-assoc* est assez simple : On regarde si le premier élément de la *a-list* correspond à la clé recherchée. Si c'est le cas, alors on renvoi le couple, sinon, on continue dans la liste de façon récursive. Si on arrive à la fin de la liste, la condition (not (null a-list)) permet de renvoyer *nil*.

```
(defun cles (a-list) ; Renvoie les clés d'une a-list
  (if (not (null a-list))
      (append (list (caar a-list)) (cles (cdr a-list)))
      nil)
  )
(cles Test)
```

On définit maintenant la fonction *cles* permettant de récupérer tous les clés d'une a-liste.

Pour cela, on parcourt la liste récursivement (avec comme condition d'arrêt que la liste ne soit pas vide) en ajoutant à la liste des clés la clé actuelle suivie des prochaines.

```
(defun creation (listeCles listeValeurs) ;Retourne une a-liste à partir d'une liste de clés et de valeurs
  (if (and (not (null listeCles)) (not (null listeValeurs)))
      (append (list (list (car listeCles) (car listeValeurs))) (creation (cdr listeCles) (cdr listeValeurs)))
      )
  )
)

(creation '(A B C) '(1 2 3 4))
```

Enfin, concernant la fonction permettant la création d'une a-liste : On prend le premier élément de la liste des clés, le premier élément de la liste des valeurs et on forme un couple avec ces deux éléments. Ensuite, on relance la fonction sur la queue de chacune de ses listes. On s'arrête dès que l'une des deux listes est vide. En procédant ainsi, on gère le cas où les deux listes ne font pas la même taille.

Exercice 4 : Gestion d'une base de connaissances en Lisp

A. Fonctions de service :

```
;Ex.4
(setq BaseTest '((" Le Dernier Jour d'un condamné " Hugo 1829 50000)
  (" Notre-Dame de Paris " Hugo 1831 3000000)
  (" Les Misérables " Hugo 1862 2000000)
  ("Le Horla " Maupassant 1887 2000000)(" Contes de la bécasse " Maupassant 1883 500000)
  ("Germinal " Zola 1885 3000000)
))

(defun auteur (ouvrage) ; Retourne l'auteur de l'ouvrage passé en argument
  (cadr ouvrage)
)

(auteur '("Notre-Dame de Paris" Hugo 1831 3000000))

(defun titre (ouvrage) ; Retourne le titre de l'ouvrage passé en argument
  (car ouvrage)
)

(titre '("Notre-Dame de Paris" Hugo 1831 3000000))

(defun annee (ouvrage) ; Retourne l'année de l'ouvrage passé en argument
  (caddr ouvrage)
)

(annee '(" Notre-Dame de Paris " Hugo 1831 3000000))

(defun nombre (ouvrage) ; Retourne le nombre d'exemplaire de l'ouvrage passé en argument
  (caddrd ouvrage)
)

(nombre '(" Notre-Dame de Paris " Hugo 1831 3000000))
```

Les fonctions de service se passent de commentaires détaillés, il s'agit la simplement de manipulation de liste à l'aide des fonctions *car* et *cdr* ainsi que de leur combinaison.

B. Autres fonctions :

La fonction FB1 doit simplement afficher tous les ouvrages de la base de données. Il suffit pour cela de parcourir la base de données avec un *dolist* et d'en afficher chacun de ses éléments.

```
(defun FB1 (database)
  (dolist (x database)
    (print x))
)
(FB1 BaseTest)
```

La fonction FB2 fonctionne sur le même principe que FB1 sauf qu'à chaque itération, il y'a une condition supplémentaire pour n'afficher que les auteurs correspondant à « Hugo ».

Pour retourner la liste des titres d'ouvrages d'un certain auteur (FB3), on utilise une variable locale s.

```
(defun FB2 (database) ; Affiche les ouvrages de l'auteur Hugo
  (dolist (x database)
    (if (equal (auteur x) 'Hugo) (print x)))
)
(FB2 BaseTest)
```

À chaque itération, si l'auteur correspond, on rajoute le titre de son œuvre à la liste s. À la fin, on renvoi simplement s.

```
(defun FB3(database auteur) ; Retourne la liste des titres d'ouvrages dont l'auteur est auteur
  (let (s '())
    (dolist (x database)
      (if (equal (auteur x) auteur) (setq s (append s (list (titre x))))))
    s
  )
)
(FB3 BaseTest 'Hugo)
```

Pour FB4, on parcourt la base de donnée en cherchant l'année x. Si celle-ci est trouvé, on utilise la fonction return pour ne pas chercher d'autres ouvrages puisqu'on souhaite seulement le premier ouvrage paru cette année.

Pour FB5, on utilise toujours la même méthode : on parcourt la liste des œuvres, si on trouve un ouvrage respectant la condition, on l'ajoute à la liste.

```
(defun FB4 (database annee) ; Retourne le premier ouvrage paru en année ann ou nil
  (dolist (x database)
    (if (equal (annee x) annee) (return x) nil))
)
(FB4 BaseTest '1831)

(defun FB5 (database) ; Retourne la liste des ouvrages dont le nb d'exemplaires vendus dépasse 1000000 ou nil
  (let (( nb '()))
    (dolist (x BaseTest)
      (if (> (nombre x) 1000000) (setq nb (append nb (list (titre x))))))
    nb
  )
)
(FB5 BaseTest)
```

Enfin, pour FB6, on utilise deux variables locales. Le compteur compte le nombre de livres pris en compte et la variable nb calcule la somme du nombre de livres vendus. Pour obtenir la moyenne, on divise simplement *compt* par *nb*.

```
(defun FB6 (database aut) ; Calcule et retourne la moyenne du nombre d'exemplaires vendus de l'auteur aut
  (let ((compt 0) (nb 0))
    (dolist (x database)
      (if (eq aut (auteur x)) (progn (setq compt (+ compt 1)) (setq nb (+ nb (nombre x))))))
    (/ nb compt)
  ))
(FB6 BaseTest 'Hugo)
```