utc
Université de Technologie
Compiègne

COMPTE RENDU :TP01

Montée en compétences Lisp

Rédigé par:
Fatima Imghi
Ndeye Aminata Aida faye

PROFESSEUR :
Mme.Marie-Hélène Abel

Rapport TP01 : Montée en compétences Lisp

Introduction

Ce TP vise à approfondir nos compétences en programmation fonctionnelle avec Lisp, un langage puissant pour la manipulation des listes, souvent utilisé en intelligence artificielle à travers ces exercices, nous explorerons les concepts clés de Lisp, comme la gestion des objets, l'utilisation des fonctions anonymes, et la manipulation des listes d'association.

Le rapport détaillera le code développé pour chaque exercice avec des explications et des justifications.

Nous avons pris des captures d'écran de nos codes sur notepad++, et les explications sont fournies en dessous de chaque figure. Ce rapport a été réalisé dans Google Docs, ce qui a permis à chacune d'entre nous de proposer des suggestions de code. Nous avons testé ensemble les différentes solutions, commenté le code de chacune, et débuggé les fonctions sur ACL. Initialement, nous utilisions SBCL, mais certaines fonctions ne fonctionnaient pas correctement.

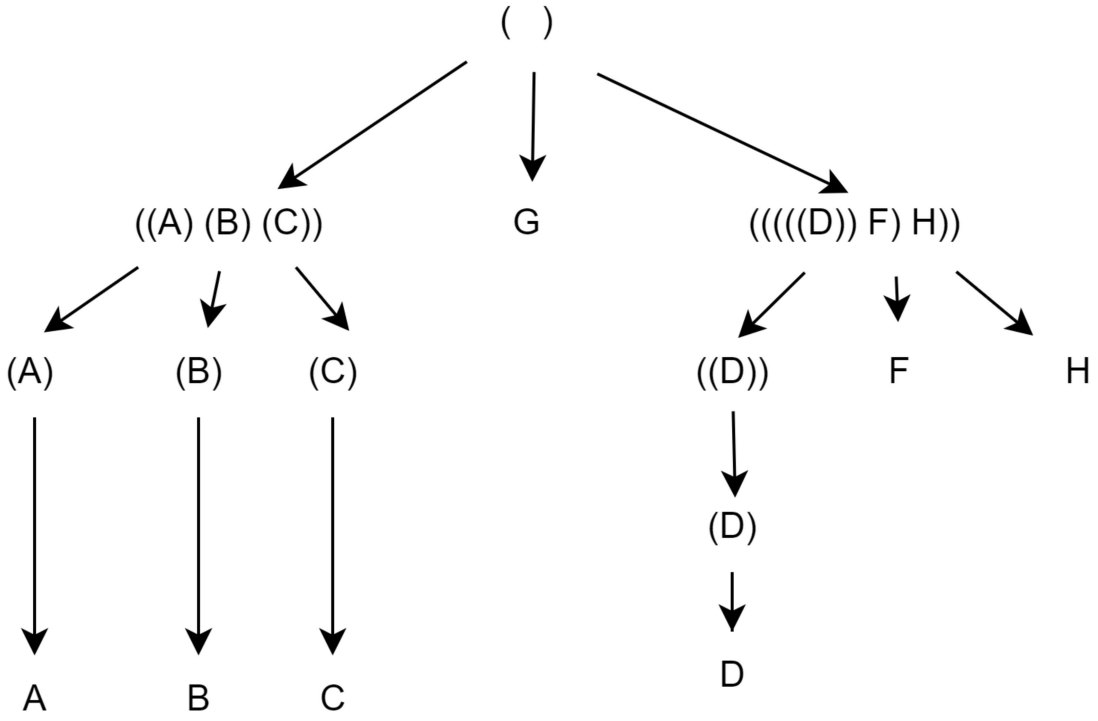
Exercice 1 : Mise en condition

1. Détermination des types d'objets Lisp

35 Numérique
(35) liste
(((3) 5) 6) liste
-34RRRR symbole
T valeur logique indiquant vrai
NIL valeur logique indiquant faux ou liste vide
() liste vide

2. Traduction sous forme d'arbre

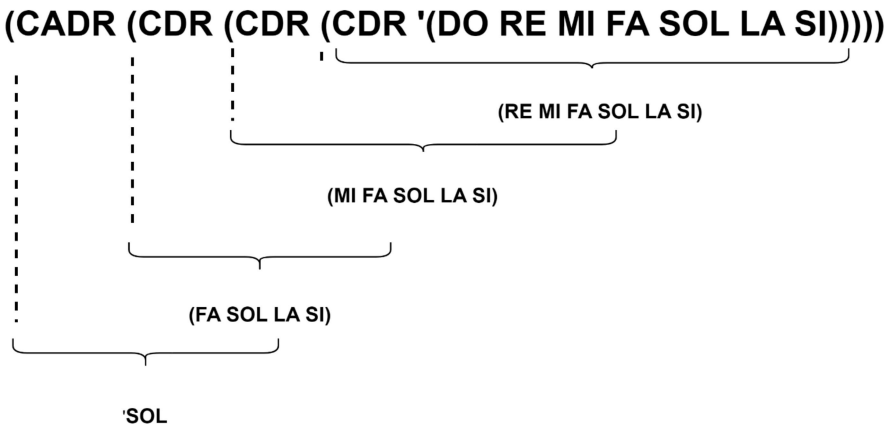
2. Traduisez sous forme d'arbre la liste suivante : (((A)(B)(C)) G ((((D)) F) H)))



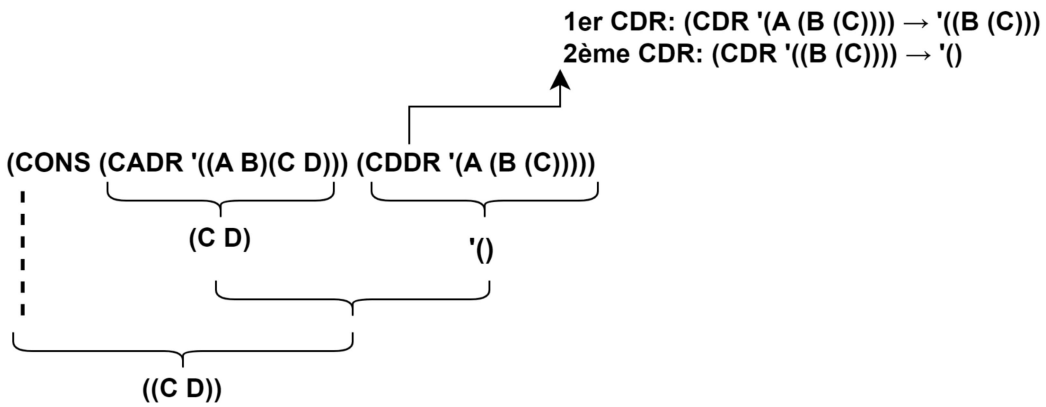
L'objet le plus profond est D car il est l'élément le plus imbriqué dans les listes

3. Explication des appels de fonctions

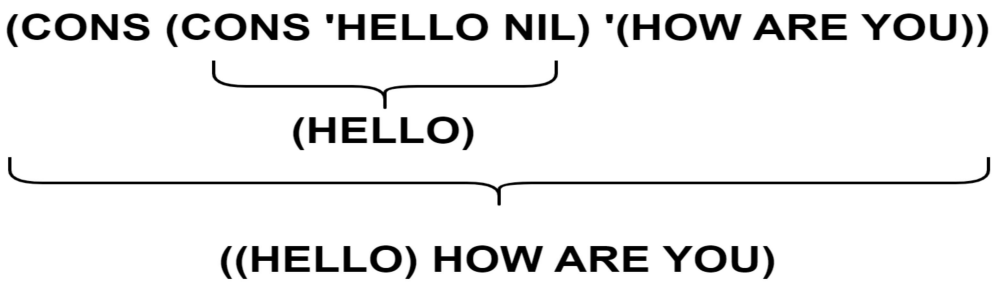
- (CADR (CDR (CDR (CDR '(DO RE MI FA SOL LA SI)))))



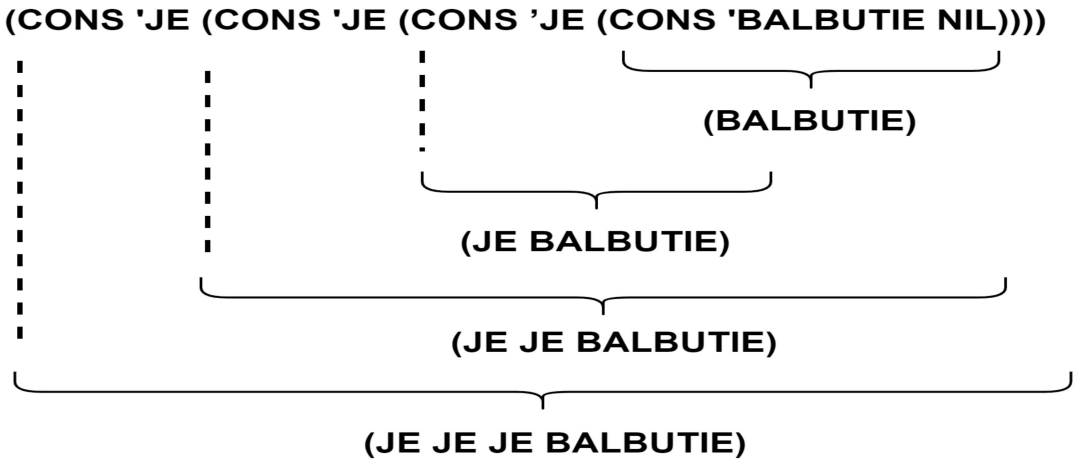
- (CONS (CADR '((A B)(C D))) (CDDR '(A (B (C)))))



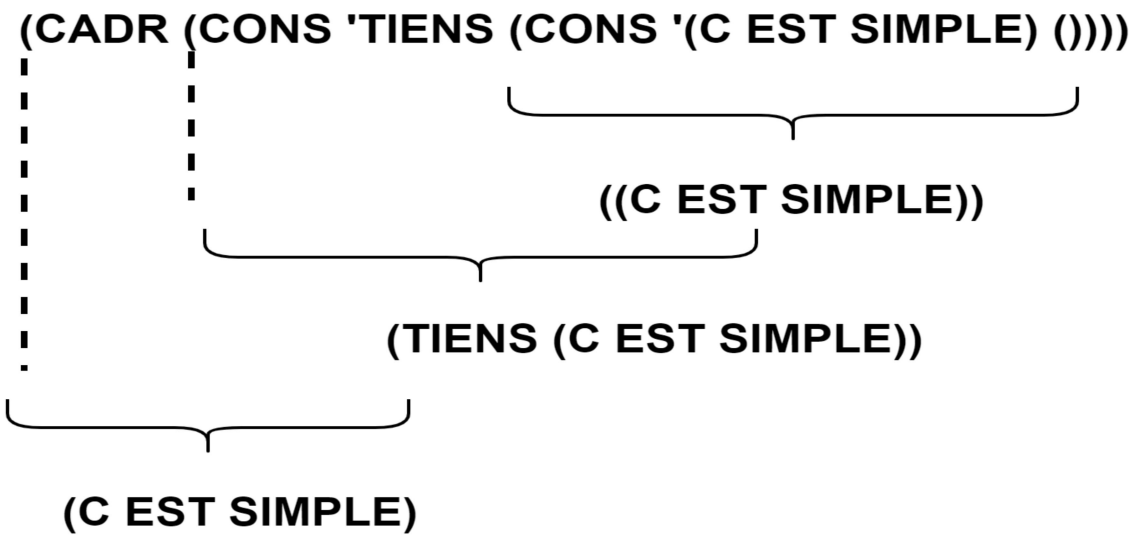
- (CONS (CONS 'HELLO NIL) '(HOW ARE YOU))



- (CONS 'JE (CONS 'JE (CONS ’JE (CONS 'BALBUTIE NIL))))



- (CADR (CONS 'TIENS (CONS '(C EST SIMPLE) ())))



4.Développement des fonctions

```
(defun nombres3 (L)     ; Fonction qui vérifie si les 3 premiers éléments de L sont des nombres
  (if (and (not (null L)) ; Vérifie que la liste n'est pas vide
           (numberp (car L)) ; Vérifie que le premier élément est un nombre
           (numberp (cadr L))  ; Vérifie que le deuxième élément est un nombre
           (numberp (caddr L)))  ; Vérifie que le troisième élément est un nombre
      "Bravo"     ; Retourne "Bravo" si toutes les vérifications sont vraies
      "Perdu"))    ; Retourne "Perdu" sinon
```

On commence par s'assurer que la liste n'est pas vide . Ensuite, on examine chacun des trois premiers éléments pour déterminer s'ils sont tous des nombres. Si c'est le cas, la fonction renvoie "Bravo". Si l'un des éléments n'est pas un nombre, elle retourne "Perdu".

```
(defun grouper (L1 L2)                  ; Fonction qui regroupe les éléments de L1 et L2
  (if (or (null L1) (null L2))          ; Vérifie si l'une des listes est vide
      nil                                ; Si oui, retourne nil
    (cons (list (car L1) (car L2))       ; Crée une nouvelle paire avec les premiers éléments de L1 et L2
          (grouper (cdr L1) (cdr L2))))) ; Appelle récursivement grouper sur les restes des listes
```

le code retourne nil, si l'une des listes est vide.
Sinon, on crée une nouvelle paire en combinant les premiers éléments des deux listes.
Ensuite, on appelle récursivement grouper pour continuer à traiter les éléments restants des deux listes.

```
(defun monReverse (L)
  (if (null L)                  ; Si la liste est vide
      '()                       ; retourner une liste vide
      (cons (car L)             ; sinon, cons avec le premier élément
            (monReverse (cdr L))))) ; et l'inverse de la queue
```

si la liste n'est pas vide on construit une nouvelle liste en utilisant le premier élément de L et en appelant récursivement monReverse sur le reste de la liste.

```
(defun palindrome (L)
  (equal L (monReverse L)))
```

En utilisant la fonction monReverse. On compare la liste d'origine L avec sa version inversée, obtenue par monReverse L. Si les deux listes sont identiques, cela signifie que la liste est un palindrome, et on retourne T. Dans le cas contraire, on retourne NIL .

Exercice 2 : Objets fonctionnels

La fonction anonyme lambda(x) (* x 3) calculant le triple de x nous a été fournie pour être utilisée dans l'exercice.. Puis on souhaiterait mettre dans une liste la liste des éléments accompagnés de leur triple, pour ce faire on utilise mapcar pour ajouter à l'élément x son triple. Pour chaque élément x, mapcar génère un couple constitué de x et de son triple, produisant ainsi une liste de couples.

```
(defun list-triple-couple(liste) ; Définition de la fonction
    (lambda (x)           ; Définition d'une fonction anonyme prenant un argument x
      (list x (* x 3)))  ; Crée une liste contenant x et son triple
    liste))  ; Applique la fonction à chaque élément de la liste
```

Cela permet de générer une nouvelle liste où chaque élément est associé à sa valeur multipliée par trois

Exercice 3 : a-list

```
(defun my-assoc (cle a-list) ; Définition de la fonction qui retourne la première paire de la liste associée à la clé donnée
  (dolist (paire a-list)    ;Parcourt chaque paire dans a-list
    (when (eq cle (car paire))  ; Vérifie si cle correspond à la clé de la paire
      (return paire))))  ; On avait utilisé print mais dans le TP il est demandé de retourner
```

Pour chaque paire,on vérifie si la clé cle correspond à la première valeur de la paire. Si une correspondance est trouvée, la fonction retourne cette paire. Si aucune correspondance n'est trouvée après avoir parcouru toute la liste, la fonction retourne nil..

```
(defun cles (a-list)
  (if (null a-list)    ; Vérifie si la liste d'associations est vide
      nil              ; Retourne nil si la liste est vide
      (loop for paire in a-list    ; Si la liste n'est pas vide, itération sur chaque paire dans la liste d'associations
            collect (car paire)))) ; Collecte la première valeur de chaque paire qui est la clé dans une nouvelle liste
```

Si la liste est vide, retourne nil. Si la liste n'est pas vide, on itère sur chaque paire dans a-list. Ensuite, pour chaque paire, elle collecte la première valeur (la clé) dans une nouvelle liste. la fonction retourne une liste contenant toutes les clés extraites

```
(defun creation (listeCles listeValeurs)
  (if (or (null listeCles) (null listeValeurs))  ; Vérifie si l'une des deux listes est vide
      nil  ; Si c'est le cas, retourne nil
      (cons (list (car listeCles) (car listeValeurs))  ; Crée une paire (clé, valeur) en utilisant 'list' sur la tête des deux listes
            (creation (cdr listeCles) (cdr listeValeurs)))))  ; Appelle récursivement 'creation' avec les queues des deux listes
```

Pour définir la fonction creation on écrit une fonction récursive. On vérifie d'abord qu'aucune des deux listes en arguments n'est nulle. Si c'est le cas on retourne Nil. Sinon on crée une liste (clé valeur) avec list(car listeCles) (car listeValeurs). On stocke le tout avec cons.

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

A- Complétons BaseTest avec les conflits des Mérovingiens et ceux des Carolingiens

(setq BaseTest
'(("Guerre de Burgondie" 523 533 (("Royaume Franc") ("Royaume des Burgondes"))
("Vezeronce" "Arles"))

("Campagnes de Clovis Ier" 486 508 (("Royaume Franc") ("Domaine gallo-romain de Soissons " "Royaume alaman" "Royaume des Burgondes " "Royaume wisigoth " "Royaume ostrogoth " " Royaume wisigoth ")) ("Aisne" "Zulpic " "Dijon " " Vouillé " "Arles"))

("Guerre de la Thuringe" 531 531(("Royaume Franc") ("Thurunges")) ("Thuringe"))

("Guerre des Goths" 535 553 (("Royaume Franc" "Royaume wisigoth Burgondes") ("Empire byzantin")) ("Péninsule italienne"))

("Conquête de l'Alémanie" 536 536 (("Royaume Franc") ("Alamans")) ("Alémanie"))

("Conquête de la Bavière" 555 555 (("Royaume Franc") ("Bavarii")) ("Bavière"))

("Campagnes de Bretagne" 560 578 (("Royaume Franc") ("Royaume du Vanatais")) ("Vanetais"))

("Guerre de succession mérovingienne" 584 585 (("Royaume Franc") ("Royaume d'Aquitaine")) ("Comminges"))

("Guerre franco-frisonne" 600 793 (("Royaume Franc") ("Royaume de Frise")) ("Pays-Bas" "Allemagne"))

("Guerre civile des Francs" 715 719 (("Neustrie") ("Austrasie")) ("Royaume Franc"))

("Invasion omeyyade en France" 719 759 (("Royaume Franc") ("Califat omeyyade")) ("Royaume d'Aquitaine" "Septimanie"))

‘(carovingiens("Guerre des Lombards" 755 758 (("Royaume franc") ("Lombards")) ("Lombardie "))

("Guerre d'Aquitaine" 761 768 (("Royaume franc") ("Aquitains")) ("Vasconie Aquitaine"))

("Guerre des Saxons" 772 804 (("Royaume franc") ("Saxons")) ("Germanie"))

("Guerre des Lombards" 773 774 (("Royaume franc") ("Lombards")) ("Lombardie "))

("Guerre des Avars" 791 805 (("Royaume de France") ("Avars")) ("Pannonie"))
("Invasions sarrasines en Provence" 798 990 (("Royaume de France") ("Sarrasins")) ("Provence"))

("Guerre civile entre les fils de Louis le Pieux" 830 842 (("Francie occidentale (Royaume de France)") ("Francie orientale") ("Francie médiane")) ("Fontenoy"))

("Guerre franco-bretonne" 843 851 (("Royaume de France") ("Royaume de Bretagne" "Vikings")) ("Royaume de Bretagne"))

("Luttes inter-dynastiques carolingiennes" 876 946 (("Francie occidentale") ("Francie orientale")) ("Royaume de Bourgogne"))

("Invasions vikings en France" 799 1014 (("Royaume de France") ("Vikings")) ("Normandie” “Bretagne")))
))

B/ Définir les fonctions de service

```
(defun dateDebut(conflit)
  (if (listp conflit) ; Vérifie si 'conflit' est une liste
      (cadr conflit))) ; Retourne le deuxième élément de la liste, qui représente la date de début du conflit
```

la date de début d'un conflit est représentée sous forme de liste. la fonction commence par vérifier si l'argument conflit est bien une liste à l'aide de listp si il l’est , elle retourne le

deuxième élément de cette liste qui est supposé être la date de début (obtenue avec cadr). Si conflit n'est pas une liste, la fonction ne retourne rien

.

```
(defun nomConflit (conflit)
(if (listp conflit)  ; Vérifie si conflit est une liste
    (car conflit))) ; Si oui, retourne le premier élément de la liste qui est le nom du conflit
```

Retourne le premier élément de cette liste, c'est à dire le nom du conflit (obtenu avec car). Si conflit n'est pas une liste, la fonction ne retourne rien.

```
(defun allies (conflit)
  (if (listp conflit)  ; Vérifie si conflit est une liste
      (let ((belligérants (cadddr conflit))) ; Récupère le quatrième élément de la liste (alliés et ennemis)
        (car belligérants)))) ; Retourne le premier élément de belligérants, qui représente les alliés
```

On a utilisé cadddr pour accéder au quatrième élément de la liste, qui contient les belligérants (alliés et ennemis). Ensuite, la fct° retourne le premier élément de cette liste de belligérants, représentant les alliés. Si conflit n'est pas une liste, la fonction ne retournera rien.

```
(defun ennemis (conflit)
  (let ((belligérants (cadddr conflit))); Récupère le quatrième élément de la liste (alliés et ennemis)
    (cadr belligérants))) ; Retourne le reste de la liste après le premier élément (ennemis)
```

On récupère le quatrième élément de la liste conflit, qui contient les belligérants (alliés et ennemis), et on l'affecte à une variable locale appelée belligérants.ensuite, cadr retournera le deuxième élément de cette liste, qui représente les ennemis

```
(defun lieu (conflit)
  (cadddr conflit)) ; Retourne le Cinquième élément de la liste conflit qui correspond au lieu
```

La fonction lieu extrait le lieu d'un conflit représenté sous forme de liste. Elle accède directement au cinquième élément de la liste conflit en utilisant cadddr. Cette valeur est supposée représenter le lieu du conflit.

FB1

```
(defun afficher_tout ()
  (dolist (conflit BaseTest)            ; Pour chaque conflit dans la liste BaseTest
    (let ((nom (nomConflit conflit)))   ; Appelle nomConflit pour obtenir le nom du conflit actuel
      (when nom                         ; Vérifie si nom n'est pas nil (donc si un nom a été trouvé)
        (print nom)))))                ; Affiche le nom du conflit
```

On parcourt tous les conflits dans la liste BaseTest. Pour chaque conflit, on utilise nomConflit

pour obtenir le nom du conflit pour l'affecter à une variable locale nom.
 Si nom n'est pas nil , on l'afficheà l'aide de print.

FB2

```
(defun royaume_franc ()
  (dolist (conflit BaseTest)
    (when (member "Royaume Franc" (allies conflit) :test 'string=)
(print (nomConflit conflit)))))
```

On a passé beaucoup de temps sur cette fonction car beaucoup de nos propositions n'ont pas marché , et au final , la solution était d'utiliser string=.
 avec member on recherche "Royaume Franc" dans la liste et on rajoute string= pour pouvoir comparer la chaine de caractère "Royaume Franc" avec les autres chaines de caractère dans la liste allies;

FB3

```
(defun liste_conflits (allie) (let ((liste '())) ; Initialisation de la liste des conflits
 (dolist (conflit BaseTest)
(when (member allie (allies conflit) :test 'string=) ; Vérifie si l'allie est dans les alliés du conflit
(setf liste (cons (nomConflit conflit) liste)))) ; Ajoute le nom du conflit à la liste liste
) ; Renvoie la liste des conflits
```

on commence ici par initialiser une liste vide pour stocker les noms des conflits et on parcourt chaque conflit dans les conflits collectés . Pour chaque conflit, on vérifie si l'allié donné est présent dans la liste des alliés associés au conflit. Si l'allié est trouvé, le nom du conflit est ajouté à la liste.

FB4

```
(defun verif_date_debut()   ; vérifie si la date du début de conflit est 523
(dolist (conflit BaseTest) ; on parcourt chaque conflit de la base
(if(= (dateDebut conflit ) 523) ; on fait appel à la fonction dateDebut
(print (nomConflit conflit)))))
```

On souhaite retourner le conflit dont la date de début est 523. On fait appel à la fonction dateDebut et on compare cette date à 523 si c'est le cas on retourne ce conflit.

FB5

```
(defun date_debut ()
  (dolist (conflit BaseTest)
    (if (and (>= (dateDebut conflit) 523) (<= (dateDebut conflit) 715))
        (print (nomConflit conflit)))))
```

Le principe est le même que pour FB4, seul le test de vérification change. On vérifie maintenant que la date de début est comprise entre 523 et 715.

FB6

```
(defun lombards ()
(let ((compteur 0))
        (dolist (conflit BaseTest)
        (when (member "Lombards" (ennemis conflit):test 'string=)
            (setf compteur (+ compteur 1))
(print (compteur))))))))
```

On souhaite retourner le nombre de conflits qui contiennent “Lombards” dans leur liste d’ennemies .Pour ce faire on initialise un compteur à 0 et on parcourt la liste des ennemis de chaque conflit dès que l’on trouve un ennemi équivalent à “Lombard” on incrémente le compteur puis on retourne ce compteur. On fait appel à la fonction ennemis définie dans la question précédente.

D. Evolution de la basetest et prise en compte de l’issue du conflit ainsi que de son appartenance aux carolingiens ou aux mérovingiens.

Nouvelle base évoluée :

Représentation: Nous rajoutons l’appartenance en début de chaque conflit (comme racine) et l’issue tout à la fin du conflit.

(setq BaseTest '(

(("Mérovingiens") ("Campagnes de Clovis Ier" 486 508 (("Royaume Franc") ("Domaine gallo-romain de Soissons" "Royaume alaman" "Royaume des Burgondes" "Royaume wisigoth" "Royaume ostrogoth")) ("Aisne" "Zulpic" "Dijon" "Vouillé" "Arles") ("Victoire franque" "Victoire franque" "Victoire franque" "Victoire franque" "Victoire Ostrogoths et Wisigoths “)))

(("Mérovingiens") ("Guerre de Burgondie" 523 533 (("Royaume Franc") ("Royaume des Burgondes")) ("Vezeronce" "Arles") ("Victoire franque")))

(("Mérovingiens") ("Guerre de la Thuringe" 531 531 (("Royaume Franc") ("Thurunges")) ("Thuringe") ("Victoire franque")))

(("Mérovingiens") ("Guerre des Goths" 535 553 (("Royaume Franc" "Royaume wisigoth Burgondes") ("Empire byzantin")) ("Péninsule italienne") ("Victoire byzantine")))

(("Mérovingiens") ("Conquête de l'Alémanie" 536 536 (("Royaume Franc") ("Alamans")) ("Alémanie") ("Victoire franque")))

(("Mérovingiens") ("Conquête de la Bavière" 555 555 (("Royaume Franc") ("Bavarii")) ("Bavière") ("Victoire franque")))

(("Mérovingiens") ("Campagnes de Bretagne" 560 578 (("Royaume Franc") ("Royaume du Vanatais")) ("Vanetais") ("Victoire roi Waroch")))

(("Mérovingiens") ("Guerre de succession mérovingienne" 584 585 (("Royaume Franc") ("Royaume d'Aquitaine")) ("Comminges") ("Victoire franque")))

(("Mérovingiens") ("Guerre franco-frisonne" 600 793 (("Royaume Franc") ("Royaume de Frise")) ("Pays-Bas" "Allemagne") ("Victoire franque")))

(("Mérovingiens") ("Guerre civile des Francs" 715 719 (("Neustrie") ("Austrasie")) ("Royaume Franc") ("Victoire Charles Martel")))

(("Mérovingiens") ("Invasion omeyyade en France" 719 759 (("Royaume Franc") ("Califat omeyyade")) ("Royaume d'Aquitaine" "Septimanie") ("Victoire franque")))

(("Carolingiens") ("Guerre des Lombards" 755 758 (("Royaume franc") ("Lombards")) ("Lombardie") ("Victoire franque")))

(("Carolingiens") ("Guerre d'Aquitaine" 761 768 (("Royaume franc") ("Aquitains")) ("Vasconie Aquitaine") ("Victoire franque"))) (("Carolingiens") ("Guerre des Saxons" 772 804 (("Royaume franc") ("Saxons")) ("Germanie") ("Victoire franque")))

(("Carolingiens") ("Guerre des Saxons" 772 804 (("Royaume franc") ("Saxons")) (Germanie") ("Victoire franque")))

(("Carolingiens") ("Guerre des Lombards" 773 774 (("Royaume franc") ("Lombards")) ("Lombardie") ("Victoire franque")))

(("Carolingiens") ("Guerre des Avars" 791 805 (("Royaume de France") ("Avars")) ("Pannonie") ("Victoire franque")))

(("Carolingiens") ("Invasions sarrasines en Provence" 798 990 (("Royaume de France") ("Sarrasins")) ("Provence") ("Victoire franque")))

(("Carolingiens") ("Guerre civile entre les fils de Louis le Pieux" 830 842 (("Francie occidentale (Royaume de France)") ("Francie orientale") ("Francie médiane")) ("Fontenoy") ("Victoire Charles II le Chauve")))

(("Carolingiens") ("Guerre franco-bretonne" 843 851 (("Royaume de France") ("Royaume de Bretagne" "Vikings")) ("Royaume de Bretagne") ("Victoire d'Erispoe")))

(("Carolingiens") ("Luttes inter-dynastiques carolingiennes" 876 946 (("Francie occidentale") ("Francie orientale")) ("Royaume de Bourgogne") ("Victoire française")))

(("Carolingiens") ("Invasions vikings en France" 799 1014 (("Royaume de France") ("Vikings")) ("Normandie" "Bretagne") ("Traité de Saint-Clair-sur-Epte")))

))

```
(defun appartenance_conflit(conflit)
(car(first conflit)))
(defun issue_conflit(conflit)
(car(cddddr conflit)))
```

En ne mettant que (first conflit) on renvoie la liste contenant le premier élément du conflit qui est ici (“Mérovingiens”) ou (“Carolingiens”) , on ajoute car pour ne renvoyer que la chaîne de caractères “Mérovingiens” ou “Carolingiens”. l’issue correspondant au dernier élément de la liste de chaque conflit, on l’obtient en prenant le car du CDDDDR.