TP01 : Montée en compétences Lisp

Léo Przewlocki - Alexandre Touzeau

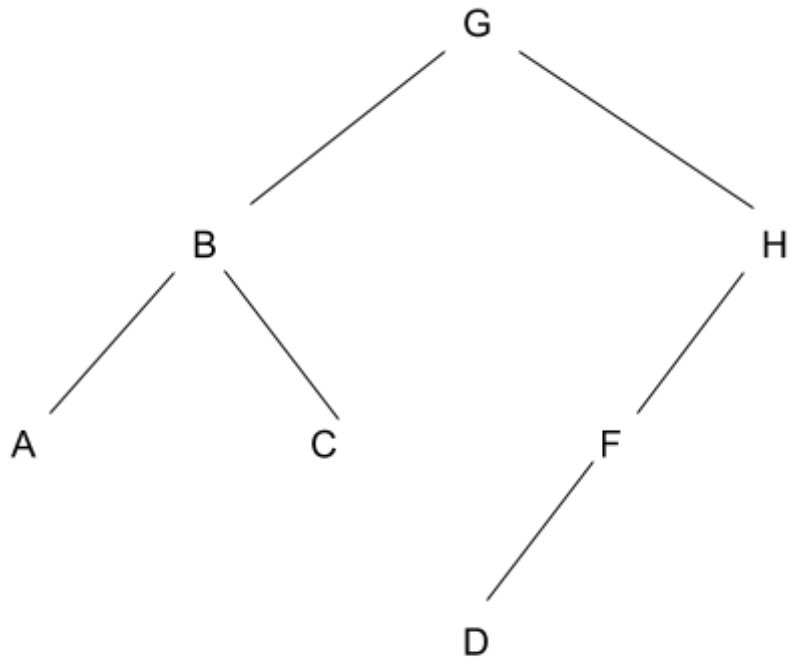
Exercice 1

1)

- 35 est un atome - Confirmation à l'aide de (atom 35) qui nous renvoie la valeur True
- (35) est une liste constitué de 1 atome - Confirmation à l'aide de (list '(35)) renvoie True
- (((3) 5) 6) est une liste
- -34RRRR est un atome
- T est un atome
- NIL est à la fois une liste et un atome
- () est équivalent à NIL c'est donc également à la fois une liste et un atome

2)

Le sujet ne spécifie pas dans quel représentation on se place pour représenter la liste (((A)(B)(C)) G (((((D)) F) H))) sous forme d'arbre. Nous avons donc choisis arbitrairement la forme infixée.



L'objet le plus profond est **D**.

3)

- (CADR (CDR (CDR (CDR '(DO RE MI FA SOL LA SI))))) nous donne SOL
- (CONS (CADR '((A B)(C D))) (CDDR '(A (B (C))))) nous donne ((C D))
- (CONS (CONS 'HELLO NIL) '(HOW ARE YOU)) donne ((HELLO) HOW ARE YOU)
- (CONS 'JE (CONS 'JE (CONS 'JE (CONS 'BALBUTIE NIL)))) donne (JE JE JE BALBUTIE)
- (CADR (CONS 'TIENS (CONS '(C EST SIMPLE) ()))) donne (C EST SIMPLE)

4)

Les 4 fonctions demandées ici sont toutes précédées d'un commentaire expliquant leur fonctionnement. De plus chaque fonction est suivie d'un test (également commenté) pour vérifier le bon fonctionnement de ces dernières.

Exercice 2

code lisp

```
`(defun list-triple-couple (x)
   (mapcar (lambda (n) (list n (* 3 n))) x)
)`
```

Il suffit de comprendre comment fonctionne '*mapcar'*. Le premier argument est la fonction qui sera appliquée à chaque élément de la liste, qui est le deuxième argument.

La fonction passé à *mapcar* est comme demandé une fonction anonyme., qui commence par un lambda, suivi la variable en argument et la fonction elle-même.

Exercice 3

my-assoc: Cette fonction est un simple parcours de liste à l'aide de "dolist", on test pour chaque élément de la liste si le car est égal à la clé passée en paramètre. Si c'est le cas alors on affiche l'élément de la liste : la clé + la valeur.

cles: Nous avons ici opté pour une fonction récursive (après avoir tenté dans un premier temps de refaire un dolist). La condition d'arrêt est que la liste passée en argument soit nulle (NIL). On utilise "append" pour chaque occurrence de la fonction sur le car du car (soit la clé du premier élément puis la clé du second etc…). On rappelle ensuite récursivement la fonction sur le cdr de la liste passée en argument. On obtient ainsi au final la liste des clés.

creation: De la même manière que pour la fonction précédente nous avons ici fait le choix de la récurrence. La condition d'arrêt étant sur la liste des clés passées en argument nous avons ici supposé que l'utilisateur entre bien deux listes de la même taille. La fonction est ici simple, il suffit d'append les car des deux listes puis de rappeler récursivement sur les cdr.

La difficulté est de ne pas oublier de mettre “list” juste après le append pour construire une liste finale constituée de plusieurs sous-liste clé-valeur.

Remarque: Les fonctions sont ici toute suivie d’un commentaire pour le test.

Exercice 4

A.
Après avoir complété BaseTest, nous nous retrouvons avec une liste de 22 éléments.

B.
Les fonctions de cette question sont assez triviales. Nous connaissons la position des éléments demandés dans les listes de conflit donc il suffit de manier les *‘car’* et ‘*cdr’*. Pour la dernière fonction, nous utilisons ‘*nthcdr 4’* afin d’appliquer quatre fois ‘*cdr’*.

C.
Nous réutilisons ici les fonctions créées à la question précédente. Nous utilisons *‘dolist’* pour parcourir tous les conflits, et des conditions *‘if’* qund nous recherchons un élément en particulier.

Dans la fonction FB6, nous créons un compteur avec *‘setq’* que nous incrémentons ensuite avec *‘incf’*. Nous affichons celui-ci en fin de fonction.