TP01 : Montée en compétences Lisp

Livrables attendus : un fichier lisp + un rapport présentant et argumentant le code lisp proposé.
Un bonus sera accordé lorsque deux solutions sont développées et comparées (itérative/récursive)

Date de remise : 6 octobre 2014 à 18H

Exercice 1 : Mise en condition

Ecrire les fonctions suivantes :

reverseA (arg1 arg2 arg3) : retourne la liste constituée des trois arguments dans l'ordre inverse.

(reverseA 1 2 3) -> (3 2 1)

reverseB (L) : retourne la liste inversée de 1, 2 ou 3 éléments.

(reverseB '(1 2)) -> (2 1)

reverseC (L) : retourne la liste constituée des éléments de L inversés.

(reverseC '(a b (c d) e f)) -> (f e (c d) b a)

double (L) : retourne la liste constituée des éléments de L en doublant les atomes.

(double '((1 2) 3 (4 5) 6)) -> ((1 2) 3 3 (4 5) 6 6)

doublebis (L) : retourne la liste constituée des éléments de L en doublant les atomes à tous les niveaux de profondeur.

(doublebis '((1 2) 3 (4 5) 6)) -> ((1 1 2 2) 3 3 (4 4 5 5) 6 6)

monAppend (L M) : retourne la concaténation des deux listes L et M

Cette fonction existe sous le nom de "append".

myEqual : retourne t ou nil selon l'égalité de ses deux arguments.

Cette fonction existe sous le nom de « equal ». Vous vous aiderez de la fonction standard EQ et de prédicats. Afin de bien comprendre, testez :

(eq 'LUC 'LUC)

(eq ‘LUC ‘DANIEL)

(eq (car ‘(do re)) (cadr ‘(mi do sol)))

(eq ‘(d p f t r) ‘(d p f t r))

Faites les mêmes tests avec la fonction standard EQUAL. Expliquez les résultats obtenus

Exercice 2 : Objets fonctionnels

L’objectif est de voir comment utiliser `mapcar` avec des fonctions anonymes, des expressions *lambda*. Une fonction anonyme est une fonction sans nom qui peut être fournie en paramètre à toute fonction LISP. Le format est:

```
(lambda (paramètres)
  body)
```

où `<body>` est une suite d'instructions LISP. Par exemple:

```
(lambda (x)
  (* x 3))
```

est une fonction anonyme qui retourne le triple de l'argument passé en paramètre.

Ecrivez une fonction *list-pair*e qui retourne la liste des éléments par paire de la liste fournie en paramètre. Utilisez `mapcar`.

Exemple:

```
(list-paire '(0 2 3 11))  -> ((0 0) (2 2) (3 3) (11 11))
```

Exercice 3 : *a-list*

Ecrivez la fonction décrite ci-dessous.

my-assoc

arguments:
`cle`: une S-Expression quelconque;

`a-liste`: une liste d'association, c'est-à-dire une liste de la forme:

`((clé`$_1$ `valeur`$_1$`) (clé`$_2$ `valeur`$_2$`) ... (clé`$_n$ `valeur`$_n$`))`

spécification:
`(my-assoc cle a-liste)` retourne `nil` si `cle` ne correspond à aucune clé de la liste d'association, la paire correspondante dans le cas contraire; cette fonction existe sous le nom de assoc.

exemples:

```
? (my-assoc 'Pierre
       '((Yolande  25) (Pierre  22) (Julie  45)))
(Pierre  22)

? (my-assoc 'Yves
       '((Yolande aime Pierre)
         (Pierre aime Julie)
         (Julie aime Pierre)))
nil
```

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

On considère une base de connaissances sur des personnes définies par les propriétés suivantes : nom, prénom, ville, âge et nombre de livres possédés.

Soit la base BaseTest suivante :

(setq BaseTest '((Dupond Pierre Lyon 45 150)

(Dupond Marie Nice 32 200)

(Dupond Jacques Lyon 69 20)

(Perrot Jacques Geneve 28 500)

(Perrot Jean Nice 55 60)

(Perrot Anna Grenoble 19 180)

))

A. Définir les fonctions de service :

nom (personne) : retourne le nom de la personne passée en argument

> (nom '(Dupond Pierre Lyon 45 150))

DUPOND

prenom (personne) : retourne le prénom de la personne passée en argument

> (prenom '(Dupond Pierre Lyon 45 150))

PIERRE

ville (personne) : retourne la ville de la personne passée en argument

>(ville '(Dupond Pierre Lyon 45 150))

LYON

age (personne) : retourne l'âge de la personne passée en argument

> (age ‘(Dupond Pierre Lyon 45 150))

45

nombre-livres (personne) : retourne le nombre de livres possédés par la personne passée en argument

> (nombre-livres ‘(Dupond Pierre Lyon 45 150))

150

B. En utilisant ces fonctions de service si nécessaire, définir les fonctions suivantes :

FB1 : affiche toutes les personnes

FB2 : affiche les personnes qui s'appellent Perrot

FB3 : retourne la liste des personnes dont le nom est précisé en argument

FB4 : retourne la première personne qui a X ans (X = un argument) ou nil

FB5 : retourne la première personne qui possède moins de 100 livres ou nil

FB6 : calcule et retourne la moyenne des âges des personnes de la famille X (X = un argument)