TP 2

1. Fonctions récursives sur les listes.

1.1. Définir la fonction *LOC*, qui appliquée à une liste *l* et un élément *e*, renvoie la position "la plus à gauche" de *e* dans *l* si *e* appartient à la liste, en comptant à partir de 1. Que faire dans le cas contraire?

LOC [4 ; 6 ; 7 ; 8] 7 ---> 3

1.2. Définir la fonction *NIEME*, qui pour *n* entier donné, fournit le *n* -ième élément d'une liste, en comptant *n* à partir de 0 pour le premier élément. Prévoir le cas où *n* est trop grand.

NIEME [2 ; 6 ; 5 ; 9 ; 1] 3 ---> 9

1.3. L'itérateur *map2* permet d'appliquer une fonction de 2 arguments en parallele sur deux listes, terme à terme, et constitue la liste des résultats.
Ainsi *map2 f [x1 ; x2 ; . . . ; xn] [y1 ; y2 ; . . . yn] ---> [f x1 y1 ; f x2 y2 . . . . ; f xn yn]*
Ecrire une définition possible de *map2*

2. Ensembles finis représentés par des listes.

On peut représenter un ensemble fini *E = {a , b , c}* par une liste de ses éléments, telle que *L1 = [a ; b ; c]*, mais l'ordre des éléments dans la liste peut être quelconque. Ainsi *L2 = [b ; a ; c]* représente aussi l'ensemble E. Par ailleurs, il ne doit pas y avoir de répétition d'éléments identiques dans la liste.
Par exemple, *L3 = [a ; a ; b ; c]* ne représente pas un ensemble.
On pourra utiliser la fonction *mem* qui teste si un élément appartient à une liste.

2.1. Définir la fonction *EST_ENS* qui teste si une liste représente un ensemble, selon le critère indiqué.

2.2. Définir la fonction *CREER* qui crée un ensemble, à partir d'une liste quelconque.

2.3. Définir l'opérateur ensembliste classique *INTER* qui donne *l'intersection* de deux ensembles. (Cette fonction existe en CaML, sous le nom *intersect*).

2.4 . Définir la relation d'inclusion *INCLUS*, puis l'égalité *EGAL* entre ensembles.

3. Vecteurs et matrices représentés par des listes

Un vecteur peut être représenté par une liste de ses composantes.

3.1. Définir une fonction *prodscal* qui calcule le produit scalaire de deux vecteurs de mêmes dimensions, représentés par des listes de flottants. Prévoir un (ou des) cas d'erreur.

3.2. Définir une fonction NORME qui calcule la norme d'un vecteur (de dimension quelconque) de nombres réels.

NORME [1. ; 3. ; 4. ; 3. ; 1.] ---> $\sqrt{}$ (1. + 9. + 16. + 9. + 1.) = 6.

3.3. Définir la fonction *transpose* qui transpose une matrice d'entiers représentée comme liste de listes
Par exemple: *transpose* [[1; 2; 3] ;
[4 ; 5 ; 6]] ;; ---> [[1; 4];
[2; 5];
[3; 6]]

3.4. Une matrice de longueur *p* et de hauteur *n* peut s'appliquer sur un vecteur *V* de dimension *p*, pour fournir un vecteur résultant *M V* de dimension *n*. Ce vecteur a pour composantes successives les produits scalaires

$L_1 .V$, $L_2 .V$, ... , $L_n .V$

En utilisant la fonction *prodscal*, définir une fonction *appliquemat* qui, étant donnée une matrice *m* de flottants et un vecteur *v* de flottants, applique *m* sur *v*. Donner une solution récursive et une solution avec itérateur.

3.5. Définir la fonction *prodmat* qui calcule le produit de deux matrices quelconques (avec cohérence des dimensions).