***Solution série de TP 4 : Les Listes***

1. ***Ecrire en Prolog le prédicat element, de deux manières : avec et sans le cut (!). element est le prédicat qui permet de savoir si X est un élément de la liste L. Tester la différence entre les deux définitions sur un exemple.***

***Sans le cut :***

* ***element(X, [X|\_]).***
* ***element(X, [\_|T]) :- element(X, T).***

***Avec le cut :***

* ***element(X, [X|\_]) :- !.***
* ***element(X, [\_|T]) :- element(X, T).***

1. ***Définir un prédicat ajoute1(L,L1) où L est une liste de nombres, et L1 une liste identique où tous les nombres sont augmentés de 1.***

* ***ajoute1([], []).***
* ***ajoute1([X], [Y]) :- Y is X + 1, !.***
* ***ajoute1([X|L], [Y|L1]) :- Y is X + 1, ajoute1(L, L1 ).***
* ***?- ajoute1([10, 20, 30], Result).***

1. **Définir le prédicat « suivants(X, Y, L) » où étant donné une liste L, le prédicat renvoie le suivant d’un élément X, avec X et Y se suivent immédiatement dans la liste L.**

* **?- suivants(a,b,[a,b,c]).**
* **true ;**
* **?- suivants(a,X,[a,b,c]).**
* **X = b ;**
* **?- suivants(X,b,[a,b,c]).**
* **X = a ;**
* **?- suivants(X,Y,[a,b,c]).**
* **X = a, Y = b ;**
* ***suivants(X, Y, [X,Y|\_]).***
* ***suivants(X, Y, [\_|L]) :- suivants(X, Y, L).***

1. ***Ecrire le prédicat qui :***
2. ***Donne l’élément maximum d’une liste d’entiers.***

* ***max([X], X).***
* ***max([X|L], X) :- max(L, MaxL), X > MaxL, !.***
* ***max([\_| L], M) :- max(L, M).***
* ***?- max([3, 1, 4, 1, 5, 9, 2, 6], Max).***

1. ***Calcule le nombre N d'occurrences de l'élément X dans la liste L (occ(L,X,N)).***

***Exemple ? occ([z,a,r,a,t],a,N).***

***N=2.***

* ***occ([], \_, 0).***
* ***occ([X|L], X, N) :- occ(L, X, N1), N is N1 + 1.***
* ***occ([Y|L], X, N) :- Y \= X, occ(L, X, N).***

1. ***Supprimer des doublons consécutifs dans une liste L pour obtenir une liste L1. Remarque: L’ordre des éléments doit être respecté.***

***Exemple: ?- compresser([a,a,a,a,b,c,c,a,a,d,e,e,e,e],L1).***

***L1 = [a,b,c,a,d,e].***

* ***compresser([], []).***
* ***compresser([X], [X]).***
* ***compresser([X, X|L1], L2) :- compresser([X|L1], L2).***
* ***compresser([X, Y|L3], [X|L2]) :- X \= Y, compresser([Y|L3], L2).***

1. ***permet de partager une liste en deux parties. On appellera ce prédicat split(L,N,L1,L2) où L est la liste de départ. N est le nombre des éléments dans la première liste L1 et L2 est la seconde liste.***

* ***split(L, 0, [], L).***
* ***split([X|L1], N, [X|L2], L3) :- N > 0, N1 is N - 1, split(L1, N1, L2, L3).***
* ***?- split([a, b, c, d, e, f, g], 3, L1, L2).***

1. **inverse les éléments d’une liste.**

**Exemple:**

**?- renverse([c, b, a, d, b],L1).**

**L1 = [b, d, a, b, c].**

* ***renverse([], []).***
* ***renverse([Tete|Queue], ListeInverse) :- renverse(Queue, QueueInverse), append(QueueInverse, [Tete], ListeInverse).***

1. **Définir le prédicat : partition(X, L, LinfX, LsupX) qui étant donné un nombre X et une liste L, partitionne cette liste en deux listes : LinfX est la liste composée des éléments de L qui sont inférieurs à X, et LsupX est la liste composée des éléments de L qui sont supérieurs ou égaux à X.**

**Exemple :**

**?- partition(4,[3,8,4,1,6,5,2],LinfX,LsupX).**

**LinfX = [3,1,2]**

**LsupX = [8,4,6,5]**

**Yes**

***partition(\_,[],[],[]).***

***partition(X,[Y|L],[Y|Linf],Lsup):- Y=<X, partition(X,L,Linf,Lsup).***

***partition(X,[Y|L],Linf,[Y|Lsup]):- Y>X, partition(X,L,Linf,Lsup).***