

# TD4 – Prolog

## Nathalie Guin & Marie Lefevre

### PARTIE 1 – TRADUCTION D'ÉNONCÉS

**Question 1 :** Traduire en Prolog l'énoncé suivant :

*Marie aime le vin*

*Pierre est un voleur*

*Pierre aime tous ceux qui aiment le vin*

*Si quelqu'un est un voleur et aime quelque chose alors il le vole*

*Qui vole quoi ?*

#### Indices de correction

```

aime(marie, vin).
aime(pierre, X) :- aime(X, vin).
voleur(pierre).
vole(X, Y) :- voleur(X), aime(X, Y).
/* vole(X, Y) donne X=pierre et Y=marie */

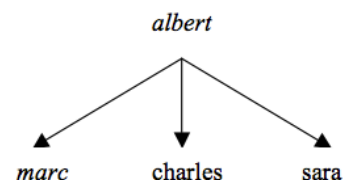
```

**Question 2 :** Soit le programme Prolog suivant :

```

homme(albert).
homme(marc).
homme(charles).
femme(sara).
pere(albert, marc).
pere(albert, charles).
pere(albert, sara).

```



qui décrit l'arbre généalogique suivant :

À partir de ces assertions (faits), définir les prédicats généraux suivants :

- enfant(X, Y) qui exprime que X est un enfant de Y ;
- fils(X, Y) qui exprime que X est un fils de Y ;
- fille(X, Y) qui exprime que X est une fille de Y ;
- frere-ou-sœur(X, Y) qui exprime que X est frère ou sœur de Y. Il est à noter qu'un individu n'est pas son propre frère ou sa propre sœur.

#### Indices de correction

Pour définir enfant, il suffit de remarquer que X est un enfant de Y si Y est le père de X. On a donc la définition :

```

enfant(X, Y) :- pere(Y, X).

```

Ensuite,  $X$  est un fils (respectivement une fille) de  $Y$  si  $X$  est un enfant de  $Y$  et si  $X$  est de sexe masculin (respectivement féminin). On obtient ainsi les deux règles :

```
fils(X, Y) :- enfant(X, Y), homme(X).
fille(X, Y) :- enfant(X, Y), femme(X).
```

Si on admet que  $X$  est le frère ou la sœur de  $Y$  si  $X$  et  $Y$  ont le même père  $Z$ , la relation frere-ou-sœur s'écrit :

```
frere-ou-sœur(X, Y) :- pere(Z, X), pere(Z, Y), dif(X, Y).
```

Le dif permet d'éviter qu'un individu soit son propre frère ou sa propre sœur en forçant  $X$  et  $Y$  à prendre des valeurs différentes. A cet endroit, on aurait aussi pu utiliser  $X \neq Y$  (non unifiable) ou  $X \neq Y$  (2 termes différents).

## PARTIE 2 – RECURSIVITE

**Question 3 :** Définir deux prédicats : l'un affiche les nombres de 1 à  $N$  par ordre croissant, l'autre par ordre décroissant.

==== Indices de correction =====

```
/* Afficher les nombres de 1 à N. */
```

```
croissant(0).
croissant(N) :- N>0, N1 is N-1, croissant(N1), write(N), nl.
croissant2(0):-!.
croissant2(N) :- N1 is N-1, croissant2(N1), write(N), nl.
```

```
/* de N à 1 */
```

```
decroissant(0).
decroissant(N) :- N>0, write(N), nl, N1 is N-1, decroissant(N1).
decroissant2(0):-!.
decroissant2(N) :- write(N), nl, N1 is N-1, decroissant2(N1).
```

**Question 4 :** Définir un prédicat qui calcule récursivement la somme des  $N$  premiers entiers.

==== Indices de correction =====

```
/* Trouver la somme des N premiers entiers.
(Ou : som(N,X) est vrai si X est la somme des entiers de 1 à N.)*/
```

```
som(0,0).
som(N,X) :- N>0, N1 is N-1, som(N1,X1), X is N+X1.
som2(0,0):-!.
som2(N,X) :- N1 is N-1, som2(N1,X1), X is N+X1.
```

## PARTIE 3 – MANIPULATION DE LISTES

**Question 5 :** Écrire un prédicat `compresse` permettant de supprimer des doublons consécutifs dans une liste `L` pour obtenir une liste `L1`. L'ordre des éléments doit être respecté.

Exemple :

```
?- compresse([a,a,a,a,b,c,c,a,a,d,e,e,e,e],L1).
L1 = [a,b,c,a,d,e]
```

==== Indices de correction =====

```
compresse([], []).
compresse([X], [X]).
compresse([X,X|Xs], Zs) :- compresse([X|Xs], Zs).
compresse([X,Y|Ys], [X|Zs]) :- X \= Y, compresse([Y|Ys], Zs).
```

**Question 6 :** Définir le prédicat `sous-ensemble` (`L1`, `L2`) qui est vrai si tous les éléments de la liste `L1` font partie de la liste `L2`.

==== Indices de correction =====

```
sous-ensemble([], _).
sous-ensemble([X|R], L2) :- member(X, L2), sous-ensemble(R, L2).
```

**Question 7 :** Ecrire un prédicat `inverse` permettant de renverser la liste `L`.

==== Indices de correction =====

```
inverse([], []).
inverse([First|Suite], Cible) :
    inverse(Suite, SuiteInversee), append(SuiteInversee, [First], Cible).
```

```
?- inverse([1,2,3,4],L).
L = [4, 3, 2, 1]
?- inverse([1,i,1,i],L).
L = [i, 1, i, 1]
?- inverse(L,[1,2,3]).
L = [3, 2, 1]
```

Deuxième solution : avec un accumulateur (meilleure complexité)

```
inverse2(L1,L2) :- inverse2(L1,[],L2).
inverse2([X|L],Acc,R) :- inverse2(L,[X|Acc],R).
inverse2([],Acc,Acc).
```

**Question 8 :** Définissez le prédicat `sous-liste` (`L1`, `L2`) qui est vrai si la liste `L1` est une sous-liste de la liste `L2`.

## Indices de correction

```
sous-liste(L1,L2) :- append(_,L3,L2), append(L1,_,L3).

?- sous-liste([], [1,2,3,4]).
true.
?- sous-liste([1,2], [1,2,3,4]).
true.
?- sous-liste([3,4], [1,2,3,4]).
true.
?- sous-liste([1,2,3,4], [1,2,3,4]).
true.
?- sous-liste([1,2,4], [1,2,3,4]).
false.
```

## BONUS...

**Question 9 :** Définir le prédicat `longueur(L, N)`, qui étant donnée la liste `L`, calcule sa longueur `N`.

**Question 10 :** Définir le prédicat `concat(L1, L2, L3)` où `L3` est le résultat de la concaténation de `L1` et `L2` (sans utiliser `append`).

**Question 11 :** Définir le prédicat `palindrome(L)` qui retourne vrai si la liste `L` est sa propre image renversée.

**Question 12 :** Définir un prédicat `rang_pair(X, Y)` qui extrait les éléments de la liste `X` qui ont des indices de rang pair afin de construire la liste `Y`.

Ex. `rang_pair([a,b,c,d,e], L) . -> L=[b,d]`

**Question 13 :** Définir le prédicat `indice(X, L, N)`, qui étant donné un élément `X` et une liste `L`, `X` appartenant à `L`, calcule `N` l'indice de la première occurrence de `X` dans `L`. Peut-on utiliser ce prédicat pour formuler une requête permettant de calculer le *i*ème élément d'une liste ?

**Question 14 :** Écrire le prédicat `remplace(X1, X2, L1, L2)` qui construit la liste `L2` qui est la liste `L1` dans laquelle `X1` est remplacé par `X2`.

**Question 15 :** Définir le prédicat `partage(L, X, L1, L2)`, qui étant donné une liste de nombre `L` et un nombre `X`, calcule la liste `L1` des nombres de `L` inférieurs à `X`, et la liste `L2` des nombres de `L` supérieurs ou égaux à `X`.

**Question 16 :** Définir le prédicat `somme(L, R)`, qui étant donnée `L` une liste de nombres `Xi`, calcule la somme des ( $i * X_i$ ).

---

 Indices de correction
 

---

```

/* Longueur d'une liste */
longueur([],0).
longueur([_|Y],N) :- longueur(Y,M), N is M+1.

/* concat (on refait append) */
concat([],L,L).
concat([X|L1],L2,[X|L3]) :- concat(L1,L2,L3).

/* L est une liste palindrome */
palindrome([]).
palindrome([_|_]).
palindrome([X|L]) :- append(L1,[X],L), palindrome(L1).

/* rang_pair(X,Y) extrait les elements de la liste X qui ont des
indices de rang pair afin de construire la liste Y */
rang_pair([],[]).
rang_pair([_|_],[]).
rang_pair([_|Y|L],[Y|L2]) :- rang_pair(L,L2).

/* indice(X,L,N), calcule N l'indice de la première occurrence de X
dans L, X appartenant a L */
indice(X,[X|_],1).
indice(X,[Y|L],N) :- X\==Y, indice(X,L,Nm1), N is Nm1+1.
/* indice(X,[a,b,c,d],2) donne le 2e élément de la liste, i.e. X=b */

/* pour trouver le ieme element d'une liste :
?- indice(X,[a,b,c,d,e],3).
X = c */

/* Remplacement des occurrences de X dans une liste par Y */
remplace(_,_,[],[]).
remplace(X,Y,[X|L1],[Y|L2]) :- replace(X,Y,L1,L2).
remplace(X,Y,[Z|L1],[Z|L2]) :- Z \== X, replace(X,Y,L1,L2).
/* peut aussi être fait avec append.. */

/* partage(L,X,L1,L2), L et X donnees, calcule L1 qui contient les
elements de L inf a X,
et L2 ceux sup ou = a X */
partage([],_,[],[]).
partage([Y|L],X,[Y|L1],L2) :- Y<X, partage(L,X,L1,L2).
partage([Y|L],X,L1,[Y|L2]) :- Y>=X, partage(L,X,L1,L2).

/* Calcul de la somme des (i*Xi) d'une liste
Utilisation d'un accumulateur */
somme(L,R) :- sommebis(L,1,R).
sommebis([],_,0).
sommebis([X|L],I,S2) :- J is I+1, sommebis(L,J,S1), S2 is X*I+S1.

```

---