# **Introduction a prolog**

## **Définition**

* Prolog : est un langage de programmation logique.
* Un programme est un ensemble de fait (prédicat) et de règles, qui forme une base de connaisse, au lieu d’une suite d’instructions
* L’utilisation du programme se fera en **interrogeant** cette base de connaissances.
* Nous utiliserons le logiciel SWI-Prolog
* Il y’a que trois types d’instruction de base dans Prolog :
* Fait : c’est un prédicat qui est toujours vrais
* Règle : c’est un prédicat conditionné.
* Requête/ question : elle sert à interroger la base de connaissance

## **Crier un nouveau fichier**

1. Interpréteur prolog --> File --> New … --> vous insérez un nom « nomfichier.pl » --> Save
2. Charger fichier : il y a plusieurs manières de le faire
   1. Dans le menu de l’éditeur cliquer sur compile --> compile buffer
   2. On se rend dans l’invite prolog, et on tape la commande **[nomFichier].**
   3. Ou bien file🡪 **consult** 🡪 **selectionner votre fichier**.
3. Une fois le fichier est compilé, il est possible d’interroger cette base donner

NB : à chaque fois que on modifier le fichier, vous devez le recompiler

## **Les termes**

### **Constante**

* Nombres : 128 ; 4.5 ; -2
* Chaîne de caractères commençant par une minuscule
* Chaîne de caractères entre **" "**

### **Variable**

* Chaîne de caractères commençant par une majuscule ou \_
* Variable anonyme (ou indéterminée) : **\_**

## **Connaissances**

Un programme en prolog est un ensemble de faits et de règle, qui constitue une base de connaissance, qui décrivent un problème

### **Fait**

* C’est un prédicat
* Syntaxe : son nom commence par une minuscule, ses termes(paramètres) sont séparés par une virgule et se termine par un point
* Exemple

### **Règles**

* C’est un prédicat conditionné
* Syntaxe :
  + : tête de clause,
  + Queue de clause.
  + Intuitivement, signifie que P est vrai s’il est possible de montrer que sont vrais
* Exemple

## **Requête**

* Questions = buts = requête
* Syntaxe : .
* Exemple

# **Corrigé Série TP N°1**

## **Exercice 1**

1. Programmer les connaissances suivantes

* Le père de ali est mohamed
* Le père de said est mohamed
* Si x et Y ont le même père ou la même mère alors ils sont des frères ou des soeurs
* La mère de souad est ferooudja
* La mère de Wissam est ferroudja

|  |
| --- |
| %Le père de ali est mohamed  pere(ali,mohamed).  %Le père de said est mohamed  pere(said,mohamed).  %   Si x et Y ont le même père ou la même mère alors ils sont des frères ou des soeurs  frereSoeur(X,Y):- (pere(X,Z), pere(X,Z)); (mere(X,Z), mere(X,Z)).  %   La mère de souad est ferroudja  mere(souad, ferroudja)  %   La mère de Wissam est ferroudja  mere(wissam,ferroudja). |

1. Ecrire un programme qui déduit tous les frères et soeurs de cette famille.

|  |
| --- |
| ?- frereSoeur(X,Y). |

## **Exercice 2**

Soit le programme Prolog suivant :

homme(ali).

homme(omar).

homme(said).

femme(céline).

femme(lila).

pere(said,ali).

pere(omar,ali).

mere(omar,céline).

mere(said,céline).

parents(X,M,P):-mere(X,M), pere(X,P).

frere(X,Y) :- homme(Y), parents(X,M,P), parents(Y,M,P).

Soit le but suivant : ?-frere(omar,said).

Donner tous les résultats possibles.

|  |
| --- |
| La requete « **?-frere(omar,said).»** signifie « **est ce que le frère de omar est said** »  On aura un seul résultat possible qui est « true » |

# **Les prédicats prédéfinie (Operateur)**

## **Egalité « = »**

* X = Y retourne vrai si X et Y peuvent être unifies
* Exemple :
* En fait Prolog n'a pas calculé le résultat mais a juste unie X avec le terme '5+4' et Y avec '4+5'
* Non-égalité : \=

## **Affectation « is »**

* Le prédicats « **is** » permet de forcer Prolog à évaluer les opérations arithmétiques
* Exemple :

## **Operations.**

|  |  |
| --- | --- |
| Arithmétique | Prolog |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## **Comparison**

|  |  |
| --- | --- |
| Arithmétique | Prolog |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# **Corrigé Série TP N°2**

## **Exercice 1**

Quelles sont les réponses de Prolog aux buts suivants.

?-2=1+1. % **false**, car « 2 » n’est pas unifiable avec « 1+1 »

?-X=2, X=1+1. % **false**, car X ne peut pas avoir deux valeurs en même temps

?- X is 3+4. % **X=7**, il va évaluer l’expression «3+4» ensuite le résultat sera affecté à X

?-X=5, X is 3+3. % **false**, car X ne peut pas avoir deux valeurs en même temps

?-p(X,X)=p(a,b). % **false**, car X ne peut pas avoir deux valeurs en même temps

?-p(X,X)=q(a,b). % **false**, car on a deux prédicat différant

## **Exercice 2**

Soit le programme PROLOG suivant :

mange(omar, pommes).

mange(omar, carottes).

mange(said, bananes).

fruit(pommes).

fruit(bananes).

legume(carottes).

sante(X) :- mange(X, Y), fruit(Y).

Donnez les réponses de PROLOG aux buts suivants :

?- sante(X). % **signifie qui est en bon-santé**

|  |
| --- |
| X=omar ; X=said. |

?- mange(X, pommes). % **signifie qui mange les pommes**

|  |
| --- |
| X=omar . |

?- fruit(X). % **qui est un fruit**

|  |
| --- |
| X=bananes ; X=pommes. |

?- legume(carottes). % **est-ce que les carottes sont des légumes**

|  |
| --- |
| True. |

?- legume(ognon). % **est-ce que ognon est un légume**

|  |
| --- |
| false. |

# **Les Listes**

## **Définition**

* Peut contenir tout type de termes
* Syntaxe : Entre crochets, éléments séparés par des virgules
* Exemple
  + , List vide
* Une liste non-vide peut être représenter par notation structurer : [tête|queue].
  + Tête : c’est le premier élément, il peut être un seul terme ou une suite de terme séparer par des virgules
  + Queue : c’est le reste de la liste, c’est une liste, et elle peut être vide.
  + La liste vide ne peut pas être représentée en notation structurée.

## **Unification des listes**

* **Une liste entière**
* **Certains éléments de la liste**

|  |  |
| --- | --- |
|  |  |
|  |  |  |

* **Avec des listes imbriquées**

|  |  |
| --- | --- |
|  |  |
|  |  |

* **Avec des listes structurées**

|  |  |
| --- | --- |
|  |  |
|  |  |
| . |  |

## **Utilisation de la documentation**

Les arguments des prédicats Prolog prédéfinie sont systématiquement précédés de leur mode d’instanciation, à savoir :

* : l’argument doit être instancie à l’appel (mode input),
* : l’argument doit être libre à l’appel (mode output),
* : l’argument peut être libre ou instancie à l’appel (mode input-output)

# **Corrigé Série TP N°3**

## **Exercice 1**

Quelles sont les réponses de Prolog aux buts suivants :

?- [X,a|Y] = [c,a,a|K].

% **X=c, Y=[a|K].**

?- [X,a|Y] = [c,c,a|K].

% **false**, car ac

?- [1|X] = [1,2,3,5].

% **X=[2,3,5].**

?- [1,2|X] = [1,2].

% **X**=[].

?- [1,2|X] = [1,2,7,3,4].

% **X**=[7,3,4].

?- [X] = [ ].

% **false**, car il n’y a pas une valeur avec la quelle on peut unifie X

?- [X] = [1,2].

% **false**, car X ne peut avoir deux valeurs.

?- [X,Y|Z] = [1,2].

% **X=1, Y=2, Z=[].**

## **Exercice 2**

Ecrire un programme PROLOG pour :

* Calculer la longueur d'une liste

**?- length([a,b,c],X].**

**?- proper\_length([a,b,c],X].**

* Afficher le dernier élément d'une liste

**?- last([a,b,c],X].**

* Inverser une liste d'entiers

**?- reverse([a,b,c],X].**

* Afficher le nième élément d'une liste

**?- nth0(3,[a,b,c,3,i],Elm].**

**?- nth1(4,[a,b,c,3,i],Elm].**

* Supprimer le dernier élément d'une liste

**?- nth1(X,[a,b,u,i], \_,Rest), length([a,b,u,i],X).**

**?- nth0(X,[a,b,u,i], \_,Rest), length([a,b,u,i],Y), X is Y-1.**

* Concaténer deux listes d'entiers

**?- append([1,2],[3,4],L).**

## **Exercice 3**

Ecrire un programme Prolog permettant de trier une liste d'entiers de façon ascendante puis descendante.

**?- msort([1,2,8,9,4,6,3],L).**

**?- msort([1,2,8,9,4,6,3],L), reverse(L,L2).**

## **Question**

* Afficher le dernier élément d’une liste en utilisant le prédicat append

**?- append(\_,[X],[1,2,5,9,4,l,o]).**

* Supprimer une valeur donner dans une liste

**?- delete([1,3,2,5,2,8],2,T).**

# **La récursivité**

|  |
| --- |
| % calculer la longueur d'une liste  longeur([],0).  longeur([\_|L],M):-  longeur(L, N), succ(N, M).  longeur1([],0).  longeur1([\_|L],M):-  longeur1(L, N), M is N + 1.  % afficher le dernier élément d'une liste  dernier([X],X).  dernier([\_|L],X):- dernier(L,X).  dernier1([X]):- write("le dernier element est "), write(X).  dernier1([\_|L]):- dernier1(L).  % inverser une liste d'entiers  inverser([X],[X]).  inverser([T|Q], ListInve):- inverser(Q,L2), append(L2,[T],ListInve).  % affichier une suite numerique  affiche(0) :- write(0).  affiche(N) :- N>0, M is N-1 , affiche(M), write(N).  % afficher le nième élément d'une liste   nieme(1,T ,[ T | \_ ]).   nieme(N ,X ,[ \_ | Q ]) :- nieme(M ,X , Q ) , N is M + 1.  % supprimer une valeur dans une liste  supprimer(\_ ,[] ,[]).  supprimer(X ,[ X | L1 ] , L2 ) :- supprimer(X , L1 , L2 ).  supprimer(X ,[ T | L1 ] ,[ T | L2 ]) :- supprimer(X , L1 , L2 ) , X \= T .  % supprimer le dernier élément d’une liste  supprimer1([\_] ,[]).  supprimer1([ T | L1 ] ,[ T | L2 ]) :- supprimer1( L1 , L2 ).  % concaténer deux listes  conc([] , L , L ).  conc([ T | Q ] ,L ,[ T | QL ]) :- conc(Q ,L , QL ).  % Suite de Fibonacci :  fibonacci(0 ,1).  fibonacci(1 ,1).  fibonacci (N,F) :-          N > 1, N1 is N -1, N2 is N -2,          fibonacci (N1 ,F1), fibonacci (N2 ,F2),          F is F1 + F2. |