Travaux Pratiques 3- Les Listes

Programmation Logique sous l'environnement SWI-Prolog

❖ Introduction :

Prolog donne la possibilité de définir une liste d'éléments ou objets de même type. Une variable de type liste peut être gérée dynamiquement pour faire toute sorte de manipulation : affichage, ajout, suppression, …

Prolog propose deux syntaxes pour représenter des variables de type liste :

Syntaxe 1: Notation d'une constante liste **[e1, e2, …, en],** où les ei sont les éléments de la liste.

Exemples :

- ✓ liste vide L1 = [].
- ✓ liste de type Liste de Caractères (LC) L2 = ['a', 'b', 'c'], write(L2).
- ✓ liste de type Liste de string (LS) L3 = [ali, driss, hamid], write(L3).
- ✓ liste de type Liste de LS (LLS) L4 = [[ali, driss], [petit, grand], [blond]].
- ✓ liste de type Liste d'objets composés (liste de voitures)

  L5 = [voiture("Peugeot 406", "blanche"), voiture("Renault 19", "rouge")].

Syntaxe 2: Ecriture récursive d'une liste **[e | Q]**

avec e l'élément en tête de liste, et Q le reste des éléments de la liste qui doit être une liste.

Exemple : [e1 | [e2, e3, …, en]]

❖ Gestion récursive d'une liste

Plusieurs types de manipulation des listes peuvent être réalisés de manière récursive. Dans cette partie, nous présentons les manipulations de base qui donnent une idée générale sur le principe de la récursivité appliqué aux listes

– Parcours et affichage des éléments d'une liste

- **Prédicat :** afficher(L), avec L est une liste qui peut être écrit sous la forme récursive [X | L1]

```
afficher([]).
afficher([ X | L ] ) :- write(X), nl, afficher(L).
```

- **Exemple:** **?-**afficher([rouge, vert, bleu]).
  rouge
  vert
  bleu

– **Recherche d'un élément dans une liste**

- **Prédicat :** appartient (X, L), avec X est l'élément à chercher dans la liste L

```
appartient (X, [X | _ ] ) .
appartient (X, [ _ | L ] ) :- element(X, L ).
```

- **Exemple:** **?-**appartient (vert, [rouge, vert, bleu]).
  true

– **Insertion en tête**

- **Prédicat :** empiler(X, L1, L2), le premier paramètre X est l'élément à empiler dans la première liste L1. Le résultat est retourné dans la deuxième L2

```
empiler( X, L, [ X | L ] ).
```

- **Exemple:** **?-** empiler(jaune, [rouge, vert, bleu], L).
  L = [jaune, rouge, vert, bleu]

– **Suppression de la tète**

- **Prédicat :** dépiler(L1, L2), L1 est la première liste qui peut être écrite sous la forme suivante [_ | L2], la deuxième liste L2 est la queue de la première L1

```
dépiler ( [] , [] ) .
dépiler ( [ _ | L], L) .
```

- **Exemple:** **?-** dépiler ([rouge, vert, bleu], L)
  L = [vert, bleu]

– **Insertion en queue**

- **Prédicat :** enfiler(X, L1, L2), le premier paramètre est le symbole à enfiler dans la première liste L1. Le résultat est retourné dans la deuxième liste L2.

```
enfiler( X, [], [ X ] ).
enfiler( X, [Y | L1], [Y | L2] ) :- enfiler( X, L1, L2 ).
```

- **Exemple:** **?-** enfiler(jaune, [rouge, vert, bleu], L).
  L = [jaune, rouge, vert, bleu]

– **Suppression de la queue**

- **Prédicat :** défiler(L1, L2), L1 est la première liste qui peut être écrit sous la forme suivante [X1, X2, …, Xn], Xn est le dernier élément supprimé de L1 pour trouver la deuxième liste L2 = [X1, …, Xn-1].

```
défiler ( [ _ ], [] ) .
défiler ( [ X | L1 ], [ X | L2] ) :- supprimer(L1, L2) .
```

- **Exemple:** **?-** défiler ([rouge, vert, bleu], L)
  L = [rouge, vert]

Exercice 1 :

Définir les prédicats suivants :

1. Ecrire un prédicat **vide(L)** pour tester si la liste L est vide.
2. Définir le prédicat **concat(L1,L2,L3)**, où L3 est le résultat de la concaténation de L1 et L2 (sans utiliser le prédicat prédéfini append).
3. Définir le prédicat **longueur(L,N)**, qui étant donnée la liste L calcule sa longueur N.
4. Définir le prédicat **indice(X,L,N)**, qui étant donnés 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 ième élément d'une liste ?
5. 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.

6. Ecrire un prédicat **inserPos(X, P, L1, L2)** qui permet d'insérer l'élément X à la position P dans la première liste L1, le résultat est dans la deuxième liste L2.
7. Ecrire un prédicat **inverser(L1, L2)** qui inverser la liste L1 pour trouver la liste L2.
8. Ecrire un prédicat **dernierElem(L, X)**, dont X est le dernier élément de la liste L.
9. Ecrire un prédicat **saisirListe(L, N)** qui construit une liste L à partir d'une suite de N éléments saisie au clavier. On demande deux versions : une insertion d'éléments dans l'ordre de leur saisie, et une deuxième version dans le sens inverse.
10. Ecrire un prrédicat **nbOccur(X, L, N)** qui calcule dans N le nombre d'occurrence d'un élément X dans la liste L.
11. Ecrire un prédicat **supprimerAllOccur(X, L1, L2)** qui supprime toute occurrence d'un élément X dans une liste L1, le résultat dans la liste L2.
12. Ecrire un prédicat **supprimerPos (X, P, L1, L2)** qui supprime un élément X à la position P d'une liste L1.
13. Ecrire un prédicat **rechercherPos(X, L, P)** qui recherche de la position P d'un élément X dans une liste.
14. Définir le prédicat **partage(L,X,L1,L2)**, qui étant donnés 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.

Exercice 2 : le tri des listes

Version 1 : le tri par sélection

L'objectif de cet exercice est de trier une liste par sélection dont le principe est le suivant :

- On cherche le minimum de la liste à trier
- On le met au début de la liste qu'on construit
- Puis on recommence avec la liste à trier privée du minimum

1. Définir **minimum(L,X)** qui calcule le minimum X de la liste L.
2. Définir le prédicat **enleve(X,L1,L2)**, qui construit la liste L2, qui est la liste L1 à laquelle on a enlevé la première occurrence de l'élément X
3. Définir le prédicat **tri_min(L,Lt)** qui construit la liste triée Lt à partir de la liste de nombres L

4. Définir un prédicat **fusion(L1,L2,L3)** qui construit L3, la liste triée obtenue en faisant la fusion des deux listes triées L1 et L2.

Exemple : fusion([1,3,5,7],[1,2,5,8,12],L3) → L3 = [1,2,3,5,7,8,12]

Version 2 : Le tri à bulles

1. Définir le prédicat **bulle(L1,L2)** qui construit la liste L2 qui est la liste L1 dans laquelle l'élément le plus petit est remonté en première place.
2. Définir le prédicat **tribulle(L1,L2)** qui implémente le tri à bulles.