TD no13	**"Le compte est bon"**

Objectif de la séance :
Résoudre des problèmes d'IA par exploration des solutions à l'aide de prolog

L'objectif de cet exercice est de développer un prédicat qui résout le problème du compte est bon de l'émission télévisuelle « Des chiffres et des lettres ». Comme le rappelle Wikipédia :

« Le but de cette épreuve est d'obtenir un nombre (de 101 à 999) à partir d'opérations élémentaires (Addition "+", Soustraction "−", Multiplication "×", Division "÷") sur des entiers naturels, en partant de nombres tirés au hasard (de 1 à 10, 25, 50, 75 et 100). Lorsque l'émission n'était pas informatisée, le jeu comportait vingt-quatre plaques : les nombres de 1 à 10 présents en double exemplaire et les nombres 25, 50, 75 et 100 présents en un seul exemplaire. Sont alors tirées 6 valeurs. »

Par exemple comment obtenir le nombre 937 à partir des nombres 2, 5, 10, 25, 1, 7 ?

Une solution est :

— $10 \times 5 = 50$
— $50 + 2 = 52$
— $25 - 7 = 18$
— $52 \times 18 = 936$
— $936 + 1 = 937$

Le prédicat `leCompteEstBon(LesNombres, NombreCherche, LesCalculs)` est vrai lorsque les calculs successifs proposés par `LesCalculs` permettent d'obtenir `NombreCherche` à partir de `LesNombres`. Ce prédicat cherchera toutes les solutions. Par exemple :

```
?- leCompteEstBon([2,5,10,25,1,7],937,R).
R = ['10*5=50', '50+2=52', '25-7=18', '52*18=936', '
    936+1=937'] ;
R = ['10*5=50', '25-7=18', '50+2=52', '52*18=936', '
    936+1=937'] ;
R = ['10*7=70', '70+5=75', '75*25=1875', '1875-1=1874', '
    1874/2=937'] ;
R = ['25-7=18', '10*5=50', '50+2=52', '52*18=936', '
    936+1=937'] ;
false.
```

swi-prolog propose nativement les prédicats suivant :

— le prédicat *select/3* tel que `select(Liste,Element,ListeSansUneOccurenceDElement)` est vrai lorsque `ListeSansUneOccurenceDElement` contient les même éléments que `Liste` exceptée une occurrence de `Element` ;
— le prédicat *concat_atom/2* tel que `concat_atom(Liste, Chaine)` est vrai lorsque `Chaine` est la concaténation des représentations en chaine de caractères des atomes de `Liste` ;
— le prédicat *append/3* tel que `append(Liste1, Liste2, Liste)` est vrai lorsque `Liste` est la concaténation de `Liste1` avec `Liste2`.

1. Développez le prédicat *tirerDeuxNombres/4*, tel que `tirerDeuxNombres(LesNombres,Nombre1,Nombre2,LesNombresSansNb1Nb2)` est vrai lorsque `Nombre1` et `Nombre2` sont deux éléments de `LesNombres`, tel que `Nombre1` $\geq$ `Nombre2` et tel que `LesNombresSansNb1Nb2` possède les même éléments que `LesNombres` sans une occurrence de `Nombre1` et `Nombre2`. Testez le :

```
?- tirerDeuxNombres([4,1,3],N1,N2,L).
N1 = 4,
N2 = 1,
L = [3] ;
N1 = 4,
N2 = 3,
L = [1] ;
N1 = 3,
N2 = 1,
L = [4] ;
false.
```

2. Développez une première clause du prédicat *leCompteEstBon/3* qui est vrai lorsque `NombreCherche` appartient à `LesNombres` et dans ce cas `LesCalculs` est associé à la liste vide. Testez la :

```
?- leCompteEstBon([1,5,1],5,[]).
true ;
false.
```

3. Développez une deuxième clause du prédicat *leCompteEstBon/3* qui est vrai lorsque `NombreCherche` est l'addition de deux nombres (le premier plus grand que le second) de `LesNombres` et dans ce cas `LesCalculs` contient le chaine de caractère représentant cette addition. Testez la :

```
?- leCompteEstBon([1,5,2],6,L).
L = ['5+1=6'] ;
false.
```

4. Améliorer la précédente clause pour qu'elle fonctionne aussi dans le cas où l'un des deux nombres a été obtenu par addition. Par exemple :

```
?- leCompteEstBon([1,5,2,3],9,L).
L = ['5+1=6', '6+3=9'] ;
L = ['5+3=8', '8+1=9'] ;
L = ['3+1=4', '5+4=9'] ;
false.
```

5. Ajouter trois clauses pour gérer la soustraction, la multiplication et la division. Attention, dans le cas de la division il ne faut l'accepter que si le reste vaut 0 (opérateur `mod`).