# Examen ELFE

durée 3h – documents de cours/td/tp autorisés
Livres et calculatrices interdits.

## Programmation Logique

Vous disposez de 3h pour traiter la partie "Programmation logique" et la partie "Programmation Fonctionnelle". Vous êtes libre de gérer votre temps comme vous le souhaitez entre ces deux parties.

Cependant il vous faut rendre des **copies séparées** pour chacun des thèmes.

Les questions sont indépendantes, c'est-à-dire que vous pouvez traitez une question sans avoir répondu aux autres, il suffit de supposer comme définis les différents prédicats réponse.

**Exercice 1 :** On sait que :

- une *chaise*, un *tabouret* et un *fauteuil* sont de *siège*s
- une *armoire*, un *placard* et une *commode* sont des *rangement*s
- un *rangement* et un *siège* sont des *meuble*s
- *tab1* et *tab2* sont des tabourets,
- *cha1* et *cha2* sont des chaises,
- *plac* est un placard,
- etc.

Ces informations sont codées par le prédicat **est_un(***?X,?Y***)** dont l'interprétation est "$X$ est un $Y$". On a par exemple les faits :

```
est_un(chaise, siege).
...
est_un(placard,rangement).
...
est_un(siege,meuble).
...
est_un(cha1,chaise).
...
est_un(plac,placard).
```

**Q 1 .** Codez un prédicat **est_un_trans(***X,Y***)** qui généralise le prédicat *est_un* dans la mesure où il en réalise la clôture transitive.
Ainsi *cha1* est une *chaise* qui est un *siège* qui est un *meuble*. Donc *est_un_trans(cha1,meuble)* sera `vrai` (de même que *est_un_trans(cha1,chaise)* et *est_un_trans(cha1,siege)*) . Il en sera de même pour *est_un_trans(plac,rangement)*, etc.

**Exercice 2 : Mener l'ours en cage**
Dans cet exercice, on appellera *mot* tout terme constant. Dans cet exercice, on supposera que tous les mots ont le même nombre de caractères.

Il s'agit d'écrire un programme qui résolve automatiquement le jeu bien connu consistant à aller d'un mot à l'autre en ne changeant qu'une lettre chaque étape. On ne limite pas le nombre d'étapes pour effectuer la transition.

| | | | |
|---|---|---|---|
| o | u | r | s |
| | | | |
| ⋮ | ⋮ | ⋮ | ⋮ |
| | | | |
| c | a | g | e |

$\longrightarrow$

| | | | |
|---|---|---|---|
| o | u | r | s |
| m | u | r | s |
| m | u | r | e |
| m | a | r | e |
| m | a | g | e |
| c | a | g | e |

Le dictionnaire des mots autorisés (en nombre fini) est défini par des faits de la forme `motdico(`$X$`).` On supposera que l'ensemble des faits `motdico` est défini.
Le prédicat prédéfini **atom_chars(***?Mot,?Liste***)** unifie le mot placé en premier argument avec la liste de ses lettres placée en second argument :

```
?- atom_chars(ours,L).
     L = [o,u,r,s] ; No.
?- atom_chars(M,[o,u,r,s]).
      M = ours ; No.
```

**Q 1 .** Codez un prédicat **distance(***+Mot1,+Mot2,?Dist***)** qui permet de connaître dans son troisième argument la distance en nombre de lettres distinctes entre deux mots situés en premier et second argument.

```
?- distance(ours,mure,D).
     D = 2 ; No.
?- distance(ours,roue,D).
     D = 4 ; No.
```

**Q 2 .** Codez un prédicat **voisin(***+Mot1,+Mot2***)** qui soit satisfait si ses deux arguments (des mots du dictionnaire) sont exactement distants d'une lettre.

**Q 3 .** Écrivez un prédicat **cherche(***+Depart,+Arrivee,?Solution***)** dont le rôle est de chercher une solution au problème posé. On passera en paramètre les deux mots de départ et d'arrivée et une variable qui devra s'unifier avec la liste des mots utilisés pour résoudre le problème. Il peut y avoir plusieurs solutions à un même problème.

```
?- cherche(ours,cage,Sol).
     Yes.  Sol = [ours,murs,mure,mare,mage,cage] ;
    Yes.  Sol = ...
```

**Q 4 .** En utilisant les prédicats prédéfinis **display(***Terme***)** qui permet l'affichage de *Terme* et **nl(***q***)** ui permet de passer une ligne, écrivez un prédicat **resoud(***+Mot1,+Mot2***)** qui résoud le problème posé pour les mots *Mot1* et *Mot2*, en ne fournissant qu'**une et une seule** solution, et affiche la solution à raison de 1 mot par ligne.

```
?- resoud(ours,cage).
ours
murs
mure
mare
mage
cage
YES
?-
```

**Exercice 3 :** Soit le programme suivant :

```
p(1).
p(2) :- !.
p(3).
```

Pour chacune des questions suivantes, dessinez l'arbre complet de la résolution PROLOG pour obtenir toutes les réponses de l'interprète.

**Q 1 .** `p(X).`

**Q 2 .** `p(X),p(Y).`

**Q 3 .** `p(X), !, p(Y).`