```
3 % TP : dictionnaire bilingue, stocké en Arbre binaire trié
% Structures :  e(motf,mote,pred,succ)

% Base de donnée stockée

dico(e(avant,before,e(a,at,_,_),e(zoo,zoo,_,_))).

% 1 & 2. rech(Mot,Dico,Traduction)
% Il existe une version avec Cut, plus efficace mais pas plus lisible.

rech(Mot,e(Mot,Trad,_,_),Trad) :- !.
rech(Mot,e(M1,_,Avant,_),Trad) :-
        avant(Mot,M1),
        rech(Mot,Avant,Trad).
rech(Mot,e(M1,_,_,Apres),Trad) :-
        \+ avant(Mot,M1),
        rech(Mot,Apres,Trad).

% Prédicat d'ordre alphabétique : "avant" (déjà vu)

avant(X,Y) :- name(X,L), name(Y,M), avantx(L,M).
avantx([],[_|_]).
avantx([X|_],[Y|_]) :-  X < Y.
avantx([T|Q],[T|S]) :- avantx(Q,S).

% 3. Pretty print qui affiche le dico dans l'ordre alphabétique

pp(e(M,T,G,D)) :- var(G), var(D), !,  write(M), write(' : '), write(T), nl.
pp(e(M,T,G,D)) :- var(G), write(M), write(' : '), write(T), nl, pp(D).
pp(e(M,T,G,D)) :- var(D), pp(G), write(M), write(' : '), write(T), nl.
pp(e(M,T,G,D)) :- pp(G), write(M), write(' : '), write(T), nl, pp(D).

% Le même en plus élégant

pp2(X) :- var(X), !.
pp2(e(M,T,G,D)) :-
        pp2(G),
        write(M), write(' : '), write(T), nl,
        pp2(D).

% Pretty print qui affiche l'arbre.

ppa(X) :- ppa(X,0).
ppa(X,_) :- var(X), !.
ppa(e(M,T,G,D),Prof) :-
        NP is Prof + 3,
        ppa(G,NP),
        tab(Prof), write('e('), write(M), write(','), write(T), write(')'), nl,
        ppa(D,NP).
```



```
% 4. Programme de recherche qui ne fait vraiment que chercher.

find(Mot,e(MotCour,Trad,_,_),Trad) :- Mot == MotCour, !.
find(_,X,_) :- var(X), !, fail.
find(Mot,e(M1,_,Avant,_),Trad) :- avant(Mot,M1),
        find(Mot,Avant,Trad).
find(Mot,e(M1,_,_,Apres),Trad) :- \+ avant(Mot,M1),
        find(Mot,Apres,Trad).

%% Boucle d'interaction.

consult :-
        retractall(dicow(_)), dico(Arbre), assert(dicow(Arbre)),
        % On passe par un prédicat intermédiaire, dicow, sur les
        % systèmes où on ne peut 'retract'-er un prédicat 'consult'-é
        menu, read(X), corps(X).

corps(fin) :- !.
corps(rech) :-
        write('Mot recherché : '), read(Mot),
        dicow(D), corps-r(D,Mot),
        menu, read(Y), !, corps(Y).
corps(inser) :-
        write('Mot à insérer : '), read(Mot),
        write('Traduction : '), read(Trad),
        dicow(D), corps-i(D,Mot,Trad),
        menu, read(Y), !, corps(Y).
corps(affd) :- dicow(A), pp(A),
        menu, read(Y), !, corps(Y).
corps(affa) :- dicow(A), ppa(A),
        menu, read(Y), !, corps(Y).
corps(X) :-
        write('Commande non reconnue'), nl,
        menu, read(Y), !, corps(Y).

corps-r(Arbre, Mot) :-
        find(Mot,Arbre,Trad), !,
        write('Traduction : '), write(Trad), nl.
corps-r(_,_) :-
        write('Désolé, mot absent'), nl.

corps-i(Arbre, Mot, Trad) :-
        retract(dicow(_)),
        rech(Mot,Arbre,Trad),
        assert(dicow(Arbre)), !.
corps-i(_,_,_) :- write('Désolé, problème.'), nl.

menu :-
        write('"fin" : Sortie'), nl,
        write('"inser" : Insertion'), nl,
        write('"rech" : Recherche'), nl,
        write('"affd" : Affichage Dico'), nl,
        write('"affa" : Affichage Arbre'), nl.
```