```
change(you,i).
change(are, [am,not]).
change(french,german).
change(do,no).
change(X,X):-
    !.

alter([],[]).

alter([H|T], [X|Y]):-
    change(H,X),
    alter(T,Y).
```

```
conv_temp(X) :-
    Temp is 9.0/5.0*X+32,
    is_it_hot(Temp).

is_it_hot(T) :-
    T > 90,
    write("Hot temperature today."),
    nl,
    get_input().

is_it_hot(T) :-
    T < 30,
    write("It is cold today"),
    nl,
    get_input().

% Neither hot or cold...
is_it_hot(T) :-
    get_input().

get_input() :-
    write("Please type the temperature in Celcius: "),
    nl,
    read(X),
    conv_temp(X).
```

```
/*
  Taken from:
  http://stackoverflow.com/questions/20256667/prolog-removing-duplicates#20264879
*/

% An empty list is a set.
set([], []).

% Put the head in the result,
% remove all occurrences of the head from the tail,
% make a set out of that.
set([H|T], [H|T1]) :-
    remv(H, T, T2),
    set(T2, T1).

% Removing anything from an empty list yields an empty list.
remv(_, [], []).

% If the head is the element we want to remove,
% do not keep the head and
% remove the element from the tail to get the new list.
remv(X, [X|T], T1) :- remv(X, T, T1).

% If the head is NOT the element we want to remove,
% keep the head and
% remove the element from the tail to get the new tail.
remv(X, [H|T], [H|T1]) :-
    X \= H,
    remv(X, T, T1).
```

```
mymember(X,[X|_]) :- !.
mymember(X,[_|T]) :- mymember(X,T).

set([],[]).
set([H|T],[H|Out]) :-
    not(mymember(H,T)),
    set(T,Out).
set([H|T],Out) :-
    mymember(H,T),
    set(T,Out).
```

```
main:-
        open('input.txt',read,Str),
        read_from_file(Str, Lines),
        close(Str),
        write(Lines),  nl.



read_from_file(Stream, [H|T]):-
         \+ at_end_of_stream(Stream),
         read(Stream,H),
         read_from_file(Stream, T).

read_from_file(Stream,[]):-
         !.
```

```
insert(X,[],[X]):-!.
insert(X,[H|T],[X,H|T]):-
       X=<H,
       !.
insert(X,[H|T],[H|T1]):-
       insert(X,T,T1).
```

```
/*Check if L2 is in the form of [Prefix, L1, Suffix]*/
includes(L1,L2):-
    append([_,L1,_],L2),!. /*We use ! so that the procedure stops before it fails*/
```

```
/*Check if the head of the first list is in L.
For H to be in L, L has to be in the form of [_,[H],_].*/
common([H|T],L):-
    append([_,[H],_],L);
    common(T,L), /*Check for tail*/
    !. /*We use ! to stop the procedure from failing*/
```

```
pairs_list([], []).
pairs_list([First, Second | Tail], [[First, Second] | Rest]) :-
    pairs_list(Tail, Rest).
```

```
precedes([],_). /*The first list is empty. Accept.*/

precedes([H|T1],[H|T2]):-
    /*The heads are the same. Check for the tails.*/
    precedes(T1,T2).
```

```
find_last([X], X).

find_last([H|T],X):-
    find_last(T,X).

proceed_list([],[]):-!.
proceed_list([H|T],[]):-!.

proceed_list([H|T],[L|R]):-
    find_last([H|T], X),
    X =< L,
    !.
```

```
removeone(_, [], []):-
    !.

removeone(X,[X|Tail], Tail):-
    !.

removeone(X, [H|T], [H|T1]):-
    X\=H,
    removeone(X,T,T1),
    !.
```