LAB 1

1) Write the max predicate that calculates the maximum between 2 values.

```
max(X,Y,Y):-X<Y,!.
max(X,_,X).                %X>=Y is implicit due to the ! above
%?-max(3,4,X).
```

2) Write the *member* and *concat* predicates.

```
member(X,[X|_]).   % member(X,[X|_]):-!.     What happens if ! is used?
member(X,[_|W]):- member(X,W).
%?-member(2,[1,2,3]).
%?-member(X,[1,2,3]).     %press ; to get all solutions


conc([],L,L).
conc([A|B],M,[A|P]):-conc(B,M,P).  %recursion by the length of the first list
%?-conc([1,2,3],[4,5],L).
```

3) Calculate the alternate sum of the elements of a list.

```
altsum([],0).
altsum([X],X).
altsum([A,B|C],S):-altsum(C,S1),S is A-B+S1.   %reduce the problem to a similar (but smaller) one
%?-altsum([1,2,3,4,5],P).
```

4) Eliminate an element from a list (one/all the occurrences of that element).

```
%del(_,[],[]).      %check with and without this rule, when X is (not) in the list. Sometimes you want
                      %the predicate to be evaluated true, sometimes false
del(X,[X|L],L):-!.    %check with and without !, when X appears multiple times in the list
del(X,[Y|L],[Y|L1]):-del(X,L,L1).
%?-del(3,[1,2,3,4,3],L)
```

```
del_all(_,[],[]).
del_all(X,[X|L],L1):-!,del_all(X,L,L1).  %check with and without ! by pressing ;
del_all(X,[Y|L],[Y|L1]):-del_all(X,L,L1).
```

5) Reverse a list; generate all the permutations of the elements of a list.

```
reverse([],[]).
reverse([A|B],C):-reverse(B,B1), append(B1,[A],C).  %it would not be correct C=[B1|A] instead of
                                                    %append(B1,[A],C) because A is not a list, but
                                                    %an item in a list

perm([],[]).
perm([A|B],C):-perm(B,B1),insert(A,B1,C). %reduce the permutation of order n to permutations of
                                          %order n-1, then build the permutation of order n by
                                          %using the predicate insert

insert(A,B,[A|B]).
insert(A,[B|C],[B|P]):-insert(A,C,P).
%?-perm([1,2,3],L).   %press ; to get all permutations
%?-insert(6,[1,2,3],L). %press ; to see what happens
```

6) Find the number of occurrences of an element in a list.

```
occur(X,L,0):- not(member(X,L)),!.    %without !, this rule may overlap with the 3rd rule, which lead
                                      %to multiple (same) solutions
occur(X,[X|L],P):-!,occur(X,L,P1),P is P1+1.
occur(X,[Y|L],P):-occur(X,L,P).   %this rule is available only if the ! above is not executed, i.e., X is
                                  %not the first element in the list

%?-occur(3,[1,2,3,4,5,6,3,7],P).
```

7) Insert an element on a certain position in a list.

```
ins_pos(X,1,L,[X|L]).
ins_pos(X,P,[A|L],[A|L1]):-P>1,P1 is P-1, ins_pos(X,P1,L,L1).
%?- ins_pos(4,2,[6,7,8,9,3,2,1],L).
```

8) Merge two ascending ordered lists.

```
merge([],L,L):-!.
merge(L,[],L).
merge([A|B],[C|D],[A|E]):-A<C,!,merge(B,[C|D],E).
merge([A|B],[C|D],[C|E]):-merge([A|B],D,E).    %A>=C is implicit due to the ! above
%?- merge([1,4,6,7],[4,5,8,9,10],L).
```

LAB 2

1) Compute the greatest common divisor of two integers – gcd/3;

```
gcd(0,X,X):-!.
gcd(X,0,X):-!.
gcd(A,B,C):-A=<B,!, D is B-A, gcd(D,A,C).
gcd(A,B,C):- D is A-B, gcd(D,B,C).    %A>B is implicit due to ! above
```

2) Split a list into 2 lists, according to a given value: the elements that are less than the value are placed in the first list and the elements greater or equal than the value are placed in the second list – split/4;

```
split([],H,[],[]).
split([H1|T],H,[H1|A],B):-H1=<H,!, split(T,H,A,B).
split([H1|T],H,A,[H1|B]):- split(T,H,A,B).
%?-split([3,2,6,1,5],4,L1,L2)
```

3) Insertion sort and quick sort – insertsort/2 and quicksort/2;

```
insertsort([],[]).
insertsort([X|T],S):-insertsort(T,ST),insertOrder(X,ST,S).
%?-insertsort([4,3,5,2],L).

%insertOrder(Elem,OrderedList,OrderedList1) -- insert the element Elem on its
right %position in OrderedList and the result is OrderedList1
%Example. Elem=3, OrderedList=[1,2,6,9] => OrderedList1=[1,2,3,6,9]

insertOrder(X,[],[X]).
insertOrder(X, [Y|T], [X,Y|T]):-X=<Y,!.
insertOrder(X, [Y|T], [Y|T1]):-insertOrder(X,T,T1).

quick([],[]).
quick([X],[X]):-!.
quick([H|T],L):-split(T,H,A,B),quick(A,A1),quick(B,B1),conc(A1,[H],C),
                        conc(C,B1,L).  %or append instead of conc
```

4) The 8 queens' problem – queen/1 - the solution may be represented as a list of lists. For example, [[**1**,3],[**2**,5],[**3**,7],…,[**8**,1]] represents the configuration where the 1st queen is on the 1st column and 3rd row, the 2nd queen on the 2nd column and the 5th row, …, the 8th queen is on the 8th column and 1st row;

```
queen([]).
queen([[X,Y]|S]):-queen(S),member(Y,[1,2,3,4,5,6,7,8]), not(atack([X,Y],S)).
atack([X,Y],S):-member([X1,Y1],S), (Y==Y1;abs(X-X1)=:=abs(Y-Y1)).
%?-queen([[1,X1],[2,X2],[3,X3],[4,X4],[5,X5],[6,X6],[7,X7],[8,X8]]).
```

Obs. If you need to trace your program, write 'trace.' in the interrogation window, then execute the predicate. To stop tracing, write 'notrace.'

For more on this, please check https://www.swi-prolog.org/pldoc/man?section=debugger

Other exercises for practice:

1) Find the position of an element in a list – e.g., for 3 and [2,5,1,3,8] the position is 4
2) Duplicate all the elements of a list – [1,2,3] -> [1,1,2,2,3,3]
3) Multiply the elements found on odd-number positions in a list by 2 – [1,2,3,4,5] -> [2,2,6,4,10]
4) Find the intersection/the difference of two lists – for [1,3,5,7] and [3,6,8], the intersection is [3] and the difference is [1,5,7]
5) Remove the multiple occurrences (if any) of the elements of a list – [1,2,4,1,3,2,1] -> [1,2,4,3]