CSc 600-01 (Section 1)

Homework 3 - Declarative Programming

prepared by Ilya Kopyl

CSc 600 Homework 3 - Declarative Programming in Prolog

March 16, 2018

Homework is prepared by: Ilya Kopyl.
It is formatted in LaTeX, using TeXShop editor (under GNU GPL license).
Diagrams are created in LucidChart online editor (lucidchart.com).

1. Write a PROLOG program that investigates family relationships using lists. The facts should be organized as follows:

```
m([first_male_name, second_male_name, ..., last_male_name]).
f([first_female_name, second_female_name, ..., last_female_name]).
family( [father, mother, [child1, child2, ..., child_n]] ).
```

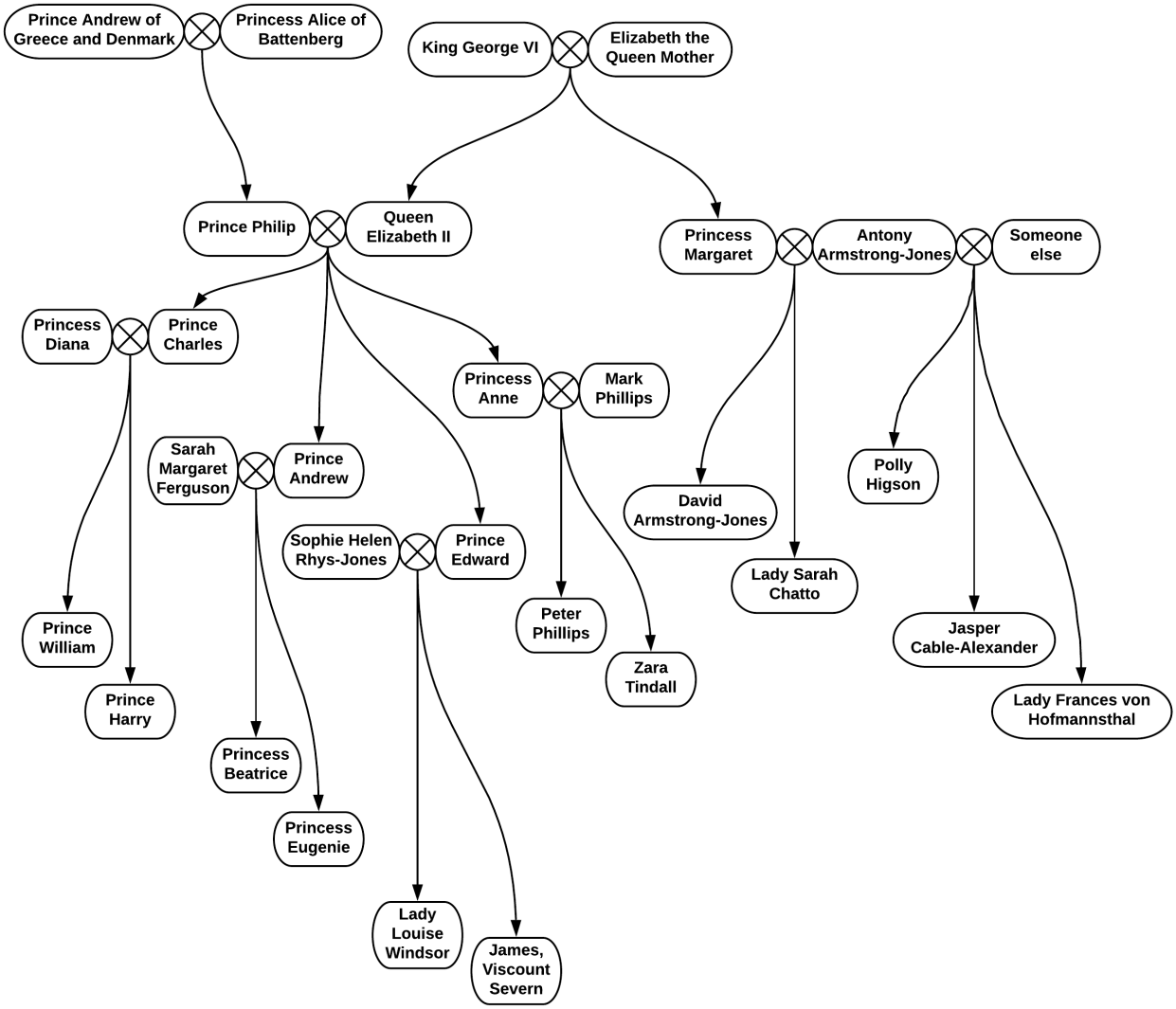
Write rules that define the following relationships:

```
male(X)
female(X)
father, mother, parent
siblings1, siblings2
brother1, brother2
sister1, sister2
cousins
uncle, aunt
grandchild, grandson, granddaughter
greatgrandparent
ancestor
```

For each of these rules show an example of its use.

The answer is listed on pages 2 through 13.

Partial diagram of the family tree of the British Royal family.



The code listing of the entire program 1:

```
% Facts:

m([
    'Prince Andrew of Greece and Denmark',
    'King George VI',
    'Prince Philip',
    'Antony Armstrong-Jones',
    'Prince Charles',
    'Mark Philips',
    'Prince Andrew',
    'David Armstrong-Jones',
    'Prince Edward',
    'Peter Philips',
    'Jasper Cable-Alexander',
    'Prince William',
    'Prince Harry',
    'James, Viscount Severn'
]).

f([
    'Princess Alice of Battenberg',
    'Elizabeth the Queen Mother',
    'Queen Elizabeth II',
    'Princess Margaret',
    'Someone else',
    'Princess Diana',
    'Princess Anne',
    'Polly Higson',
    'Sarah Margaret Ferguson',
    'Sophie Helen Rhys-Jones',
    'Lady Sarah Chatto',
    'Zara Tindall',
    'Lady Frances von Hofmannsthal',
    'Princess Beatrice',
    'Princess Eugenie',
    'Lady Louise Windsor'
]).

family([
    'Prince Andrew of Greece and Denmark',
    'Princess Alice of Battenberg',
    [
        'Prince Philip'
    ]
]).
```

```
family([
    'King George VI',
    'Elizabeth the Queen Mother',
    [
        'Queen Elizabeth II',
        'Princess Margaret'
    ]
]).

family([
    'Prince Philip',
    'Queen Elizabeth II',
    [
        'Prince Charles',
        'Princess Anne',
        'Prince Andrew',
        'Prince Edward'
    ]
]).

family([
    'Antony Armstrong-Jones',
    'Princess Margaret',
    [
        'David Armstrong-Jones',
        'Lady Sarah Chatto'
    ]
]).

family([
    'Antony Armstrong-Jones',
    'Someone else',
    [
        'Polly Higson',
        'Jasper Cable-Alexander',
        'Lady Frances von Hofmannsthal'
    ]
]).

family([
    'Prince Charles',
    'Princess Diana',
    [
        'Prince William',
        'Prince Harry'
    ]
]).
```

```
family([
    'Mark Philips',
    'Princess Anne',
    [
        'Peter Philips',
        'Zara Tindall'
    ]
]).

family([
    'Prince Andrew',
    'Sarah Margaret Ferguson',
    [
        'Princess Beatrice',
        'Princess Eugenie'
    ]
]).

family([
    'Prince Edward',
    'Sophie Helen Rhys-Jones',
    [
        'Lady Louise Windsor',
        'James, Viscount Severn'
    ]
]).

% Rules:

% X is male if it is a member of a list m.
male(X) :-
    m(List),
    member(X, List), !.     % the first match would suffice

% Y is female if it is a member of a list f.
female(Y) :-
    f(List),
    member(Y, List), !.

% parent is either a first or second element of the list family
parent(X, Child) :-
    family([X, _, Children]),
    member(Child, Children);
    family([_, X, Children]),
    member(Child, Children).
parent(X) :-
    parent(X, _), !.
```

```
% X is father if X is a parent and is male.
father(X) :-
    parent(X),
    male(X).

% Y is a mother if Y is a parent and is female.
mother(Y) :-
    parent(Y),
    female(Y).

% X and Y are siblings2 if they have both parents in common
siblings2(X, Y) :-
    family([_,_,Children]),
    member(X, Children),
    member(Y, Children),
    X \= Y.

% X and Y are siblings1 if they come from
% different families, but have 1 parent in common.
siblings1(X, Y) :-
    family([Father,_,Children1]),
    family([Father,_,Children2]),
    Children1 \= Children2,
    member(X, Children1),
    member(Y, Children2).

siblings1(X, Y) :-
    family([_,Mother,Children1]),
    family([_,Mother,Children2]),
    Children1 \= Children2,
    member(X, Children1),
    member(Y, Children2).

% X is brother1 to Y if X is sibling1 to Y and is male.
brother1(X, Y) :-
    siblings1(X, Y),
    male(X).

% Y is sister1 to Y if X is sibling1 to Y and is female.
sister1(X, Y) :-
    siblings1(X, Y),
    female(X).

% X is brother2 to Y if X is sibling2 to Y and is male.
brother2(X, Y) :-
    siblings2(X, Y),
    male(X).
```

```
% X is sister2 to Y if X is sibling2 to Y and is female.
sister2(X, Y) :-
    siblings2(X, Y),
    female(X).

% X and Y are cousins if a Parent of X is a sibling2 to a Parent of Y
cousins(X, Y) :-
    parent(Parent1, X),
    parent(Parent2, Y),
    siblings2(Parent1, Parent2).

% X is uncle of Y if X is a brother of a parent of Y
uncle(X, Y) :-
    brother2(X, Parent),
    parent(Parent, Y).

% X is aunt of Y if X is a sister of a parent of Y
aunt(X, Y) :-
    sister2(X, Parent),
    parent(Parent, Y).

% child X of Y if Y is a father of X
child(X, Y) :-
    parent(Y, X).

% X is grandchild of Y if Y is a parent of a parent of X.
grandchild(X, Y) :-
    parent(Parent, X),
    parent(Y, Parent).

% X is grandson of Y if X is grandchild of Y and is male.
grandson(X, Y) :-
    grandchild(X, Y),
    male(X).

% X is a granddaughter of Y if X is granddaughter of Y and if female.
granddaughter(X, Y) :-
    grandchild(X, Y),
    female(X).

% X is a grandparent of Y if X is a parent of a parent of Y
grandparent(X, Y) :-
    parent(X, Parent),
    parent(Parent, Y).

% X is a greatgrandparent of Y if X is a parent of a grandparent of Y
greatgrandparent(X, Y) :-
    parent(X, Parent),
    grandparent(Parent, Y).
```

```
% Base case: X is ancestor of Y if X is a parent of Y
ancestor(X, Y) :-
    parent(X, Y).

% Recursive defintion:
ancestor(X, Y) :-
    parent(X, Parent),
    ancestor(Parent, Y).
```

Examples of use of each rule in problem 1:

```
?- male('Prince Philip').
true.

?- male('Princess Diana').
false.

?- female('Princess Diana').
true.

?- female('Prince Philip').
false.

?- parent('Prince Philip').
true.

?- parent('Princess Beatrice').
false.

?- parentof('Prince Philip', 'Prince Charles').
true.

?- parentof('Princess Diana', 'Prince Harry').
true.

?- parentof('Princess Margaret', 'Prince Charles').
false.

?- mother('Prince William').
false.

?- father('Ilya').
false.

?- mother('Prince Philip').
false.

?- father('Prince Philip').
true.
```

```
?- mother('Princess Diana').
true.

?- father('Princess Diana').
false.

?- siblings2('Princess Diana', 'Prince Philip').
false.

?- siblings2('Prince William', 'Prince Harry').
true ;
false.

?- siblings2('Prince Charles', X).
X = 'Princess Anne' ;
X = 'Prince Andrew' ;
X = 'Prince Edward' ;
false.

?- siblings1(X, Y).
X = 'David Armstrong-Jones',
Y = 'Polly Higson' ;
X = 'David Armstrong-Jones',
Y = 'Jasper Cable-Alexander' ;
X = 'David Armstrong-Jones',
Y = 'Lady Frances von Hofmannsthal' ;
X = 'Lady Sarah Chatto',
Y = 'Polly Higson' ;
X = 'Lady Sarah Chatto',
Y = 'Jasper Cable-Alexander' ;
X = 'Lady Sarah Chatto',
Y = 'Lady Frances von Hofmannsthal' ;
X = 'Polly Higson',
Y = 'David Armstrong-Jones' ;
X = 'Polly Higson',
Y = 'Lady Sarah Chatto' ;
X = 'Jasper Cable-Alexander',
Y = 'David Armstrong-Jones' ;
X = 'Jasper Cable-Alexander',
Y = 'Lady Sarah Chatto' ;
X = 'Lady Frances von Hofmannsthal',
Y = 'David Armstrong-Jones' ;
X = 'Lady Frances von Hofmannsthal',
Y = 'Lady Sarah Chatto' ;
false.
```

```
?- brother1(X, Y).
X = 'David Armstrong-Jones',
Y = 'Polly Higson' ;
X = 'David Armstrong-Jones',
Y = 'Jasper Cable-Alexander' ;
X = 'David Armstrong-Jones',
Y = 'Lady Frances von Hofmannsthal' ;
X = 'Jasper Cable-Alexander',
Y = 'David Armstrong-Jones' ;
X = 'Jasper Cable-Alexander',
Y = 'Lady Sarah Chatto' ;
false.

?- sister1(X, Y).
X = 'Lady Sarah Chatto',
Y = 'Polly Higson' ;
X = 'Lady Sarah Chatto',
Y = 'Jasper Cable-Alexander' ;
X = 'Lady Sarah Chatto',
Y = 'Lady Frances von Hofmannsthal' ;
X = 'Polly Higson',
Y = 'David Armstrong-Jones' ;
X = 'Polly Higson',
Y = 'Lady Sarah Chatto' ;
X = 'Lady Frances von Hofmannsthal',
Y = 'David Armstrong-Jones' ;
X = 'Lady Frances von Hofmannsthal',
Y = 'Lady Sarah Chatto' ;
false.

?- brother2(X,Y).
X = 'Prince Charles',
Y = 'Princess Anne' ;
X = 'Prince Charles',
Y = 'Prince Andrew' ;
X = 'Prince Charles',
Y = 'Prince Edward' ;
X = 'Prince Andrew',
Y = 'Prince Charles' ;
X = 'Prince Andrew',
Y = 'Princess Anne' ;
X = 'Prince Andrew',
Y = 'Prince Edward' ;
X = 'Prince Edward',
Y = 'Prince Charles' ;
X = 'Prince Edward',
Y = 'Princess Anne' ;
X = 'Prince Edward',
Y = 'Prince Andrew' ;
X = 'David Armstrong-Jones',
```

```
Y = 'Lady Sarah Chatto' ;
X = 'Jasper Cable-Alexander',
Y = 'Polly Higson' ;
X = 'Jasper Cable-Alexander',
Y = 'Lady Frances von Hofmannsthal' ;
X = 'Prince William',
Y = 'Prince Harry' ;
X = 'Prince Harry',
Y = 'Prince William' ;
X = 'Peter Philips',
Y = 'Zara Tindall' ;
X = 'James, Viscount Severn',
Y = 'Lady Louise Windsor' ;
false.

?- sister2(X,Y).
X = 'Queen Elizabeth II',
Y = 'Princess Margaret' ;
X = 'Princess Margaret',
Y = 'Queen Elizabeth II' ;
X = 'Princess Anne',
Y = 'Prince Charles' ;
X = 'Princess Anne',
Y = 'Prince Andrew' ;
X = 'Princess Anne',
Y = 'Prince Edward' ;
X = 'Lady Sarah Chatto',
Y = 'David Armstrong-Jones' ;
X = 'Polly Higson',
Y = 'Jasper Cable-Alexander' ;
X = 'Polly Higson',
Y = 'Lady Frances von Hofmannsthal' ;
X = 'Lady Frances von Hofmannsthal',
Y = 'Polly Higson' ;
X = 'Lady Frances von Hofmannsthal',
Y = 'Jasper Cable-Alexander' ;
X = 'Zara Tindall',
Y = 'Peter Philips' ;
X = 'Princess Beatrice',
Y = 'Princess Eugenie' ;
X = 'Princess Eugenie',
Y = 'Princess Beatrice' ;
X = 'Lady Louise Windsor',
Y = 'James, Viscount Severn' ;
false.

?- cousins('Prince Charles', X).
X = 'David Armstrong-Jones' ;
X = 'Lady Sarah Chatto' ;
false.
```

```
?- uncle('Prince Charles', X).
X = 'Peter Philips' ;
X = 'Zara Tindall' ;
X = 'Princess Beatrice' ;
X = 'Princess Eugenie' ;
X = 'Lady Louise Windsor' ;
X = 'James, Viscount Severn' ;
false.

?- aunt('Princess Margaret', X).
X = 'Prince Charles' ;
X = 'Princess Anne' ;
X = 'Prince Andrew' ;
X = 'Prince Edward' ;
false.

?- child('Queen Elizabeth II', X).
X = 'King George VI' ;
X = 'Elizabeth the Queen Mother' ;
false.

?- grandchild('Prince Charles', X).
X = 'Prince Andrew of Greece and Denmark' ;
X = 'Princess Alice of Battenberg' ;
X = 'King George VI' ;
X = 'Elizabeth the Queen Mother' ;
false.

?- grandson(X, 'Queen Elizabeth II').
X = 'Prince William' ;
X = 'Prince Harry' ;
X = 'James, Viscount Severn' ;
X = 'Peter Philips' ;
false.

?- granddaughter(X, 'Queen Elizabeth II').
X = 'Princess Beatrice' ;
X = 'Princess Eugenie' ;
X = 'Lady Louise Windsor' ;
X = 'Zara Tindall' ;
false.

?- grandparent(X, 'Prince Andrew').
X = 'Prince Andrew of Greece and Denmark' ;
X = 'King George VI' ;
X = 'Princess Alice of Battenberg' ;
X = 'Elizabeth the Queen Mother' ;
false.
```

```
?- greatgrandparent(X, 'Princess Beatrice').
X = 'Prince Andrew of Greece and Denmark' ;
X = 'King George VI' ;
X = 'Princess Alice of Battenberg' ;
X = 'Elizabeth the Queen Mother' ;
false.

?- ancestor('Queen Elizabeth II', X).
X = 'Prince Charles' ;
X = 'Princess Anne' ;
X = 'Prince Andrew' ;
X = 'Prince Edward' ;
X = 'Prince William' ;
X = 'Prince Harry' ;
X = 'Peter Philips' ;
X = 'Zara Tindall' ;
X = 'Princess Beatrice' ;
X = 'Princess Eugenie' ;
X = 'Lady Louise Windsor' ;
X = 'James, Viscount Severn' ;
false.
```

2. Write a PROLOG program that includes the following operations with lists:

```
membership testing (is an element member of a list?)
first element
last element
two adjacent elements
three adjacent elements
append list1 to list2 producing list3
delete element from a list
append element to a list
insert element in a list
compute the length of list
reverse a list
check whether a list is a palindrome
display a list
```

For each of these operations write your implementation of the operation and show an example of its use. If a predicate already exists (predefined in Prolog), modify its name (e.g. myappend or append1). Lists to be processed can be created by an auxiliary program, defined as facts, or entered from the keyboard.

The answer is listed on pages 14 through 17.

The code listing of the entire program 2.

```
% Operations on lists.

% append list1 to list2 producing list3
% Base case: if List1 is empty list, then result is List2.
% Otherwise, if List1 is not empty, chop the head from both
% List1 & List3 until you approach the base case.
append1([], L2, L2).
append1([H|T1], L2, [H|T3]) :-
    append1(T1, L2, T3).

% auxiliary definition
prefix1(Prefix, List) :-
    append1(Prefix, _, List).

% auxiliary definition
suffix(Suffix, List) :-
    append1(_, Suffix, List).

% auxiliary definition
% sublist is a prefix of a suffix of a list
sublist(Sublist, List) :-
    prefix(Sublist, Suffix),
    suffix(Suffix, List).

% membership testing (checks if an element is a member of a list)
member1(X, List) :-
    sublist([X], List).

% first element of the list
first_element(X1, List) :-
    prefix([X1], List).

% last element of the list
last_element(Xn, List) :-
    suffix([Xn], List).

% two adjacent elements in the list
two_adjacent(X, Y, List) :-
    sublist([X, Y], List).

% three adjacent elements in the list
three_adjacent(X, Y, Z, List) :-
    sublist([X, Y, Z], List).

% auxiliary definition
select1(X, [X|Xs], Xs).
select1(X, [Y|Ys], [Y|Zs]) :-
    select1(X, Ys, Zs).
```

```
% delete an element from a list (delete 1 occurrence)
del1(X, List, Result) :-
    select1(X, List, Result).

% delete element from a list (delete all occurrences)
del_all(_, [], []).
del_all(H, [H|T], Result) :-
    del_all(H, T, Result).
del_all(X, [H|T1], [H|T2]) :-
    H=\=X,
    del_all(X, T1, T2).


% append element to a list
append_element(X, List, Result) :-
    append1(List, [X], Result).


% insert element in a list
insert_element(X, List, Result) :-
    select1(X, Result, List).


% compute the length of a list
len(L, N) :-
    len(L, 0, N).
len([], N, N).
len([_|Tail], N0, N) :-
    N1 is N0 + 1,
    len(Tail, N1, N).


% reverse a list in linear time:
reverse1(List, Reversed) :-
    reverse1(List, [], Reversed).
reverse1([], Reversed, Reversed).
reverse1([H|T], Acc, Reversed) :-
    reverse1(T, [H|Acc], Reversed).

% check whether a list is a palindrome
palindrome(List) :-
    reverse1(List, List).


% display a list
puts([]).
puts([H|T]) :-
    write(H),
    write(' '),
    puts(T).
```

Examples of use of each rule in problem 2:

```
?- append1(X, Y, [1,2,3]).
X = [],
Y = [1, 2, 3] ;
X = [1],
Y = [2, 3] ;
X = [1, 2],
Y = [3] ;
X = [1, 2, 3],
Y = [] ;
false.

?- prefix1(X, [1,2,3]).
X = [] ;
X = [1] ;
X = [1, 2] ;
X = [1, 2, 3] ;
false.

?- suffix(X, [1,2,3]).
X = [1, 2, 3] ;
X = [2, 3] ;
X = [3] ;
X = [] ;
false.

?- member1(X, [1,2,3]).
X = 1 ;
X = 2 ;
X = 3 ;
false.

?- first_element(X, [1,2,3]).
X = 1.

?- last_element(X, [1,2,3]).
X = 3 ;
false.

?- two_adjacent(X, Y, [1,2,3,4]).
X = 1,
Y = 2 ;
X = 2,
Y = 3 ;
X = 3,
Y = 4 ;
false.
```

```
?- three_adjacent(X, Y, Z, [1,2,3,4,5]).
X = 1,
Y = 2,
Z = 3 ;
X = 2,
Y = 3,
Z = 4 ;
X = 3,
Y = 4,
Z = 5 ;
false.

?- del1(1, [1,2,3,21,1,2,1], X).
X = [2, 3, 21, 1, 2, 1] ;
X = [1, 2, 3, 21, 2, 1] ;
X = [1, 2, 3, 21, 1, 2] ;
false.

?- del_all(3, [1,2,3,2,3,4,3,5,3,1], X).
X = [1, 2, 2, 4, 5, 1] ;
false.

?- append_element(213, [1,2,3], X).
X = [1, 2, 3, 213].

?- insert_element(2, [1,4,9], X).
X = [2, 1, 4, 9] ;
X = [1, 2, 4, 9] ;
X = [1, 4, 2, 9] ;
X = [1, 4, 9, 2] ;
false.

?- len([1,2,3,2,5,1], N).
N = 6.

?- reverse1([1,2,3,4,5,6,7], X).
X = [7, 6, 5, 4, 3, 2, 1].

?- palindrome([1,2]).
false.

?- palindrome([1,1]).
true.

?- palindrome([1,0,0,1,0,0,1]).
true.

?- puts(['Apple', 'Banana', 'Kiwi', 'Mango', 'Papaya']).
Apple Banana Kiwi Mango Papaya
true.
```

3. Write a PROLOG program that solves the 8 queens problem (location of 8 queens on a chess board so that no queens have each other in check, i.e. are not located in the same row/column/diagonal).

The code listing of the entire program 3:

```
% N Queens problem.
%
% Predicates nocheck and legal initially were
% intellectual property of Adam Brooks Webber,
% but I took some liberty and modified them to
% solve N Queens problem.
%
% I also took a liberty of representing
% the queen's position with just one number -
% the second coordinate is implied by the index
% of that element in the list.

% nocheck(Y,L) takes a queen Y and a list
% of other queens. Succeeds if and only if
% the Y queen holds none of the others in
% check.
nocheck(_, [], _) :- !.
nocheck(Y, [Y1 | Rest], N) :-
    Y =\= Y1,
    abs(Y1-Y) =\= N,
    N1 is N + 1,
    nocheck(Y, Rest, N1).


legal([], _).
legal([Y | Rest], NumList) :-
    legal(Rest, NumList),
    member(Y, NumList),
    nocheck(Y, Rest, 1).


n_queens(N, L) :-
    natural_number(N),
    numlist(1, N, NumList),
    permutation(NumList, L),
    legal(L, NumList).


natural_number(0).
natural_number(N) :-
    M is N - 1,
    natural_number(M).
```

Examples of the program execution:

```
?- n_queens(8,L).
L = [1, 7, 5, 8, 2, 4, 6, 3] ;
L = [1, 7, 4, 6, 8, 2, 5, 3] ;
L = [1, 6, 8, 3, 7, 4, 2, 5] ;
L = [1, 5, 8, 6, 3, 7, 2, 4] ;
L = [6, 1, 5, 2, 8, 3, 7, 4] ;
L = [4, 1, 5, 8, 2, 7, 3, 6] ;
L = [5, 1, 8, 4, 2, 7, 3, 6] ;
L = [3, 1, 7, 5, 8, 2, 4, 6] ;
L = [5, 1, 4, 6, 8, 2, 7, 3] ;
L = [7, 1, 3, 8, 6, 4, 2, 5] ;
L = [5, 1, 8, 6, 3, 7, 2, 4] ;
L = [4, 1, 5, 8, 6, 3, 7, 2] ;
L = [2, 6, 1, 7, 4, 8, 3, 5] ;
L = [5, 3, 1, 7, 2, 8, 6, 4] ;
L = [8, 3, 1, 6, 2, 5, 7, 4] ;
L = [4, 6, 1, 5, 2, 8, 3, 7] ;
L = [5, 7, 1, 4, 2, 8, 6, 3] ;
L = [6, 3, 1, 8, 4, 2, 7, 5] ;
L = [5, 3, 1, 6, 8, 2, 4, 7] ;
L = [6, 3, 1, 8, 5, 2, 4, 7] ;
L = [4, 8, 1, 3, 6, 2, 7, 5] ;
L = [8, 4, 1, 3, 6, 2, 7, 5]

?- n_queens(9, L).
L = [1, 5, 2, 6, 9, 3, 8, 4, 7] ;
L = [1, 4, 2, 8, 6, 9, 3, 5, 7] ;
L = [1, 6, 2, 9, 7, 4, 8, 3, 5] ;
L = [1, 3, 7, 2, 8, 5, 9, 4, 6] ;
L = [1, 6, 4, 2, 8, 3, 9, 7, 5] ;
L = [1, 5, 7, 2, 6, 3, 9, 4, 8] ;
L = [1, 6, 4, 2, 7, 9, 3, 5, 8] ;
L = [1, 5, 9, 2, 6, 8, 3, 7, 4] ;
L = [1, 8, 4, 2, 7, 9, 6, 3, 5] ;
L = [1, 3, 6, 8, 2, 4, 9, 7, 5] ;
L = [1, 4, 6, 8, 2, 5, 3, 9, 7] ;
L = [1, 7, 5, 8, 2, 9, 3, 6, 4]

?- n_queens(5, L).
L = [1, 3, 5, 2, 4] ;
L = [1, 4, 2, 5, 3] ;
L = [2, 4, 1, 3, 5] ;
L = [2, 5, 3, 1, 4] ;
L = [3, 1, 4, 2, 5] ;
L = [3, 5, 2, 4, 1] ;
L = [4, 1, 3, 5, 2] ;
L = [4, 2, 5, 3, 1]
```