This topic covers logic programming in more detail, including further examples.
Examples: Prefix and Suffix

- We can use the previous definition of appending to define a relationship for one list is a prefix of another, and similarly for when it is a suffix. (A list is a prefix if it is the same as an front part of another list, and a suffix if it is the same as the back part of another list.)

  append([], YS, YS).
  append([X|XS], YS, [X|ZS]) :- append(XS, YS, ZS).

  prefix(XS, YS) :- append(XS, REST, YS).
  suffix(XS, YS) :- append(REST, XS, YS).

Then we can query to generate the prefixes of a known list:

  ?- prefix(P, [1,2,3]).
  P = []
  P = [1]
  P = [1,2]
  P = [1,2,3]
Examples: 8 queens problem

- The eight queens problem involves finding all ways that 8 queens can be placed on a chess board such that none is “attacking” another.

```prolog
queens([]).  // succeed when empty
queens([ Row/Col | Rest]) :-  // otherwise, for each row
    queens(Rest),
    member(Col, [1,2,3,4,5,6,7,8]),
    safe( Row/Col, Rest).

safe(Anything, []).
safe(Row/Col, [Row1/Col1 | Rest]) :-
    Col =\= Col1,  // =\= is “not equals”
    Col1 - Col =\= Row1 - Row,
    Col1 - Col =\= Row - Row1,
    safe(Row/Col, Rest).
```
Examples: 8 queens problem

Here member is a built-in relationship for an element being in a list. It can also be implemented as follows.

```
member(X, [X | Tail]).

member(X, [Head | Tail]) :-
    member(X, Tail).
```

[Note that member is called with the second argument being known to backtrack through the possibilities.]

Then we can query to generate solutions:

```
?- queens([1/C1, 2/C2, 3/C3, 4/C4, 5/C5, 6/C6, 7/C7, 8/C8]).
    C1 = ...
    C2 = ...
```