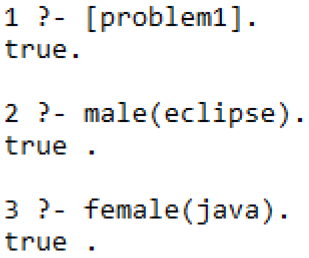
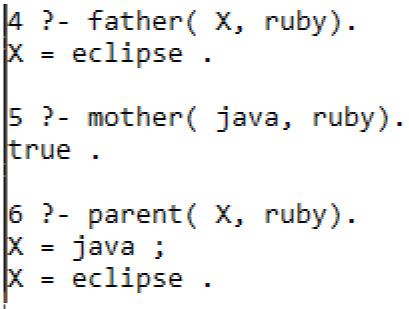
1) Family Tree

```
%Definitions
m([eclipse, atom, euler, waring, jack, jimmy, chad]).
f([java, ruby, curie, sql, jill]).
family([eclipse, java, [ruby]]).
family([atom, ruby, [curie, waring]]).
family([euler, curie, [jill, jack]]).
family([waring, sql, [jimmy, chad]]).
%Rules
male(X) :- m(M), member(X,M).
female(X) :- f(F), member(X,F).

father(Father, Child) :- family([Father, _, Children]),
                    member(Child, Children).

mother(Mother, Child) :- family([ _, Mother, Children]),
                    member(Child, Children).

parent(Parent, Child) :- mother(Parent, Child);
                    father(Parent, Child).

grandFather(GrandFather, GrandChild) :- father(GrandFather, Parent),
                              parent(Parent, GrandChild).

grandMother(GrandMother, GrandChild) :- mother(GrandMother, Parent),
                              parent(Parent, GrandChild).

grandParent(GrandParent, GrandChild) :- grandFather(GrandParent, GrandChild);
                              grandMother(GrandParent, GrandChild).

greatGrandParent(GreatGrandParent, GreatGrandChild) :- grandParent(GreatGrandParent, Parent),
                                          parent(Parent, GreatGrandChild).

siblings1(SiblingX, SiblingY) :- parent(Parent, SiblingX), parent(Parent, SiblingY),
                         SiblingX \== SiblingY.
siblings2(SiblingX, SiblingY) :- father(Father, SiblingX), father(Father, SiblingY),
                         mother(Mother, SiblingX), mother(Mother, SiblingY),
                         SiblingX \== SiblingY.

aunt(Aunt, Person) :- parent(Parent, Person),
                  siblings1(Parent, Aunt),
                  female(Aunt).

uncle(Uncle, Person) :- parent(Parent, Person),
                   siblings1(Parent, Uncle),
                   male(Uncle).

cousins(CousinX, CousinY) :- parent(ParentX, CousinX),
                      parent(ParentY, CousinY),
                      siblings1(ParentX, ParentY).

ancestor(Ancestor, Person) :- parent(Ancestor, Person);
                       parent(Ancestor, Z), ancestor(Ancestor, Z).
```

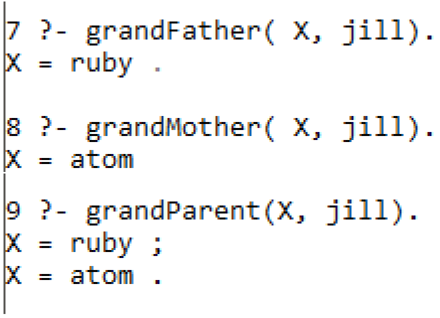
OUTPUTS: Showed a variety of test cases. Every relationship was used at least once.



```
1 ?- [problem1].
true.

2 ?- male(eclipse).
true .

3 ?- female(java).
true .
```

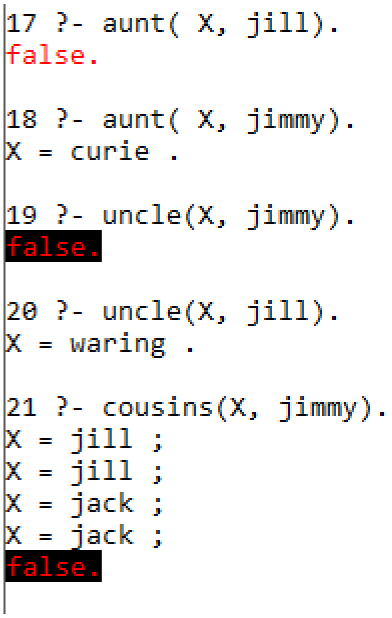


```
4 ?- father( X, ruby).
X = eclipse .

5 ?- mother( java, ruby).
true .

6 ?- parent( X, ruby).
X = java ;
X = eclipse .
```



```
7 ?- grandFather( X, jill).
X = ruby .

8 ?- grandMother( X, jill).
X = atom

9 ?- grandParent(X, jill).
X = ruby ;
X = atom .
```

```
10 ?- [problem1].
true.

11 ?- grandFather( X, jimmy).
X = atom .

12 ?- grandMother( X, jimmy).
X = ruby .

13 ?- grandParent( X, jimmy).
X = atom
Unknown action: l (h for help)
Action? ;
X = ruby .

14 ?- greatGrandParent(X, jimmy).
X = eclipse ;
X = java .

15 ?- siblings1(curie, waring).
true .

16 ?- siglings2(X, waring).
Correct to: "siblings2(X,waring)"? yes
X = curie .
```



```
17 ?- aunt( X, jill).
false.

18 ?- aunt( X, jimmy).
X = curie .

19 ?- uncle(X, jimmy).
false.

20 ?- uncle(X, jill).
X = waring .

21 ?- cousins(X, jimmy).
X = jill ;
X = jill ;
X = jack ;
X = jack ;
false.
```

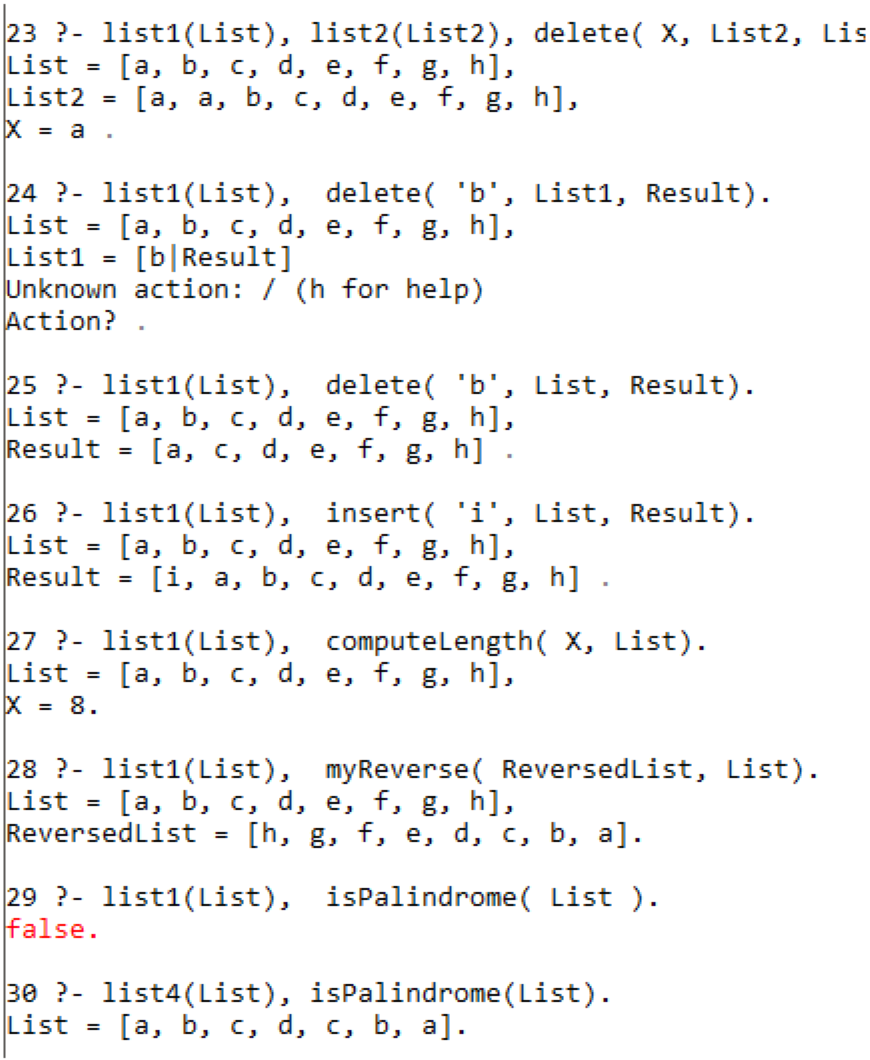
```
28 ?- ancestor( eclipse, jimmy).
true .

29 ?- ancestor( jimmy, eclipse).
false.
```

2) List Operations

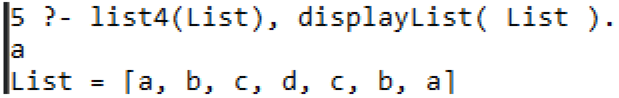
```
%Definitions
list1([a, b, c, d, e, f, g, h]).
list2([a, a, b, c, d, e, f, g, h]).
list3([a, a, a, b, c, d, e, f, g, h]).
list4([a, b, c, d, c, b, a]).



%Rules
firstElement( Element, [ H|_ ] ) :- Element is H.

lastElement( Element, [Element] ).
lastElement( Element, [_|T] ) :- lastElement(Element, T).

twoAdjacent( X, Y, [X,Y|_] ).
twoAdjacent( X,Y, [_|T] ) :- twoAdjacent(X, Y, T).

threeAdjacent( X, Y, Z, [X,Y,Z|_] ).
threeAdjacent( X, Y, Z, [_|T] ) :- threeAdjacent(X, Y, Z, T).

myAppendList( [], List, List ).
myAppendList( [X|TX], List, [X|T] ):- myAppendList( TX, List, T ).


delete( Element, [Element|T], T ).
delete( Element, [H|T], [H|T2]) :- delete(Element, T, T2).

insert( Element, List, ExpandedList ) :- delete(Element, ExpandedList, List).

computeLength( 0, []).
computeLength( Length, [_|T] ) :- computeLength( CurrentLength, T), Length is CurrentLength + 1.


myReverse( [], [] ).
myReverse( Reversed, [H|T] ) :- myReverse( RT, T ),
                append( RT, [H], Reversed ).

isPalindrome( List ) :-  myReverse(Reversed, List),
            List = Reversed, !.

displayList( [] ).
displayList( [H|T] ) :- write( H ); write(‘ ‘);
             displayList( [T|_] ).
```

OUTPUTS: Every rule is shown working, some have multiple test cases shown

```
10 ?- firstElement( X, [a,b,c,d] ).
X = a.

11 ?- lastElement( X, [a,b,c,d] ).
X = d .

12 ?- twoAdjacent( X, Y, [a,b,c,d] ).
X = a,
Y = b ;
X = b,
Y = c ;
X = c,
Y = d ;
false.

14 ?- list1(List), threeAdjacent(X,Y,Z,List).
List = [a, b, c, d, e, f, g, h],
X = a,
Y = b,
Z = c ;
List = [a, b, c, d, e, f, g, h],
X = b,
Y = c,
Z = d .


20 ?- list1(List), list2(List2), myAppendList( List, List2, Result ).
List = [a, b, c, d, e, f, g, h],
List2 = [a, a, b, c, d, e, f, g, h],
Result = [a, b, c, d, e, f, g, h, a|...].

21 ?- myAppendList( [a,b,c], [b,a], Result ).
Result = [a, b, c, b, a].
```



```
23 ?- list1(List), list2(List2), delete( X, List2, Lis
List = [a, b, c, d, e, f, g, h],
List2 = [a, a, b, c, d, e, f, g, h],
X = a .

24 ?- list1(List),  delete( 'b', List1, Result).
List = [a, b, c, d, e, f, g, h],
List1 = [b|Result]
Unknown action: / (h for help)
Action? .

25 ?- list1(List),  delete( 'b', List, Result).
List = [a, b, c, d, e, f, g, h],
Result = [a, c, d, e, f, g, h] .

26 ?- list1(List),  insert( 'i', List, Result).
List = [a, b, c, d, e, f, g, h],
Result = [i, a, b, c, d, e, f, g, h] .

27 ?- list1(List),  computeLength( X, List).
List = [a, b, c, d, e, f, g, h],
X = 8.

28 ?- list1(List),  myReverse( ReversedList, List).
List = [a, b, c, d, e, f, g, h],
ReversedList = [h, g, f, e, d, c, b, a].

29 ?- list1(List),  isPalindrome( List ).
false.

30 ?- list4(List), isPalindrome(List).
List = [a, b, c, d, c, b, a].
```



```
5 ?- list4(List), displayList( List ).
a
List = [a, b, c, d, c, b, a]
```

3) 8- Queens

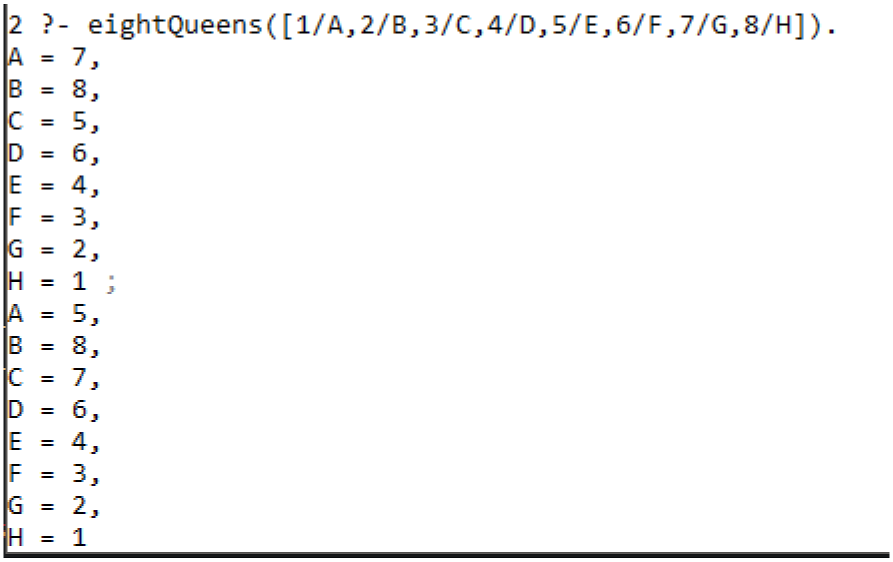
My original approach. This simply returned “True” and gave me a gigantic headache.
The revised solution before is more efficient, can have its results checked, and is more clear yet I felt as if I had to share my struggle and my initial approach.

```
eightQueens(  )  :-
                        permuteEight( X ),
                        permuteEight( Y ),
                        checkSets( X, Y ).


permuteEight( List ) :- findall(X, permutation([1,2,3,4,5,6,7,8], X), List).

checkSets([], []).
checkSets([A,B,C,D,E,F,G,H | RestX], [A2,B2,C2,D2,E2,F2,G2,H2 | RestY] ) :-
  checkSet( [A,B,C,D,E,F,G,H], [A2, B2, C2, D2, E2, F2, G2, H2]);
  checkSets( RestX, RestY).


checkSet([], []).
checkSet([X | RestX], [Y | RestY]) :-
   checkSet(X, Y, RestX, RestY),
   checkSet(RestX, RestY).

checkSet(X, Y, [X2 | RestX], [Y2 | RestY]) :-
   Y2 - Y \== X2 - X,
   Y2 - Y \== X - X2,
   checkSet( X,Y, RestX, RestY).
```

My revised solution, after a few hours of googling and comparing methods of solving this problem:

```
% This solution solves the 8 Queens Problem
% I had to look at 3-4 different solutions for help
% I understand the solution myself now, I did not know before htat you could
% group X and Y with any separator between the two before

eightQueens([]).
eightQueens([X/Y|T]) :- eightQueens(T),
                    member(Y,[1,2,3,4,5,6,7,8]),
                    safeArea(X/Y, T).

safeArea(_,[]).
safeArea(X1/Y1, [X2/Y2 | T ]) :-
                 Y1 \== Y2,       %Two Y Coords Cannot Be Equal
                 Y2-Y1 \== X2-X1,  %Diagonal Must Be Clear
                 Y2-Y1 \== X1-X2,  %Diagonal Must Be Clear
                 safeArea( X1/Y1, T).

% Template for solution-- each column must have a queen
template([1/A,2/B,3/C,4/D,5/E,6/F,7/G,8/H]).
% I used A-H because chest boards are keyed by <Letter><Number>
```

OUTPUT:



```
2 ?- eightQueens([1/A,2/B,3/C,4/D,5/E,6/F,7/G,8/H]).
A = 7,
B = 8,
C = 5,
D = 6,
E = 4,
F = 3,
G = 2,
H = 1 ;
A = 5,
B = 8,
C = 7,
D = 6,
E = 4,
F = 3,
G = 2,
H = 1
```

I displayed two answers. All results could be enumerated and stored in a list if desired by using the find all rule built into the language, but there are too many solutions to practically view them all here in this document.

CSC 600
Programming Languages

Richard Robinson
HW 3: Prolog

Instructor: Jozo Dujmović