**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),

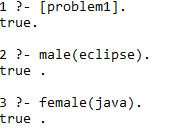
parent(ParentY, CousinY),

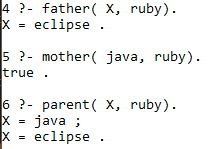
siblings1(ParentX, ParentY).

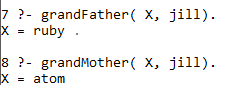
ancestor(Ancestor, Person) :- parent(Ancestor, Person);

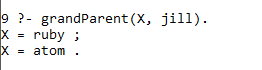
parent(Ancestor, Z), ancestor(Ancestor, Z).

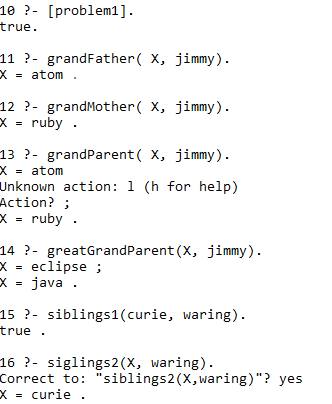
**OUTPUTS: Showed a variety of test cases. Every relationship was used at least once.**

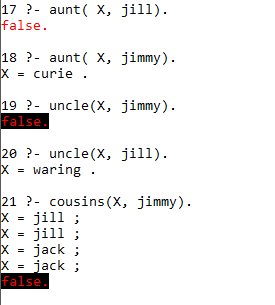


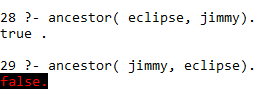












**2) List Operations**

%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),

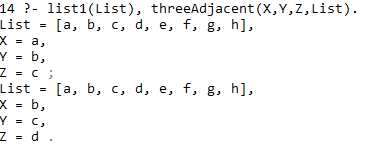
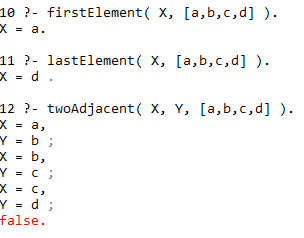
List = Reversed, !.

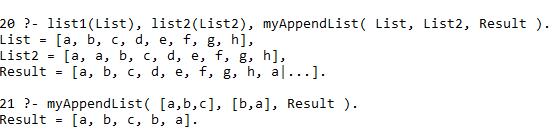
displayList( [] ).

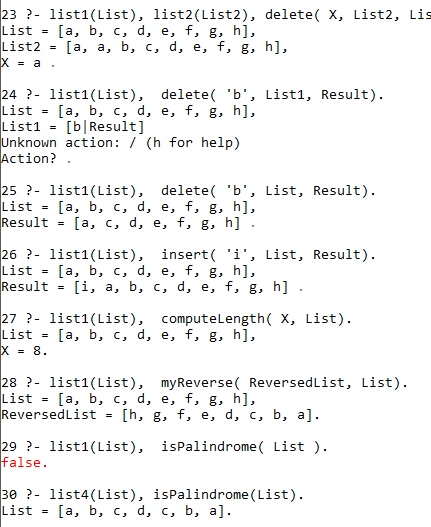
displayList( [H|T] ) :- write( H ); write(‘ ‘);

displayList( [T|\_] ).

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







**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

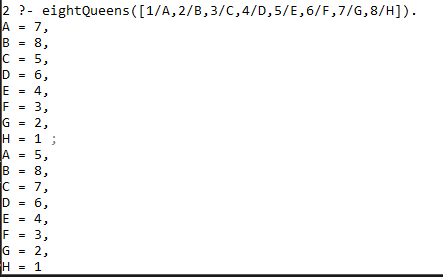
safeArea( X1/Y1, T).

% 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:**



**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**