SOLUTIONS - EXERCISES ON PROLOG

Write the following as Prolog rules:

1. Implement a rule "**dogEnthusiast**".

   Someone is a "dogEnthusiast" if they own AT LEAST TWO dogs.
   Assume that the only types of facts available are:

   "owner" facts of the form owner(p,n), meaning that person p owns animal named n.
   "breed" facts of the form breed(a,b), meaning that animal named a is of type b.

   For example:

   owner(fred,fido) means that a person named "fred" owns an animal named "fido".
   breed(fido,dog) means that the animal named "fido" is a dog.
   breed(leo,cat) means that the animal named "leo" is a cat.

   dogEnthusiast(sam) would be true if sam owned two different animals that were dogs.

   dogEnthusiast (X) :- owner (X, Y), owner (X, Z), not (Y = Z), breed(Y, dog), breed(Z, dog).

2. Implement the "**listMins**" function takes two equal-length lists of numbers, and returns a single list consisting of the smaller of the two lists, position by position.
   For example, the query: listMins([2,8,7], [5,4,4], M) should result in Prolog's response:
   M = [2, 4, 4].

   ```
   listMins([], [], []).
   listMins([H1|T1], [H2|T2], [H1|Y]) :- H1<H2, listMins(T1,T2,Y).
   listMins([H1|T1], [H2|T2], [H2|Y]) :- H1>=H2, listMins(T1, T2, Y).
   ```

3. Convert "**digitinc**" takes as input an integer, and returns another integer constructed of the original input integer's digits, each incremented by 1. When a digit is a 9, the corresponding output digit should be a 0.

   For example:
   ?- digitinc(22897,X) should respond X=33908.

   ```
   onedigitinc(9,0).
   onedigitinc(X,Y) :- X<9, Y is X+1.
   digitinc(X,Y) :- X<10, onedigitinc(X,Y).
   digitinc(X,Y) :- X>=10, Z is mod(X,10),U is div(X,10),onedigitinc(Z,V), digitinc(U,T), Y is T*10+V.
   ```

4. Implement a rule "**crypto**" that solves the following cryptarithmetic multiplication problem:

   TOCK * TOCK = GRIPTOCK

   Each of the 4 letters (T,O,C,K) stands for a different digit.
   The 4 letters (G,R,I,P) can each be matched by any digit.
   The aim is to find a substitution of digits for the letters
   such that the above stated product is arithmetically correct.
   Your program should find all answers.

   It should be possible to query your solution in this manner:

   ?- crypto(G,R,I,P,T,O,C,K).

   Your solution should then produce all of the combinations of the digits that satisfy the multiplication problem above.Don't get confused between the letter "O" and the number "0" (zero).

   Make sure you never let T=C, or C=K, etc... all of the distinct letters T,O,C,K have to stand for distinct digits.

   Use generate-and-test!

```
digit(9). digit(8). digit(7). digit(6). digit(5). digit(4). digit(3). digit(2). digit(1).

crypt ( G, R, I, P, T, O, C, K) :-.
        digit(G), digit(R), digit(I), digit(P), digit(T), digit(O), digit(C), digit(K),

T\=O, T\=C, T\=K,
O\=C, O\=K,
C\=K,

TOCK is T*1000+O*100+C*10+K,
RESULT is G*10000000+R*1000000+I*100000+P*10000+T*1000+O*100+C*10+K,
PRODUCT is TOCK*TOCK,
RESULT=PRODUCT.
```

5. Write a Prolog rule "**overlap2**", that takes two lists, and returns true if the lists have at least two elements in common. For example:

   overlap2([3,4,5],[7,8,3,9]). should return false.
   overlap2([3,4,5],[4,7,3,8,6]). should return true.

```
overlap2(L,M) :- intersect(L,M,I), listlen(I,N), N>=2.

listlen([],0).
listlen([X|Y],Z) :- listlen(Y,W), Z is W+1.
```

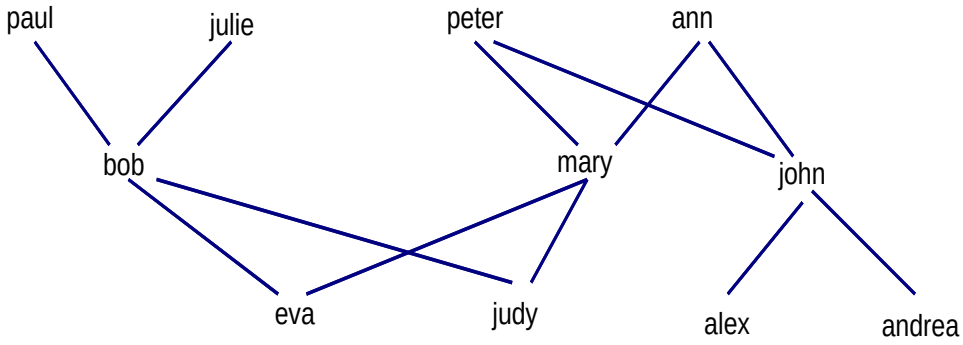
```
intersect([],M,[]).
intersect([X|Y],M,[X|Z]) :- memberof(X,M), intersect(Y,M,Z).
intersect([X|Y],L,U) :- intersect(Y,L,U).

memberof(X,[X|M]).
memberof(X,[Y|M]) :- memberof(X,M).
```

For the following questions assume the following relations are in a Prolog database:

parent(X, Y)
female(X)
male(X)

1. Create the database that will reflect the following family relationships:




```
female ( eva).
female ( judy).
female ( andrea).
female ( mary).
female ( julie).
female ( ann).

male ( alex).
male ( bob).
male ( john).
male ( paul).
male ( peter).

parent(mary, eva).
parent(mary, judy).
parent(bob, eva).
parent(bob, judy).
parent(ann, mary).
parent(peter, mary).
parent(ann, john).
parent(peter, john).
parent(john, alex).
parent(paul, bob).
parent(julie, bob).
parent(john, andrea).
parent(paul, bob).
parent(julie, bob).
parent(john, andrea).
```

2. Define a *mother* predicate so that *mother*(X, Y) says that X is the mother of Y.

```
mother(X, Y)        :-      parent(X, Y), female(X).
```

3. Define a *father* predicate so that *father*(X, Y) says that X is the father of Y.

```
father(X, Y)          :-    parent(X, Y), male(X).
```

4. Define a *sister* predicate so that *sister*(X, Y) says that X is the sister of Y (Note: a person cannot be her own sister).

```
sibling(X, Y)         :-    parent(M, X), parent(M, Y), not(X= Y).
sister(X, Y)          :-    parent(M, X), parent(M,Y), female(X), not(X=Y).
```

5. Define a *grandson* predicate so that *grandson*(X, Y) says that X is a grandson of Y.

```
grandchild(X, Y)      :-    parent(M, X), parent(Y, M).
grandson(X, Y)        :-    male(X), grandchild(X, Y).
granddaughter(X, Y) :-    female(X),grandchild(X, Y).
```

6. Define the *firstCousin* predicate so that *firstCousin*(X, Y) says that X is a first cousin of Y (Note: a person cannot be his or her first cousin, nor can a brother or sister also be a cousin.).

```
firstCousin(X, Y)     :-    parent(P, X), parent(Q, Y), sibling(P, Q).
```

7. Define the *descendant* predicate so that *descendant*(X, Y) says that X is a descendant of Y.

```
descendant(X, Y)      :-    parent(Y, X).
descendant(X, Y)      :-    parent(P, X), descendant(P, Y).
```