1) Create a database of your relatives using predicates **child_of/2**, **male/1**, **female/1** and **married_to/2**. child_of(x,y) means x is the child of y, and married_to(x,y) means x is married to y. If two persons have the same name (e.g. joe), make them distinct by calling one joe1 and the other joe2 etc. Make sure you have all the information up to your grandparents from both the mother side and the father side. (to be done indivudually)

2) Write definitions for the following predicates.

father(X,Y):- .... /* X is the father of Y */

mother(X,Y):- .... /* X is the mother of Y */

aunt_m(X,Y):- .... /* X is the aunt of Y (mother's sister) */

aunt_f(X,Y):- .... /* X is the aunt of Y (father's sister) */

grandfather(X,Y):- .... /* X is the grandfather of Y (both mother and father side) */

grandmother(X,Y):- .... /* X is the grandmother of Y (both mother and father side) */

cousin(X,Y):- ... /* X is the cousin of Y */

niece(X,Y):- /* X is the niece of Y */

nephew(X,Y):- .... /* X is the nephew of Y */

brother(X,Y):- ... /* X is the brother of Y */

sister(X,Y):- ... /* X is the brother of Y */

3) Implement the following predicates in Prolog.

countHowMany(E,L,N). N is the number of times E occurs in list L. For example, ?-countHowMany(b,[a,s,b,d,b,e,b],R) should bind R to 3.

reversePairs(X,Y). Y is a list obtained by reversing the first two elements, followed by reversing the next two elements, etc., in X. For example, ?-reversePairs([1,2,3,4,5,6,7,9],R) should bind R to [2,1,4,3,6,5,9,7].

split(X,N,Y,Z). Y is a list obtained by taking the first N elements of list X and putting them in Y, and the remaining elements in Z. For example, split([a,s,d,f,g,h,j],2,R1,R2) should bind R1 to [a,s], and R2 to [d,f,g,h,j]).

minMax(L,Min,Max). Min is the smallest element of L, and Max is the largest. For example, ?-**minMax**([5,3,4,6,3,8],R,S) should bind R to 3, S to 8.

vector_add(A,B,C). If A and B are lists of numbers, then C is a list of numbers such that an element of C at position i is the sum of the element of A at position i and the element of B at position i. For example, ?-vector_add([1,3],[5,8], R) should bind R to [6,11].

4) A tree can be represented in Prolog in the following way.

- An empty tree is represented by the atom **empty**.
- A tree with a left child **l** and a right child **r** as node value **v** is represented by **node(l,v,r).**

Some predicates are given for you below for manipulating binary search trees with this representation.

```
insert(V,empty,node(empty,V,empty)). /* insert a value into a binary search tree */
insert(V, node(L,V2,R), node(L2,V2,R)):- V=<V2, insert(V,L,L2).
insert(V, node(L,V2,R), node(L,V2,R2)):- V>V2, insert(V,R,R2).

insert_list([],empty). /* insert a list of numbers into an empty binary search tree */
insert_list([H|T],Tree):- insert_list(T, Temp), insert(H,Temp,Tree).

size(empty, 0). /* size of the tree - how many nodes it has */
size(node(L,_,R),S):- size(L,X), size(R,Y), S is X+Y+1.

in_order(empty,[]). /* inorder traversal */
in_order(node(L,V,R),Res):- in_order(L,LT), in_order(R,RT), app(LT,[V],X),
app(X,RT,Res).

app([],L,L).  /* append two lists */
app([H|T],L,[H|TL]):- app(T,L,TL).

min(node(empty,V,empty),V). /* find the minimum value in a binary search tree */
min(node(L,_,R),V):- min(L,V).
```

Implement the following predicates on binary search trees.

find(V,T) succeeds if value V is in the T

post_order(T,L) succeeds if L list containing the values in the tree T in the same order as in a "postorder" traversal.

inner_nodes(T,L) succeds if L is a list containing the values in the inner nodes of the tree T.

max(T, Max) succeeds if Max is the the maximum value in tree T (hint: the maximum is "rightmost value" in the tree)