Problems 1, 2 and 3 are due Fri., Dec. 3. Problems 4, 5 and 6 are due Fri., Dec. 10.

1. Suppose that you have a database of facts of the form
   
   male(He)
   female(She)
   parent(Parent, Child)
   
   with the obvious meanings. Write Prolog predicates that could deduce the following relationships:
   (a) mother(Mom, Child)
   (b) father(Dad, Child)
   (c) daughter(Girl, Parent)
   (d) son(Boy, Parent)
   (e) brother(Boy, Sibling)
   (f) sister(Girl, Sibling)
   (g) grandfather(OldMan, Child)
   (h) aunt(Lady, Child).
   (i) ancestor(Person, Descendent).

   In parts (e) and (f), be careful that no one is his or her own brother or sister. If part (i), a person should not count as his or her own ancestor.

   A sample database containing some facts of these forms is available as family.pl. You may use this to test your program, or make up a database of your own.

2. Write and test a Prolog predicate, extremes, which finds the maximum and minimum values in a list of numbers. That is, extremes(+L, -Max, -Min) should succeed when Max and Min are the maximum and minimum numbers in the list L. Try to do this by recursing through the list only once, rather than once to find the minimum and again to find the maximum.

3. Solve the hailstone problem in Prolog. hailstone(+H, -T) should succeed when a hailstone at initial height H takes T time units to hit the ground. As a side effect, the heights should be printed out as the hailstone falls (that is, as the satisfaction of the goal is approached).
4. Implement *Mergesort* in Prolog. `mergesort(+L1, -L2)` should succeed when `L2` is the sorted version of the list `L1`. (Assume that `L1` is a list of numbers.)

5. In *Peano arithmetic*, natural numbers are defined as follows:
   - 0 is a natural number.
   - If `n` is a natural number then `s(n)` is a natural number. (Think of `s(n)` as the *successor* of `n`, or `n + 1`.)
   - Nothing else is a natural number.

   or, in Prolog,
   ```prolog
   nat(0).
   nat(s(X)) :- nat(X).
   ```

   Implement the operations of addition and multiplication for the natural numbers in Peano arithmetic, so that
   ```prolog
   natplus(X, Y, Z) succeeds if Z = X + Y
   nattimes(X, Y, Z) succeeds if Z = XY.
   ```

   Then, for example
   ```prolog
   ?- natplus(s(s(s(0))), s(s(0)), Sum).
   Sum = s(s(s(s(s(0))))).
   ```
   ```prolog
   ?- nattimes(s(s(s(0))), s(s(0)), Product).
   Product = s(s(s(s(s(s(0)))))).
   ```

   What happens if you call your predicates with the third argument instantiated to a natural number, but one or both of the first two arguments as variables?

   Note: do **not** do this by converting back and forth to ordinary integers. Use the Peano definition directly.
6. The “Twenty-four” puzzle is the following:

Given a list of four numbers, try to build an arithmetic expression using each number exactly once and the operations of addition, subtraction, multiplication and division so that the value of the expression is exactly 24.

For example, if the numbers are 1, 3, 5 and 7 then one solution would be

\[(3 - 1) \times (5 + 7) = 24\].

Solve this puzzle in Prolog. Write a predicate

`puzzle(+L, -E, +N)`

which succeeds when `L` is a list of numbers and `E` is an expression whose value is `N`. It is no harder to program the more general case of a list of any number of numbers, and any desired value. The above example would be solved as:

```
?- puzzle([1, 3, 5, 7], E, 24).
E = (3 - 1) * (5 + 7)
```

There are other solutions in this example so your program might find a different one. It is not required, but hopefully would turn out to be the case that, if you reject a solution your program will generate more solutions until it has found them all. Some of these solutions might look the same as earlier solutions; don’t worry about that.