**Exercise 1.4**:

Represent the following in Prolog:

1. Butch is a killer.
2. Mia and Marsellus are married.
3. Zed is dead.
4. Marsellus kills everyone who gives Mia a footmassage.
5. Mia loves everyone who is a good dancer.
6. Jules eats anything that is nutritious or tasty.

Add a single fact that is not kills(marsellus,zed) to make prolog deduce that Marsellus killed Zed.

**Solution 1.4**:

killer(butch).

married(mia,marsellus).

dead(zed).

kills(marsellus,X) :- gaveFootMassage(X,mia).

loves(mia,X) :- goodDancer(X).

eats(jules,X) :- nutoritious(X) ; tasty(X).

gaveFootMassage(zed,mia).

**Exercise 1.5**:

Suppose we are working with the following knowledge base:

wizard(ron).

wizard(X):-  hasBroom(X),  hasWand(X).  
hasWand(harry).  
quidditchPlayer(harry).  
hasBroom(X):-  quidditchPlayer(X).

How does Prolog respond to the following queries?

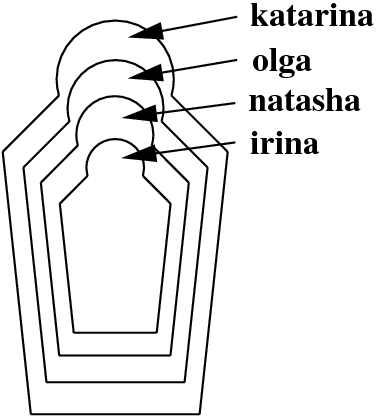
1. wizard(ron).
2. witch(ron).
3. wizard(hermione).
4. witch(hermione).
5. wizard(harry).
6. wizard(Y).
7. witch(Y).

**Solution 1.5**:

1: true, 2, 4, 7: Undefined procedure: witch/1, 3: false, 5: true, 6: Y = ron; Y = harry.

**Exercise 3.2**:

Do you know these wooden Russian dolls (Matryoshka dolls) where the smaller ones are contained in bigger ones? Here is a schematic picture:



First, write a knowledge base using the predicate directlyIn/2 which encodes which doll is directly contained in which other doll. Then, define a recursive predicate in/2 that tells us which doll is (directly or indirectly) contained in which other dolls. For example, the query in(katarina,natasha) should succeed and in(olga, katarina) should fail.

**Solution 3.2**:

directlyIn(irina,natasha).

directlyIn(natasha,olga).

directlyIn(olga,katarina).

in(X,Y) :- directlyIn(X,Y).

in(X,Y) :- directlyIn(X,Z), in(Z,Y).

**Exercise 3.4**:

Define a predicate greater\_than/2 that takes two numerals in the notation that we introduced in the text (that is, 0, succ(0), succ(succ(0)), and so on) as arguments and decides whether the first one is greater than the second one. For example:

   ?-  greater\_than(succ(succ(succ(0))),succ(0)).  
   yes  
   ?-  greater\_than(succ(succ(0)),succ(succ(succ(0)))).  
   no

**Solution 3.4**:

greater\_than(succ(X), X).

greater\_than(succ(X), Y) :- greater\_than(X,Y).

**Exercise 4.1**:

How does Prolog respond to the following queries?

1. [a,b,c,d] = [a,[b,c,d]].
2. [a,b,c,d] = [a|[b,c,d]].
3. [a,b,c,d] = [a,b,[c,d]].
4. [a,b,c,d] = [a,b|[c,d]].
5. [a,b,c,d] = [a,b,c,[d]].
6. [a,b,c,d] = [a,b,c|[d]].
7. [a,b,c,d] = [a,b,c,d,[]].
8. [a,b,c,d] = [a,b,c,d|[]].
9. [] = \_.
10. [] = [\_].
11. [] = [\_|[]].

**Exercise 4.2**:

Which of the following are syntactically correct lists? If the representation is correct, how many elements does the list have?

1. [1|[2,3,4]]
2. [1,2,3|[]]
3. [1|2,3,4]
4. [1|[2|[3|[4]]]]
5. [1,2,3,4|[]]
6. [[]|[]]
7. [[1,2]|4]
8. [[1,2],[3,4]|[5,6,7]]

**Exercise 4.3**:

Write a predicate second/2 which checks if the first argument is the second element of the List given as the second argument. Use just a single variable in your predicate.

**Solution 4.3**:

second(X,[\_,X|\_]).

**Exercise 4.5**:

Suppose we are given a knowledge base with the following facts:

   tran(eins,one).  
   tran(zwei,two).   
   tran(drei,three).   
   tran(vier,four).   
   tran(fuenf,five).   
   tran(sechs,six).   
   tran(sieben,seven).   
   tran(acht,eight).   
   tran(neun,nine).

Write a predicate listtran/2 which translates a list of German number words to the corresponding list of English number words. For example:

listtran([eins,neun,zwei],X) should give: X = [one,nine,two].

Your program should also work in the other direction. For example, if you give it the query listtran(X,[one,seven,six,two]) it should return X = [eins,sieben,sechs,zwei].

**Solution 4.5**:

listtran([],[]).

listtran([X|TG],[Y|TE]) :- tran(X,Y), listtran(TG,TE).

**Exercise 4.6**:

Write a predicate twice/2 whose left argument is a list, and whose right argument is a list consisting of every element in the left list written twice. For example, the query twice([a,4,buggle],X) should return X = [a,a,4,4,buggle,buggle]). And the query twice([1,2,1,1],X) should return X = [1,1,2,2,1,1,1,1].

**Solution 4.6**:

twice([],[]).

twice([X|T],[X,X|T2]) :- twice(T,T2).

**Exercise 5.1**:

How does Prolog respond to the following queries?

1. X  =  3\*4.
2. X  is  3\*4.
3. 4  is  X.
4. X  =  Y.
5. 3  is  1+2.
6. 3  is  +(1,2).
7. 3  is  X+2.
8. X  is  1+2.
9. 1+2  is  1+2.
10. is(X,+(1,2)).
11. 3+2  =  +(3,2).
12. \*(7,5)  =  7\*5.
13. \*(7,+(3,2))  =  7\*(3+2).
14. \*(7,(3+2))  =  7\*(3+2).
15. 7\*3+2  =  \*(7,+(3,2)).
16. \*(7,(3+2))  =  7\*(+(3,2)).

**Exercise 5.2**:

1. Define a 2-place predicate increment that holds only when its second argument is an integer one larger than its first argument. For example, increment(4,5) should hold, but increment(4,6) should not.
2. Define a 3-place predicate sum that holds only when its third argument is the sum of the first two arguments. For example, sum(4,5,9) should hold, but sum(4,6,12) should not.

**Solution 5.2**:

increment(X,Y) :- Y is X+1.

sum(X,Y,Z) :- Z is X+Y.

**Exercise 5.3**:

Write a predicate addone/2 whose first argument is a list of integers, and whose second argument is the list of integers obtained by adding 1 to each integer in the first list. For example, the query addone([1,2,7,2],X) should give X = [2,3,8,3].

**Solution 5.3**:

addOne([],[]).

addOne([X|T],[Y|T2]) :- Y is X+1, addOne(T,T2).

**Exercise 5.4**:

In mathematics, an n-dimensional vector is a list of numbers of length n. For example, [2,5,12] is a 3-dimensional vector, and [45,27,3,-4,6] is a 5-dimensional vector. One of the basic operations on vectors is scalar multiplication. In this operation, every element of a vector is multiplied by some number. For example, if we scalar multiply the 3-dimensional vector [2,7,4] by 3 the result is the 3-dimensional vector [6,21,12] .

Write a 3-place predicate scalarMult whose first argument is an integer, whose second argument is a list of integers, and whose third argument is the result of the scalar multiplication of the second argument by the first. For example, the query scalarMult(3,[2,7,4],Result) should yield Result = [6,21,12].

**Solution 5.4**:

scalarMult(\_,[],[]).

scalarMult(Mult,[H|T],[H2|T2]) :- H2 is H\*Mult, scalarMult(Mult,T,T2).

**Exercise 5.5**:

Another fundamental operation on vectors is the dot product. This operation combines two vectors of the same dimension and yields a number as a result. The operation is carried out as follows: the corresponding elements of the two vectors are multiplied, and the results added. For example, the dot product of [2,5,6] and [3,4,1] is 6+20+6, that is, 32. Write a 3-place predicate dot whose first argument is a list of integers, whose second argument is a list of integers of the same length as the first, and whose third argument is the dot product of the first argument with the second. For example, the query dot([2,5,6],[3,4,1],Result) should yield Result = 32.

**Solution 5.5**:

With accumulator:

dot(X,Y,Z) :- dot(X,Y,0,Z).

dot([],[],Acc,Acc).

dot([H1|T1],[H2|T2],Acc,Result) :- AccNew is Acc + H1\*H2, dot(T1,T2,AccNew,Result).

Without accumulator:

dot([],[],0).

dot([[H1|T1],[H2|T2],Result) :- dot(T1,T2,TailResult), Result is TailResult + H1\*H2.

**Exercise 9.1**:

Recall the definition of jealousy given in Chapter 1.

   jealous(X,Y):-  loves(X,Z),  loves(Y,Z).

In a world where both Vincent and Marsellus love Mia, Vincent will be jealous of Marsellus, and Marsellus of Vincent. But Marsellus will also be jealous of himself, and so will Vincent. Revise the Prolog definition of jealousy in such a way that people can’t be jealous of themselves.

**Solution 9.1**:

loves(vincent,mia).

loves(marsellus,mia).

jealous(X,Y) :- loves(X,Z), loves(Y,Z), X\==Y.

**Exercise 10.7**:

Write a predicate set/2 which takes a list as input, and returns a list in which each element of the input list appears only once. For example, the query set([2,2,foo,1,foo,[],[]],X) should yield the result X  =  [2,foo,1,[]].

Hint: use the member\2 predicate to test for repetitions of items.

**Solution 10.7**:

set([],[]).

set([X|T],T2) :- member(X,T), set(T,T2).

set([X|T],[X|T2]) :- \+member(X,T), set(T,T2).

**Exercise 6.3**:

Write a predicate choppedList/2 that returns true if the second argument equals to the list in the first argument without its first and last elements. If the first list has less than two elements it returns false. You can use the append/3 predicate in your solution. For example:

           choppedList([a],T).  
           no  
           choppedList([a,b],T).  
           T=[]    
           choppedList([a,b,c],T).  
           T=[b]

**Solution 6.3**:

choppedList([\_|Tail],ChoppedList) :- append(ChoppedList,[\_],Tail).

**Exercise 6.4**:

Write a predicate last/2 which is true only when the second argument is the last element of the list given as the first argument. It should return false if the list is empty. Do this in two different ways:

1. Define lastWithReverse/2 using the predicate reverse/2.
2. Define last/2 using recursion without the use of any other predicate.

**Solution 6.4**:

lastWithReverse(List,Last) :- reverse(List,[Last|\_]).

last([X],X).

last([\_|T],Last) :- last(T,Last).