

[Dashboard](#) / [My courses](#) / [COSC367-2020S2](#) / [Weekly quizzes](#) / [4. Declarative programming with Prolog \(i\)](#).

Started on	Thursday, 6 August 2020, 6:48 PM
State	Finished
Completed on	Friday, 14 August 2020, 5:06 PM
Time taken	7 days 22 hours
Marks	8.00/8.00
Grade	100.00 out of 100.00

Getting started with Prolog

Start a terminal and type `swipl`. You should see something like this:

```
Welcome to SWI-Prolog

...

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```

Open an editor. It is better to use an editor that supports syntax highlighting for Prolog programs (e.g. emacs, vim, atom). Paste the following in the editor and save it as `kb1.pl`.

```
wizard(harry).
wizard(ron).
wizard(hermione).
muggle(uncle_vernon).
muggle(aunt_petunia).
chases(crookshanks, scabbers).
```

Now type `consult('kb1.pl').` at the prompt. Equivalently you can type `consult(kb1).` Now you can ask questions like:

Query you may have in mind Query to type

- Is ron a wizard? `wizard(ron).`
- Is ron a muggle? `muggle(ron).`
- Who is a muggle? `muggle(X).`
- Who does crookshanks chase? `chases(crookshanks, X).`
- Who chases whom? `chases(X, Y).`

Don't forget to put a full stop at the end of each query before pressing return. If you forget to put a full stop, Prolog will react by just doing nothing. This is what you will see on the screen:

```
?- wizard(harry)
|
```

To fix it just type a full stop and press return again:

```
?- wizard(harry)
|      .

Yes
```

If something else goes wrong and Prolog starts writing crazy things on the screen without getting back to showing you the `?- prompt`, type `Ctrl-C` . This should show you the following line:

```
Action (h for help) ?
```

Type `a` (for abort) and press return. This should get you back to the `?- prompt`.

Change the file or paste the content of some of the examples in the lecture notes. You need to make Prolog to read the file again by using `consult`.

In order to leave Prolog type `halt` .

How to test your code locally

If you want to test your code locally before submitting it (which is a good idea), you can paste the content of a test case in a file after your solution; save it; consult it; and then ask the query `test_answer` . at the prompt.

Question **1**

Correct

Mark 1.00 out of 1.00

Define a set of relations in the form of Prolog rules about people according to the following statements.

1. *A person eats something if the person likes that thing.* Use the predicate `eats/2` where the first argument is a person and the second argument is a thing, and the predicate `likes/2` where the first argument is a person and the second is a thing. For example if the fact `likes(bob, chocolate)` is in the knowledge base then the query `eats(bob, chocolate)` must always succeed (be true), even if the fact is not explicitly in the knowledge base.
2. *A person eats something if the person is hungry and the thing is edible.* Use predicates `hungry/1` and `edible/1`.

Please note that you do not need to provide any facts. Only write two rules (two lines).

For example:

Test	Result
<pre>likes(bob, chocolate). hungry(alice). test_answer :- eats(bob, chocolate), writeln('Bob eats chocolate.').</pre>	Bob eats chocolate.
<pre>edible(crisps). hungry(bob). likes(bob, sushi). test_answer :- eats(bob, crisps), writeln('Bob eats crisps.').</pre>	Bob eats crisps.
<pre>/* This example shows how our incomplete definition of rules can lead to unexpected (nonsense) answers. */ likes(alice, rock). likes(alice, jazz). edible(pizza). hungry(bob). test_answer :- eats(alice, rock), writeln('Alice eats rock!').</pre>	Alice eats rock!

Answer: (penalty regime: 0, 15, ... %)

1

`eats(Person, Thing) :- likes(Person, Thing).`

2

`eats(Person, Thing) :- hungry(Person), edible(Thing).`

	Test	Expected	Got	
✓	<pre>likes(bob, chocolate). hungry(alice). test_answer :- eats(bob, chocolate), writeln('Bob eats chocolate.').</pre>	Bob eats chocolate.	Bob eats chocolate.	✓
✓	<pre>edible(crisps). hungry(bob). likes(bob, sushi). test_answer :- eats(bob, crisps), writeln('Bob eats crisps.').</pre>	Bob eats crisps.	Bob eats crisps.	✓

	Test	Expected	Got	
✓	<pre>/* This example shows how our incomplete definition of rules can lead to unexpected (nonsense) answers. */ likes(alice, rock). likes(alice, jazz). edible(pizza). hungry(bob). test_answer :- eats(alice, rock), writeln('Alice eats rock!').</pre>	Alice eats rock!	Alice eats rock!	✓

Passed all tests! ✓

Correct
Marks for this submission: 1.00/1.00.

Information

Unification (matching)

Prolog provides a built-in predicate which takes two arguments and checks whether they unify (match). This predicate is `=/2`. Prolog allows you to use `=` as a prefix or infix operator (both with the same meaning). For example, you can write `=(harry,harry)` or `harry = harry`.

How does Prolog answer the following queries? Think about it first, and then type it into Prolog to check whether you are right.

```
?- =(harry,harry).
?- =(harry,'Harry').
?- =(harry,Harry).
?- =(f(a,b),f(a(b))).
```

When matching, Prolog may instantiate variables. For example:

```
?- X = harry, Y = hermione.
X = harry
Y = hermione
```

Instantiating turns variables to new terms. For example, the following fails:

```
?- X = harry, X = hermione.
false.
```

This is because after the first goal (which is `x = harry`) has been processed and as a result variable `X` is instantiated to `harry` which no longer unifies with `hermione` (as they are different constants), thus the second goal fails.

The built-in predicate `\=/2` works in the opposite way: `Arg1 \= Arg2` is true if `Arg1 = Arg2` fails, and vice versa. (Note that not all Prolog implementations provide `\=/2`. SWI Prolog does, but Sicstus, for example, doesn't.)

Try some queries involving `=` and `\=`. For example:

```
?- s(np(pn(dobby)),vp(v(likes),np(pn(harry)))) = s(np(pn(dobby)),v(likes),np(pn(harry))).
?- s(X,vp(v(sings))) = s(np(pn(dobby)),Y).
?- s(X,vp(v(sings))) = s(np(pn(dobby)),vp(X)).
?- father(X) = X.
?- sarah = AName.
?- lora = AName, sarah \= AName. % this succeeds.
```

Question **2**

Correct

Mark 1.00 out of 1.00

Which of the following *unifies*?

Select one or more:

- ☐ brothers(bob, sam) = brothers(sam, bob)
- ☒ Auckland = Wellington ✓
- ☒ 2 * 2 = 2 * 2 ✓
- ☒ f(g(X)) = f(g(Y)) ✓
- ☐ 2 * 2 = 4 (you may want to try this in Prolog first)
- ☒ anything = anything ✓
- ☒ chch = Christchurch ✓
- ☒ f(X) = f(2) ✓
- ☒ christchurch = Christchurch ✓

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

Let `point(X,Y)` represent a point in a 2D Euclidean space. Define a predicate `reflection/2` that takes two points and is true when the two points are the reflection of each other across the line $y=x$.

For example:

Test	Result
<code>test_answer :- reflection(point(3, 6), point(6, 3)), writeln('OK').</code>	OK
<code>test_answer :- reflection(point(-5, 8), point(X, Y)), writeln(point(X, Y)).</code>	<code>point(8, -5)</code>

Answer: (penalty regime: 0, 15, ... %)

```
1 reflection(point(X1, Y1), point(X2, Y2)) :-
2     X1 = Y2,
3     Y1 = X2.
```

	Test	Expected	Got	
✓	<code>test_answer :- \trreflection(point(3, 6), point(6, 3)), writeln('OK').</code>	OK	OK	✓
✓	<code>test_answer :- \trreflection(point(-5, 8), point(X, Y)), writeln(point(X, Y)).</code>	<code>point(8, -5)</code>	<code>point(8, -5)</code>	✓

Passed all tests! ✓

Correct
Marks for this submission: 1.00/1.00.

Information

Here is a tiny lexicon and mini grammar with only one rule which defines a sentence as consisting of five words: an article, a noun, a verb, and again an article and a noun.

```
word(article,a).
word(article,every).
word(noun,criminal).
word(noun,'big kahuna burger').
word(verb,eats).
word(verb,likes).
```

```
sentence(Word1,Word2,Word3,Word4,Word5) :-
    word(article,Word1),
    word(noun,Word2),
    word(verb,Word3),
    word(article,Word4),
    word(noun,Word5).
```

What query do you have to pose in order to find out which sentences the grammar can generate? In which order will Prolog generate the sentences? Make a prediction about the order, then try it out.

Question 4

Correct

Mark 1.00 out of 1.00

Consider the following Prolog program.

```
f(a).
f(b).
f(c).
g(b).
g(c).
h(c).
k(X):-f(X), g(X), h(X).
```

Assume the user asks the query `?- k(X)`. When searching for the proof (solution), which of the listed (sub)goals are tried, and in what order? If the search stops in fewer than 10 steps, use `'.'` for the remaining steps.

You may want to see the lecture notes or read section 2.2 of the Prolog book on proof search.

- 1

f(a)

✓
- 2

g(a)

✓
- 3

f(b)

✓
- 4

g(b)

✓
- 5

h(b)

✓
- 6

f(c)

✓
- 7

g(c)

✓
- 8

h(c)

✓
- 9

.

✓
- 10

.

✓

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Question 5

Correct

Mark 1.00 out of 1.00

Prolog is commonly used in implementing *expert systems*. In simple words, an expert system emulates the decision-making ability of a human expert. In this question you have to define a set of rules that together work as a very simple expert system to give advice on what type of lenses (if any) should be prescribed for a patient.

First create a new knowledge base and include the following predicates in it:

```
/* tear rate related clauses */
normal_tear_rate(RATE) :- RATE >= 5.
low_tear_rate(RATE) :- RATE < 5.

/* age-related clauses */
young(AGE) :- AGE < 45.
```

Now add a new predicate of the form `diagnosis(Recommend, Age, Astigmatic, Tear_Rate)` to the knowledge base where the arguments are:

- `Recommend` is either `hard_lenses`, `soft_lenses`, or `no_lenses`;
- `Age` will be an integer;
- `Astigmatic` will be either `yes` or `no` (note that these are atoms 'yes' and 'no' not true or false.); and
- `Tear_Rate` will be a positive integer.

The predicate must be the translation of the following natural-language rules to logic:

1. If the patient is young and has a normal tear rate then the type of lenses will depend on astigmatism. If the patient is astigmatic then hard lenses must be recommended, otherwise soft lenses.
2. If the person has a low tear rate then 'no lenses' must be recommended.

Once finished, paste the entire knowledge base into the answer box.

Note that there are some questions (queries) that this expert system cannot answer. For example it cannot recommend anything for people who are over 45 years old and do not have low tear rate.

For example:

Test	Result
test_answer :- diagnosis(hard_lenses, 21, yes, 11), writeln('OK').	OK
test_answer :- diagnosis(X, 45, no, 4), writeln(X).	no_lenses
test_answer :- diagnosis(X, 19, no, 5), writeln(X).	soft_lenses

Answer: (penalty regime: 0, 15, ... %)

```
1  /* tear rate related clauses */
2  normal_tear_rate(RATE) :- RATE >= 5.
3  low_tear_rate(RATE) :- RATE < 5.
4
5  /* age-related clauses */
6  young(AGE) :- AGE < 45.
7
8  diagnosis(no_lenses, Age, Astigmatic, Tear_Rate) :- low_tear_rate(Tear_Rate).
9  diagnosis(hard_lenses, Age, yes, Tear_Rate) :- young(Age), normal_tear_rate(Tear_Rate).
10 diagnosis(soft_lenses, Age, no, Tear_Rate) :- young(Age), normal_tear_rate(Tear_Rate).
11
```

	Test	Expected	Got	
✓	test_answer :- diagnosis(hard_lenses, 21, yes, 11), writeln('OK').	OK	OK	✓

	Test	Expected	Got	
✓	test_answer :- diagnosis(X, 45, no, 4), writeln(X).	no_lenses	no_lenses	✓
✓	test_answer :- diagnosis(X, 19, no, 5), writeln(X).	soft_lenses	soft_lenses	✓

Passed all tests! ✓

Correct

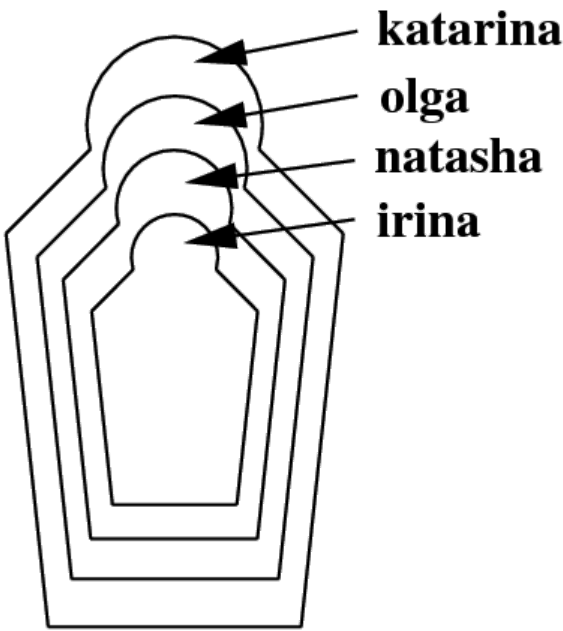
Marks for this submission: 1.00/1.00.

Question **6**

Correct

Mark 1.00 out of 1.00

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` where `directlyIn(X,Y)` means that X is directly in Y. Then, define a recursive predicate `contains/2`, that tells us which doll (directly or indirectly) contains which other doll. For example, the query `contains(katarina,natasha)` should succeed (be true), while `contains(olga,katarina)` should fail.

Please note the spelling of `katarina` and other names.

In some test cases you may see the symbol `\+` which means negation (not).

For example:

Test	Result
<pre>test_answer :- directlyIn(irina, natasha), writeln('OK').</pre>	OK
<pre>test_answer :- \+ directlyIn(irina, olga), writeln('OK').</pre>	OK
<pre>test_answer :- contains(katarina, irina), writeln('OK').</pre>	OK
<pre>test_answer :- contains(katarina, natasha), writeln('OK').</pre>	OK
<pre>% Here we look for all of the dolls which contain irina. test_answer :- findall(P, contains(P, irina), Output), sort(Output, SortedOutput), foreach(member(X,SortedOutput), (write(X), nl)).</pre>	katarina natasha olga

Answer: (penalty regime: 0, 15, ... %)

```
1 | directlyIn(irina, natasha).
2 | directlyIn(natasha, olga).
3 | directlyIn(olga, katarina).
4 | contains(X, Y) :- directlyIn(Y, X).
5 | contains(X, Y) :- directlyIn(Y, Z), contains(X, Z).
6 |
```

	Test	Expected	Got	
✔	test_answer :- directlyIn(irina, natasha), writeln('OK').	OK	OK	✔
✔	test_answer :- \+ directlyIn(irina, olga), writeln('OK').	OK	OK	✔
✔	test_answer :- contains(katarina, irina), writeln('OK').	OK	OK	✔
✔	test_answer :- contains(katarina, natasha), writeln('OK').	OK	OK	✔
✔	% Here we look for all of the dolls which contain irina. test_answer :- findall(P, contains(P, irina), Output), sort(Output, SortedOutput), foreach(member(X,SortedOutput), (write(X), nl)).	katarina natasha olga	katarina natasha olga	✔
✔	directlyIn(patty, lettuce). directlyIn(lettuce, bread). directlyIn(bread, wrapper). test_answer :- contains(wrapper, patty), writeln('OK').	OK	OK	✔

Passed all tests! ✔

Correct
Marks for this submission: 1.00/1.00.

Question **7**

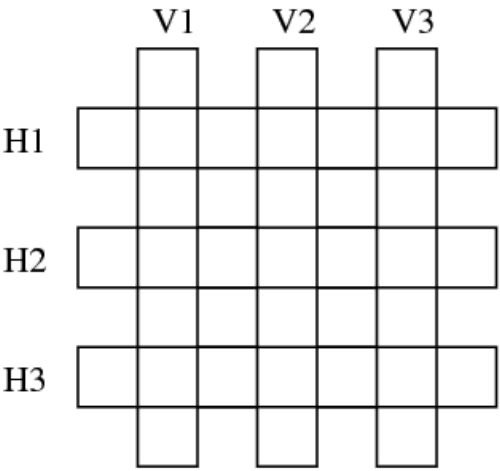
Correct

Mark 1.00 out of 1.00

Here are six Italian words:

astante , astoria , baratto , cobalto , pistola , statale .

They have to be assigned to rows and columns (crossword puzzle fashion) in the following grid:



The following knowledge base represents a lexicon containing these words:

```
word(astante, a,s,t,a,n,t,e).
word(astoria, a,s,t,o,r,i,a).
word(baratto, b,a,r,a,t,t,o).
word(cobalto, c,o,b,a,l,t,o).
word(pistola, p,i,s,t,o,l,a).
word(statale, s,t,a,t,a,l,e).
```

Write a predicate `solution/6` in the form of `solution(V1,V2,V3,H1,H2,H3)` that tells us how to fill in the grid.

Notes

- In this question, the same word can appear in different rows or columns. [As an additional exercise for yourself, think what rule needs to be added in order to have each word used only once.]
- The rules you provide will be used with different sets of six-letter words.
- Prolog has some facilities that make it possible to solve this sort of problems in simpler ways but at this stage we have to solve it in a somewhat tedious way (you need some copy and paste).

For example:

Test	Result
<pre>word(astante, a,s,t,a,n,t,e). word(astoria, a,s,t,o,r,i,a). word(baratto, b,a,r,a,t,t,o). word(cobalto, c,o,b,a,l,t,o). word(pistola, p,i,s,t,o,l,a). word(statale, s,t,a,t,a,l,e). test_answer :- findall((V1,V2,V3,H1,H2,H3), solution(V1,V2,V3,H1,H2,H3), L), sort(L,S), foreach(member(X,S), (write(X), nl)). test_answer :- write('Wrong answer!'), halt.</pre>	<pre>astante,baratto,statale,astante,baratto,statale astante,cobalto,pistola,astoria,baratto,statale astoria,baratto,statale,astante,cobalto,pistola astoria,cobalto,pistola,astoria,cobalto,pistola baratto,baratto,statale,baratto,baratto,statale cobalto,baratto,statale,cobalto,baratto,statale</pre>
<pre>word(abalone, a,b,a,l,o,n,e). word(abandon, a,b,a,n,d,o,n). word(enhance, e,n,h,a,n,c,e). word(anagram, a,n,a,g,r,a,m). word(connect, c,o,n,n,e,c,t). word(elegant, e,l,e,g,a,n,t). test_answer :- findall((V1,V2,V3,H1,H2,H3), solution(V1,V2,V3,H1,H2,H3), L), sort(L,S), foreach(member(X,S), (write(X), nl)). test_answer :- write('Wrong answer!'), halt.</pre>	<pre>abalone,anagram,connect,abandon,elegant,enhance abandon,elegant,enhance,abalone,anagram,connect</pre>

Answer: (penalty regime: 0, 15, ... %)

```
1 solution(V1,V2,V3,H1,H2,H3) :-
2   word(V1, _, A, _, B, _, C, _),
3   word(V2, _, D, _, E, _, F, _),
```

```
4 | word(V3, _, G, _, H, _, I, _),
5 | word(H1, _, A, _, D, _, G, _),
6 | word(H2, _, B, _, E, _, H, _),
7 | word(H3, _, C, _, F, _, I, _).
```

	Test	Expected	Got
✓	<pre>word(astante, a,s,t,a,n,t,e). word(astoria, a,s,t,o,r,i,a). word(baratto, b,a,r,a,t,t,o). word(cobalto, c,o,b,a,l,t,o). word(pistola, p,i,s,t,o,l,a). word(statale, s,t,a,t,a,l,e). test_answer :- findall((V1,V2,V3,H1,H2,H3), solution(V1,V2,V3,H1,H2,H3), L), sort(L,S), foreach(member(X,S), (write(X), nl)). test_answer :- write('Wrong answer!'), halt.</pre>	<pre>astante,baratto,statale,astante,baratto,statale astante,cobalto,pistola,astoria,baratto,statale astoria,baratto,statale,astante,cobalto,pistola astoria,cobalto,pistola,astoria,cobalto,pistola baratto,baratto,statale,baratto,baratto,statale cobalto,baratto,statale,cobalto,baratto,statale</pre>	<pre>astante,baratto,st astante,cobalto,pi astoria,baratto,st astoria,cobalto,pi baratto,baratto,st cobalto,baratto,st</pre>
✓	<pre>word(abalone,a,b,a,l,o,n,e). word(abandon,a,b,a,n,d,o,n). word(enhance,e,n,h,a,n,c,e). word(anagram,a,n,a,g,r,a,m). word(connect,c,o,n,n,e,c,t). word(elegant,e,l,e,g,a,n,t). test_answer :- findall((V1,V2,V3,H1,H2,H3), solution(V1,V2,V3,H1,H2,H3), L), sort(L,S), foreach(member(X,S), (write(X), nl)). test_answer :- write('Wrong answer!'), halt.</pre>	<pre>abalone,anagram,connect,abandon,elegant,enhance abandon,elegant,enhance,abalone,anagram,connect</pre>	<pre>abalone,anagram,co abandon,elegant,er</pre>

	Test	Expected	Got
✔	<pre>word(cookies,c,o,o,k,i,e,s). word(squawks,s,q,u,a,w,k,s). word(balloon,b,a,l,l,o,o,n). word(banquet,b,a,n,q,u,e,t). word(bejewel,b,e,j,e,w,e,l). word(bloated,b,l,o,a,t,e,d). test_answer :- findall((V1,V2,V3,H1,H2,H3), solution(V1,V2,V3,H1,H2,H3), L), sort(L,S), foreach(member(X,S), (write(X), nl)). test_answer :- write('Wrong answer!'), halt.</pre>	<pre>balloon,squawks,bejewel,banquet,bloated,cookies banquet,bloated,cookies,balloon,squawks,bejewel bejewel,bejewel,bejewel,bejewel,bejewel,bejewel bloated,banquet,bejewel,bloated,banquet,bejewel</pre>	<pre>balloon,squawks,be banquet,bloated,co bejewel,bejewel,be bloated,banquet,be</pre>

Passed all tests! ✔



Correct

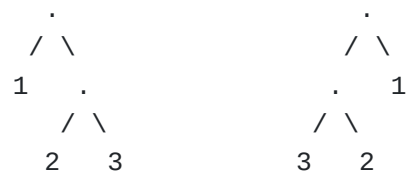
Marks for this submission: 1.00/1.00.

Question 8

Correct

Mark 1.00 out of 1.00

Binary trees are trees where all internal nodes have exactly two children. The mirror of a binary tree is obtained by exchange all the left and right children. For example the following two trees are the mirror of each other:



The smallest binary trees consist of only one leaf node. We will represent leaf nodes as `leaf(Label)` . For instance, `leaf(1)` and `leaf(3)` are leaf nodes, and therefore small binary trees. Given two binary trees `B1` and `B2` we combine them into one binary tree by `tree(B1,B2)`.So, from the leaves `leaf(2)` and `leaf(3)` we can build the binary tree `tree(leaf(2),leaf(3))` . And from the binary trees `leaf(1)` and `tree(leaf(2),leaf(3))` we can build the binary tree `tree(leaf(1), tree(leaf(2), leaf(3)))`.

Define a predicate `mirror/2` that succeeds when the two arguments are binary trees and are the mirror of each other.

For example:

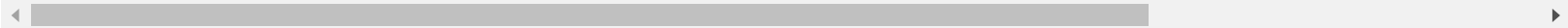
Test	Result
<pre>test_answer :- mirror(leaf(foo), leaf(foo)), write('OK'), halt. test_answer :- write('Wrong answer!'), halt.</pre>	OK
<pre>test_answer :- mirror(tree(leaf(foo), leaf(bar)), tree(leaf(bar), leaf(foo))), write('OK'), halt. test_answer :- write('Wrong answer!'), halt.</pre>	OK
<pre>test_answer :- mirror(tree(tree(leaf(1), leaf(2)), leaf(4)), T), write(T), halt. test_answer :- write('Wrong answer!'), halt.</pre>	<code>tree(leaf(4),tree(leaf(2),leaf(1)))</code>
<pre>test_answer :- mirror(tree(tree(leaf(1), leaf(2)), tree(leaf(3), leaf(4))), T), write(T), halt. test_answer :- write('Wrong answer!'), halt.</pre>	<code>tree(tree(leaf(4),leaf(3)),tree(leaf(2),leaf(1)))</code>

Answer: (penalty regime: 0, 15, ... %)

```
1 mirror(leaf(X), leaf(X)).
2 mirror(tree(A, B), tree(X, Y)) :-
3     mirror(B, X),
4     mirror(A, Y).
5
```

	Test	Expected	Got
✔	<pre>test_answer :- mirror(leaf(foo), leaf(foo)), write('OK'), halt. test_answer :- write('Wrong answer!'), halt.</pre>	OK	OK
✔	<pre>test_answer :- mirror(tree(leaf(foo), leaf(bar)), tree(leaf(bar), leaf(foo))), write('OK'), halt. test_answer :- write('Wrong answer!'), halt.</pre>	OK	OK
✔	<pre>test_answer :- mirror(tree(tree(leaf(1), leaf(2)), leaf(4)), T), write(T), halt. test_answer :- write('Wrong answer!'), halt.</pre>	tree(leaf(4),tree(leaf(2),leaf(1)))	tree(leaf(4),tree(J
✔	<pre>test_answer :- mirror(tree(tree(leaf(1), leaf(2)), tree(leaf(3), leaf(4))), T), write(T), halt. test_answer :- write('Wrong answer!'), halt.</pre>	tree(tree(leaf(4),leaf(3)),tree(leaf(2),leaf(1)))	tree(tree(leaf(4),J

Passed all tests! ✔



Correct
Marks for this submission: 1.00/1.00.