If the description of the problem relies on given facts, these should not be part of the program you submit as a solution. Our test suite will (only) call the predicates which are mentioned in the title of the sections and with free arguments unless otherwise stated.
The state of the Prolog (and recorded) database before and after a program run, should be the same, i.e. if your method of solution involves asserting some clauses, they should be cleaned up by your program and without us having to invoke a special predicate. The cleanup time will be counted in the solution time. For some problems, efficiency is important for the ranking: this is mentioned. For all problems, there is a timeout on the testing, which can make a 'correct' solution unacceptable for the organisation.
Do not use predicates with a name starting with ilps94 or test.

Some problems were adapted or copied from the Internet Programming Contest: we are grateful to Vivek Khera and his team for making the problem statements of the Internet Programming Contest publicly available. Other problems come from folklore/real life/ published papers/student assignments ...

1. triplets/0

Generate all triples (X,Y,Z) that satisfy the conditions that all three are different integers between 0 and 9 (both included), and that

(10*X + Y) / (10*Y+Z) equals X / Z (with infinite precision)

Output of your program should be on the screen and in the form:

```
3 5 9
3 1 6
```

if (3,5,9) and (3,1,6) happen to be the solutions.

Predicate you must supply: triplets/0.
No input; no given facts; writes on screen.

2. spiral/2

Given two positive integers **a** and **b**, output on the screen a rectangle with width **b** and heigth **a** containing the numbers from 1 to **a***b in a spiraling fashion. For **a**=4 and **b**=3, the output is:

```
 1   2   3
10  11   4
 9  12   5
 8   7   6
```

The columns must be properly aligned to the right and occupy N+1 positions, where N is the number of digits in the decimal representation of **a***b.

Predicate you must supply: spiral/2.
Two strictly positive integers as input; no given facts; writes on screen.

3. domino/1

You are given a set of domino stones; these stones have 2 numbers on the same side, one at the left, one at the rigth, more or less like in the example:

2	6

3	3

In the usual game, each participant is given a number of stones, some stone is put on the table, and each player in turn must either add to the stones already on the table, or pass. You win, if you got rid of your stones first. You add legally to the stones already on the table, if there is a stone on the table with a free end containing number x, and you put your stone with its x-side next to the free x-side on the table. Stones with two different numbers have 2 free sides initially - one for each number. Stones with 2 equal numbers on them, have 3 free sides. When a stone is put with a free side next to a free side of another stone, both free sides become occupied.
You will play a variant called **domino solitaire**: you are given a set of stones and you are to finish the game all by yourself, i.e. start with one of your stones on the table (you choose which one), and continue adding stones, until you have no more stones. This can fail of course: the time to fail (or first success) is important for the score on this problem.

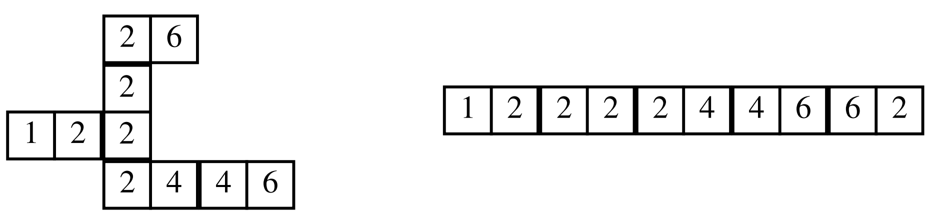
An example: you have the facts:

stone(2,2). stone(4,6). stone(1,2). stone(2,4). stone(6,2).

You write a predicate domino/1 which returns e.g. the list:

[stone(1,2),stone(2,2),stone(2,4),stone(4,6),stone(6,2)]

Only the first solution should be returned.
This list can represent more than one end configuration on the table (see the picture), but the order in the list gives you a correct order for the stones to be added incrementally to a configuration that is always correct according to the rules of the game.



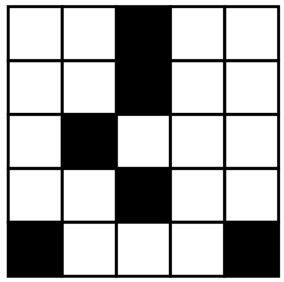
Predicate you must supply: domino/1.
No input; given facts stone/2; unifies answer with argument.

4. crossword/1

You are given an empty crossword puzzle exemplified by the facts

```
dimension(5) . % it is always a square crossword puzzle
black(1,3) .      % meaning: square 1,3 is black
black(2,3) . black(3,2) . black(4,3) . black(5,1) . black(5,5) .
```

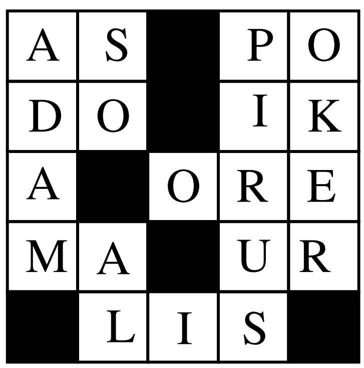
These example facts represent the empty puzzle:



You also get a list of words - example

```
words([do,ore,ma,lis,ur,as,po,so,pirus,oker,al,adam,ik]) .
```

All words have at least two characters. Fill the puzzle with all the words exactly once. If that is impossible, fail. Filling the above puzzle with the above words gives:



Your program should give the solution as the output parameter of crossword/1 and in the form of a list of words in the following order:

```
[as,po,do,ik,ore,ma,ur,lis,adam,so,al,pirus,oker]
```

i.e. first all horizontal words, row by row and in a row as they occur from left to rigth, then all the vertical words, column by column.

If there is more than one solution, your program should produce them all by backtracking. Every non-black square belongs to a word of 2 or more letters.

Your score on this problem depends also on the efficiency of your solution, but correctness comes first.

> ***Predicate you must supply: crossword/1.***
> ***No input; given facts words/1, black/2, dimension/1; unifies answer with argument***

5. loops/1

You get a directed graph in the form:

```
arrow(a,b) .
arrow(b,c) .
arrow(c,c) .
arrow(a,d) .
arrow(d,a) .
```

with obvious meaning. You are to write a predicate loops/1 that returns a list (possibly empty) of all the (minimal) cycles in the graph. A cycle is represented by a list which contains all the nodes in the cycle in the order in which arrows link them and it starts and ends with the same node: a minimal cycle does not properly contain any other cycle. Every minimal cycle of the graph should be given exactly once. For the example, your program could deliver as solution only:

[[c,c],[a,d,a]] or [[c,c],[d,a,d]] or [[a,d,a],[c,c],] or [[d,a,d],[c,c],]

i.e. the order in the list of cycles is irrelevant and so is the actual start node of a cycle.

Predicate you must supply: loops/1.
No input; given facts arrow/2; unifies answer with argument.

6. path/1

On a square board of size N (say 4 in the example, given by the fact size(4) .), there is a starting point (always square 1,1) and an end point (given by a fact e.g.goto(1,4,f) .) There is a correct path between the starting and the end point and it is marked: each square of the path contains the information how to get to the next square in the path, in the form of a fact:

```
goto(1,1,u) . % meaning from square 1,1 go up - i.e. to square 1,2
goto(1,2,u) .
goto(1,3,r) . % meaning from square 1,3 go rigth - i.e. to square 2,3
goto(2,3,r) .
goto(3,3,d) . % meaning from square 3,3 go down - i.e. to square 3,2
goto(3,2,d) .
goto(3,1,r) .
```

An l would have meant 'left' of course and f means final destination reached.
So, the above information represents the board:

r	r	d	
u		d	
u		r	f

There is of course no problem in finding the correct path from 1,1 to 1,4 on such a board. However, a mean demon has removed some information from the board: it has taken away some goto/3 facts, so that the path is no longer completely represented. Still, it wasn't that mean, since no two neighbouring squares in the path have their information removed, the end point is not removed and at most two neighbours of a removed square belong to the correct path (diagonally touching squares are not considered neighbours): you can still easely find the path. However, a second, much meaner demon has introduced information for every square NOT on the path and in such a way, that if you follow this information, you will either run around in circles or fall off the board (in particular, you will not meet 'empty' squares).
Here is an example of such a mutilated board:

l	l	u	r
r		d	d
	d		u
u	d	r	f

You are asked to write a predicate path/1 which delivers the correct path, as a list starting with (1,1) and ending in the endpoint and ordered along the path. So, in the above example, path/1 delivers

X = [(1,1),(1,2),(1,3),(2,3),(3,3),(3,2),(3,1),(4,1)]

Predicate you must supply: path/1.
No input, given facts goto/3; unifies answer with argument.