CSC 372, Spring 2014
Assignment 6
Due: Wednesday, April 16 at 23:00

Use SWI Prolog!

Use SWI Prolog for this assignment. On lectura that's `swipl`.

Use a symbolic link for easy access to the tester and data files

This write-up assumes you've made a symbolic link named `a6`:

```
% ln -s /cs/www/classes/cs372/spring14/a6 .
```

See the assignment 4 write-up for how-to details.

Use `a6/tester`

Use `a6/tester` on lectura to test your solutions. There will be little chance for any fix-up and retest for any mistakes that could have been caught by using `a6/tester`. See the assignment 4 write-up for how-to details.

There are no `tester` cases for `rsg`—see the write-up below for details.

Easy Money!

Due to the time frame for this assignment and not wanting to underweight problems on assignments 7 and 8, I think you'll find that the time required to do this assignment is relatively small with respect to the points assigned.

Singleton variables

`print_stars` is a predicate that prints a specified number of asterisks:

```
?- print_stars(10).
**********
true.
```

Here's a first version of it:

```
% cat print_stars.pl
print_stars(N) :- between(1,N,X), write('*'), fail.
print_stars(N).
```

When we load it, we get two warnings:

```
% pl -l print_stars
...
Warning: /Users/whm/372/a6/print_stars.pl:1:
        Singleton variables: [X]
Warning: /Users/whm/372/a6/print_stars.pl:2:
        Singleton variables: [N]
```

Prolog is pointing out that in the rule on line 1 there's only one occurrence of the variable `X`. If a variable is only used once in a rule it may be being instantiated but its value is not being used. In this case, we don't use the value that `between` instantiates `X` to. Instead, we're just using `between` to produce `N` calls to `write('*')`.

Similarly, the second clause, `print_stars(N).`, is like the `showfoods.` fact on slide 74—it's present to make `print_stars` succeed after the first clause has done the required printing, but failed in the end.

Prolog warns about singletons because it's often due to a variable being misspelled/misnamed.

The proper thing to do in `print_stars` is to use anonymous variables, indicating that we need a placeholder for the term but we don't care about the value.

```
print_stars(N) :- between(1,N,_), write('*'), fail.
print_stars(_).
```

Problem 1. (3 points, ½ point each) `queries.txt`

Create a plain text file named `queries.txt` with queries that answer the questions below.

Using the facts in `a6/fcl.pl`, write queries that answer the following questions:

Who likes carrots?

Who likes baseball and a food?

Who likes baseball and a red food?

Who likes somebody who likes baseball?

Using the facts in `a6/things.pl` write queries that answer the following questions:

What are the colors of non-foods?

What are colors that are a color of both a food and a non-food?

Include the questions with your answers. You do not need to include generated output. We'll grade these by hand. Here's the format we'd like to see in `queries.txt`:

```
What foods are green?
     food(X), color(X,green).
```

That's simply the question you're answering followed by the appropriate query on the next line. Follow each answer/query pair with a blank line.

Please submit a plain text file, not a Word document or PDF, for example.

Problem 2. (2 points) `sequence.pl`

Write a predicate `sequence/0` that outputs the sequence shown in the box on the right ride of slide 66.

```
?- sequence.
10101000
10101001
10101010
10101011
10111000
10111001
10111010
10111011
true.
```

Be sure that `sequence` produces `true` when done, as shown above.

Two notes: (1) Don't overthink this one. (2) Don't just "wire-in" the whole output, like `writeln(10101000), writeln(10101001), ...`

Problem 3. (6 points) `rect.pl`

In this problem you are to implement several simple predicates that work with `rect(width,height)` structures, which represent position-less rectangles that have only a width and height.

`square(+Rect)` asks whether a rectangle is a square.

```
?- square(rect(3,4)).
false.

?- square(rect(5,5)).
true.
```

`landscape(+Rect)` is true iff (if and only if) a rectangle is wider than it is high. `portrait` tests the opposite—whether a rectangle is higher than wide. A square is neither landscape nor portrait.

```
?- landscape(rect(16,9)).
true.

?- landscape(rect(3,4)).
false.

?- portrait(rect(3,4)).
true.

?- portrait(rect(10,1)).
false.

?- landscape(rect(3,3)).
false.

?- portrait(rect(3,3)).
false.
```

`rotate(?R1, ?R2)` has three distinct behaviors:

(1) If `R1` is instantiated and `R2` is not, rotate instantiates `R2` to the rotation of `R1`.
(2) If `R2` is instantiated and `R1` is not, rotate instantiates `R1` to the rotation of `R2`.
(3) If both are instantiated, `rotate` succeeds iff `R1` is the rotation of `R2`.

Examples:

```
?- rotate(rect(3,4),R).
R = rect(4, 3).

?- rotate(R,rect(3,4)).
R = rect(4, 3).

?- rotate(rect(5,7),rect(7,5)).
true.

?- rotate(rect(3,3),R).
R = rect(3, 3).
```

`rotate` should also handle cases like these:

```
?- rotate(rect(3,4),rect(W,H)).
W = 4,
H = 3.

?- rotate(rect(3,X),rect(Y,4)).
false.
```

`smaller(+R1, +R2)` succeeds iff both the width and height of `R1` are respectively less than the width and height of `R2`. Rotations are not considered.

```
?- smaller(rect(3,5), rect(5,7)).
true.

?- smaller(rect(3,5), rect(7,5)).
false.
```

`add(+R1, +R2, ?RSum)` follows the idea of "adding" rectangles that was shown on Ruby slide 224.

```
?- add(rect(3,4),rect(5,6),R).
R = rect(8, 10).

?- add(rect(3,4),rect(5,6),rect(W,H)).
W = 8,
H = 10.

?- add(rect(3,4),rect(5,6),rect(10,10)).
false.

?- X = 10, add(rect(3,4),rect(5,6),rect(X,X)).
false.
```

Assume both terms of `rect` structures are non-negative integers.

If you need more than six fairly short lines of Prolog to implement all the above, you're making the problem way too hard!

Problem 4. (3 points) `bases.pl`

Write a predicate `bases/2` such that `bases(Start,End)` prints the integers from `Start` through `End` in decimal, hex, and binary. Assume that `Start` is non-negative and that `End` is greater than `Start`. Examples:

```
?- [bases].
% bases compiled 0.00 sec, ...
true.

?- bases(0,5).
 Decimal     Hex        Binary
     0         0             0
     1         1             1
     2         2            10
     3         3            11
     4         4           100
     5         5           101
true.

?- bases(1022,1027).
 Decimal     Hex        Binary
  1022       3FE    1111111110
  1023       3FF    1111111111
  1024       400   10000000000
  1025       401   10000000001
  1026       402   10000000010
  1027       403   10000000011
true.
```

Be sure that your predicate succeeds, showing `true`, not `false`.

Below is a predicate `fmttest/0` that shows <u>almost exactly</u> the specifications to use with `format/2`. However, you'll need to do `help(format/2)` and figure out how to output numbers in hex and binary.

```
?- listing(fmttest).
fmttest :-
        format('~tDecimal~t~10|~tHex~t~20|~tBinary~t~35|\n'),
        format('~t~d~6|~t~d~16|~t~d~30|\n', [10, 20, 30]).

true.

?- fmttest.
 Decimal     Hex        Binary
    10        20            30
true.
```

Problem 5. (12 points) **`grid.pl`**

Write a predicate `grid(+Rows, +Cols)` that prints an ASCII representation of a grid based on a specification of rows and columns in English.

Here's an example of a grid with three rows and four columns:

```
?- grid(three,four).
+--+--+--+--+
|  |  |  |  |
+--+--+--+--+
|  |  |  |  |
+--+--+--+--+
|  |  |  |  |
+--+--+--+--+
true.
```

The grid is built with plus signs, minus signs, vertical-bars ("or" bars), and spaces. Lines have no trailing whitespace.

Unless a specification is invalid, `grid` always succeeds, producing the `true` that follows the output.

Here are two more examples:

```
?- grid(two,ten).
+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+
true.

?- grid(one,one).
+--+
|  |
+--+
true.
```

Widths and heights from `one` through `ten` are recognized. The behavior is undefined if other atoms are used.

If a number is used for either dimension instead of an atom, the user is reminded to use English:

```
?- grid(3,four).
Use English, please!
true .

?- grid(three,4).
Use English, please!
true .

?- grid(5,5).
Use English, please!
true .
```

In the three examples immediately above note there's a space between `true` and the period. In fact, they print `true` and then pause for input. In the cases above, ENTER was pressed. Here's what happens with my solution when I type a semicolon instead:

```
?- grid(three,4).
Use English, please!
true ;
false.
```

After we've learned about "cuts" we'll see how to make things like `grid(three,4)` succeed and show just `true` but that comes later. For this problem your solution should behave like mine, first displaying `true` and then pausing for input.

`a6/grid-hint.html` shows a solution for a simplified version of this problem, a predicate `box` that simply prints a rectangle of asterisks. To provide a little extra challenge for those who want it, I'm not showing that code here but please don't hesitate to take a look if you're stumped by `grid`.

Don't make a big problem out of turning `one` into 1, `two` into 2, etc.! Remember that you only need to handle `one` through `ten`.

Problem 6. (9 points) `rsg.pl`

In this problem you are to write two predicates, `rsg/0` and `rsg/1`. `rsg/0` generates a simple random sentence in SVO (subject-verb-object) form. Examples:

```
?- rsg.
Rush Limbaugh cooks pizzas.
true.

?- rsg.
President Obama faxes memos.
true.

?- rsg.
Jim eats memos.
true.
```

A set of facts for `subject`, `verb`, and `object` specify the building blocks:

```
subject(0,'Jim').
subject(1,'Rush Limbaugh').
subject(2,'President Obama').

verb(0,eats).
verb(1,faxes).
verb(2,cooks).

object(0,pizzas).
object(1,memos).
object(2,burgers).
```

You may choose to create a different set of subject, verb, and object facts, hopefully far more creative than mine. If you wish, you can go further than SVO form. Perhaps add an adjective, or maybe even do something in a Mad Libs style (`http://en.wikipedia.org/wiki/Mad_Libs`). Anything with

three or more fields whose contents vary is fine.

The second predicate, `rsg/1`, generates N random sentences using `rsg/0`.

```
?- rsg(5).
Jim cooks burgers.
Jim cooks burgers.
Jim faxes pizzas.
President Obama cooks memos.
Jim faxes burgers.
true.

?- rsg(5).
President Obama cooks burgers.
Jim cooks pizzas.
President Obama cooks pizzas.
Jim eats burgers.
Rush Limbaugh cooks pizzas.
true.
```

`rsg/1` is `rsg(+N)`, i.e., `N` must be instantiated. `N` is assumed to be an integer greater than zero.

Implementation notes

Use `random` to generate three random numbers that are used to select a random subject, verb, and object, or, for the creative, whatever building blocks you pick.

As slide 92 notes, `random(N)` is a structure evaluated by `is/2`.

Picking an appropriate value for `N` in `random(N)` requires you to know how many facts there are for subjects, verbs, and objects. There are ways to compute that with Prolog code but the techniques are beyond what we've covered; just count the facts yourself and use a numeric literal. The example above has the same number of subjects, verbs, and objects but that is not required.

Random numbers are, of course, random. N consecutive `rsg` queries might produce the same verb N times but as N grows, so should the distribution of results.

Your `rsg.pl` should contain whatever set of facts your `rsg/0` uses.

Because you're free to vary the facts and/or sentence structure there's no simple way to test `rsg` in an automated fashion. `a6/tester` has no tests for `rsg`.

Problem 7. Extra Credit `observations.txt`

Submit a plain text file named `observations.txt` with...

(a) (1 point extra credit) An estimate of how long it took you to complete this assignment. To facilitate programmatic extraction of the hours from all submissions have an estimate of hours on a line by itself, more or less like one of these:

```
Hours: 10
Hours: 12-15.5
Hours: 16+
```

Other comments about the assignment are welcome, too. Was it too long, too hard, too detailed? Speak up! I appreciate all feedback, favorable or not.

(b) (1-3 points extra credit) Cite an interesting course-related observation (or observations) that you made while working on the assignment. The observation should have at least a little bit of depth. Think of me saying "Good!" as one point, "Interesting!" as two points, and "Wow!" as three points. I'm looking for quality, not quantity.

Turning in your work

Use the D2L Dropbox named `a6` to **submit a single zip file named `a6.zip` that contains all your work. Do not submit individual files!** If you submit more than one `a6.zip`, we'll grade your final submission. Here's the full list of deliverables:

```
queries.txt
rect.pl
sequence.pl
bases.pl
grid.pl
rsg.pl
observations.txt (for extra credit)
```

Note that all characters in the file names are lowercase.

Have all the deliverables in the uppermost level of the zip. It's ok if your zip includes other files, too.

Miscellaneous

You can use any elements of Prolog that you desire, but the assignment is written with the intention that it can be completed easily using only the material presented on Prolog slides 1-93. In particular, lists are not required. If you think you need you need lists you're overlooking the simpler, intended solution.

Point values of problems correspond directly to assignment points in the syllabus. For example, a 10-point problem on this assignment corresponds to 1% of your final grade in the course.

Feel free to use comments to document your code as you see fit, but note that no comments are required, and no points will be awarded for documentation itself. (In other words, no part of your score will be based on documentation.) A % is comment to end of line. `/* ... */` can be used for block comments, just like in Java.

Remember that late assignments are not accepted and that there are no late days; but if circumstances beyond your control interfere with your work on this assignment, there may be grounds for an extension. See the syllabus for details.

My estimate is that it will take a typical CS junior from 3 to 4 hours to complete this assignment.

Keep in mind the point value of each problem; don't invest an inordinate amount of time in a problem or become incredibly frustrated before you ask for a hint or help. Remember that the purpose of the assignments is to build understanding of the course material by applying it to solve problems. If you reach the three-hour mark, regardless of whether you have specific questions, it's probably time to touch base with us. Give us a chance to speed you up! **Our goal is that everybody gets 100% on this assignment**

AND gets it done in an amount of time that is reasonable for them.

I hate to have to mention it but keep in mind that cheaters don't get a second chance. If you give your code to somebody else and they turn it in, you'll both likely fail the class, and more. (See the syllabus for the details.)