CSC 372, Spring 2015
Assignment 7
Due: Thursday, April 16 at 23:59:59

Use SWI Prolog!

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

Use the tester!

Don't just "eyeball" your output—use the tester! I'll be delighted to help you with using the tester and understanding its output. However, I won't have any sympathy for those who fail tests during grading simply because they didn't want to bother with the tester

Make symbolic links for `a7` and `t` in your assignment 7 directory on lectura, for easy access to the tester and the data files. (And, the tester assumes the `a7` symlink is present.) See the assignment 5 write-up for how-to details.

About the `if-then-else` structure (`->`) and disjunction (`;`)

To encourage thinking in Prolog, you are strictly prohibited from using the `if-then-else` structure, which is represented with `->`. (Section 4.7 in Covington talks about it.)

Disjunction, represented with a semicolon (`;`), is occasionally very appropriate but it's easy to misuse and make a mess. Section 1.10 in Covington talks about it. Here's the rule for us: If you think you've found a good place to use disjunction, ask me about it; but unless I grant you a specific exemption, you are not allowed to use disjunction. (My general rule is this: don't use disjunction unless it avoids significant repetition.)

Easy Money!

Due to the time frame for this assignment and not wanting to underweight problems on assignments 8 and 9, I think you'll find that the time required to do this assignment is relatively small with respect to the points assigned.

Problem 1. (5 points, ½ point each) `queries.pl`

For this problem you are to write some simple queries, packaged up as rules.

`a7/queries-starter.pl` starts like this:

```
:-[a7/fcl].

% Who likes foods with the same color as foods that Mary likes?
q0(Who) :- likes(mary,F), food(F), color(F, C), food(F2),
color(F2,C), likes(Who,F2).

% Who likes carrots?
q1(Who) :- true.

% Who likes baseball and a food?
q2(Who) :- true.
```

q0 above is a completed example. The comment just prior specifies a question, "Who likes foods with the same color as foods that Mary likes?" Following that comment is a query that will answer that question. Let's load up the file and try `q0`:

```
$ swipl -l a7/queries-starter.pl
[...lots of singleton warnings due to the uncompleted queries...]

?- q0(Who).
Who = mary ;
Who = joe ;
false.
```

Your task is to replace the dummy bodies (just `true`) for rules `q1` through `q10`. The first six use the facts in `a7/fcl.pl`; the last four use the facts in `a7/things.pl`. Begin by copying `a7/queries-starter.pl` to `queries.pl`, and then edit `queries.pl`.

When your `queries.pl` is complete you should see behavior like this:

```
$ swipl -l queries.pl
...

?- q1(Who).
Who = bob.

?- q2(Who).
Who = joe ;
Who = mary ;
Who = jim.

...

?- q10(Food).
Food = apple ;
Food = carrot ;
Food = orange ;
Food = rice ;
Food = bagel.
```

Leave the sample rule `q0` in place—the tester uses it.

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

Write a predicate `sequence/0` that outputs the sequence below.

```
?- 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 output verbatim, 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 that represent position-less rectangles having 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.
```

`classify(+Rect,-Which)` instantiates `Which` to `portrait`, `landscape` or `square`, depending on the width and height. If `Rect` is not a two-term `rect` structure, then `Which` is instantiated to `wat`.

```
?- classify(rect(3,4),T).
T = portrait.

?- classify(rect(10,1),T).
T = landscape.

?- classify(rect(3,3),T).
T = square.

?- classify(rect(3),T).
T = wat.
```

```
?- classify(10,T).
T = wat.
```

You may need to use some cuts (slide 112+) to prevent `classify` from producing bogus alternatives.
Here is an example of BUGGY behavior:

```
?- classify(rect(5,7),T).
T = portrait ;   First answer is correct but there should be no alternatives!
T = square ;
T = wat.
```

Needless to say, use your `portrait/1`, `landscape/1`, and `square/1` predicates to write `classify/2`.

`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 257.

```
?- 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 ten mostly short lines of Prolog to implement all the above, you're probably not making good use of unification.

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:

```
$ swipl -l bases
...

?- 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 almost exactly 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. (13 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(three,twenty-one).
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
true.

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

Widths and heights, in English, from `one` through `ninety-nine` are recognized; numbers are one or two hyphen-separated words.

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

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

Hint: Use `number/1` to see if a value is a number rather than a structure.

Invalid specifications produce `Huh?`:

```
?- grid(testing,this).
Huh?
true.

?- grid(one-hundred,twenty-five).      one-hundred is out of range
Huh?
true.

?- grid(---,+++).
Huh?
true.
```

Be careful not to accept invalid combinations of words representing numbers, like `ten-four`, `twenty-twenty`, and `one-fifty`; they, too, should produce the `Huh?` diagnostic. Example:

```
?- grid(ten-four,twenty-twenty).
Huh?
true.
```

`a7/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`.

Note that terms like `ninety-nine`, `thirty-seven`, `fifty-two` are simply two-atom structures with the functor `'-'`. Here's a predicate that simply prints the terms of such a structure:

```
parts(First-Second) :-
     format('First word: ~w; second word: ~w\n', [First,Second]).

?- parts(twenty-one).
First word: twenty; second word: one
true.
```

Problem 6. (6 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 possibilities:

```
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(+N)`, generates N random sentences using `rsg/0`. `N` is assumed to be an integer greater than zero.

```
?- 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.
```

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.

`random(N)` is a structure evaluated by `is/2`. If `N` is an integer, `0 <= random(N) < N`. Examples:

```
?- X is random(5).
X = 4.

?- X is random(5).
X = 1.

?- X is random(5).
X = 3.
```

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 as of press time; 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 random, of course! 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/0` in an automated fashion. For this problem the tester only confirms that `rsg/1` produces the right number of lines of output.

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 the following three examples:

```
Hours: 6
Hours: 3-4.5
Hours: ~8
```

If you want the one-point bonus, be sure to report your hours on a line that starts with `"Hours:"`. Some students are including per-problems times, too. That's useful and interesting data—keep it coming!—but `observations.txt` should have only one line that starts with `Hours:`. If you care to report per-problem times, impress me with a good way to show that data.

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, "Excellent!" as two points, and "Wow!" as three points. I'm looking for quality, not quantity.

Turning in your work

Use the D2L Dropbox named `a7` to **submit a single zip file named `a7.zip` that contains all your work**. If you submit more than one `a7.zip`, your final submission will be graded. Here's the full list of deliverables:

- `queries.pl`
- `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.

Miscellaneous

Here's what `wc` shows for my current solutions:

```
$ wc rect.pl sequence.pl bases.pl grid.pl rsg.pl
  10   40  378 rect.pl
   2    7  108 sequence.pl
  17   29  540 bases.pl
  49  103 1283 grid.pl      (over half the lines are simple facts)
  21   44  406 rsg.pl
  99  223 2715 total
```

You can use any elements of Prolog that you desire other than if-then-else (`->`) and disjunction (`;`), but the assignment is written with the intention that it can be completed easily using only the material presented on Prolog slides 1-119. Note that lists are not required! If you think you need you need lists to do any of the problems on this assignment, 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.) In Prolog, 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 me. Give me a chance to speed you up! **My 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.)