CSC 372, Spring 2014
Assignment 7
Due: Wednesday, April 23 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 `a7`:

```
% ln -s /cs/www/classes/cs372/spring14/a7 .
```

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

Use `a7/tester`

Use `a7/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 `a7/tester`. See the assignment 4 write-up for how-to details.

Problem 1. (2 points) `splits.pl`

This problem reprises `splits.hs` from assignment 1. In Prolog it is to be a predicate `splits(+List,-Split)` that unifies `Split` with each "split" of `List` in turn. Example:

```
?- splits([1,2,3],S).
S = [1]/[2, 3] ;
S = [1, 2]/[3] ;
false.
```

Note that `Split` is not an atom. It is a structure with the functor `/`. Observe:

```
?- splits([1,2,3], A/B).
A = [1],
B = [2, 3] ;
A = [1, 2],
B = [3] ;
false.
```

Here are additional examples. Note that splitting a list with less than two elements fails.

```
?- splits([],S).
false.

?- splits([1],S).
false.

?- splits([1,2],S).
S = [1]/[2] ;
false.
```

```
?- atom_chars('splits',Chars), splits(Chars,S).
Chars = [s, p, l, i, t, s],
S = [s]/[p, l, i, t, s] ;
Chars = [s, p, l, i, t, s],
S = [s, p]/[l, i, t, s] ;
Chars = [s, p, l, i, t, s],
S = [s, p, l]/[i, t, s] ;
Chars = [s, p, l, i, t, s],
S = [s, p, l, i]/[t, s] ;
Chars = [s, p, l, i, t, s],
S = [s, p, l, i, t]/[s] ;
false.
```

My solution is less than 100 bytes long. It uses only these predicates: `append`, `length`, and `>`.

Problem 2. (3 points) `repl.pl`

Write a predicate `repl(?E, +N, ?R)` that unifies `R` with a list that is `N` replications of `E`. If `N` is less than 0, `repl` fails.

```
?- repl(x,5,L).
L = [x, x, x, x, x] ;
false.

?- repl(1,3,[1,1,1]).
true ;
false.

?- repl(X,2,L), X=7.
X = 7,
L = [7, 7] ;
false.

?- repl(a,0,X).
X = [] ;
false.

?- repl(a,-1,X).
false.
```

My solution is 84 bytes.

Problem 3. (5 points) `pick.pl`

Write a predicate `pick(+From, +Positions, -Picked)` that unifies `Picked` with an atom consisting of the characters in `From` at the zero-based positions in `Positions`.

```
?- pick('testing', [0,6], S).
S = tg.

?- pick('testing', [1,1,1], S).
S = eee.

?- pick('testing', [10,2,4], S).
S = si.
```

```
?- between(0,6,P), P2 is P+1, pick('testing', [P,P2], S),
writeln(S), fail.
te
es
st
ti
in
ng
g
false.

?- pick('testing', [], S).
S = ''.
```

If a position is out of bounds, it is silently ignored. My solution uses `atom_chars`, `findall`, `member`, and `nth0`.

Problem 4. (12 points) `polyperim.pl`

Write a predicate `polyperim(+Vertices,-Perim)` that unifies `Perim` with the perimeter of the polygon described by the sequence of Cartesian points in `Vertices`, a list of `pt` structures.

```
?- polyperim([pt(0,0),pt(3,4),pt(0,4),pt(0,0)],Perim).
Perim = 12.0 ;
false.

?- polyperim([pt(0,0),pt(0,1),pt(1,1),pt(1,0),pt(0,0)],Perim).
Perim = 4.0 ;
false.

?- polyperim([pt(0,0),pt(1,1),pt(0,1),pt(1,0),pt(0,0)],Perim).
Perim = 4.82842712474619 ;
false.
```

The polygon is assumed to be closed explicitly, i.e., **assume** that the first point specified is the same as the last point specified.

There is no upper bound on the number of points but at least four points are required, so that the minimal path describes a triangle. (Think of it as ABCA, with the final A "closing" the path.) If less than three points are specified, a message is produced:

```
?- polyperim([pt(0,0),pt(3,4),pt(0,4)],Perim).
At least a four-point path is required.
false.
```

This is not a course on algorithms so keep things simple! Calculate the perimeter by simply summing the lengths of all the sides; don't worry about intersecting sides, coincident vertices, etc.

Be sure that `polyperim` produces only one result.

Implementation notes

Consider writing a predicate `pair(+List,-Pair)`:

```
?- pair([a,b,c,d],Pair).
Pair = [a, b] ;
Pair = [b, c] ;
Pair = [c, d] ;
false.
```

Use `append` to write `pair`. Then use `pair` with `findall`, `sumlist`, and a predicate that computes the length of a line between two points.

Note that `is/2` supports `sqrt`:

```
?- Y = 3, X is sqrt(Y**2+3).
Y = 3,
X = 3.4641016151377544.
```

Problem 5. (16 points) `switched.pl`

This problem is a reprise of `switched.rb` from assignment 4. `a7/births.pl` has a subset of the baby name data, represented as facts. Here are the first five lines:

```
% head -5 a7/births.pl
births(1950,'Linda',f,80437).
births(1950,'Mary',f,65461).
births(1950,'Patricia',f,47942).
births(1950,'Barbara',f,41560).
births(1950,'Susan',f,38024).
```

`births.pl` only holds data for 1950-1959, and names with less than 70 births are not included.

Your task is to write a predicate `switched(+First,+Last)` that prints a table much like that produced by the Ruby version. To save a little typing, `switched` assumes that the years specified are in the 20th century.

```
?- switched(51,58).
            1951  1952  1953  1954  1955  1956  1957  1958
Dana        1.19  1.20  1.26  1.29  1.00  0.79  0.67  0.64
Jackie      1.40  1.29  1.14  1.13  1.11  0.94  0.72  0.57
Kelly       4.23  2.74  3.73  2.10  2.32  1.77  0.98  0.51
Kim         2.58  1.82  1.47  1.08  0.61  0.30  0.17  0.12
Rene        1.43  1.32  1.15  1.24  1.13  0.88  0.87  0.89
Stacy       1.06  0.81  0.62  0.47  0.44  0.36  0.29  0.21
Tracy       1.51  1.14  1.02  0.73  0.56  0.55  0.59  0.59
true.
```

If no names are found, `switched` isn't very smart; it goes ahead and prints the header row:

```
?- switched(52,53).
            1952  1953
true.
```

If you want to make your `switched` smarter, that's fine—we won't test with any spans that produce no

names. Also, we'll only test with spans where the first year is less than the last year.

In the Ruby version I made the mistake of directing that names with less than 100 births be dropped, creating some pesky holes in the data. My bad! The Prolog version doesn't do that dropping, but a name is not shown in the table unless there are at least 100 male and 100 female uses in the first and last years of the range being queried.

Names are left-justified in a ten-wide field. Below is a `format` call that does that. Note that the dollar sign is included only to clearly show the end of the output.

```
?- format("~w~t~10|$", 'testing').
testing   $
true.
```

Outputting the ratios is a little more complicated. I'll spare you the long story but I use `sformat`, like this:

```
?- sformat(Out, '~t~2f~6|', 2.34), write(Out).
  2.34
Out = "  2.34".
```

The call above instantiates `Out` to a six-character string (which is actually a list of ASCII character codes) and `write(Out)` outputs it.

You can use `help(format/2)` if you're curious about the details of using `~t` but the essence is that you can use `~N|` to specify that a field extend to column `N`, and then put a `~t` on the left, right, or both sides of a specifier like `~w` or `~f` to get right, left, or center justification, respectively.

To consult `a7/births.pl` when you consult `switched.pl`, put the following line in your `switched.pl`.

```
:-['a7/births.pl'].
```

That construction, `:-` followed by a query, causes the query to be executed when the file is consulted.

You might use that to cause a couple of tests to be run when the file is loaded. Below I define `test/0` to do a couple of `switched` queries, putting a line of dashes between them. I then invoke it with `:-test.`

```
test :- switched(51,58), writeln('-----'), switched(51,52).

:-test.
```

That invocation of `test` must follow the definition of `switched` and consulting `a7/births.pl`.

You'll see that with `swipl -l switched` the output from `test` appears before "`Welcome to SWI-Prolog...`"

Be sure to comment out lines like `:-test.` before turning in your solution. (That output will cause `a7/tester` failures, too, as you'd expect.)

My Prolog solution is less than half the length of my Ruby solution but it's easy to get lost on this problem if you don't come up with a good set of helper predicates. I suggest that you give it a try on your own but if it starts to get ugly, take a look at /cs/www/classes/cs372/spring14/a7/switched-hints.pdf to see how

I broke it down. The points assigned to this problem are based on the assumption that you will take a look at the hints. Without the hints, and based on your current level of Prolog knowledge, I might assign this problem 25 points.

Problem 6. (16 points) `iz.pl`

In this problem you are to write a predicate `iz/2` that evaluates expressions involving atoms and a set of operators. Let's start with some examples:

```
?- S iz abc+xyz.        % + concatenates two atoms.
S = abcxyz .

?- S iz (ab + cd)*2.    % *N produces N replications of the atom.
S = abcdabcd .

?- S iz -cat*3.         % - is a unary operator that produces a reversed copy of the atom.
S = tactactac .

?- S iz -cat+dog.
S = tacdog .

?- S iz abcde / 2.      % / N produces the first N characters of the atom.
S = ab .

?- S iz abcde / -3.     % With -N, / produces last N characters
S = cde .

?- N is 3-5, S iz (-testing)/N.
N = -2,
S = et .

?- S iz aaa // 'X'.     % // Atom wraps the operand with Atom on both ends.
S = 'XaaaX' .

?- S iz abc // ['*'*2,'<'*3]. % //List wraps the left operand with the first and
S = '**abc<<<' .              % second elements of List, always a two-element list.
```

Behavior is undefined if the user asks `iz` for an alternative (by typing a semicolon). In all the cases above the user hit ENTER when presented with the first result and Prolog responded by printing a period.

The atoms `comma`, `dollar`, `dot`, and `space` do not evaluate to themselves but instead evaluate to `','`, `'$'`, `'.'`, and `' '`, respectively. (They are similar to `e` and `pi` in arithmetic expressions evaluated with `is/2`.)

```
?- S iz space.
S = ' ' .

?- S iz dollar*3.
S = $$$ .

?- S iz (comma+dot*(3+2)+space+dollar) // ['>>>', '<<<'].
S = '>>>,..... $<<<' .
```

Here is a summary for `iz/2`.

> `-Atom iz +Expr` unifies `Atom` with the result of evaluating `Expr`, a structure representing a calculation involving atoms. The operators (functors) are as follows:
>
> | | |
> |---|---|
> | `E1+E2` | Concatenates the atoms produced by evaluating `E1` and `E2` with `iz`. |
> | `E*N` | Concatenates `E` (evaluated with `iz`) with itself `N` times. (Just like Ruby.) `N` is a term that can be evaluated with `is/2` (repeat, `is/2`). |
> | `E/N` | Produces the first (last) `N` characters of `E` if `N` is greater than (less than) 0. If `N` is zero, an empty atom (two single quotes with nothing between them) is produced. `N` is a term that can be evaluated with `is/2`. The behavior is undefined if `abs(N)` is greater than the length of `E`. |
> | `E1//E2` | Produces `E2+E1+E2`. |
> | `E1//[E2,E3]` | Produces `E2+E1+E3`. |
> | `-E` | Produces reversed `E`. |
>
> The behavior of `iz` is undefined for all cases not covered by the above. We simply won't test with things like `1+2`, `abc*xyz`, `a//[b]`, etc.

Here are some cases that demonstrate that the right-hand operand of `*` and `/` can be an arithmetic expression:

```
?- X = 2, Y= 3, S iz 'ab' * (X+Y*3).
X = 2,
Y = 3,
S = abababababababababab .

?- S = '0123456789', R iz S + -S, End iz R / -(2+3).
S = '0123456789',
R = '01234567899876543210',
End = '43210' .
```

Implementation notes

One of the goals of this problem is to reinforce the idea that Prolog creates a tree structure that corresponds to an expression with operators like `a+b*3`. `is/2` interprets a tree as an arithmetic expression. `iz/2` interprets a tree as a "string expression". Note the contrast:

```
?- X is pi + e*3.          % using is
X = 11.296438138966929.

?- X iz pi + e*3.          % using iz
X = pieee .
```

It's important to understand Prolog itself parses the expression and builds a corresponding structure that takes operator precedence into account. `display/1` shows the tree:

```
?- display(pi + e*3).
+(pi,*(e,3))
true.
```

Processing of syntactically invalid expressions like `abc + + xyz` never proceeds as far as a call to `iz`.

Along with arithmetic and comparisons my current solution uses only these predicates: `append`, `atom`, `atom_chars`, `concat_atom`, `length`, `repl` (from problem 2), and `reverse`. It is 898 bytes long.

Below is some code to get you started. It fully implements the + operation.

```
% cat a7/iz0.pl
:-op(700, xfx, iz).   % Declares iz to be an infix operator. The leading :- causes the
                      % line to be evaluated as a goal, not consulted as a fact.

iz(A, A) :- atom(A).
iz(R, E1+E2) :- iz(R1,E1), iz(R2,E2), concat_atom([R1,R2],R).
```

Here are examples that use the version of `iz` just above.

```
?- ['a7/iz0.pl'].
% a7/iz0.pl compiled 0.00 sec, 3 clauses
true.

?- X iz abc+def.
X = abcdef .

?- X iz abc+def, Y iz X+'...'+X.
X = abcdef,
Y = 'abcdef...abcdef' .

?- X iz a+b+(c+(de+fg)+hij+k)+l.
X = abcdefghijkl .
```

Let's look at the code provided above. Here's the second clause:

```
iz(R, E1+E2) :- iz(R1,E1), iz(R2,E2), concat_atom([R1,R2],R).
```

Consider the goal '`X iz ab+cd`'. It unifies with the head of the above rule like this:

```
?- (X iz ab+cd) = iz(R,E1+E2).
X = R,
E1 = ab,
E2 = cd.
```

The first goal in the body of the rule is `iz(R1,ab)`, observing that `E1` is instantiated to `ab`. That goal unifies with the head of this rule:

```
iz(A,A) :- atom(A).
```

This is the base case for the recursive evaluation of `iz`. It says, "If `A` is an atom then `A` is the result of evaluating that atom." Another way to read it: "An atom evaluates to itself." The result is that `iz(R1,ab)` instantiates `R1` to `ab`.

It's important to recognize that because the `iz(R, E1+E2)` rule is recursive, it'll handle every tree composed of + operations.

Here are the heads for the other `iz` rules that I've got:

```
iz(R, E1 * NumExpr) :- ...
iz(R, E1 / NumExpr) :- ...
iz(R, E1 // [First0,Last0]) :- ...
iz(R, E1 // E2) :- ...
iz(R, -(E)) :- ...
```

Via recursion these five heads and the one above for + handle all possible combinations of operations, like this one:

```
?- X iz (-(ab+cde*4)/6+xyz)//['Start>','<'+(end*3+zz*2)].
X = 'Start>edcedcxyz<endendendzzzz' .
```

If you find yourself wanting to write rules like `iz(R, (E1+E2) * NumExpr) :- ...` then STOP! You're probably not recognizing that the above rules cover everything.

On the slides I fail to mention that Prolog requires some sort of separation between operators. Consider this:

```
?- X iz abc+-abc.    % No space between * and -
ERROR: Syntax error: Operator expected
ERROR: X iz abc
ERROR: ** here **
ERROR: +-abc .
```

To make it work, add a space or parenthesize:

```
?- X iz abc+ -abc.
X = abccba .

?- X iz abc+(-abc).
X = abccba .
```

This issue only arises with unary operators, of course.

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: 6
Hours: 3-4
Hours: 8+
```

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 `a7` to **submit a single zip file named `a7.zip` that contains all your work. Do not submit individual files!** If you submit more than one `a7.zip`, we'll grade your final submission. Here's the full list of deliverables:

```
splits.pl
repl.pl
pick.pl
polyperim.pl
switched.pl
iz.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 1-134.

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.

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 7-9 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 six-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.)