CSc 372, Fall 2006
Assignment 8
Due: Thursday, November 16 at 23:59:59 MST

Problem 1. (5 points) ruler.pl

Write a predicate `ruler(+N)` that prints a two-line "ruler" of length `N`. `ruler` produces no output and fails if `N` is less than 10 or greater than 99.

```
?- ruler(25).
         1         2
1234567890123456789012345

Yes
?- ruler(40).
         1         2         3         4
1234567890123456789012345678901234567890

Yes
?- ruler(5).

No
?- ruler(500).

No
```

The lines printed by `ruler` never have trailing whitespace.

Problem 2. (7 points) splits.pl

This problem reprises `splits.sml` 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] ;
No
```

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

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

Here are the other interesting cases:

```
?- splits([],S).
No

?- splits([1,2],S).
S = [1]/[2] ;
No
```

```
?- splits([a,b,c,d],S).
S = [a]/[b, c, d] ;
S = [a, b]/[c, d] ;
S = [a, b, c]/[d] ;
No
```

My solution is 157 bytes long and makes use of `append`, `findall`, `length`, and `member`.

Problem 3. (15 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(2,0),pt(2,1),pt(0,1)],Perim).
Perim = 6.0

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

?- polyperim([pt(-3,2),pt(7,2),pt(7,-4),pt(-3,-4)],Perim).
Perim = 32.0
```

There is no upper bound on the number of points but at least three points (a triangle) are required. If less than three points are specified, a message is produced:

```
 ?- polyperim([pt(2,0),pt(2,1)],Perim).
At least three points are required.

No
```

Because this is not an algorithms class 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. If asked for an alternative, it fails:

```
?- polyperim([pt(0,0),pt(3,0),pt(3,4)],Perim).
Perim = 12.0 ;
No
```

Implementation notes

There is a square root available:

```
?- X is sqrt(2).
X = 1.41421
```

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

```
?- pair([1,2,3,4],Pair).
Pair = [1, 2] ;
Pair = [2, 3] ;
Pair = [3, 4] ;
No
```

Use `append` to write `pair`. Then use `pair` with `findall` and `sumlist`.

Problem 4. (25 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 reverses the atom.
S = tactactac

?- S iz abcde / 2.      % /N produces the first or (if N negative) characters of the atom.
S = ab

?- S iz abcde / -3.
S = cde

?- 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
                             % second elements of List, always a two-element list.

S = '**abc<<<'
```

Four atoms are interpreted as standing for a single character: `comma`, `dot`, `space`, and `dollar`.

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

Here is a complete but concise 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, `i`**<u>`S`</u>**`/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`	Reverses `E`.

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`, shown on slide 68.)

The behavior of `iz` is undefined for all undescribed cases. We simply won't test with things like `1+2`, `abc*xyz`, `a//[b]`, etc.

Implementation notes

This problem is similar in concept to the example on slides 184-187 in the Standard ML set but the specification of expressions is very natural in Prolog.

Note that Prolog itself handles the parsing of the expressions; 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`, and `reverse`. It is 875 bytes long. That includes a predicate `repl(+X, +N, -List)` that unifies `List` with a list consisting of `N` copies of `X`. If we get into low-level list processing on Thursday, November 9, you'll be able to write that yourself, if you find it useful. If we don't get that far, remind me and I'll post that predicate to the list.

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

```
% cat /home/cs372/fall06/a8/iz0.pl
iz(A, A) :- atom(A).
iz(R, E1+E2) :- iz(R1,E1), iz(R2,E2), concat_atom([R1,R2],R).

:-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.
```

Here is the above code in use: (the `Yes`'s are not shown)

```
?- [iz0].

?- X iz abc.
X = abc

?- 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 = _G347
R = _G347
E1 = ab
E2 = cd
```

The first goal in the body of the rule is `iz(R1,ab)`, noting 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.

YOUR SOLUTION DOES NOT NEED TO ACCOMMODATE BACKTRACKING. You may find that your solution does something like this when asked for alternatives:

```
?- X iz dot+dot.
X = .. ;
X = '.dot' ;
X = 'dot.' ;
X = dotdot ;
No
```

You only need to be sure that the first result produced by `iz` is correct, as is the first result above (`X = ..`). We'll later see a way to prevent undesired backtracking in this problem.

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 5. (Extra Credit) `extra.txt`

Create a text file named `extra.txt` with answers to the following questions. **DO NOT submit a Word document, PDF, rich text file, etc.**—I want plain text.

(a) (1 point extra credit) Estimate how long it took you to complete this assignment. Other comments about the assignment are welcome, too—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 "OK" as 1 point, "Interesting!" as 2 points, and "WOW!" as 3 points. I'm looking for quality, not quantity.

Deliverables

The deliverables for this assignment are these files: `ruler.pl`, `splits.pl`, `polyperim.pl`, `iz.pl`, and if so inclined, `extra.txt`. That list can be found on `lectura` in `/home/cs372/fall06/a8/delivs`.

Use the `turnin` tag `372_8` to submit your solutions.

Corrections and FAQs, late submissions, `turnin`, retests, etc.

Refer to the write-up for the first assignment for details on these topics and similar ones.

Miscellaneous

Aside from writing something like the `repl` function mentioned in the `iz` implementation notes, slides 1-101 have all the information you need to do this assignment.

If time permits `fall06/a8/tester` will appear. If it does, use it! If not, eyeball it!

Solutions will be scored only on correctness.

Keep in mind the point value of each problem; don't invest an inordinate amount of time in a problem before you ask for help with it. Remember that the purpose of the assignments is to build understanding of the course material by applying it to solve problems. Seek the help of Poorna and me as needed to meet your time budget. Poorna's been focusing on Prolog for over a month. Take advantage of his work!

Feel free to use comments to document your code as you wish but note that no comments, not even your name, are required.

I hate to have to mention it but keep in mind that I don't give cheaters a second chance to waste everybody's time. 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.)