Homework #3: Chapter 3 & Prolog

Issued: Tuesday, October 10
Due: Tuesday, October 24

There are two parts to this homework assignment. The first part allows you to reinforce your reading, by performing Exercises from the end of Chapter 3 (Names, Scopes, and Bindings). The second part allows you to program in a language introduced in lecture: Prolog.

Part 1: Textbook Exercises

First, read the question in or textbook. Then, read what this handout has to say about the question.

3.2 This question asks you to differentiate static and stack allocation. Use Java and Scheme for your two examples, respectively.

3.4 This question asks you to differentiate lifetime and visibility (aka, extent and scope). Use Java and Scheme for your examples.

3.5 This question asks you to differentiate declaration-order rules, for different languages.

3.6 This question asks you to draw stack frames.

3.13 This question asks you to differentiate shallow and deep binding (aka, late and early binding), in dynamically-scoped languages.

The question asks "What does this program print?" Of course, it doesn't *print* anything. The real question is: What is the result of evaluating the expression `(A)`?

We discussed `define` and `lambda` during the lecture on Scheme. Recall that Scheme is actually statically scoped. In Scheme, the result of evaluating `(A)` is `2`.

3.19 This question asks you to differentiate static and dynamic scope *and* deep and shallow binding.

Part 2: Prolog

Why Prolog?

Prolog is a logic-based and declarative language. It may not be "popular," but it is the most popular language of its kind. It has been used for artificial intelligence, natural language understanding, and expert systems (e.g., IBM's chess/Jeopardy maven: Watson). Prolog and SQL queries can be seen as similar.

Prolog was designed by Alain Colmerauer, from Aix-Marseille University, in 1972.

Translator

In our lab, `onyx` is the home-directory file server for its nodes (e.g., `onyxnode01`). There is also a shared directory for "apps" at `/usr/local/apps`. Nodes share a translator for Prolog, named `gprolog`, which is installed below `/usr/local/apps`, which is a non-standard location.

Due to network constraints, `onyx` can be reached from the public Internet, but a node can only be reached from `onyx`. So, you can SSH and login to `onyx`, then SSH and login to a node.

An easy way to use `gprolog`, from a node, is to permanently add a line to the end of your `.bashrc` file. To do so, login to a random node, from `onyx`, by executing the script:

```
pub/bin/sshnode
```

Then, execute the script:

```
pub/bin/bashrc
```

Don't change your `$PATH`; just execute the script. Then, logout from the node and login to a node.

Documentation

Prolog lecture slides are at:

```
pub/slides/slides-prolog.pdf
```

Prolog is demonstrated by:

```
pub/sum/prolog
```

Prolog is also described in Section 12.2 of our textbook.

Links to programming-language documentation can be found at:

```
http://csweb.boisestate.edu/~buff/pl.html
```

Assignment

Write and fully demonstrate a program that selects a set of acceptable meeting times for a set of people. Each person provides a set of free time slots. For example:

```
pub/hw3/data.pl
```

shows that Bob has three time slots that are free, one of which is 7:00AM–8:30AM. Bob's not very busy!

Hints and Advice

Start with a simpler problem. A skeletal solution is:

```
pub/hw3/meetone.pl
```

When complete, this program will print the names of people who can meet from 8:30AM–8:45AM. This will be Ann, Carla, and Dave. If the "query" was for 5:30AM–6:45AM there would be no solutions.

Then, extend your solution to the complete problem. A skeletal solution is:

```
pub/hw3/meet.pl
```

When complete, this program will print a list of compatible meeting times for Ann, Bob, and Carla: 8:00AM–8:30AM, 10:00AM–10:15AM, and 8:00PM–8:30PM.

Given two times slots, there are several ways in which they can overlap, but some are symmetric. Draw pictures. Your predicate will need a rule for each way. Use the `\==` predicate to avoid zero-length meetings. A big hint is for your predicate to "return" a new time slot modeling the overlap.

A general Prolog hint is to use one or more variables to pass data "into" a predicate, and another variable to "return" data.

You'll also need a predicate for comparing two times (e.g., `lte`).

Test your solution thoroughly, by modifying `data.pl` and the list of people who want to meet.