CSC 372, Spring 2015
Assignment 9
Due: Wednesday, May 6 at 23:59:59

Warm-up suggestions

Here are some built-in predicates I suggest writing just for practice: `length`, `member`, `append`, `numlist`, `sumlist`, `delete`, `reverse`, and `nextto`. Where applicable, like for `reverse`, try writing one version that uses `append` and another version that instead uses the `[H|T]` construct to build lists.

General advice

If you think you need to use arithmetic or something like `between` on the first four problems, you're overlooking how Prolog naturally generates alternatives. Take a look at the cons-based version of `pair` in my `polyperim` solution.

`pipes.pl` is the only problem on which using `assert` and `retract` is appropriate.

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 `a9` and `t` in your assignment 9 directory on lectura, for easy access to the tester and the data files. (And, the tester assumes the `a9` symlink is present.) See the assignment 5 write-up for how-to details.

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

The rules for `if-then-else` and disjunction are the same as for assignment 7.

Problem 1. (3 points) `noconsec.pl`

Write a Prolog predicate `noconsec(+L)` that succeeds iff (if and only iff) the list `L` has no consecutive elements that are identical. Assume that `L` is a list.

Restriction: You may not use `nextto/3`.

```
?- noconsec([3,1,5,4,3,4]).
true.

?- noconsec([3,1,5,4,4,3]).
false.

?- atom_chars('noon',L), noconsec(L).
```

```
false.

?- atom_chars('nope',L), noconsec(L).
L = [n, o, p, e].

?- noconsec([1]).
true.

?- noconsec([]).
true.
```

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

Write a Prolog predicate `rotate(+L,?R)` that for an instantiated list `L` instantiates `R` to each unique list that is a left rotation of `L`. For example, the list `[1,2,3]` can be rotated left to produce `[2,3,1]` which in turn can be rotated left again to produce `[3,1,2]`.

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

?- rotate([1,2,3],L), writeln(L), fail.
[1,2,3]
[2,3,1]
[3,1,2]
false.

?- rotate([1], R).
R = [1] ;
false.

?- rotate([], R).
false.
```

Additionally, `rotate` can be asked whether the second term is a rotation of the first term:

```
?- rotate([a,b,c],[c,a,b]).
true ;
false.

?- rotate([a,b,c],[c,b,a]).
false.
```

Problem 3. (3 points) `outin.pl`

Write a Prolog predicate `outin(+L,?R)` that generates the elements of the list `L` in an "outside-in" sequence: the first element, the last element, the second element, the next to last element, etc. If the list has an odd number of elements, the middle element is the last one generated.

Restriction: You may not use `is/2`.

```
?- outin([1,2,3,4,5],X).
```

```
X = 1 ;
X = 5 ;
X = 2 ;
X = 4 ;
X = 3 ;
false.

?- outin([1,2,3,4],X).
X = 1 ;
X = 4 ;
X = 2 ;
X = 3 ;
false.

?- outin([1],X).
X = 1 ;
false.

?- outin([],X).
false.
```

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

Write a Prolog predicate `btw(+L,+X,?R)` that instantiates `R` to copies of `L` with `X` inserted between each element in turn.

Restriction: You may not use `append` or `between`.

```
?- btw([1,2,3,4,5],---,R).
R = [1, ---, 2, 3, 4, 5] ;
R = [1, 2, ---, 3, 4, 5] ;
R = [1, 2, 3, ---, 4, 5] ;
R = [1, 2, 3, 4, ---, 5] ;
false.

?- btw([1,2],***,R).
R = [1, ***, 2] ;
false.

?- btw([1],***,R).
false.

?- btw([],x,R).
false.
```

Problem 5. (20 points) `fsort.pl`

Imagine that you have a stack of pancakes of varying diameters that is represented by a list of integers. The list `[3,1,5]` represents a stack of three pancakes with diameters of 3", 1" and 5" where the 3" pancake is on the top and the 5" pancake is on the bottom. If a spatula is inserted below the 1" pancake (putting the stack `[3,1]` on the spatula) and then flipped over, the resulting stack is `[1,3,5]`.

In this problem you are to write a predicate `fsort(+Pancakes,-Flips)` that instantiates `Flips` to a sequence of flip positions that will order `Pancakes`, an integer list, from smallest to largest, with the largest pancake (integer) on the bottom (at the end of the list). `fsort` stands for "flip sort".

`fsort` does not produce a sorted list—its only result is the flip sequence.

The flip position is defined as the number of pancakes on the spatula. In the above example the flip position is `2`. `Flips` would be instantiated to `[2]`.

Below are some examples. Note the use of a set of `case` facts to show a series of examples with one query.

```
% cat fsortcases.pl
case(a, [3,1,5]).
case(b, [5,4,3,2,1]).
case(c, [3,4,5,1,2]).
case(d, [5,1,3,1,4,2]).
case(e, [1,2,3,4]).
case(f, [5]).

% swipl
...
?- [fsortcases,fsort].
% fsortcases compiled 0.00 sec, 7 clauses
% fsort compiled 0.00 sec, 10 clauses
true.

?- case(_,L), fsort(L,Flips).
L = [3, 1, 5],
Flips = [2] ;

L = [5, 4, 3, 2, 1],
Flips = [5] ;

L = [3, 4, 5, 1, 2],
Flips = [3, 5, 2] ;

L = [5, 1, 3, 1, 4, 2],
Flips = [6, 2, 5, 2, 4, 3] ;

L = [1, 2, 3, 4],
Flips = [] ;

L = [5],
Flips = [].
```

Your solution needs only to produce a sequence of flips that results in a sorted stack; the sequences it produces do NOT need to match the sequences shown above. There are some requirements on the flips, however: (1) All flips must be between 2 and the number of pancakes, inclusive. (2) There must be no consecutive identical flips, like `[5,3,3,4]`. (3) `fsort` must always generate exactly one solution.

You may assume that stacks always have at least one pancake and that pancake sizes are always greater than zero.

"Pancake sorting" is a well-known problem. I first encountered it in 1993's Internet Programming Contest. There's even a Wikipedia article about pancake sorting. (Read it!) I debated whether to go with this problem because it's so well known but it's a fun problem and it's interesting to solve in Prolog, so here it is. I did Google up one solution but it's got some issues! I strongly encourage you to build your Prolog skills by solving this problem without Google's assistance—the final exam will be closed-Google!

The clauses in my current solution have a total of seventeen goals. I don't use `is/2` at all. I do use `max_list`. You might find `nth0` and/or `nth1` to be useful; if you look at the documentation closely you'll see they can be used to extract and find values.

I've placed this problem, `fsort.pl`, early in the line-up in hopes of getting you thinking about it early but don't get hung up on it.

Problem 6. (15 points) `pipes.pl`

In this problem you are to write a simple command interpreter to perform manipulations on a set of named pipes having given lengths and diameters. (Picture pipes like you buy at Home Depot, not UNIX pipes!) The commands are in the form of Prolog terms. The calculator shown on slide 192 and following is a good starting point for this problem.

The interpreter provides the following commands:

`pipes` Show the current set of pipes. The pipes are shown in alphabetical order by name.

`weld(A, B)`
The pipe named `B` is welded onto the pipe named `A`. `A` and `B` must have the same diameter. After welding, `A` has the combined length of `A` and `B`. Pipe `B` no longer exists.

`cut(A, L, B)`
A section of length `L` is cut from the pipe named `A`. The section cut off becomes a pipe named `B` having the same diameter as `A`. `L` must be less than the length of `A`.

`trim(A,L)` A section of length `L` is cut from the pipe named `A` and discarded. `L` must be less than the length of `A`.

`help` Print a help message with a brief summary of the commands.

`echo` Toggle command echo and prompting. (See below for details on this.)

Assume that all lengths are integers.

Use a predicate such as `setN` to establish a set of pipes to work with:

```
set1 :-
    retractall(pipe(_,_,_)),
    assert(pipe(a,10,1)),
    assert(pipe(b,5,1)),
    assert(pipe(c,20,2)).
```

The command interpreter is started with the predicate `run/0`.

A session with the interpreter is shown below. No blank lines have been inserted or deleted.

```
% swipl -l pipes
...

?- set1.
true.
```

```
?- run.

Command? help.
pipes -- show the current set of pipes
weld(P1,P2) -- weld P2 onto P1
cut(P1,P2Len,P2) -- cut P2Len off P1, forming P2
trim(P,Length) -- trim Length off of P
echo -- toggle command echo
help -- print this message
q -- quit

Command? pipes.
a, length: 10, diameter: 1
b, length: 5, diameter: 1
c, length: 20, diameter: 2

Command? weld(a,b).
b welded onto a

Command? pipes.
a, length: 15, diameter: 1
c, length: 20, diameter: 2


Command? trim(a,12).
12 trimmed from a

Command? pipes.
a, length: 3, diameter: 1
c, length: 20, diameter: 2

Command? cut(c,10,d).
10 cut from c to form d

Command? pipes.
a, length: 3, diameter: 1
c, length: 10, diameter: 2
d, length: 10, diameter: 2


Command? cut(c,6,c1).
6 cut from c to form c1

Command? pipes.
a, length: 3, diameter: 1
c, length: 4, diameter: 2
c1, length: 6, diameter: 2
d, length: 10, diameter: 2

Command? weld(d,c1).
c1 welded onto d

Command? pipes.
a, length: 3, diameter: 1
c, length: 4, diameter: 2
d, length: 16, diameter: 2
```

```
Command? q.
true.
```

Your implementation must handle four errors:

Cutting or welding a pipe that doesn't exist.

Cutting with a result pipe that does exist.

Cutting the full length (or more) of a pipe with cut or trim.

Welding pipes with differing diameters.

If an error is detected, the pipes are unchanged. Here is an example of error handling:

```
Command? pipes.
a, length: 10, diameter: 1
b, length: 5, diameter: 1
c, length: 20, diameter: 2

Command? cut(x,10,y).
x: No such pipe

Command? weld(x,y).
x: No such pipe

Command? weld(a,x).
x: No such pipe

Command? cut(a,5,b).
b: pipe already exists

Command? cut(a,10,a2).
Cut is too long!

Command? trim(a,15).
Cut is too long!

Command? weld(a,c).
Can't weld: differing diameters

Command? pipes.
a, length: 10, diameter: 1
b, length: 5, diameter: 1
c, length: 20, diameter: 2
```

Don't use `write('\nCommand? ')` to prompt the user. Instead, use the built-in `prompt/2` to set the prompt, like this: `prompt(_, '\nCommand? ')`. Then when `read(X)` is called, `'\nCommand? '` will automatically be printed first.

To make `tester` output more usable there is an `echo` command. By default, the `Command?` prompt is printed and the command entered is not echoed. The `echo` command toggles both behaviors: entering `echo` causes prompting to be turned off and echo to be turned on. A subsequent `echo` command reverts to the default behavior. In the following example, the text typed by the user is in bold and underlined:

```
?- run.

Command? cut(a,1,a2).
1 cut from a to form a2

Command? echo.
Echo turned on; prompt turned off
cut(a,1,a3).

Command: cut(a,1,a3)
1 cut from a to form a3
pipes.

Command: pipes
a, length: 8, diameter: 1
a2, length: 1, diameter: 1
a3, length: 1, diameter: 1
b, length: 5, diameter: 1
c, length: 20, diameter: 2
weld(a,a2).

Command: weld(a,a2)
a2 welded onto a
echo.

Command: echo
Echo turned off; prompt turned on
Command? q.
true.

?-
```

Hint: Manipulate an `echo/0` fact with `assert(echo)` and `retract(echo)`. To produce no prompt at all, use the built-in `prompt/2` like this: `prompt(_, '')`.

Important: to allow that manipulation of `echo/0` with `assert` and `retract` you'll need to declare `echo` as dynamic. Have the following line as the first line in your `pipes.pl`:

```
:-dynamic(echo/0).
```

TL;DR?

The built-in help provides a quick summary:

```
?- run.

Command? help.
pipes -- show the current set of pipes
weld(P1,P2) -- weld P2 onto P1
cut(P1,P2Len,P2) -- cut P2Len off P1, forming P2
trim(P,Length) -- trim Length off of P
echo -- toggle command echo
help -- print this message
q -- quit
```

And, handle these errrors:

Cutting or welding a pipe that doesn't exist.
Cutting with a result pipe that does exist.
Cutting the full length (or more) of a pipe with cut or trim.
Welding pipes with differing diameters.

I won't test with cases involving multiple errors, like a too-long cut that names an existing pipe as the result.

Problem 7. (20 points) `connect.pl`

In this problem you are to write a predicate `connect` that finds and displays a suitable sequence of cables to connect two pieces of equipment that are some distance apart. Each cable is specified by a three element list. Here is a list that represents a twelve-foot cable with a male connector on one end and a female connector on the other:

```
[m,12,f]
```

Let's consider an example of using `connect` to produce a sequence of cables. Imagine that to your left is a piece of equipment with a male connector. On your right, fifteen feet away, is a piece of equipment with a female connector. To connect the equipment you have two cables:

- A ten-footer with a male connector on one end and a female on the other.
- A seven-footer with a female connector on end and a male on the other.

The following query represents the situation described above.

```
?- connect([ [m,10,f], [f,7,m] ], m, 15, f).
```

`connect`'s first argument is a list with the two cables. The second, third, and fourth arguments respectively represent the gender of the connector of the equipment on the left (male—`m`), the distance between the equipment (15 feet), and the gender of the connector of the equipment on the right (female—`f`).

Here's the query and its result:

```
?- connect([ [m,10,f], [f,7,m] ], m, 15, f).
F----------MF-------M
true.
```

We see that a connection is possible in this case; a valid sequence of connections is shown. Observe that the first cable was reversed to make the connection. The number of dashes is the length of the cable. There's some slack in the connection—only fifteen feet needs to be spanned but the total length of the tables is seventeen feet. That's fine.

Note that the output has no representation of the pieces of equipment on the left and right that we're connecting with the cables.

Only male/female connections are valid in the world of `connect.pl`.

In some cases, a connection cannot be made, but `connect` always succeeds:

```
?- connect([[m,10,f], [f,7,m] ], m, 25, f).
Cannot connect
true.
```

```
?- connect([[m,10,f], [f,7,m] ], m, 15, m).
Cannot connect
true.
```

More examples:

```
?- connect([[m,1,m],[f,1,f],[m,10,m],[f,5,f],[m,3,f]], m, 20, f).
F-FM-MF-----FM----------MF---M
true.

?- connect([[m,1,m],[f,1,f],[m,10,m],[f,5,f],[m,3,f]], m, 20, m).
Cannot connect
true.

?- connect([[m,1,m],[f,1,f],[m,10,m],[f,5,f],[m,3,f]], m, 10, f).
F-FM-MF-----FM----------M
true.

?- connect([[m,10,f]], m, 1, f).
F----------M
true.
```

IMPORTANT: The ordering of cables your solution produces for a particular connection need NOT match that shown above. Any valid ordering is suitable. (A Ruby program, `a9/pc.rb`, analyzes the output.

Assume the arguments to `connect` are valid—you won't see two-element lists, non-numeric or non-positive lengths, ends other than `f` and `m`, etc. Assume that all lengths are integers. Assume that the distance to span is greater than zero.

You can approach this problem using an approach similar to that in the brick laying example in the slides. The built-in predicate `select/3` is like `getone/3` shown in the brick laying example. You don't have the complication of multiple rows but you will have to worry about mating the cables and trying both orientations of cables.

Note that you do not need to use all the cables or exactly span the distance.

My current solution is around 25 goals; about a third of those are related to producing the required output.

Problem 8. (10 points) `buy.pl`

In this problem the task is to print a bill of sale for a collection of items. Several predicates provide information about the items. The first is `item/2`, which associates an item name with a description:

```
item(toaster, 'Deluxe Toast-a-matic').
item(antfarm, 'Ant Farm').
item(dip, 'French Onion Dip').
item(twinkies, 'Twinkies').
item(lips, 'Chicken Lips').
item(hamster, 'Hamster').
item(rocket, 'Model rocket w/ payload bay').
item(scissors, 'StaySharp Scissors').
item(rshoes, 'Running Shoes').
item(tiger, 'Sumatran tiger').
item(catnip, '50-pound bag of catnip').
```

The second predicate is `price/2`, which associates an item name with a price in dollars:

```
price(toaster, 14.00).
price(antfarm, 7.95).
price(dip, 1.29).
price(twinkies, 0.75).
price(lips, 0.05).
price(hamster, 4.00).
price(rocket, 12.49).
price(scissors, 2.99).
price(rshoes, 59.99).
price(tiger, 749.95).
```

The third is `discount/2`, which associates a discount percentage with some, possibly none, of the items:

```
discount(antfarm, 20).
discount(lips, 40).
discount(rshoes, 10).
```

Finally, state law prohibits same-day purchase of some items. `dontmix/2` specifies prohibitions. Here are some examples:

```
dontmix(scissors,rshoes).
dontmix(hamster,rocket).
dontmix(tiger,catnip).
```

You can only ensure that any prohibited pairings are not included in a single purchase; the well-intentioned prohibitions can be thwarted by making multiple trips to the store!

You are to write a predicate `buy(+Items)` that prints a bill of sale for the specified items. If any mutually prohibited items are in the list, that should be noted and no bill printed.

```
?- buy([hamster,twinkies,hamster,toaster]).
Hamster.............................4.00
Twinkies............................0.75
Hamster.............................4.00
Deluxe Toast-a-matic...............14.00
----------------------------------------
Total                               $22.75
true.

?- buy([lips,lips,lips,dip]).
Chicken Lips........................0.03
Chicken Lips........................0.03
Chicken Lips........................0.03
French Onion Dip....................1.29
----------------------------------------
Total                                $1.38
true.

?- buy([scissors,dip,rshoes]).
State law prohibits same-day purchase of "StaySharp Scissors" and
"Running Shoes".
true.
```

You may assume that all items named in a `buy` are valid and that a price exists for every item. If several mutually prohibited items are named in the same `buy` only the first conflict is noted.

Here's the `format/2` specification I use to produce the per-item lines: `'~w~`.t~2f~40|~n'`. The backquote-period sequence causes the enclosing tab to fill with periods.

`a9/buyfacts.pl` has a collection of facts but I may tests with other sets of facts, too.

Problem 9. (8 points) `puzzle.pl`

UPDATE: See page 14 for this problem!

~~I'm considering a problem similar to the Zebra Puzzle shown on slides 216-214 but as of press time I don't have a good problem ready. By Tuesday, April 28, I'll either finalize a problem and publish an addendum with it or scratch this problem and evenly redistribute the eight points of this problem to the four previous problems.~~

Problem 10. 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 `a9` to **submit a single zip file named `a9.zip` that contains all your work**. If you submit more than one `a9.zip`, your final submission will be graded. Here's the full list of deliverables:

```
noconsec.pl
rotate.pl
outin.pl
btw.pl
fsort.pl
pipes.pl
connect.pl
buy.pl
```

`puzzle.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, which are typically written with one goal per line for the longer predicates.

```
$ wc noconsec.pl rotate.pl outin.pl btw.pl fsort.pl pipes.pl
connect.pl buy.pl
   3    8   76 noconsec.pl
   1    7   56 rotate.pl
   2    7   64 outin.pl
   2    4   82 btw.pl
  28   63  593 fsort.pl
  87  264 2607 pipes.pl
  36  116 1197 connect.pl
  24   68  768 buy.pl
 183  537 5443 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 1-224.

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 10 to 12 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 eight-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.)

Problem 9. (8 points) `puzzle.pl`

The following logic puzzle, "Rural Mishaps", was written by Margaret Shoop. It was published in *The Dell Book of Logic Problems #2*.

> "A butt by the family cow was one of the five different mishaps that befell Farmer Brown, his wife, his daughter, his teenage son, and his farmhand one summer morning. From the rhyme that follows, can you determine the mishap that happened to each of the five, and the order in which the events occurred?
>
> > "The garter snake was surprised in a patch
> > And bit a grown man's finger.
> > One person who weeded a flower bed
> > Received a nasty stinger.
> > The farmer's mishap happened first;
> > Son Johnny's happened third.
> > When Mr. Reston was kicked by the mule,
> > He said, "My word! My word!"
> > The sting of the bee was the fourth mishap
> > To befall our rural cast.
> > Neither it nor the wasp attacked Mrs. Brown
> > Whose mishap wasn't the last."

For `puzzle.pl` you are to encode as Prolog goals the pertinent information in the above rhyme and then write a predicate `mishaps/0` that uses those goals to solve the puzzle.

`a9/puzzle.out` shows a run of `mishaps/0` with the exact output you are to produce, but you might enjoy the challenge of solving the puzzle using Prolog **before** looking at the expected output.

The challenge of this problem is of course the encoding of the information as Prolog goals. **Solutions must both produce the correct output and accurately encode the information as stated in the rhyme to earn full credit.** The tester will check for correct output; I'll check manually for accurate encoding.