CSC 372, Spring 2014
Assignment 8
Due: Wednesday, May 7 at 23:00

Warm-up suggestions

Here are some built-in predicates I suggest writing just for practice: `length`, `member`, `append` (tough!), `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.

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 `a8`:

```
% ln -s /cs/www/classes/cs372/spring14/a8 .
```

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

Also, if you'd like to run `a8/tester` with just "`./t`", do this:

```
% ln -s /cs/www/classes/cs372/spring14/a8/tester t
```

Depending on whether `.` is in your `$PATH`, just "`t`" instead of "`./t`" might work. Another option is to put a `t` or `t8` alias in your `~/.bashrc`:

```
alias t8=/cs/www/classes/cs372/spring14/a8/tester
```

Use `a8/tester`

Use `a8/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 `a8/tester`. See the assignment 4 write-up for how-to details.

Problem 1. (2 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. (2 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. (2 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. (2 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. (17 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)` 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 put this problem second in the line-up in hopes of getting you thinking about it early but don't get hung up on it.

Problem 6. (12 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. The commands are in the form of Prolog terms. The command interpreter shown on slide 169 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? ')`, 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(_, '')`.

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.

Problem 7. (17 points) **`connect.pl`**

In this problem you are to write a predicate `connect` that finds a suitable sequence of cables to reach from one point to another. 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]
```

Here is an example of using `connect` to determine a configuration of cables to reach between a male connector and a female connector that are fifteen feet apart. A connection is possible in this case; a valid sequence of connections is shown. The number of dashes is the length of the cable.

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

Observe that the first cable was reversed to make the above the connection. 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.

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. (8 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(farm, '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').
```

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).
```

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 those prohibitions:

```
dontmix(scissors,rshoes).
dontmix(hamster,rocket).
dontmix(dip,twinkies).
```

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.

`a8/buyfacts.pl` has a collection of facts but we may tests with other sets, too.

Problem 9. (9 points) `range.pl`

Note: This problem and the following problem, `listsum`, won't be "in range" until we've covered the *Parsing and Grammars* section in the slides, which starts on slide 202.

Using Prolog's grammar rule notation write a parser `range(+S)` that recognizes Ruby ranges with integer literals. If the range is valid, the predicate prints a line in the form of a Prolog structure. If the range has no values (as defined by Ruby), a warning is also printed. If the range is syntactically invalid, "`Invalid Range`" is printed. `range` always succeeds. Integer literals with underscores, like `1_000_000` are not supported.

```
?- range('1..10').
range(1,10,inclusive)
true.

?- range('-100...10').
range(-100,10,exclusive)
true.

?- range('100...10').
range(100,10,exclusive)
Warning: Range is empty
true.
```

```
?- range('-10...-10').
range(-10,-10,exclusive)
Warning: Range is empty
true.

?- range('10..x').
Invalid range
true.

?- range('1..10..').
Invalid range
true.
```

Problem 10. (5 points) `listsum.pl`

Extend the list recognizer in the slides into a predicate `listsum(+String, -Sum)` that computes the sum of all the integer values found in the list specified by `String`. `listsum` fails if the list is syntactically invalid. Start with `a8/listsum0.pl`. The code in `listsum0.pl` does not handle negative numbers; neither does `listsum`.

```
?- listsum('[1,2,[3]]', Sum).
Sum = 6.

?- listsum('[1,20,[30,[[40]],6,7],[]]', Sum).
Sum = 104.

?- listsum('[1,,[30,[[40]],6,7],[]]', Sum).
false.

?- listsum('[]', Sum).
Sum = 0.
```

Problem 11. (10 points) `rules.txt`

In this problem you are to define a Prolog predicate possibly consisting of several clauses that implements each of the ten rules described below in English. As an example consider the rule "If something has webbed feet, waddles, flies, and quacks then the probability is .99 that it is a duck." That might be expressed in Prolog like this:

```
what(X,What,Prob) :-
      feet(X,webbed), movement(X,waddles), movement(X, flies),
      sound(X, quack), What=duck, Prob=0.99.
```

Another example: "A right triangle is a triangle with at least one angle of 90 degrees." In Prolog:

```
right_triangle(T) :-
      triangle(T), angles(T, Angles), member(90, Angles).
```

As shown here, your predicate may use other predicates that are not specified. In such a case those predicates should have a suggestive name as those above or you should include a comment briefly describing their operation.

Here are the rules you are to describe:

1. Anything stocked in the produce section of a grocery store or appearing in a ingredient list in a

cookbook is a food.

2 A person is a female if the person has a name that is typically given to females but not to males.

3. Never play poker with someone wearing a hat or whose nickname is "Doc".

4. Doubling a number is like multiplying it by two.

5. If the stoplight is green, maintain current speed. If the stoplight is red, stop. If the stoplight is yellow, step on the gas!

6. A polygon with four sides of equal length and with four equal angles is a square.

7. A bicycle is a cycle with two wheels. A tandem bicycle is a bicycle with two seats.

8. Ticket prices are $6.00 for shows before 6pm on weekdays. The first show on holidays and weekends is $6.00. The price for all other shows is $9.00.

9. If today is the fourth Thursday in November, then today is Thanksgiving.

10. Public members of a class `X` can be accessed by any method. Protected members of a class X can be accessed by methods of `X` and methods of subclasses of `X`. Private members of a class X can only be accessed by methods of `X`.

Note that because you don't need to further specify the predicates used in your rules, you have quite a bit of freedom to be creative on this problem. The essential task is to cast the rule as one or more rules in Prolog that are expressed with a set of reasonable predicates. If you're concerned whether you're on the right track, do two or three and mail them to us for some feedback. Important: The first few students to mail me and say they read this sentence in this write-up will get three points of extra credit on this assignment! Call it a close reading bonus. No posts on Piazza about this, please.

Submit your answers in a text file named `rules.txt`. **DO NOT submit a Word document, PDF, rich text file, etc.**—I want plain text.

Problem 12. 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: 10+
```

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 `a8` to **submit a single zip file named `a8.zip` that contains all your work. Do not submit individual files!** If you submit more than one `a8.zip`, we'll grade your final submission. Here's the full list of deliverables:

`nonconsec.pl`
`rotate.pl`
`outin.pl`
`btw.pl`
`fsort.pl`
`pipes.pl`
`connect.pl`
`buy.pl`
`range.pl`
`listsum.pl`
`rules.txt`
`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 in Prolog slides.

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 10-15 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 ten-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.)