CSc 372, Fall 2006
Assignment 9
Due: **Wednesday**, December 6 at 23:59:59 MST

Problem 1. (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] ;

No

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

No

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

No

?- rotate([],R).

No
```

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

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

Yes

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

No
```

Problem 2. (20 points) `fs.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 `fs(+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). `fs` stands for "flip sort".

`fs` 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 fscases.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]).

% pl
?- [fscases,fs].
?- case(_,L), fs(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) `fs` must always generate exactly one solution.

If you generate no more flips than my solution does for every case that we test with (not just the cases above) you'll earn five points of extra credit.

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've done a little Googling and haven't turned up a solution in Prolog but there may well be one out there. 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 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 solution have a total of seventeen goals, not counting a predicate to find the maximum value in a list. I don't use `is/2` at all. You may 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.

Problem 3. (20 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 137 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. (Hint: One way to produce this ordering is to use `findall` and `msort`.)
`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`.
`info`	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 `pipes/0`. Don't confuse that with the `pipes` command of the interpreter you're implementing—they are distinct!

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

```
?- set1.

Yes

?- pipes.

Command? info.
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) -- shorten P by Length
echo -- toggle command echo
info -- print this message

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

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

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
```

To make `tester` output more usable there is an `echo` command. By default, a prompt (`Command?`) 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:

```
?- pipes.

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
|: echo.
```

```
Command: echo
Echo turned off; prompt turned on

Command? cut(a,1,a4).
1 cut from a to form a4
Command?
```

Hint: Manipulate an `echo/0` fact with `assert(echo)` and `retract(echo)`. Here's a little of my code relating to echoing and prompting:

```
prompt :- \+echo,  write('Command? ').
prompt :- echo.
```

Problem 4. (20 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

Yes
```

Observe that the first cable was reversed to make the above the connection. Perhaps needless to say, only male/female connections are valid.

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

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

Yes

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

Yes
```

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

Yes

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

```
Yes

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

Yes

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

Yes
```

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 that begins on slide 148. The built-in predicate `select/3` is like `getone/3` shown on slide 149. 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.

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

Problem 5. (15 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).
```

We'll only ensure that any prohibited pairings are not included in a single purchase; the 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

Yes

?- 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

Yes

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

Yes
```

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.

`fall06/a9/buyfacts.pl` has a collection of facts but we may tests with other sets, too.

Problem 6. (7 points) `range.pl`

Using Prolog's grammar rule notation write a parser `range(+S)` that recognizes Ruby ranges. 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.

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

Yes

?- range('-100..10').
range(-100, 10, inclusive)

Yes

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

Yes

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

Yes

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

Yes

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

Yes
```

Problem 7. (8 points) `listsum.pl`

Extend the list recognizer on slide 177 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 `fall06/a9/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).
No
```

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

Problem 8. (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 $4.00 for shows before 6pm on weekdays. The first show on holidays and weekends is $4.00. The price for all other shows is $7.00.

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 `cs372-staff` for some feedback.

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 9. (7 points) `pw.pl`

In this problem you to write a predicate `pw(+Words, ?MVs, ?MCs, ?Same)` that partitions a list of words based on whether a word has more vowels than consonants, more consonants than vowels, or the same number of each. There is no upper limit on the number of words or letters per word. Assume all words consist exclusively of lower-case letters. `pw` always produces exactly one result.

There is a big restriction on this problem! Your solution may use only the following elements: `append`, `atom_chars`, `!` (cut), and list construction (`[]`, `[X1,_|]`, etc.) A few of the things YOU CANNOT USE are `is`, `length`, `=/2`, and the various comparison operators. As usual when restrictions are in force, we'll be happy to inspect solutions for compliance before the deadline.

Examples of usage:

```
?- pw([and,oops,are],MV,MC,Same).

MV = [are]
MC = [and]
Same = [oops]

?- pw([assume,all,letters,are,lowercase],MV,MC,Same).

MV = [are]
MC = [all, letters, lowercase]
Same = [assume]

?- pw([a],MV,MC,Same).

MV = [a]
MC = []
Same = []

?- pw([],MV,MC,Same).

MV = []
MC = []
Same = []

?- pw([aaaa,bbbb],MV,MC,Same).

MV = [aaaa]
MC = [bbbb]
Same = [] ;

No
```

DON'T FORGET ABOUT THE RESTRICTION ON THIS PROBLEM!!

Problem 10. (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: `rotate.pl`, `fs.pl`, `pipes.pl`, `connect.pl`, `buy.pl`, `range.pl`, `listsum.pl`, `rules.txt`, `pw.pl` and if so inclined, `extra.txt`. That list can be found on `lectura` in `/home/cs372/fall06/a9/delivs`.

Use the `turnin` tag `372_9` 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

`fall06/a9/tester` will appear as soon as possible. Use 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!

Be especially disciplined when working on `pw.pl`. Don't get obsessed and spend hours on it unless you can afford to. Instead, ask for some hints. Also, `fs.pl` can easily turn into a time sink. It took me about five minutes to write the essence of a solution for it but I then spent several hours trying to come up with a solution that didn't disgust me!

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.)