Solutions for
CSC 372 Final Exam
Wednesday, May 14, 2014

Problem 1: (6 points)

Mean and median: 5.1, 6

Cite three things about programming languages you learned by watching your classmates' video projects. Each of the three should be about a different language.

I tallied the languages cited in the responses and came up with these numbers:

Java	34
PHP	19
Lua	17
Python	15
Ruby	12
Lisp	8
C#	7
MATLAB	5
Haskell	5
C++	4
C	4
vimscript	3
R	3
Dart	3
JavaScript	2

The relatively high interest in Lisp reflected above reminds me that I should probably be whipped for not spending at least a little time on Lisp in this course. I did do a sidebar slide on Lisp in the Haskell set (#127) but ended up skipping it due to time. If you want to see a little bit of Lisp in the context of Emacs Lisp, see http://www.cs.arizona.edu/~whm/352/elisp.sli.pdf.

Problem 2: (6 points)

Mean and median: 5.1, 5.5

Write the following simple Prolog predicates. There will be a half-point deduction for each occurrence of a singleton variable or failing to take full advantage of unification.

<u>Restriction: No library predicates may be used other than `is/2`.</u>

`last(?List,?Elem)` *specifies the relationship that* `Elem` *is the last element of* `List`*, which is assumed to be non-empty.*

```
last([X],X).
```

```
        last([_|T], X) :- last(T,X).
```

`member(?Elem, ?List)` specifies the relationship that `Elem` is a member of `List`.

```
        member(X,[X|_]).
        member(X,[_|T]) :- member(X,T).
```

`length(?List, ?Len)` specifies the relationship that `List` is `Len` elements in length.

```
        length([],0).
        length([_|T],Len) :- length(T,N), Len is N+1.
```

Problem 3: (8 points)

Mean and median: 6.8, 7.5

Write a Prolog predicate `insrel(+List, -WithRels)` that instantiates `WithRels` to a copy of `List` with one of the atoms <, >, and = inserted between each pair of values to reflect the relationship between those values. Assume all values are numbers. Examples:

```
        ?- insrel([5,3,7,7,0],L).
        L = [5, >, 3, <, 7, =, 7, >, 0] .
        ...
```

Solution:

```
        insrel([],[]).
        insrel([X],[X]).
        insrel([X,Y|T], [X,Sym|R]) :- which(X,Y,Sym), insrel([Y|T], R).

        which(X,Y,'<') :- X < Y.
        which(X,X,'=').
        which(X,Y,'>') :- X > Y.
```

Problem 4: (7 points)

Mean and median: 5.7, 7

Write a Prolog predicate `alltails(+List, -T)` that first instantiates `T` to the tail of `List`. If an alternative is requested, it generates the tail of the tail of `List`, and so forth, thus generating "all the tails".

```
        alltails([_|T],T).
        alltails([_|T],R) :- alltails(T,R).
```

Problem 5: (10 points)

Mean and median: 8.2, 9

Write a Prolog predicate `pinch(+List, -Pair)` that instantiates `Pair` to a series of `pair/2` structures. The first `pair` structure has the first and last values of `List`. The second `pair` has the second and next-to-last values of `List`. And so forth. Example:

```
        ?- pinch([a,b,c,d,e,f],P).
        P = pair(a, f) ;
        P = pair(b, e) ;
        P = pair(c, d) ;
        false.
```

Solution:

```
head([H|_],H).
pinch(L, pair(First,Last)) :- head(L,First), last(L,Last).
pinch([_|T], pair(F,L)) :- append(Rest,[_],T), pinch(Rest,pair(F,L)).
```

Problem 6: (5 points)

Mean and median: 4.2, 5

Write a Prolog predicate `sumvals/0` *that consolidates all* `v/1` *facts into a single fact that contains the sum of the terms of those facts. Assume all terms are numbers. There may be any number of* `v/1` *facts.*

```
sumvals :- findall(Val, v(Val), Vals), sumlist(Vals,Sum), retractall(v(_)),
      assert(v(Sum)).
```

The first version of my solution had the following no-facts case but just as I was about to deduct points for a solution not having it, I realized it wasn't needed!

```
sumvals :- \+v(_), assert(v(0)), !.   % Not needed!
```

Problem 7: (12 points)

Mean and median: 9.2, 11.5

`path(+Start, +End, +Moves, -Path)` *instantiates* `Path` *to a series of values from* `Moves` *that describe a path from* `Start` *to* `End`*, two points on a Cartesian plane. All alternatives are produced if requested. Each move can be used only once in a given path. Example:*

```
?- path(p(0,0), p(3,4), [m(4,4),m(3,3),m(0,1),m(-1,0)], Moves).
Moves = [m(4, 4), m(-1, 0)] ;
Moves = [m(3, 3), m(0, 1)] ;
Moves = [m(0, 1), m(3, 3)] ;
Moves = [m(-1, 0), m(4, 4)] ;
false.
```

Solution:

```
path(p(X,Y),p(X,Y),_,[]).

path(p(X,Y), p(DX,DY), Moves, [m(MX,MY)|MoreMoves]) :-
    select(m(MX,MY), Moves, Remaining),
    NX is X + MX, NY is Y + MY,
    path(p(NX,NY), p(DX,DY), Remaining, MoreMoves).
```

Problem 8: (10 points)

Mean and median: 6.4, 7.5

Using the parsing method supported by Prolog's grammar rule notation *write a predicate* `ptime(+Spec,-Mins)` *that parses atoms that represent time durations, like* `'10m'`*,* `'5h'` *and* `'10:20'`*, and instantiates* `Mins` *to the number of minutes represented by* `Spec`*.*

```
ptime(Spec,Mins) :- atom_chars(Spec,Chars), spec(Mins,Chars,[]).

spec(Mins) --> int(Hours), ['h'], { Mins is Hours * 60 }.
spec(Mins) --> int(Mins), ['m'].
spec(Mins) --> int(H), [':'], int(M), {Mins is H * 60 + M}.
```

Problem 9: (7 points)

Mean and median: 5.7, 6

Write a ***Haskell*** *function* `ckconn` *that takes a list similar to that used by a8's* `connect.pl` *and returns* `True` *or* `False`*, depending on whether the exact sequence and orientation of cables represents a valid connection.*

Example:

```
> ckconn 'm' [('f',10,'m'), ('f',7,'m')] 'f'
True
```

For the type, it was fine to have something like this:

```
ckconn :: Char -> [(Char,Int,Char)] -> Char -> Bool
```

but the type that Haskell infers is this:

```
ckconn :: Eq a => a -> [(a, t, a)] -> a -> Bool
```

Here's the executable code:

```
ckconn _ [] _ = False

ckconn from [(left,_,right)] to =
      from /= left && to /= right

ckconn from ((left,_,right):t) to =
      from /= left && ckconn right t to
```

Problem 10: (7 points)

Mean and median: 5.3, 6

Write a ***Haskell*** *function* `connlen` *that takes a list of the same type as* `ckconn` *and* ***prints*** *either the length of the configuration, like "*`25 feet`*" or "*`nope`*", if the configuration is not valid.*

```
> connlen 'm' [('f',10,'m'), ('f',7,'m')] 'f'
17 feet

> connlen 'm' [('f',10,'m'), ('f',7,'m')] 'm'
nope
```

Solution:

```
connlen :: Char -> [(Char,Int,Char)] -> Char ->  IO ()
connlen from cables to = putStr result
    where
      result =
        if ckconn from cables to then
            (show $ sum $ map ( \(_,len,_) -> len) cables)
                  ++ " feet\n"
        else
            "nope\n"
```

Problem 11: (14 points)

Mean and median: 11.9, 14

Write a ***Ruby program*** `shuffile.rb` *that reads lines from standard input, shuffles them, and then writes them to standard output.*

`shuffile` *requires one command line argument,* `-N`*, which specifies that lines are to be handled in blocks of N lines. Assume that standard input consists of a multiple of N lines.*

Here's a "plug and chug" solution:

```
if ARGV.size != 1 || (not ARGV[0] =~ /^-\d$/)
    puts "Oops!"
    exit 1
end

n = -ARGV[0].to_i

blocks = []
while true do
    block = []
    1.upto(n) {
        line = STDIN.gets
        if !line then
            blocks.shuffle.each { |block|
                block.each {|line| puts line }
                }
            exit
        end
        block << line
        }

    blocks << block
end
```

Those nested `each` blocks for output can be replaced with just this:

```
puts blocks.shuffle
```

Note that I "punt" on getting out of the `1.upto(n)` block by printing and exiting in the middle of the block when end of file is reached.

If you lean on the library a little more you can come up with this:

```
if ARGV.size != 1 || (not ARGV[0] =~ /^-\d$/)
    puts "Oops!"
    exit 1
end

n = -ARGV[0].to_i

blocks = []

lines = STDIN.readlines.each_slice(n) { |slice| blocks << slice }

puts blocks.shuffle
```

My son pointed out that PHP has `array_chunk`. That suggests a one-liner like this,

```
puts STDIN.readlines.chunks(n).shuffle
```

but I can't find anything like `chunks` in Ruby, which PHP programmers know as `array_chunk`. However, we can correct that omission with this:

```
class Array
    def chunks(n)
        result = []
        self.each_slice(n) { |slice| result << slice }
        result
    end
end
```

I first encountered "shuffile" years ago as an Icon program. At the time I assumed that Ralph Griswold came up with that clever name but maybe it was somebody else. There's a `shuf` Linux command, too. It has the useful behavior of being able to shuffle command-line arguments as well but its `-i` and `-n` options make me wonder if the author knew of `seq` and `head`.

Problem 12: (8 points, one point each)
Mean and median: 5.7, 6

(a) *Who founded UA's CS department, and in what year?*

Ralph Griswold, in 1971. I hope the CS majors will always remember who got this program going.

Ralph once said that on his first day here, he arrived to find a number of students waiting in line outside his office, for advising. Once in his office, he found that he had a desk but no chair.

Ralph said he took the position here with the specific condition that he would not have to serve as department head but when no other suitable candidate was found, he took on that role.

A random memory about Ralph: A tradition started by his graduate students was showing up at his house on Halloween night, and inviting themselves in for a chat. Those Halloween visits by students and former students continued until his death, in 2006.

(b) *Why are Prolog warnings about singleton variables worth paying attention to?*

A singleton warning often indicates a misspelled variable name. In other cases it might indicate that a value is being computed but never used, so why compute it.

(c) *What's the fundamental difference between predicates like `member/2` and `append/3` in Prolog and their superficial analogs in other languages?*

Prolog's `member` predicate can be used to both test for membership in a list, generate all elements in a list and more.

(d) *Draw the box for Prolog's four-port model and label the ports.*

I'm lazy! See slide 55.

(e) *"cons" lists are found in many languages that make use of recursion. What is it about cons lists that makes them well-suited for recursive functions?*

Many problems have a natural recursive formulation of operating on the first element of the list and combining that with the result of the function on the rest of the list.

(f) *Consider this claim: Prolog's grammar rule notation, like `sentence --> article, noun, verb`, is an example of syntactic sugar. Present an argument that either supports that claim or refutes it.*

Grammar rules are a great example of syntactic sugar. Any grammar rule can be brainlessly rewritten by adding

"In" and "Out" arguments to each non-terminal, replacing terminals with list unifications, and removing curly braces around ordinary goals.

(g) *Write a simple* ***Haskell function*** *and show an example of using a partial application of that function.*

```
> let add x y = x + y
> map (add 5) [1,2,3]
[6,7,8]
```

(h) *What's a very good reason that* `ckconn` *above uses tuples instead of lists to represent the cables?*

Those Prolog lists are heterogeneous.

Extra Credit Section (½ point each unless otherwise noted)

Mean and median: 2.3, 2.5

(a) *What movie inspired the "Original Thought" bonus?*

Broadcast News

(b) *whm came to graduate school here at UA specifically to join a research team developing a programming language. What was the language?*

Icon

(c) *To what current CS faculty member does whm attribute the following quotation?*
"When you come to a problem you can lean forward and type, or you can sit back and think."

Dr. Proebsting

(d) *The Prolog 1000 is a compilation of applications written in Prolog and related languages. What's a little odd about it?*

There are less than 1000 applications on the list.

(e) *Jean Ichbiah, Ada's designer, was said to have once predicted that in ten years only two programming languages would remain in use. Ada was one of the languages. What was the other?*

Lisp

(f) *Cite a significant contribution to computer science made by Grady Booch.*

He was one of the "Three Amigos"—along with James Rumbaugh and Ivar Jacobson—who created UML.

(g) *What's a language that has/had an "arithmetic if", one form of which looks like this:*

```
     IF (I-J) 100, 110, 120
```

FORTRAN. Control branches to the statement labeled `100`, `110`, or `120` respectively, depending on whether the result of `I-J` is negative, zero, or positive.

(h) *In terms of creators, what do* `vim` *and Java have in common?*

Bill Joy, later a founder of Sun Microsystems, wrote `vi`, which of course inspired `vim`. Joy was also involved with Java early on.

Several students found success with something like "They were both created by programmers."

(i) *Market research has determined that the course title Comparative Programming Languages is dry and unappealing. Think up a new name for 372. It needn't be less dry but should be funny!*

The responses are below, in random order. It seems like not everybody got the "It needn't be less dry but should be funny!" part, but I suppose good humor is where you find it. I also posted the list on the board outside the 733 corridor. If you're in the building, go by and vote for your favorite.

Learn 3+ Languages in Too Short of Time
Brain Buster: The Class
So Much Recursion
Get Confused by Small Differences 101
372: Fiesta of Programming
Resume Booster
Why Ruby is So Much Better Than Haskell [Ed. note: That wasn't my intention!]
The Many Languages of Pancakes
Dr. Strangelanguage or: How I Learned to Stop Worrying and and Love Prolog
Super Comparative Programming Languages
Actually Learn Programming
372: The Wrath of Cons
More, Harder, and Funner Homework Than You Could Hope For
Study of Mind-Warping
Trying to Keep 3 Languages Separate in Your Head
Multiple Languages Just in One Semester
Exploration into the World of Machine Languages
Heads and Tails, Let's Rock!
Three Fun Ways to Scramble Your Brains
Multiple Language Ingestion
How to Talk To Computers
Introduction to Different Styles of Thinking About Programming General Purpose Computing Devices
An Introduction to Remaining Competitive In Your Field, Good Luck.
Bunch of Random Languages
No Language to Beat 'em All
Learn to Hate Pancakes in 3 Different Languages
372: Too Much Information You Won't Retain
Learn All the Languages!!! or Just 3 More
Completely Pleasant Languages
Hardcore 3 Programming Language Learning in 3 Months
Learn You a Haskell /[Haskell|Ruby|Prolog]/ for Great Good
Foreign Languages for Programmers
Superfast Computer Language Learning Time
A Class About Not Java
whm and the 3 bears
Haskell, Ruby, Prolog! (so they know the languages we're learning)
Any Way You Want It: 3 Languages for the Programmer on the Go
A Programmer's Nightmare
1001 Ways to Write Fast
Wow So Many Languages in One Course
Too Many Languages!
Not Java—the Class!
French 102 and Other Languages (so us B.S. students wouldn't have to take Foreign Language in place of a CS elective!)
Pancake Hell
Programming in Latin
How Many Examples Can We Fit Into One Slideshow
Programming Languages: The Good, The Bad, and Haskell
Blow Your Mind With Unorthodox Programming Languages

Functional, Objective, Logical—A Plethora of Paradigms
Programming Languages, A Love Story
Learning How to Stroke Egos: New With Haskell!
Programming Paradigms Explored (not funny, but EC!)

(j) *whm would like to recycle a8's* `buy.pl` *in a future semester but needs more* `dontmix` *facts with an element of humor. What's another pair he could use?*

My favorites of these were `dontmix('Godzilla','NYC')` and the classic but elegant `dontmix(oil,water)`.

(k) *How many students earned the "close reading" bonus on assignment 8?*

Eleven.

Answers like "A few." were counted as correct, too.

Statistics

All scores:

```
100, 100, 99.5, 98.25, 98, 97.5, 97, 97, 96.75, 96.5, 96, 95, 94.5, 94.5, 93.5,
92.75, 91.75, 91, 91, 90.5, 90.5, 89.75, 89, 88, 87.5, 87, 86.75, 86.5, 84,
82.5, 82.5, 82.5, 81, 80.5, 80, 80, 77.5, 76.5, 76.25, 76, 75.5, 74.25, 72.5,
70, 65, 62, 61, 60, 59, 59, 58.25, 57.5, 54, 44.5, 41
```

```
N = 55
mean = 81.6136
median = 86.5
```