# CSC 372 Final Exam Solutions
# Tuesday, March 12, 2015

## Quiz 18, May 12, 2015
## 2 questions; 4 points

1. *For one point each, cite three things you learned from Dr. Proebsting's presentation in 372 on Tuesday, May 5, the last day of class.*

   Here are some of my jots...

   Story of Richard Hamming's three questions for new hires at Bell Labs.

   "I think programming languages are the key to productivity."

   "Perl is the only popular language I hold in lower esteem than C."

   Recommended book *The Innovator's Dilemma*.

   Academics are batting zero for popular languages. [But I'd say Haskell is a still-rising counter-example.]

   Thinks Visual Basic was most influential language of the 1990s.

   Statistics gathered by Microsoft's Watson tool (not to be confused with IBM's Watson!) show that about 50% of crashes are caused by 1% of the bugs.

   His mention of "flight data recorders" for programs reminded me that Adobe had something like that in the works for its ActionScript/Flash Player stack. I've got a distinct memory of seeing a demo of it but a quick Google just now didn't turn up anything that looked familiar.

   He said that Go is the fastest growing language right now. It was also interesting to hear about a single code formatting standard being enforced by the Go formatter. There are benefits to freedom from choice, and I sure like a one-for-all-formatter better than Python's enforcement of indentation rules.

   And last but not least there was that incredible story about the birth of Erlang.

2. *On a scale of 1 to 5, with 1 being least important and 5 being most important, how would Dr. Proebsting rate the importance of language choice for a programming project?*

   I believe he'd give it a 5, but I counted a 4 or 5 as correct.

---

**Problem 1: (6 points) <u>mean = 5.12, median = 6.00</u>**

*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 the numbers below. I believe Mr. Nelson's video is destined to become a cult classic, so I gave it its own category, and didn't count it toward Ruby. I especially loved how he fit ten minutes of video into seven minutes by simply speeding it up, taking a lesson from Alvin and the Chipmunks.

| | |
|---|---|
| Swift | 27 |
| Python | 26 |
| Ruby | 17 |
| Go | 14 |
| The Hardest Unsolved Problem in Computer Science | 12 |
| Java | 10 |
| Haskell | 8 |
| PHP | 8 |
| Lisp | 6 |
| C++ | 6 |
| C | 5 |
| JavaScript | 4 |
| Erlang | 4 |
| Icon | 3 |
| Lisp | 3 |
| Perl | 3 |
| io | 2 |
| MATLAB | 2 |
| C# | 2 |
| Mathematica | 2 |
| F# | 1 |

Lots of people commented about the ability to have emojis in Swift but just like Swift, Java allows programs to be written in Unicode. You can thus use emojis in Java, too; the question is whether your IDE or editor displays the full Unicode character set properly.

**Problem 2: (3 points) mean = 1.09, median = 1.00**

(1) *(Two points) Imagine that a Haskell function* `f` *has the type* `Char -> Bool -> Char`*. Explain the meaning of that type in a way that demonstrates you have significant knowledge of Haskell.*

*Hint: here's an answer that's worth zero points: "*`f` *takes two arguments, a* `Char` *and a* `Bool`*, and returns a* `Char`*."*

Calling `f` with a `Char` produces a function that can called with a `Bool` to produce a final result, of type `Char`. `f` is most likely defined in curried form.

(2) *(One point) Fully parenthesize the following Haskell expression.*

```
((((f  1) 2) g) + (x x)) + (((f g) f) 1)
```

**Problem 3: (6 points) mean = 2.48, median = 2.00**

*Write a Haskell function* `longpos :: [String] -> Int` *that returns the one-based position of the longest string in a list of strings. Assume the list has at least one string. Don't worry about ties.*

```
longpos strings = snd $ head $ reverse $ sort $ zip (map length strings) [1..]
```

**Problem 4: (9 points) mean = 6.82, median = 8.00**

*Write a Ruby program that reads integers, fractions, and mixed numbers, one per line from standard input, and prints their sum when end of file is reached.*

```
sum = 0
while (print "Number? "; line = gets)
    line.chomp!

    if line =~ /^\d+$/
        sum += $&.to_f
    elsif line =~ /^((\d+) )?(\d+)\/(\d+)$/ then
        sum += $2.to_i + $3.to_f/$4.to_f
    else
        puts "Ignored: #{line}"
    end
end
puts "\nTotal: #{sum}"
```

**Problem 5: (9 points) mean = 7.24, median = 8.50**

*In this problem you are to write a Ruby version of* `buy.pl` *from assignment 9. The Ruby version reads a file named* `buy.data` *for information about items and discounts.*

```
items = {}

f = File.open("buy.data")

while line = f.gets
    type,item,f3,f4 = line.chomp.split("|")
    if type == "item"
        items[item] = [f3, f4.to_f]
    else
        items[item][1] *= 1 - f3.to_f*0.01
    end
end
f.close
```

```
total = 0
for item in ARGV
    if !items[item]
        puts "Price check for #{item}!\n"
        exit(1)
    else
        cost = items[item][1]
        printf("%s...%.2f\n", items[item][0], cost)
        total += cost
    end
end

printf("Total: $%.2f\n", total)
```

I was appalled by the number of students who used this structure, which turns an *O(n)* computation into an $O(n^2)$ computation:

```
for item in ARGV
    read buy.data, looking for item
```

**Problem 6: (6 points) mean = 4.72, median = 6.00**

*Implement assignment 9's* `outin` *as a Ruby iterator. Assume its argument is always a non-empty array.*

```
module Outin # as a mixin...
    def outin
        a = self.dup
        while a.size > 0
            yield a[0]
            a = a[1..-1].reverse
        end
        nil
    end
end
```

Some students imagined that an iterator can be be called recursively but that doesn't work. However, I let those pass.

**Problem 7: (5 points) mean = 2.87, median = 3.00**

*Ruby defines a meaning for* `String * Fixnum` *but the operation is not commutative—*`Fixnum * String` *produces an error:*

```
>> 4 * "abc"
TypeError: String can't be coerced into Fixnum
```

For this problem you are to create a file named `symmul.rb` such that after `symmul.rb` is loaded, `Fixnum * String` works.

```
class Fixnum
    def *(rhs)
        if rhs.class == String
            rhs * self
        else
            self * rhs
        end
    end
end
```

**Problem 8: (10 points) mean = 8.16, median = 9.00**

*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.*

***Restriction: No library predicates may be used other than `is/2`.*** *(But note exception for `init`!)*

*(a)* `f(?X)` *expresses the relationship that* `X` *is 3 or 4.*

```
f(3).
f(4).
```

(b) `head2(?L, ?Heads)` *expresses the relationship that* `Heads` *is a list that consists of two copies of the head of* `L`.

```
head2([H|_],[H,H]).
```

(c) *Write the library predicate* `member/2`.

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

*(d)* `sum(+L, -Sum)` *instantiates* `Sum` *to be the sum of the elements in* `L`*, which are assumed to all be numbers.*

```
sum([],0).
sum([H|T],Sum) :- sum(T,Sum0), Sum is H + Sum0.
```

*(e)* `init(?L, ?Init)` *is an analog to Haskell's* `init`*: the list* `Init` *is all but the last element of* `L`. `init` *fails if* `L` *is empty.* ***Restriction EXCEPTION: Your solution for `init` may use `append`.***

```
init(L,Init) :- append(Init,[_],L).
```

**Problem 9: (4 points) mean = 2.91, median = 3.00**

*Write a Prolog predicate* `zip(+L1, +L2, ?Pair)` *that produces a series of* `pair/2` *structures:*

```
?- zip([1,2,3,4],[a,b,c],R).
R = pair(1, a) ;
R = pair(2, b) ;
R = pair(3, c) ;
false.
```

```
zip([H1|_],[H2|_], pair(H1,H2)).
zip([_|T1],[_|T2], R) :- zip(T1,T2,R).
```

**Problem 10: (6 points) mean = 4.15, median = 5.00**

*Write a Prolog predicate* `swapends(?L1,?L2)` *that expresses the relationship that* `L1` *is a copy of* `L2` *with the first and last elements swapped.* `swapends` *is only meaningful for lists with two or more elements.*

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

?- swapends([1,2,3],L).
L = [3, 2, 1] ;
false.

?- numlist(1,7,Nums), swapends(Nums,R).
Nums = [1, 2, 3, 4, 5, 6, 7],
R = [7, 2, 3, 4, 5, 6, 1] ;
false.
```

```
swapends([First|T],[Last|Rest]) :-
      append(Middle,[Last],T), append(Middle,[First],Rest).
```

I was very impressed with the elegance and symmetry of answers by Mr. Britton,

```
swapends([H|T],[R|F]) :- reverse(T,[R|RS]), reverse([H|RS],F).
```

and Mr. S. Stephens:

```
swapends([X|T],[Y|S]) :- reverse(T,[Y|R]), reverse(S,[X|R]).
```

In case you're wondering, yes, I had to type those in to convince myself they worked.

Mr. Wolfe had an elegant recursive solution: (minus an overlooked unification)

```
swaphelp([X],[Y],Y,X).
swaphelp([X|XS],[Y|YS],In,Out) :- X = Y, swaphelp(XS,YS,In,Out).
swapends([X|XS],[Y|YS]) :- swaphelp(XS,YS,X,Y).
```

**Problem 11: (6 points) mean = 2.98, median = 4.00**

*On assignment 9 you wrote* `outin(+L, -R)`*, which worked like this:*

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

*On this problem you are write* `inout(+L, -R)`*, which generates elements in the opposite order from* `outin`*.*

***IMPORTANT: You may use `outin` in your solution!*** *This problem is intended to be easy; don't make it hard!*

```
inout(L,R) :-
    findall(Val, outin(L,Val), Vals), reverse(Vals,RVals), member(R,RVals).
```

A common short but wrong answer was to simply reverse the list and call `outin` on the reversed list.

**Problem 12: (12 points)** **mean = 6.06, median = 7.00**

*In this problem you are to write a Prolog predicate* `cross/3` *that finds a way to cross over a series of pits using wooden planks as bridges. ...*

```
cross(Planks,Distance,Solution) :-
      layplanks(Planks, 0, Distance, Solution).

layplanks(_,Current,Distance,[]) :- Current >= Distance.

layplanks(Planks, Current, Distance, [Plank|MorePlanks]) :-
    select(Plank, Planks, LeftOver),
    NewEnd is Current + Plank,
    \+overpit(NewEnd),
    layplanks(LeftOver,NewEnd,Distance,MorePlanks).

overpit(N) :- pit(Start,Width), End is Start + Width, N >= Start, N =< End.
```

A common mistake was to have code that looked like this,

```
..., pit(Start,Width), End is Start, NewEnd < Start, NewEnd >= End, ...
```

where I've got `\+overpit(NewEnd)` above. With code like that you're asking, "Is there is **any** pit that the joint is not over?"

**Problem 13: (7 points)** (one point each unless otherwise indicated) **mean = 5.09, median = 5.00**

***The following questions and problems are related to Prolog.***

*(1) Haskell strings are actually just lists of characters. Similarly, Prolog lists are actually instances of a more basic element of Prolog. What's that element?*

Structures, but lots of people said "atoms".

*(2) What's an appropriate situation in which to use the "cut-fail" combination?*

When a predicate detects a condition that guarantees the predicate should fail. Or, "My final answer is no!"

*(3) Consider the following goal written by a novice Prolog programmer:* `append(A,B,A)`*. What's a likely problem with that goal?*

`append(A,B,A)` can only be true if `B` is an empty list. Perhaps the programmer thinks that `append` works like `A = A + B`.

*(4) Consider this claim: Aside from what* `is/2` *provides, Prolog has nothing that corresponds to the concept of an expression like that found in Java, Haskell, and Ruby. Present an argument either in favor or against this claim. (2 points)*

The hallmark of an expression is that it produces a value but predicates can only succeed or fail. Any values produced by a predicate must be via instantiation.

In retrospect I decided this wasn't a very good question. Any non-trivial answer for it was counted as correct.

*(5) What's the most important fundamental difference between a Prolog predicate like* `member/2` *and a function/method of the same name in Haskell, Ruby, or Java? (2 points)*

`member` can both test for membership and generate values.

**Problem 14: (5 points)** (one point each unless otherwise indicated) **mean = 5.95, median = 6.00**

*The following is a Sather program, from* `http://rosettacode.org/wiki/Sum_of_squares#Sather`

```
class MAIN is

  sqsum(s, e:FLT):FLT is
    return s + e*e;
  end;

  sum_of_squares(v :ARRAY{FLT}):FLT is
    return (#ARRAY{FLT}(|0.0|).append(v)).reduce(bind(sqsum(_,_)));
  end;

  main is
    v :ARRAY{FLT} := |3.0, 1.0, 4.0, 1.0, 5.0, 9.0|;
    #OUT + sum_of_squares(v) + "\n";
  end;

end;
```

*Based on the code above, tell me five things about Sather. Excellent or additional observations may earn up to three points of extra credit.*

Here's what I see:

The big one: Sather appears to be statically typed.

Types follow the entity whose type is being declared.

`|val1, val2, ..., valN|` appears to be array initializer syntax but there's also `#ARRAY{FLT}(|0.0|)`; the distinction between the two is not clear. `#ARRAY` is apparently the same kind of thing that `#OUT` is.

`reduce(bind(sqsum(_,_)))` implies that reduce is a higher-order function, but `bind` is apparently needed to create an intermediate value that refers to `sqsum`.

Last value in method does not implicitly become return value; explicit `return` statement is used instead.

One form of initialization is `:=`.

`#OUT` is a special value that produces output when a value is concatenated with it.

String literals are enclosed in double quotes and have backslash escapes.

Semicolons are required to terminate statements.

`class` may imply that Sather is object-oriented. I believe that only Mr. Bernth mentioned that `.append` implies object-orientation, too.

A number of students said that the free-standing `s` in `sqsum(s, e:FLT)` implied that types need not be specified, but in fact the type simply follows the identifier(s). (Note that in the Java declaration `int x, y;` the type `int` applies to both `x` and `y`.)

A handful of students said that `:` was cons, but I see no grounds for that whatsoever.

**Problem 15: (6 points) mean = 5.46, median = 6.00**

Answer the following general questions.

*(1) What is the fundamental characteristic of a statically typed language? (1 point)*

The type of every expression can be determined at compile time.

*(2) A language designer must decide whether an operation is to have a symbolic form, like `+`, `==`, and `x[y]` or an alphabetic form like `length`, `insert`, and `charAt`. What are some factors that must be considered when deciding between an symbolic or alphabetic form for an operation? (2 points)*

Symbolic forms allow more concise expression and that's good for common operations like concatenation and indexing.

Sometimes using a symbol avoids a naming problem. For example, Icon's unary `*` operator, which returns the length of a string or a list, the number of key/value pairs in a table, the numbers of characters in a character set, and more. Instead of deciding between `length`, `size`, `count`, `cardinality` (true story), etc., `*x` provides a helpful indefiniteness.

Designers must also consider whether there's a common underlying idea that makes a symbolic form appropriate, like using `+` for both addition and concatenation, although the latter is not commutative.

*(3) Imagine that for some reason all your knowledge and memories of two of the three languages we covered this semester must be erased! Which of the three languages do you wish to retain your memories of, and why? (3 points)*

I tallied the answers. Here are the numbers:

| | |
|---|---|
| Haskell: | 16 |
| Ruby: | 24 |
| Prolog: | 28 |

The rationales for retention were myriad. Among them were which language was most interesting, which language would likely be most useful and/or marketable, which language had the most concepts useful in other settings, which language was least likely to be independently learned, and which language "I don't ever want to learn again."

**Extra Credit Section (½ point each unless otherwise noted) mean = 2.85, median = 3.00**

*(1) What programming language had a typo in the first example of the first book published about the language?*

The first example in the *The Java Programming Language* by Ken Arnold and James Gosling had "`Sytem.out.println(...)`" in the first example.

*(2) The first machine named lectura was a VAX-11/785. Who picked the name "lectura"?*

The first round of CS host names were font names. Dr. Peter Downey found "Lectura", and it seemed perfect for an instructional machine. Some others were Bocklin, Megaron, Caslon, and Bembo.

*(3) What would be a better name for the Prolog 1000, and why?*

Maybe the Prolog 500—the list appears to have about 502 entries.

*(4) What does DWIM mean?*

"Do what I mean." Runner-up, by P. Martin: "Drunken whales injure millions."

*(5) What is the official name of Gould Simpon 942? (Ok to be funny!)*

"Programming Languages Lab" is on the plaque beside the door, and Mr. Hickey knew that, but here are other titles:
The Lion's Den
The Den of Crushed Freshmen
Ephiphany H.Q.
Home [two entries!—whm]
The Thunderdome
A long staircase away.
the Lab
CS Dungeon
The "Please Help me" Lab
"When you're desperate"
The Help me room
Late-night Debugging Center
Big Poppa's House
The Hurt Locker
Last minute disaster
Panic Bunker
The Romper Room
Where the Problems Are Solved
This is where you go to finish your program.
Buckle your Seat Belts Room
Help Center for Sick 372 Students
The Meat Grinder
A room where people inside cry 9 times a semester.
Help Center
think tank
The Simpsons
whm's coder clubhouse
The Cool Kids Club
Broken Elevators
Windowless Dungeon of Office Hours [submitted by a fellow correctional officer]
Office hours central
my second home
Mitchell's Lab (?) [that does have a nice ring!]
The Last Minute Help Room
"The assignment is due tonight?!"
The computer lab that eats all of your time away.
The "Help Me" group
The penthouse
my apartment
Gould Simpson 942 (spelling counts) [ouch!]

Names for the building, apparently:
My girlfriend calls it the "Nerd Palace".
Palace of Science

*(6) Ralph Griswold is known for designing languages like SNOBOL4 and Icon but he's also known for his work related to the mathematical aspects of weaving. What's a <u>one word</u> connection between those two areas?*

My word is "strings", and Mr. Nelson came up with that, too.

J. Naranjo and T. Romero said "threading".

E. Harris, C. Jensen, C. Johnson, R. Macdonald, B. Streeter, and C. Troy all came up with "patterns", and that's surely true!

K. Enami pointed out that both SNOBOL4 and "weaving" have an "n" in them.

K. Hudspeth was creative with "weavematical". (Zero Google hits just now.)

*(7) In what year did Ralph Griswold found The U of A's Department of Computer Science?*

1971

*(8) On Ralph's first day here he arrived to find a number of students waiting outside his office for advising. When he entered his office he found a common piece of office furniture was absent. What was it?*

Ralph had a desk, but no chair! However, any answer was counted as correct. Twenty-two said "desk", twenty-six said "chair(s)", and one said "coffee maker".

Here's one last story about Ralph. As dial-up modems started becoming popular the phone company started worrying about modem users monopolizing circuits. There was a Tucson TV news story about this in the early 70s in which a phone company representative talked about the problems that heavy modem users could create. As a case in point they cited a customer in town whose modem had been continuously connected for over a month! Ralph told me that story and added, "That customer was me."

*(9) There are many measures of success. What do you think must be true in order for a programming language to be considered a success?*

The first answer I read was by Mr. Dimka: "One person considers it their favorite language." Interestingly, that's an answer I had in mind, too. I believe I mentioned in class that I discovered the language RATSNO (Rational SNOBOL) among the Icon stuff. Dave Hanson developed it as an experiment in software adaptability, to see how hard it would be to retarget the RATFOR preprocessor (about 1500 lines), which had a design principle of "RATFOR does not know any FORTRAN". Hanson completed the conversion in about 8 hours.

I loved RATSNO and used it for several projects as an undergraduate at N.C. State, including a Pascal compiler. I've never gone looking for other users (I should!), but as I far as I know, I was the only regular user of RATSNO. But for me, RATSNO was a success. http://www.cs.arizona.edu/classes/cs372/spring15/csc412project.pdf is an example of some RATSNO code. It solves language equations using Arden's Lemma. You'll see that thirty-five years ago I wrote more comments than I do now.

Here are other answers that caught my eye:

"A successful language displaces a previous language."—S. Stephens
"It can solve a problem."—J. Weiser
"If a lot of people are using it without complaining."—N. Gelo
"People use it."—J. Tom
"Mother's approval."—M. Curry
"Helped solve one problem better than any other language."

*(10) Under what circumstances should a programming language be considered dead?*

I've pondered this question a fair amount over the years. During the Icon segment of 372 in Fall 1996, Larry Johnson said, "I'm just worried we're studying some dead language just because it was invented at the U of A.".

As a practical matter I think it takes a lot to kill a programming language. One of my first thoughts was that if a language is written in assembler and no machines with that architecture still exist, that language is surely dead. However, one could write an emulator for that architecture and revive the language. Even if all implementations were lost but specifications or programs remained, again the language could be revived.

I think I may have mentioned that there was never a implementation of the Alphard language but several papers were published about it. (Ralph thought that was "nuts!")

Here are some of the many answers I enjoyed:
"No one considers it their favorite."—A. Dimka
"No one can write a program in it without Googling."—J. Wolfe
"When there are no new programs being written in it."—C. Johnson
"When you can't Google it."—M. Justice
"When the tools to run the programming language no longer exist."—J. Wynne
"When it has no more followers."—E. Saetren
"They don't die; they just go to sleep for a very long time."—B. Hammond
"When it has C in the name."

**Statistics and an adjustment**

While grading I ultimately came to not like problems/questions 2(1), 13(1), and 15(2). Five points were added to all scores to compensate for those questions. The scores written on your exams do not reflect that five-point adjustment.

Here are all scores, with that five-point adjustment applied:

```
107.00, 106.50, 106.50, 106.00, 105.25, 102.50, 100.50, 97.00, 96.50, 96.00,
93.00, 91.00, 90.50, 90.00, 89.50, 89.50, 89.50, 88.00, 87.50, 87.00, 87.00,
86.50, 86.50, 86.00, 84.50, 83.50, 83.00, 83.00, 83.00, 82.50, 82.00, 81.00,
80.50, 79.50, 78.00, 78.00, 76.50, 76.50, 76.00, 75.50, 74.50, 74.00, 73.50,
73.50, 72.00, 72.00, 69.50, 68.00, 66.00, 64.50, 64.50, 63.00, 62.50, 62.00,
62.00, 61.50, 61.50, 61.50, 60.50, 60.50, 57.75, 56.50, 54.50, 51.00, 37.50

n = 65
mean = 78.9615
median = 80.5
```

All overall final averages for the course are shown below, with grade lines drawn in. You can find your final letter grade in 372 by looking at your "Overall average" on D2L and seeing where you fall in the list below.

Drawing grade lines is always tough. For those just short of a letter grade I double-checked his/her exam for any additional possible points.

```
------ A's below
104.70
104.26
104.24
103.50
102.92
101.96
101.53
101.23
100.15
100.00
96.83
96.75
```

```
96.67
96.39
96.38
96.04
95.87
95.51
94.54
93.85
93.57
92.92
92.87
92.65
91.44
90.58
89.99
89.88
------ A's above; B's below
88.89
87.97
87.22
86.15
86.02
85.16
84.15
82.41
82.25
82.11
82.05
80.33
79.57
----- B's above; C's below
77.74
77.70
77.69
76.94
74.58
72.35
70.85
70.59
69.98
69.87
69.20
69.09
68.49
----- C's above; D's below
66.95
65.54
64.69
64.32
63.61
60.06
----- D's above; E's below
53.28
52.83
31.09
30.67
```