# CSC 372, Fall 2022
## Final Exam Solutions
## Thursday, December 15, 2022

**Problem 1: (6 points)** (mean: 5.38, median: 6, 3rd quartile: 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 and have a bit of depth, as described in the Piazza post that announced this problem.*

All the videos this semester were good and some were great. Videos are great in different ways. It'd be pretty hard for me to pick a best video but I will pick an All-Star team, and in alphabetic order by title, here it is:

== in JavaScript
https://www.youtube.com/watch?v=8m3DWtoumNs

Coding your own Music w/ Sonic Pi
https://youtu.be/DRmJVM9iY3E

Bash Arrays
https://drive.google.com/file/d/1kp9K3JHY5Z5SarV-Bi3KPSPP55v15w1n

Goroutines and Channels: Concurrency in Go
https://youtu.be/ZwY3kRMlenw

How I Typed "Toy Story 3" Without Any Characters in JavaScript: Automatic Type Conversion By Chase K
https://youtu.be/zjC7jfyrOZc

Make a Class Really Easy to Use in Python with Magic Methods by Amber Converse
https://drive.google.com/file/d/1jVgayWJ69iEP8_i1OlYBKobKojIOHrhb/view

A number of students made their YouTube videos "Unlisted" but I encourage all to list them—they might accumulate an interesting number of views over time. For example, Ryan Melzer's Spring 2016 video, *Haskell Monads in 8 Minutes*, has accumulated 53k views! You can see Ryan's video here: https://www.youtube.com/watch?v=gEoruozy3mk

**Problem 2: (10 points)** (mean: 8.9, median: 10, 3rd quartile: 10)

The following program is in Tcl and was adapted from `rosettacode.org/wiki/Ordered_words#Tcl`. Read through the code and then for one point each, briefly answer the questions that follow below the code.

```
 1      # Return a list of the ordered words (of maximal length) in a list
 2      proc chooseOrderedWords list {
 3          set maxlen 0
 4          set orderedOfMaxlen {}
 5          foreach word $list {
 6              # Condition to determine whether a word is ordered;
 7              # are its characters in sorted order?
 8              if {$word eq [join [lsort [split $word ""]] ""]} {
 9                  set wordlen [string length $word]
10                  if {$wordlen > $maxlen} {
11                      set maxlen $wordlen
12                      set orderedOfMaxlen {}
13                  }
14                  if {$wordlen == $maxlen} {
15                      lappend orderedOfMaxlen $word
16                  }
17              }
18          }
19          return $orderedOfMaxlen
20      }
21      # Get the dictionary and print the ordered words from it
22      set t [http::geturl "http://cs.arizona.edu/classes/cs372/fall22/web2"]
23
24      set result [chooseOrderedWords [http::data $t]]
25
26      puts "Longest ordered words:"
27      for {set i 0} {$i < [llength $result]} {incr i} {
28          puts [lindex $result $i]
29          }
```

*(1)* *What does the syntax for procedure <u>definitions</u> seem to be?*
`proc NAME PARAMS { ... }`

*(2)* How does one apparently create and initialize a variable?
`set VAR VALUE`

*(3)* *What appears to be the rule about when a variable name needs a sigil?*
When we want the value of the variable.

*(4)* *What purpose do square brackets (`[` and `]`) seem to serve?*
They indicate a procedure call, and enclose both the procedure's name and its arguments, a bit like Lisp S-expressions.

(5) *At line 8, `eq` is used to test equality but at line 14, == is used to test for equality. Why are two different operators used?*
Looks like perhaps `eq` is numeric equality and == is string equality.

*(6)* *What does the `{}` in line 4 apparently represent?*
An empty list.

*(7)* *Roughly translate line 5 (`foreach...`) into Ruby code that uses an iterator.*
`for word in list do`

*(8) Roughly translate line 28 (`puts [lindex $result $i]`) into Ruby.*

```
puts result[i]
```

*For one point each, **make up to four more significant observations** about the code, but maximum score for the problem is 10.*

Here are a few more observations:

- Tcl seems to be dynamically typed.
- The for-loop at 27 seems to have init/condition/advance elements just like Java, C, and others.
- "`http::...`" seems to reference elements in an `http` package/module.
- The `return` at line 19 implies that Tcl doesn't have "last value return" like Ruby.
- Indexing seems to be zero-based.
- `incr VAR` apparently increments the value of `VAR`.
- # is comment to end of line.

I wish I'd thought to include a question that asked whether you'd seen Tcl before, but I didn't!

You'll see on its Wikipedia page that Tcl, pronounced "tickle", was created in 1988 and its latest release was less than a month ago. I see about ten Tcl books on learning.oreilly.com at the moment.

https://wiki.tcl-lang.org/page/Applications+in+Tcl+and+Tcl%2FTk lists a number of large applications written in Tcl. At a Usenix conference years ago I remember a speaker mentioning a 250,000 line Tcl application that was integral to the operation of an oil rig in the Gulf of Mexico.

If you've done any graphics in Python with `tkinter`, take a look at https://docs.python.org/3/library/tk.html. It says, "The `tkinter` package is a thin object-oriented layer on top of Tcl/Tk."

**Problem 3: (11 points)** (mean: 6.95, median: 7, 3rd quartile: 10.125)

*Write a Haskell function `avgvows` of type `[String] -> Double` that returns the average (mean) number of vowels per word in a body of text, ignoring one-letter words. Assume all letters are lower-case. Examples:*

```
> avgvows ["what a", "treat"]
1.5

> avgvows ["ouch oops", "my bad"]
1.25

> avgvows ["my my"]
0.0
```

My solution:

```
avgvows lns = fromIntegral total_vowels / fromIntegral (length keepers)
    where
        all_words = words $ unwords lns
        keepers = filter (\w -> length w > 1) all_words
        vowel_counts = map (filter (\c -> c `elem` "aeiou")) keepers
        total_vowels = sum $ map length vowel_counts
```

**Problem 4: (8 points)** (mean: 5.43, median: 6, 3rd quartile: 7.625)

*Write a Prolog predicate `all_atoms(+L)` that succeeds iff the list `L` is not empty and all its values are atoms. Examples:*

```
?- all_atoms([just,a,test]).
true.

?- all_atoms([just,1,test]).
false.

?- all_atoms([]).
false.
```

This problem is a good example of the hazard of looking for a problem with a solution in hand! I was looking for a `findall` problem and, sure enough, the problem can be solved like this:

```
all_atoms(L) :-
      findall(A, (member(A,L),atom(A)), Atoms), length(Atoms,N),
      length(L,N), N > 0.
```

However, if you approach the problem without a hammer that's looking for a nail, you'll see a simple recursive solution:

```
all_atoms([X]) :- atom(X).
all_atoms([H|T]) :- atom(H), all_atoms(T).
```

**Problem 5: (20 points)** (mean: 14.3, median: 17, 3rd quartile: 18)

*This is the Prolog problem like `connect` and pit-crossing that you were told to expect.*

*Write a Prolog predicate `makepath/1` that takes a list of "layers" and prints a sequencing of **ALL** of them that has a continuous path of asterisks from top to bottom. Here's an example of a path through five layers:*

```
?- makepath(['---*-','--*--','----*','--*--', '---*-']).
---*-
--*--
--*--
---*-
----*
true.

?- makepath(['*--','--*','--*']).
No sequencing for those 3 layers
true.
```

My solution:

```
findseq([],_,[]).
findseq(Layers,N,[Layer|Seq]) :-
      select(Layer, Layers, Remaining),
      atom_chars(Layer, Chars),
      nth0(Pos, Chars, '*'),
      fit(N,Pos),
      findseq(Remaining, Pos, Seq).

fit(N, _) :- N < 0.
fit(N, Pos) :- abs(N-Pos) =< 1.
```

```
makepath(Layers) :-
      findseq(Layers, -1, Seq), print_layers(Seq), !.
makepath(Layers) :-
      length(Layers,N),
      format('No sequencing for those ~a layers~n', N).

print_layers(Layers) :- member(Layer, Layers), writeln(Layer), fail.
print_layers(_).
```

**Problem 6: (15 points)** (mean: 13.34, median: 15, 3rd quartile: 15)

*Write a Ruby program `hmt.rb` that, based on command-line arguments, prints the "head", "tail" and/or "middle" of its input lines, possibly multiple times. This problem was inspired by the `head` and `tail` UNIX commands.*

*The command*

***`$ ruby hmt.rb ht 5 < x.txt`***

*specifies that the "head" (`h`) and "tail" (`t`) of the input are to be printed. This causes the first five lines and then the last five lines of `x.txt` to be printed; a total of ten lines.*

*The command*

***`$ ruby hmt.rb tth 20 < x.txt`***

*causes the last twenty lines of `x.txt` to be printed twice, followed by the first twenty lines; a total of sixty lines.*

*If the number of lines to print is not specified, it **defaults to three**.*

My solution:

```
if ARGV.size == 2
    n = ARGV[1].to_i
else
    n = 3
end

lines = STDIN.readlines

ARGV[0].each_char do
    |c|
    if c == "h"
        puts lines[0...n]
    elsif c == "t"
        puts lines[-n..-1]
    elsif c == "m"
        start = lines.size/2 - n/2
        puts lines[start,n]
    end
end
```

**Problem 7: (9 points)** (mean: 7.04, median: 8, 3rd quartile: 9)

Write a Ruby method `lc_occurs(s,n)` that returns a string of the lower-case letters that occur **n** or more times in the string `s`. Characters in the result are in alphabetical order.

Examples:

```
>> lc_occurs("just a test", 1)
=> "aejstu"

>> lc_occurs("just a test", 3)
=> "t"

>> lc_occurs("",0)
=> "abcdefghijklmnopqrstuvwxyz"
```

**RESTRICTION: You may not use 'for' or 'while' statements in your solution.**

My solution:

```
def lc_occurs(s, n)
    counts = Hash.new 0
    s.each_char {|c| counts[c] += 1 }
    result = ""
    ('a'..'z').each {|c| result += c if counts[c] >= n}
    return result
end
```

**Problem 8: (12 points)** (mean: 10.43, median: 11, 3rd quartile: 12)

*In this problem you are to implement a Ruby class named `IntRange` that represents a non-empty range of integers.*

```
>> r = IntRange.new(3, 7)

>> r.inspect
=> "3 4 5 6 7 (5 values)"

>> r.size
=> 5

>> r.bounds
=> [3, 7]

>> r[1]
=> 3

>> r[5]
=> 7

>> [r[0], r[6]]
=> [nil, nil]
```

*Earn a point of extra-credit by not having `return` appear in your code!*

```
class IntRange
    def initialize first, last
        @first = first
        @last = last
    end

    def size
        @last - @first + 1
    end

    def inspect
        "#{(@first..@last).to_a * " "} (#{size}
values)"
    end

    def bounds
        [@first,@last]
    end

    def [](i)
        if i > 0 && i <= size
            @first + (i - 1)
        else
            nil
        end
    end
end
```

**Problem 9: (5 points)** (one point each unless otherwise indicated) (mean: 3.22, median: 3.5, 3rd quartile: 4)

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

*(1)* *What are two examples of sigils in Ruby and what does each indicate?*

| | |
|---|---|
| $ | global variable |
| @ | instance variable |
| @@ | variable |

Several students cited the `!` and `?` method suffixes but I don't believe I ever described those as sigils, nor did I find anything on the net that suggests they, too, should be considered sigils.

*(2)* *A novice programmer might simply understand that the following loop terminates when end of file is reached on standard input. What's a more thorough explanation of what happens when end of file is reached on standard input?*

```
while line = STDIN.gets
    puts line
end
```

At end of file, `STDIN.gets` returns `nil`. `nil` gets assigned to `line` and the value produced by the assignment operator is the value assigned, `nil` in this case. That causes the `while` to reduce to `while nil`, and because `nil` is regarded as "falsey", the `while` terminates.

*(3)* *For any Ruby Integer `n`, I'd like `n.big?` to return a "truthy" value if `n` is greater than 1000 and a "falsey" value if not. Write Ruby code to fulfill my wish.*

```
class Integer
    def big?
        self > 1000
    end
end
```

A lot of students got this half-right but both `class Integer` and `self > 1000` were required for full credit.

*(4)* *What is the essence of the duck typing mindset?*

A few students simply used James Whitcomb Riley's quote from Ruby slide 134. That made me wish I'd prohibited the word "duck" in the answer, but having failed to do that, I counted those recitations as correct.

I'd say that Mr. Ricci's answer was perhaps the most concise:

> "I don't care if you have the right type. Can you do this method?"

*(5)* *If you only remember one thing about Ruby, what will it be?*

I was hoping for some interesting things here but the most common answer by far was something about the `end` keyword—needing it, frequently forgetting it, etc.

**Problem 10: (2 points)** (mean: 0.89, median: 0, 3rd quartile: 2)

*Write a SNOBOL4 program `twice.sno` that prompts for the user for a number and then prints twice that number, like this:*

```
$ snobol4 twice.sno
Number?
7
Twice 7 is 14
```

*For a point of extra credit, make it loop until the user hits ^D (end-of-file) to terminate it.*

```
loop    output = "Number?"
        n = input                              :f(end)
        output = "Twice " n " is " (n * 2)     :s(loop)
end
```

**Problem 11: (2 points)** (mean: 0.84, median: 1, 3rd quartile: 1.5)

*The following questions about Icon are worth one point each. You may answer as many as you want but the maximum score on this problem is 2 points.*

*(1) Write two distinctly different Icon expressions that fail to produce a value.*

```
2 < 3
"x"[2]
```

*(2) Write an expression that produces the length of `s`, where `s` is an Icon string.*

```
*s
```

*(3) Write an expression that produces the sum of the first and last elements of `L`, where `L` is a list of strings that represent integers. If `L` is `["7", "45", "8"]` the sum would be `15`. Assume that `L` has at least one element.*

```
L[1] + L[-1]
```

*(4) What is a major aspect of Icon that Ralph Griswold was disappointed with?*

String scanning

*(5) Regarding language design, what was Ralph Griswold's opinion about whether a language should be easy to implement?*

Icon slide 5 mentions that there was generally a "disregard for implementation problems and efficiency".

**Extra Credit Section (½ point each unless otherwise noted)** (mean: 1.99, median: 2, 3rd quartile: 2.5)

*(1)* *With respect to programming languages, the term "mixin" was said to have been borrowed from the product line of a Somerville, Massachusetts business. What was the primary product of that business?*

I assume that ice cream was the primary product of Steve's Ice Cream in Somerville. When grading I got a smile from these guesses: cement, blenders, and cocktails.

*(2)* *What's the language in the SNOBOL family that was developed between SNOBOL4 and Icon and that "had everything".*

Lots of students said SNOBOL5 but it was SL5, short for SNOBOL Language 5.

*(3)* *Brooks Law says that "Adding manpower to a late software project makes it* later *."*

*(4)* *To the best of `whm`'s knowledge, what was the size of the RATSNO (Rational SNOBOL4) user community at its peak?*

My search for other RATSNO users continues but to the best of my knowledge, I was the only regular user of RATSNO. I used it for several undergraduate projects including a compiler for a subset of Pascal, an information retrieval system, and an extra-credit project involving Arden's Lemma/Rule. I couldn't even get my partner on the Arden's project to use RATSNO; he did his portion in plain old SNOBOL4!

Here is Dr. David R. Hanson's RATSNO paper: https://drhanson.s3.amazonaws.com/storage/documents/ratsno.pdf. Dr. Hanson was CS department head here for a time. When the question of "What's the best class you ever had?" comes up, my answer is Hanson's CSC 453—we wrote a C compiler, a linker, and a debugger. They were simplified in various ways, but they were also very "real".

*(5)* *Write the Prolog predicate `member/2` in terms of `nth0/3`.*

```
member(E,L) :- nth0(_,L,E).
```

*(6)* *Write a SNOBOL4 pattern that would match decimal fractions like `123.45`, `0.789`, and `1000.1`.*

```
decfrac = span(&digits) "." span(&digits)
```

*(7)* *Among Haskell, Icon, Java, Prolog, Python, Ruby, and SNOBOL4, which are the three oldest languages?*

SNOBOL4, Prolog, and Icon

*(8)* *What's relatively unique about when Ruby was created?*

I don't know of any other language that has an exact date cited for its "birth" but Matz wrote, "Well, Ruby was born on February 24, 1993."

*(9)* *Is Java a compiled language or an interpreted language? Briefly justify your answer. (**1 point**)*

See Ruby slide 132. I say that compilation or interpretation is an attribute of an implementation of a language, not the language itself. For any language, one can write a compiler or an interpreter; a great number of language implementations use a mix of compilation and interpretation techniques.

*(10)* *Write a good extra credit question related to the course material and answer it. (**1 point**)*
Many answers were very creative but I'll say that I like Ms. Ying's best:

*Q: Who would win in a fight: Haskell, Prolog, or Ruby?*
*A: Haskell be on the defense and guard itself.*
*Prolog would logic its way out of the fight.*
*Ruby could win in multiple ways.*

The non-lazy side of me aspires to add a few more notable answers here. Check back to see if that ever happens.

## Statistics

Here are all 48 scores, in descending order:

```
103.50, 100.50, 100.00, 99.50, 97.50, 97.00, 97.00, 94.00, 92.50, 92.00, 90.50,
89.50, 89.50, 87.00, 86.00, 86.00, 85.50, 85.00, 85.00, 84.00, 82.50, 81.50,
81.50, 80.50, 80.50, 80.00, 80.00, 79.50, 79.50, 78.50, 78.00, 76.50, 76.00,
75.00, 73.50, 73.50, 70.50, 70.00, 69.00, 69.00, 64.50, 62.50, 60.00, 57.00,
54.50, 51.50, 43.00, 8.00
```

Mean: 78.70
Median: 80.50
3rd Quartile: 89.5

Here's a table with per-problem statistics. Median/possible shows per-problem median scores divided by possible points, expressed as a percentage.

| Last Name | Videos | Tcl | avgvows | all_atoms | makepath | hmt | lc_occurs | IntRange | Ruby s/a | SNOBOL | Icon | EC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mean | 5.38 | 8.90 | 6.95 | 5.43 | 14.30 | 13.34 | 7.04 | 10.43 | 3.22 | 0.89 | 0.84 | 1.99 |
| Median | 6 | 10 | 7 | 6 | 17 | 15 | 8 | 11 | 3.5 | 0 | 1 | 2 |
| 3rd quartile | 6 | 10 | 10.125 | 7.625 | 18 | 15 | 9 | 12 | 4 | 2 | 1.5 | 2.5 |
| Median/possible | 100.0 | 100.0 | 63.6 | 75.0 | 85.0 | 100.0 | 88.9 | 91.7 | 70.0 | 0.0 | 50.0 | |

Here's a histogram of scores: