CSC 372 Final Exam Solutions
Tuesday, May 8, 2018

Problem 1: (6 points) (mean: 5.58, median: 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.

When grading I tallied the video topics that were cited and came up with the counts below, with a cut-off of three. There were multiple videos on some topics; a combined total is shown for those.

Python Magic Methods (combined)	15
PHP Arrays (combined)	11
Optionals in Swift	11
Tables in Lua	10
Java Lambdas and Streams (combined)	10
Bash Arrays (combined)	10
For-else clause in Python	9
Ownership in Rust	7
String Scanning in Icon	6
LINQ (Language Integrated Query) in C#	6
Functors in Haskell	5
Lambdas + Macros in Clojure	4
C++ Classes and Objects	3
INLINE X86 in C: A New Hope	3
Ruby Threads (combined)	3

There were four videos that I felt were particularly outstanding for one reason or another. Authors are not cited because I promised the potential of anonymity. In alphabetical order, here are those four videos:

Constraint Logic Programming Over Finite Domains

Feature and Usage of java.util.stream

Optionals in Swift

Ownership in Rust

Problem 2: (6 points) (mean: 4.62, median: 6)

Given the following Haskell function definition,

```
f xs ys = foldr (++) [] (map m (zip xs ys))
    where
        m (a,b) = [a,b]
```

What does the following expression produce?

```
f ['h','e','l','l','o'] "hello"
```

a) `"hellohello"`
b) `["hh","ee","ll","oo"]`
c) `[('h','h'),('e','e'),('l','l'),('o','o')]`
d) **`"hheellllloo"`**
e) A type error of some sort.

Michael wrote this question for the mid-term. I ended up without a good place for it then but I found it perfect for the final, both due to its use of higher-order functions and a well-designed set of distractors.

Problem 3: (4 points) (mean: 3.53, median: 4)

Write a Haskell function `again list`*, of type* `Eq t => [t] -> Bool`*, that returns* `True` *if and only if the first element of* `list` *appears again elsewhere in the list.*

`again` *returns* `False` *if* `list` *has less than two elements.*

This problem can be solved in many ways but here's a solution facilitated by the higher-order function `any`, of type `Foldable t => (a -> Bool) -> t a -> Bool`:

```
again (x:xs) = any (==x) xs
again _ = False
```

Problem 4: (3 points) (mean: 2.68, median: 3)

Write a Ruby iterator `inorder(x,y,z)` *that yields the values* `x`*,* `y`*, and* `z` *in ascending order. Assume that* `x`*,* `y`*, and* `z` *can be sorted with Ruby's* `Enumerable#sort` *method.* `inorder` *always returns* `nil`*.*

```
def inorder(x,y,z)
    [x,y,z].sort.each { |x| yield x }
    nil
end
```

Problem 5: (6 points) (mean: 4.85, median: 5.75)

This Ruby problem is similar to `btw.pl` from assignment 9. Write a Ruby iterator `btw(a,sep)` that yields copies of the array `a` with `sep` inserted between each element of `a` in turn. `btw(a,sep)` returns `a`.

My first solution was this,

```
def btw(a,sep)
    for i in 1...(a.size)
        yield a[0...i] + [sep] + a[i..-1]
    end
    a
end
```

but it took me a little time to convince myself that the subscripts were right. Both Mr. Duvall and Mr. Wan used `Array`'s `take` and `drop` instead, and I found that far easier to reason about. Here's an example of that approach:

```
def btw(a,sep)
    for n in 1...(a.size)
        yield a.take(n) + [sep] + a.drop(n)
    end
    a
end
```

A simpler solution that might well run as fast as the above two is to repeatedly make a copy of the array `a`, and insert the separator at the appropriate position for each `yield`:

```
def btw(a,sep)
    for i in 1...(a.size)
        a2 = a.dup
        a2.insert(i, sep)
        yield a2
    end
    a
end
```

A common error was to end up with multiple copies of `sep` in the yielded array.

Problem 6: (5 points) (mean: 4.24, median: 5)

This problem is similar to `re.rb` on assignment 7. Write a method `zip_re` that returns a regular expression that matches the city/state/zip line of postal addresses, like these:

```
Tucson, AZ 85750
Beverly Hills, CA  90210
Walkertown, NC 27051
```

Solution:

```
def zip_re
    /^(?<city>[A-Z][A-Za-z ]*[a-z]), +(?<state>[A-Z]{2}) +(?<zip>\d{5})$/
end
```

Problem 7: (9 points) (mean: 7.80, median: 8.5)

For this problem you are to write a Ruby program that implements a ***very simple*** *simple line-oriented calculator. It supports three commands/operations:*

`variable=integer`

Assigns the integer value to the specified variable.

`var1=var2+var3`

Assigns the sum of `var2` *and* `var3` *to* `var1`*.*

`/vars`

Print all variables and their values. ***Simplification****: Variables can be printed in any order.*

A number of students used regular expressions and/or `split` to parse the commands but I'd written the problem with the intention of simple string indexing being sufficient, like this:

```
vars = {}
while line = (print "calc> "; gets)
    line.chomp!
    if line == "/vars"
        for var,val in vars
            print("#{var}: #{val}\n")
        end
    elsif line =~ /\+/
        vars[line[0]] = vars[line[2]] + vars[line[4]]
    elsif
        vars[line[0]] = line[2..-1].to_i
    end
end
```

Problem 8: (5 points) (mean: 4.8, median: 5)

In this problem you are to implement a Ruby class named `Rope` that represents a length of rope that can be cut into two pieces.

```
class Rope
    def initialize(len)
        @pieces = [len]
    end

    def cut(len)
        if @pieces.size == 1 and len < @pieces[0]
            @pieces = [@pieces[0] - len, len]
            true
        else
            false
        end
    end

    attr_reader :pieces
end
```

Most students used two or three instance variables, but as you can see above, one is all you need.

Problem 9: (3 points) (mean: 2.40, median: 3)

Write a Prolog predicate `ints_atoms(+L,-E)` that first instantiates `E` to each integer in `L`, and then instantiates `E` to each atom in `L`.

```
ints_atoms(L,E) :- member(E,L), integer(E).
ints_atoms(L,E) :- member(E,L), atom(E).
```

Problem 10: (5 points) (mean: 3.39, median: 4.5)

Write a predicate `middle(+List, -M)` that instantiates `M` to the middle element of the list `L`. `middle` fails if the list has an even number of elements.

RESTRICTIONS:

- Just like on `append.pl` on assignment 9, **the only predicates you can use are `length` and `append`**.
- `middle` cannot be recursive.

```
middle(L,M) :-
    append(First,Rest,L), append([M],Last,Rest),
    length(First,Len), length(Last,Len).
```

Problem 11: (7 points) (mean: 5.44, median: 6.5)

For this problem you are to write a Prolog predicate `sumneq(+L, -Sum)`. The first argument, `L`, is a list of `vp/2` structures where both terms are integers. `sumneq` looks for `vp` structures with unequal terms, like `vp(3,4)` and `vp(-10,2)`, but not `vp(3,3)`. For each `vp` structure with unequal terms, `sumneq` computes the product of those two integer terms and then instantiates `Sum` to the sum of all those products.

```
sumneq(L,Sum) :-
    findall(Sum, (member(vp(X,Y),L), X\==Y, Sum is X*Y), Sums),
    sumlist(Sums,Sum).
```

Problem 12: (7 points) (mean: 5.32, median: 6.5)

Write a predicate `updown(+Ints, -Atom)` that instantiates `Atom` to a series of atoms in turn, based on the integers in `Ints`. Example:

```
?- updown([3,1,5], A).
A = 'up three' ;
A = 'down three' ;
A = 'up one' ;
A = 'down one' ;
A = 'up five' ;
A = 'down five'.
```

My solution:

```
updown(Nums, A) :-
    member(Num, Nums), member(UD, ['up ','down ']),
    n(Word,Num), atom_concat(UD,Word,A).
```

Problem 13: (10 points) (mean: 6.24, median: 8)

Write a Prolog predicate `seating(+Host, +Guests, -Seating)` that determines the seating for a dinner party to be attended by some number of conservatives and liberals.

```
seating(Host, Guests, [Host|Seating]) :-
     seatingH(Host,Guests,Seating),
     append(_,[Last],Seating), ok(Host,Last),!.

seatingH(_,[],[]).
seatingH(Seated, People, [Person|Seating]) :-
    select(Person, People, Remaining),
     ok(Seated,Person),
     seatingH(Person, Remaining, Seating).

ok(lib(_,_),lib(_,_)).
ok(con(_,_),con(_,_)).
ok(con(_,D1),lib(_,D2)) :- D1 + D2 =< 4.
ok(lib(_,D1),con(_,D2)) :- D1 + D2 =< 4.
```

`ok/2` can be reduced to three clauses by using a cut in a way we barely discussed, to prevent later clauses from being tried if we simply unify with the head in either of the first two clauses:

```
ok(con(_,D1),lib(_,D2)) :- !, D1 + D2 =< 4.
ok(lib(_,D1),con(_,D2)) :- !, D1 + D2 =< 4.
ok(_,_).
```

The predicate `=../2`, mentioned in the extra credit section, can be used to produce a less repetitious `ok/2` that is perhaps also less readable:

```
ok(P1,P2) :-
     P1 =.. [T1,_,D1], P2 =.. [T2,_,D2], T1 \== T2, !, D1 + D2 =< 4.
ok(_,_).
```

Problem 14: (10 points) (one point each unless otherwise indicated) (mean: 7.71, median: 8)

<u>The following questions and problems are related to Prolog.</u>

(1) *Write a sentence that expresses the relationship between the terms "fact", "clause", and "rule".*
Both facts and rules are clauses.

(2) *Prolog goals were modeled using the "Four port model". What are the four ports?*
call, exit, redo, fail

(3) *If a predicate with three clauses is being evaluated and the first clause fails, what does Prolog do?*
It tries the second clause.

(4) *What's the most important fundamental difference between a Prolog predicate like `length/2` and a function/method of the same name in Haskell, Ruby, or Java?*
In Haskell, Ruby, Java, and other conventional languages, `length` simply returns the length of a list. In Prolog, `length(?List,?Len)` can (1) produce the length of `List`, (2) produce a list of `Len` uninstantiated variables, (3) see if the length of `List` is `Len`, (4) produce all `List/Len` pairs.

(5) *What did you find hardest to understand about Prolog?*

(6) *In your mind, what's the most interesting thing about Prolog?*

A wide variety of answers earned points on these two questions.

(7) *What does the predicate `trace/0` do?*

Turns on tracing.

(8) *Write a Prolog clause that would generate a singleton warning.*

`f(X).`

(9) *(Two points) What does the `findall` predicate do?*

Evaluates a query and for each result produced adds an instance of a template to a list that is ultimately produced as the result of `findall`.

Problem 15: (2 points) (mean: 1.21, median: 1)

The following questions about SNOBOL4 are one point each. You may answer as many as you want but the maximum score on this problem is 2 points.

(1) *Write a SNOBOL4 program that copies standard input to standard output, just like the `cat` command. (For a point of extra credit, implement `tac` instead, copying lines in reverse order—last line first, first line last. Hint: remember that `char(10)` produces a newline character.)*

```
$ cat cat.sno
loop  output = input   :s(loop)
end

$ cat tac.sno
loop result = input char(10) result   :s(loop)
     output = result
end
```

Note: The `tac.sno` above is acceptable but in fact it produces an extra newline at end of the output. We can avoid that by using `rtab` (tab to from-right position) to match all but the last character and assign it to `output`.

```
loop result = input char(10) result   :s(loop)
     result rtab(1) . output
end
```

(2) *Write a SNOBOL4 expression that concatenates the strings held in the three variables `x`, `y`, and `z`.*

`x y z`

(3) *In the following SNOBOL4 statement, which would start in column 1, what does the "`. x`" mean?*

```
loop    line span(&digits) . x        :f(p)
```

Assign the spanned digits to the variable `x`.

(4) *In the SNOBOL4 statement above, what does `:f(p)` mean?*

If the pattern match fails, go the statement whose label is `p`.

(5) *Tell me something else of significance about SNOBOL4.*

Lots of answers sufficed here, but a common one was that Ralph Griswold played a central role in the creation of the SNOBOL family.

One student wrote, "This is the language that Mr. Mitchell coded in during his prime."

Problem 16: (2 points) **(mean: 1.33, median: 2)**

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

(1) *Briefly describe Icon.*
Ruby meets Prolog

(2) *What is the output of the following Icon expression?* `write(2 > 3, 2+3, !"abc")`
The expression `2 > 3` fails. That failure propagates and causes the `write` to fail, in turn causing no output whatsoever to be produced.

(3) *What was a fundamental design goal of Icon's string scanning mechanism?*
Be able to interleave string matching operations with regular computation—compute a little, match a little, ...

(4) *What's something that Prolog and Icon have in common?*
Backtracking

(5) *Tell me something else of significance about Icon.*
Just like in the SNOBOL4 section, a common answer was that Ralph Griswold played a central role in the creation of Icon.

Problem 17: (10 points) (one point each unless otherwise indicated) (mean: 6.9, median: 7)

Answer the following general questions.

(1) *This is a "matching" problem. Draw a line connecting each of the following language facilities on the left with the best characterization of primitives vs. idioms/techniques for that language on the right.*
Here are the correspondences:

Icon's string scanning:	Few primitives but many idioms/techniques
Regular expressions in Ruby:	Many primitives but few idioms/techniques
SNOBOL4 patterns:	Few primitives and few techniques

(2) *What is a fundamental characteristic of a statically typed language?*
The type of every expression can be determined by analyzing the code.

(3) *What is the essence of the duck typing mindset?*
Writing code that is concerned only with whether a value supports the operations that are used, not the type of the value.

(4) *With programming languages in general, what's the fundamental difference between a statement and an expression?*
Expressions produce values; statements don't.

(5) *What is the basic idea of type inferencing?*

Mr. Chambers wrote the following, which I considered to be an Original Thought:

"You act like this type, so you must be it."

(6) *In any language you know, write an expression that has both a value and a side-effect. Cite the language, too.*

In C and Java, `x=1` and `x++` are two examples. In Python, `x=1` is statement, not an expression.

(7) *What is a fundamental benefit provided by a REPL?*

I failed to note who wrote it, but I loved this answer: "Fun"

(8) *Tell me something about the concept of paradigms that is unrelated to programming paradigms in particular.*

A paradigm has a vocabulary.

(9) *(Two points) Which of the three main languages we covered this semester are you most glad we covered, and why?*

Some students answered with multiple languages; I divided such "votes" accordingly. The counts:

Haskell: 11.33
Ruby: 11.33
Prolog: 24.17

Extra Credit Section (½ point each unless otherwise noted) (mean: 3.59, median: 4)

(1) *The slides had some quotes from prior 372 students. What's something you remember about one of them?*

Eleven students cited Astier's Analogy—Prolog predicates are like D/C motors. Others cited Swartz re `append` , Nguyen re expressions vs. statements, and Sleiman re `findall`.

(2) *What does the Prolog predicate `=..(?Struct, ?List)` do?*

It expresses an equivalence between lists and structures:

```
?- con(david,3) =.. L.
L = [con, david, 3].
```

(3) *What is the Chomsky hierarchy a hierarchy of?*

Languages

(4) *What's an example of a list that Prolog can sort but that Ruby and Python 3 can't?*

`[1, "two"]`

(5) *(One point) Present a brief argument either in favor of keeping Prolog as a 372 language or replacing it.*

The count was 27 to 2 in favor of keeping Prolog.

(6) *Pick a nickname for one of the five languages we studied and explain why it's appropriate.*

Mr. Morgan wrote,

"Truth or Dare: Prolog. Either check and see if it is true or I dare you to try and make it so."

That looks like an Original Thought to me!

Here are the others: (apologies if I overlooked any!)

"Haskiller"
"Haskill, it will kill your fear of recursion."
"Truly Functional Fun = Haskell"
"Pro-long for Prolog because most of the clauses in Prolog are really long."
"Haskell -> Lazy Language"

"Prolog = Lil Logic. The Lil makes it cool, and Logic because Prolog is a fact based language."
"Haskell -> Nightmare. Enough said."
"Hask(hell) because it is hell to learn"
"Ruby = Diamond. More pretty."
"Prolog = APE. I chose this nickname because I feel we use append in the all the programs. :)"
"Mathskell, relation to math functions"
"It might be Ruby but it's a diamond in the rough..."
"Husky for Haskell 'coz both are fun."
"SNOBOL4 = Out-dated regEx's, typically new languages provide great libraires and regExs in place of SNOBOL4 features."
"Stupid - Haskell, name fits."
"Prolog: "Just the facts, ma'am." Entirely logic based."
"Haskell - recursion hell"
"Prolog - One liners—mostly gigantic single line expression"
"Friendly: Ruby = so many options; friendly to old and new programmers"
"Ruby: Typing Ducks"
"Ruby - Shape Shifter—it looks familiar but it's not as easy as it looks"
"Haskell -> Hasl-ALL, everything is a hassle"

(7) Put the five languages we studied in order by their age, oldest first.
SNOBOL4, Prolog, Icon, Haskell, Ruby. When grading, any order for the five was accepted.

(8) What's a piece of wisdom from Ralph Griswold that `whm` *passed along to you?*
Eight students cited some form of "If you're going to design a language, design a language you want to use."

Six students remembered that Ralph said I could add some number of features to Icon if I could find a corresponding number of features to remove.

Others:
"Humans start counting at 1."
"Simplicity is important."
"Less is more in a language."

(9) What is a non-control, non-alphanumeric character and its ASCII code?
Results on this question were disappointing. Do `man ascii` on lectura.

If you ever take time to memorize the ASCII character set, wait a year or two, and then let me know whether you thought the effort was worthwhile.

(10) Where in Gould-Simpson could leftover 372 handouts be found?
On the shelves at the 8th floor freight elevator.

(11) The TAs were required to learn all of your names. What are the TAs' names? (First and last!)
Alex Koltz and Michael Ordaz

(12) Late in the semester the class was invited to a sporting outing. What was the sport?
Bowling. Other answers: soccer, riding, "drinking and talking".

(13) Finish this sentence: "If I only remember one thing about 372 it will be ...". (apologies if I overlooked any!)
"Haskell is not the language for me!"
"Haskell"
"To always appreciate a REPL."

"Prolog"
"Spending 20+ hours on an assignment and feeling very proud/accomplished despite wanting to pull my hair out."
"the night club on every fridays"
"The University of Arizona was well-known by creating programming languages like Icon."
"The reason I no longer eat pancakes."
"Pancakes"
"Haskell's a pain / take all time"
"long office hours"
"Pancakes"
"Prepare for Prolog"
"Always have office hour in 914 until midnight."
"How great it was."
"How it feels to turn in this final."
"That some languages can handle tasks much more concisely than mainstream languages."
"Haskell gave me scars."
"Pancakes"
"Prolog is hard."
"Being surprised on day 1 when whm knew my name."
"Pancakes"
"Mr. Mitchell"
"That Java and Python and the other imperatives are computing but computing is not imperative. I.e., there are other ways to think about computation than imperatively."
"The huge assignment 9"
"Bowling is a sport."
"Putting the fun in functional programming."
"The size of the spec as I awake from my nightmare."
"Mr. Mitchell's handouts"
"Haskell will kick your butt and laugh at you while you cry."
"How much whm cares for the success of his students."
"STAY AWAY FROM HASKELL"
"The importance of choosing the right language for a task."
"What it's like to be a good teacher."
"How much time I spent on open frames in bowl.rb."
"whm worked on Icon."
"That I am not a fan of Prolog."
"whm"
"Haskell is hard."
"Maybe I should stick to Java."

(14) *(Zero points)* `whm` *booked his best of time of the semester ascending the steps to the eighth floor before Sunday's practice session! What was his time? (a) 25 seconds* ***(b) 50 seconds*** *(c) 1:40 (d) 3:20 (e) 6:40*

Overall statistics

For the record, the exam started with a twelve minute walk-through that ended at 3:42pm. The exam ran until 5:40pm, a total of 118 minutes.

All final exam scores, in order:

```
104.50, 102.00, 100.00, 98.25, 98.00, 96.75, 96.75, 96.50, 95.00, 94.00,
93.50, 92.25, 91.50, 91.00, 90.75, 90.50, 90.00, 87.25, 86.50, 86.00, 85.50,
85.00, 85.00, 83.50, 83.00, 83.00, 82.50, 81.25, 80.75, 79.75, 79.25, 78.50,
76.25, 76.00, 74.75, 72.25, 71.25, 69.00, 68.25, 66.00, 65.00, 62.00, 51.00,
46.50, 45.00, 44.25

N = 46
mean = 81.6
median = 84.25
```