CSC 372 Midterm Exam Solutions
Wednesday, November 2, 2022

Problem 1: (5 points) (one point each) (mean: 3.4, median: 3.5, 3rd quartile: 4)

What is the **type** of each of the following Haskell expressions? If the expression is invalid, briefly state why.

```
[1..3]
     [Int]

('x',['x'])
     (Char, [Char])

length
     [a] -> Int

tail "head"
     [Char] (or String)

map isDigit
     [Char] -> [Bool]
```

Problem 2: (6 points) (mean: 4.5, median: 5.3, 3rd quartile: 5.5)

This problem is like `ftypes.hs` on `a3`. Write functions `f1`, `f2`, and `f3` having each of the following types. There are no restrictions other than you may not use explicit type declarations. (e.g. `f1::...`)

```
f1 :: [a] -> Int
     f1 [a] = length [a]


f2 :: a -> b -> c -> [b]
     f2 a b c = [b]


f3 :: [(a, b)] -> ([b], [c])
     f3 [(a,b)] = ([b],[])
```

Problem 3: (4 points, as indicated) (mean: 2.75, median: 3, 3rd quartile: 4)

This problem is like `warmup.hs` on the assignments—write the following Haskell Prelude functions.

`tail` *[1 point] (Assume the list is never empty.)*

```
tail (_:t) = t
```

`last` *[1 point] (Assume the list is never empty.)*

```
last [x] = x
last (_:t) = mylast t
```

`zip` *[2 points] (Remember that the length of the shorter list is the length of the result.)*

```
zip (a:as) (b:bs) = (a,b) : zip as bs
zip _ _ = []
```

Problem 4: (18 points) (8 points each + 2 points for types)

(recursive—mean: 6.1, median: 8, 3rd quartile: 8)
(non-recursive—mean: 6.2, median: 8, 3rd quartile: 8)
(types—mean: 1.8, median: 2, 3rd quartile: 2)

For this problem you are to **write both recursive and non-recursive versions** of a Haskell function `mkintlists` that takes a list of strings of digits and returns a list of `Int`s with corresponding values.

```
> mkintlists ["315","91", "", "713"]
[[3,1,5],[9,1],[],[7,1,3]]
```

In the recursive version **ONLY**, use `error "even!"` to produce an error if an even digit is encounted:

```
> mkintlists ["31","12"]
*** Exception: even!   (Non-recursive version would produce [[3,1],[1,2]]).
```

BEFORE writing your two versions of `mkintlists`, what is the type of...

```
 mkintlist?  [[Char]] -> [[Int]]
digitToInt?  Char -> Int
```

Recursive:

```
mkintlists [] = []
mkintlists (s:ss) = doStr s : mkintlists ss

doStr [] = []
doStr (d:ds)
  | even val = error "even!"
  | otherwise = val : doStr ds
  where
    val = digitToInt d
```

Non-recursive:

```
mkintlists strings = map (map digitToInt) strings
```

Problem 5: (15 points) (mean: 10.1, median: 14, 3rd quartile: 15)

Without writing any recursive code, write a Haskell function `vcnp :: String -> Int` that counts the number of vowels in a string that are not immediately preceded by a vowel. Vowels may be in upper or lower case. Examples:

```
> vcnp "ate"
2

> vcnp "Oopsie!"
2

> vcnp "a-e-i-o-u-A-E-I-O-U"
10
```

Solution:

```
isv c = toLower c `elem` "aeiou"

f (last,count) c
    | isv last = (c, count)
    | isv c = (c, count+1)
    | otherwise = (c, count)

vcnp s = snd $ foldl f ('x',0) s
```

I was dismayedby the number of students who wrote out `['a','e','i','o','u']` instead of `"aeiou"`. And, at least one student, wrote that out twice.

Problem 6: (17 points) (mean: 10.7, median: 15.5, 3rd quartile: 16.5)

Write a Haskell function `coords rows cols` with type `Int -> Int -> IO ()` that PRINTS row and column coordinates for a grid with the given number of rows and columns. Example:

```
> coords 3 4
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
(2,0) (2,1) (2,2) (2,3)
```

Solution:

```
-- my first version of mkrc:
-- mkrc row col = concat ["(",show row,",",show col,")"]
mkrc row col = show (row, col)

mkrow cols row = unwords $ map (mkrc row) [0..cols-1]

coords rows cols =
     putStr $ unlines $ map (mkrow cols) [0..rows-1]
```

Problem 7: (5 points) (one point each unless otherwise indicated) (mean: 2.9, median: 3, 3rd quartile: 4)
The following questions and problems are related to Haskell.

(1) *The Haskell expression below has more parentheses than are needed! Mark **ALL** the parentheses that can be removed without changing the value of the expression.*

Before: `(take 3) (show x) ++ (replicate 5) (chr 66)`
After: `take 3 (show x) ++ replicate 5 (chr 66)`

(2) *Given the type of a function, how can we quickly tell if the function is polymorphic?*

A function is polymorphic if any parameter has a type variable.

(3) *Add parentheses to the type below to fully show the right-associativity of the `->` type operator.*

Before: `Int -> [Int] -> (Char -> Bool -> Bool) -> String`
After: `Int -> ([Int] -> ((Char -> (Bool -> Bool)) -> String))`

(4) *Briefly explain how the following `map` works, paying particular attention to the function being mapped. (That function is the result of **`(uncurry $ flip replicate)`**.)*

```
> map (uncurry $ flip replicate) [('a',3),('b',2)]
["aaa","bb"]
```

`flip replicate` creates a version of `replicate` with the parameters swapped:

```
> flip replicate
<function>

> :type it
it :: a -> Int -> [a]
```

Then, uncurrying the result function produces a function that can be applied to an `(a, Int)` tuple:

```
> uncurry it
<function>
> :type it
it :: (a, Int) -> [a]

> it ('a',3)
"aaa"
```

It was sufficient to say something like, "flip swaps the order of replicate's arguments and then uncurrying it lets it be mapped onto the tuples."

This question was mentioned as a possible midterm question in the `a5` solution write-up for `rtext.hs`.

(5) Consider the Haskell expression below. Does it have any partial applications? If so, briefly describe what constitutes each of the partial applications.

```
map (take 5)
```

First, `take 5` produces a partial application. Then `map` is applied to that function, producing a second partial applications.

Most students identified `take 5` as a partial application but only a few recognized that there is also a partial application of `map`.

Problem 8: (7 points) (mean: 5.1, median: 6.5, 3rd quartile: 7)

Using our DIY cons lists, write a Prolog predicate `eqalen/2` that succeeds iff its two arguments are lists that consist entirely of atoms **and** the length of all atoms in corresponding positions are equal. Examples:

```
?- eqalen(a:test:now:empty, i:went:too:empty).
true.       (Atoms in both lists have lengths of 1, 4, and 3, respectively.)

?- eqalen(a:test:now:empty, i:went:too:far:empty).
false.      (Four atoms in second list.)
```

RESTRICTION: The symbol = may NOT APPEAR in your solution! (This rules out == and \==, too, for example.)

Solution:

```
eqalen(empty, empty).
eqalen(A1:T1,A2:T2) :-
     atom(A1), atom_length(A1,L),
     atom(A2), atom_length(A2,L),
     eqalen(T1,T2).
```

Problem 9: (12 points) (mean: 6.4, median: 7.5, 3rd quartile: 10)

Write a Prolog predicate `lines(+Pairs, +Separator)` with this behavior:

```
?- lines(three-x:five-ok:four-oops:empty,'.').
x.x.x
ok.ok.ok.ok.ok
oops.oops.oops.oops
true.

?- lines(four-four:five-[]:empty,<>).
four<>four<>four<>four
[]<>[]<>[]<>[]<>[]
true.

?- lines.
Usage: lines(+Pairs,+Sep)
true.
```

Solution:

```
lines(Pairs,Sep) :-
     member_cl(English-Atom,Pairs), n(English,N),
     print(N,Atom,Sep), fail.
lines(_,_).

lines :- writeln('Usage: lines(+Pairs,+Sep)').

print(N,Atom,Sep) :-
     between(2,M,_),
     format('~w~w', [Atom, Sep]), fail.
print(_,Atom,_) :- writeln(Atom).
```

Problem 10: (5 points) (one point each unless otherwise indicated) (mean: 2.7, median: 3, 3rd quartile: 4)

The following questions and problems are related to Prolog.

(1) `x(3,4)` *is an example of a Prolog structure. Without using any parentheses, commas, or square brackets, write another example of a Prolog structure.*

```
3+4
```

(2) Prolog anatomy question: What are the three parts of a Prolog rule? (½ point)

head, neck, and body

(3) What are three distinct ways in which `length/2` *can be used? (1½ points)*

Get the length of a list.
See if a list has a specific length.
Make a list with a specific number of uninstantiated variables.

(4) What is the ***output*** *of the following query?*

```
?- between(1,3,A), writeln(A), A > 5,
   between(5,7,B), writeln(B), B < 10,
   writeln('Done!').
```

The output is:

```
1
2
3
```

It was also fine to include the result, "`false.`", but that's not part of the output.

(5) How do the following two goals differ in meaning?

`A+B == 5`

Compares the structure 3+5 to the number 5; always fails.

`A+B =:= 5`

Evaluates `A+B` *as an arithmetic expression and tests whether the result is equal to 5.*

Problem 11: (6 points) (one point each unless otherwise indicated) (mean: 3, median: 3, 3rd quartile: 4)

Briefly answer the following general questions.

(1) Who founded The University of Arizona's Computer Science department and when?

Ralph Griswold, in 1971

(2) What is the value and side effect of the following Python expression?

```
print("testing")
```

The value is `None`. The side effect is that `"testing"` is printed on standard output.

(3) What's the one feature/property that a language must have in order to do anything that even remotely resembles functional programming?

Functions must be first-class values. In general, it must be possible to use a function value in all the contexts where values of others types can be used—be held in variables and data structures , be passed as parameters and returned as results, etc.

(4) Which <u>one</u> of the following language elements is most essential for imperative programming AND why? (2 points)

(a) some sort of looping construct like a `while` or a `for`
<u>(b) an assignment operation</u>
(c) some sort of "print" statement
(d) procedures

Much of imperative programming is basically orchestration of changes to the value of variables but without an assignment operation, the value of a variable can't be changed.

(5) Early in the semester we talked about three aspects of expressions that are often important to understand and reason about. What are those three aspects?

Value, type, and side effect

Extra Credit Section (½ point each unless otherwise noted) (mean: 0.8, median: 0.5, 3rd quartile: 1.5)

(1) *Collectively, the body of facts and rules that implement a Prolog predicate is known as the* __procedure__ for the predicate.

Note: "clauses" (**plural**) was also counted as correct.

(2) Sadly, I never got around to answering this Haskell puzzle posed on 284:
Make the list `[take, tail, init]` valid by adding two characters.
Answer it now!

My thought was `[take 5, tail, init]` but Mr. Fielding came up with `[(take, tail, init)]`. Mr. Meyer, Ms. Rahman, and Mr. Waugaman came up with `["take, tail, init"]`

(3) What's a fundamental difference between using `>>> type(x)` in the Python shell and using `> :type x` in `ghci`?

`type` is a Python function—we can use it in a Python program—but `:type` is a `ghci` command, and not an element of Haskell itself. Similarly, `help` is a Python function but `:help` is a `ghci` command. **This distinction—whether something is part of a language or part of a tool—is important to understand!**

(4) Write our beloved `map` function in Python. (1 point)

Here are two versions; the second uses a list comprehension.

```
def map(f,L):
    result = []
    for e in L:
        result.append(f(e))
    return result

def map(f,L):
    return [f(e) for e in L]
```

(5) Several places in the Haskell slides mention "H10". Example: "Lambda abstraction (H10)". What is H10?

The Haskell 2010 Report. (Slide 37)

Lots of students said that "H10" referred to a version of Haskell but only Ms. Saran correctly identified it as my shorthand for a document.

(6) What would be a big simplification in the following Haskell code?

```
g list = foldl1 (\acm elem -> f acm elem) list
```

The anonymous function does nothing but pass its two arguments to `f`, so we could just call `f` directly, and that's the big simplification:

```
g list = foldl1 f list
```

A minor improvement is that we could also turn `g` into a partial application:

```
g = foldl1 f
```

(7) Write a non-imperative version of C's `strcpy(...)` function. (If you haven't had 352 nor are taking it now, plead ignorance for a half-point!)

Nobody recognized that this is simply impossible. At the heart of `strcpy` is copying characters from one location in memory to another, and without assignment—essential for imperative programming—that's impossible

(8) *Once `whm` decided he should teach Racket instead of Ruby, he held onto Racket like a monkey holding onto a piece of fruit!* [See also `youtube.com/watch?v=9jBgo7UipqY`.]

(9) Write a good extra credit question related to the course material and answer it. (1 point)

There were several good ones but relatively few students responded to this question.

Statistics

Here are all fifty scores, in descending order:

```
98.50, 97.00, 95.50, 95.50, 92.50, 91.50, 91.50, 89.00, 89.00,
88.00, 85.00, 84.00, 83.50, 79.00, 78.00, 78.00, 76.00, 75.50,
75.00, 74.00, 74.00, 74.00, 73.50, 72.50, 71.50, 70.00, 68.50,
67.50, 66.50, 66.00, 65.50, 65.00, 60.00, 60.00, 59.50, 58.00,
53.00, 53.00, 52.50, 52.50, 52.00, 48.50, 43.00, 41.00, 39.00,
31.50, 24.50, 21.00, 20.00, 3.00
```

Mean: 66.45
Median: 70.75
3rd Quartile: 82.38

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

	exprtypes	ftypes	simple	mkints rec	mkints nr	mkints type	vcnp	coords	Haskell S/A	eqalen	lines	Prolog S/A	Gen S/A	EC	Total
Mean	3.41	4.46	2.75	6.08	6.2	1.83	10.11	10.74	2.91	5.04	6.38	2.72	2.99	0.83	66.45
Median	3.5	5.25	3	8	8	2	14	15.5	3	6.5	7.5	3	3	0.5	70.75
3rd quartile	4	5.5	4	8	8	2	15	16.5	4	7	10	4	4	1.5	82.375
Median/possible	70.0	87.5	75.0	100.0	100.0	100.0	93.3	91.2	60.0	92.9	62.5	60.0	50.0	11.1	

Adjustment

I'm thinking the exam was perhaps a half-problem too-long, and some of the short answer questions perhaps took a little longer to answer than I'd anticipated. Taking all things into account, I've decided to add seven points to all scores.

With that adjustment applied, here's a histogram of scores: