Last name
____

CSC 372 Mid-term Exam
Thursday, March 10, 2016

READ THIS FIRST

Read this page now but do not turn this page until you are told to do so. Go ahead and fill in your last name in the box above.

<u>This is a 60-minute exam</u> with a total of 100 points of regular questions and an extra credit section.

<u>The last five minutes of the exam is a "seatbelts required" period</u>, to avoid distractions for those who are still working. If you finish before the "seatbelts required" period starts, you may turn in your exam and leave. If not, you must stay quietly seated—no "packing up"— until time is up for all.

<u>You are allowed no reference materials whatsoever.</u>

If you have a question, raise your hand. I will come to you. DO NOT leave your seat.

If you have a question that can be safely resolved with a minor assumption, like the name of a function or the order of function arguments, state the assumption and proceed.

Feel free to use abbreviations, like "`otw`" for "`otherwise`".

It's fine to use helper functions unless a specific form, such as point-free style, is specifically requested.

<u>Don't make problems hard</u> by assuming that they need to do more than is specifically mentioned in the write-up or that the solution that comes to mind is "too easy."

<u>If you're stuck on a problem, please ask for a hint.</u> Try to avoid leaving a problem completely blank—that's a sure zero.

<u>It is better to put forth a solution that violates stated restrictions than to leave it blank</u>—a solution with violations may still be worth partial credit.

When told to begin, double-check that your name is at the top of this page, and then **<u>put your initials in the lower right hand corner of the top side of each sheet, checking to be sure you have all six sheets.</u>**

<u>BE SURE to enter your last name on the sign-out log when turning in your completed exam.</u>

Problem 1: (8 points)

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

Assume numbers have the type `Int`. Remember that `String` is a type synonym for `[Char]`; those two can be used interchangeably.

Important: Remember that the type of a function includes the type of the arguments as well as the type of the value returned.

```
"abc"
```

```
('4', 52, [5,2])
```

```
last
```

```
[[True]]
```

```
filter even
```

```
map head
```

```
[head, head . tail]
```

```
(++"x")
```

Problem 2: (10 points) (two points each)

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

<u>Instances of poor style or needlessly using other Prelude or helper functions will result in deductions</u>.

Remember this order of preference for handling cases: patterns, guards, `if-else`. Be sure to use the wildcard pattern (underscore) when appropriate.

<u>There's no need to specify function types.</u>

`snd` (Returns second element of a two-tuple)

`head` (Ignore empty-list case)

`last` (Ignore empty-list case)

`map`

`zipWith` (Reminder: `zipWith (+) [1,2,3] [10,20,30]` is `[11,22,33]`)

<u>DID YOU REMEMBER TO USE WILDCARDS WHEN APPROPRIATE?</u>

Problem 3: (16 points) (8 points each)

For this problem you are to **write both recursive and non-recursive versions** of a function `numlist`, with type `[[Char]] -> IO ()`, that outputs a numbered list of the strings in a list:

```
> numlist ["just", "testing", "this"]
1. just
2. testing
3. this

> numlist []
[no items]
```

Just as you did with `street` and other problems on the Haskell assignments that produced output, build up a string containing newlines and output it with `putStr` as a final step.

Just like on assignment 3, the recursive version must not use any higher-order functions. Just like on assignment 4, write the non-recursive version imagining that you just don't know to how to write a recursive function.

`unlines` and/or `concat` may be useful:

```
> unlines ["a","b","c"]
"a\nb\nc\n"

> concat ["a","bc","d"]
"abcd"
```

Problem 4: (4 points)

Consider the following interaction with `ghci`:

```
> map count [10,20,30]
[1,2,3]

> map count [7,8,9,10]
[1,2,3,4]
```

For this problem you are to either (1) write a function `count` that behaves as shown above or (2) explain why it is impossible to write such a function.

Problem 5: (8 points)

The following function, `revtups`, takes a list of two-tuples and produces a new list with the tuples' elements swapped **and** the order of the tuples reversed:

```
revtups list = foldl ff [] list
```

Usage:

```
> revtups [(1,'a'),(2,'b'),(3,'c')]
[('c',3),('b',2),('a',1)]

> revtups [("one",[1]), ("two",[2])]
[([2],"two"),([1],"one")]
```

For this problem you are to:

1. State the type of `revtups` (1 point)
2. Write a folding function `ff` that will work with `revtups` as shown above.

Problem 6: (12 points)

Write a Haskell function `co chars string`, with type `[Char] -> [Char] -> Int`, that counts how many times the characters in `chars` occur in `string`.

Usage:

```
> co "aeiou" "just a test"
3

> co ['0'..'9'] "12:15pm"
4

> co "xyz" ""
0

> co "" "12:15pm"
0
```

You may use only Prelude functions on this problem but there are no other restrictions.

Remember that the Prelude has a `sum` function: `sum [3,1,5]` produces `9`.

Assume the characters to count are unique; you won't see something like `co "aaa" ....`

Problem 7: (22 points)

Write a Ruby program `adjcols.rb` that makes specified adjustments to columns of numeric values in a CSV (comma-separated values) file.

Adjustments are specified as command-line arguments. Here's a sample invocation:

```
ruby adjcols.rb 2:+10 3:=7 5:-1 < x.csv
```

Adjustments have the form *`COLUMN-NUMBER:CHANGE`*. Examples:

`2:+10`	means to add `10` to all values in column `2` (column numbers are 1-based and positive)
`3:=7`	means to change all values in column `3` to `7`
`5:-1`	means to subtract `1` from all values in column `5`

Here's a CSV file:

```
% cat x.csv
name,q1,q2,q3,q4
whm,2,5,20,5
jmc,7,6,18,10
chris,5,0,10,15
```

`adjcols` reads lines from standard input (use `STDIN.gets` to read lines) and writes the lines to standard output. `adjcols` assumes the first line is a header row and outputs it unchanged.

Execution:

```
% ruby adjcols.rb 2:+10 3:=7 5:-1 < x.csv
name,q1,q2,q3,q4
whm,12,7,20,4
jmc,17,7,18,9
chris,15,7,10,14
```

Columns can be specified in any order. Changes are cumulative. An example of both:

```
% ruby adjcols.rb 2:+100 3:=1 2:+1000 < x.csv
name,q1,q2,q3,q4
whm,1102,1,20,5
jmc,1107,1,18,10
chris,1105,1,10,15
```

The CSV file may have any number of lines, and lines may have any number of columns. Assume that adjustments are well-formed and that the specified column is always present and contains an integer value. Any number of adjustments may be present.

Ruby note: `"+3".to_i` produces `3` and `"-5".to_i` produces `-5`.

THERE'S SPACE FOR YOUR SOLUTION ON THE NEXT PAGE

For reference:

```
% ruby adjcols.rb 2:+10 3:=7 5:-1 < x.csv
name,q1,q2,q3,q4
whm,12,7,20,4
jmc,17,7,18,9
chris,15,7,10,14
```

```
% cat x.csv
name,q1,q2,q3,q4
whm,2,5,20,5
jmc,7,6,18,10
chris,5,0,10,15
```

Problem 8: (5 points) (one point each unless otherwise indicated)

The following questions and problems are related to Haskell.

(1) Add parentheses to the following type to explicitly show the associativity of the -> operator.

```
[Int] -> (Int -> Bool) -> [Bool]
```

(2) Fully parenthesize the following expression to show the order of operations.

```
f  2  g  3  +  x  f  g  [ a  b  c  *  1 ]
```

(3) Rewrite the following function binding to use point-free style:

```
f1 x = f (g x)
```

(4) Briefly, how does Haskell's pattern matching contribute to the readability of Haskell code?

(5) Once upon a time I had this line of code in a file:

```
f x y = x*3 + y
```

I wanted to produce an error on that line, so I typed some junk into the line:

```
asdsf sdfs 3 3 3 f x y = x*3 + y
```

But, that goofy code loaded without error! Why? (Hint: Be sure your answer is more than something like "It's still valid code.")

Problem 9: (5 points) (one point each unless otherwise indicated)

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

(1) Aside from the fact that strings are mutable in Ruby and are immutable in Java, what's an important difference between the string-handling facilities in Ruby and Java?

(2) Imagine that a Ruby class has methods named `mudge` and `mudge!`. What does that imply?

(3) What's a fundamental semantic (i.e., non-syntactic) difference between `if-else` in Ruby and `if-else` in Java?

(4) Aside from syntax, what's a fundamental difference between `:load x.hs` in `ghci` and `load "x.rb"` in `irb`?

(5) What's the value of each of the two following Ruby expressions?

```
"abc"[5] && false
```

```
0 || 1
```

Problem 10: (10 points) (one point each unless otherwise indicated)

Briefly answer the following general questions.

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

(2) What are two aspects of a "paradigm", as described in Kuhn's *The Structure of Scientific Revolutions*? Here are two **wrong** answers: "functional" and "object-oriented". (Two points!)

(3) What's a fundamental characteristic of a language that uses static typing?

(4) What's a fundamental characteristic of a language that uses dynamic typing?

(5) Show an example of syntactic sugar in any language you know.

(6) Aside from the intrinsic elements of a language, like its design and performance, what are two factors that can influence the popularity of a language?

(7) whm often says "In Haskell we never change anything; we only make new things." What's an example of that?

(8) When a new graduate research assistant wanted to add a bunch of new features to Icon, what did the project's wise leader say?

(9) What's one way in which having a REPL available makes it easier to learn a programming language?

Extra Credit Section (½ point each unless otherwise noted)

(1) whm believes the abbreviation LHtLaL appears nowhere on the web except in his slides. What does it stand for?

(2) To whom does whm attribute the following quote?
"When you hit a problem you can lean forward and type or sit back and think."

(3) Consider this Haskell function: `add x y = x + y`. What is the exact type of `add`?

(4) Double tricky: What is the exact type of the Haskell list `[foldl, foldr]`?

(5) The Haskell expression `[`*`FROM`*`..`*`TO`*`]` produces an empty list if *FROM* > *TO*. Write a non-recursive function `range from to` that's a little smarter: `range 1 5` returns `[1,2,3,4,5]`, and `range 5 1` returns `[5,4,3,2,1]`.

(6) Suppose the second example for problem 4 (page 5) had been this:

```
> map count [6,7,8,9]
[1,2,3,4]
```

Briefly, how would that have changed your answer for that question?

(7) If you only remember one thing about the Haskell segment of 372, what will it be? (Ok to be funny!)

(8) What is "duck typing"?

(9) With Ruby in mind, define the term "iterator".

(10) Name a language other than Icon that was created at The University of Arizona.