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

Problem 1: (8 points) (mean = 5.6, median = 5.5)

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

```
"abc"
      [Char]

('4', 52, [5,2])
      (Char, Int, [Int])

last
      [a] -> a

[[True]]
      [[Bool]]

filter even
      [Int] -> [Int]

map head
      [[a]] -> [a]

[head, head . tail]
      [[a] -> a]

(++"x")
      [Char] -> [Char]
```

Problem 2: (10 points) (two points each) (mean = 8.3, median = 8.5)

```
snd          (Returns second element of a two-tuple)
      snd (_,x) = x

head         (Ignore empty-list case)
      head (h:_) = h

last         (Ignore empty-list case)
      last [x] = x
      last (_:t) = last t

map
      map _ [] = []
      map f (h:t) = f h : map f t

zipWith        (Reminder: zipWith (+) [1,2,3] [10,20,30] is [11,22,33])
      zipWith f (x:xs) (y:ys) = f x y : zipWith f xs ys
      zipWith _ _ _ = []
```

Note the ordering of the clauses: The first clause handles the case that both lists have at least one element. If that's not true, we're done, and that's handled by the second clause.

Problem 3: (16 points) (8 points each) (recursive: mean = 6.2, median = 7; non-recursive: mean = 5.0, median = 6)

*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]
```

Solutions:

```
numlist [] = putStr "[no items]\n"
numlist items = putStr $ unlines $ helper 1 items
    where
        helper _ [] = []
        helper n (item:items) =
            (show n ++ ". " ++ item) : helper (n+1) items


numlist [] = putStr "[no items]\n"
numlist items = putStr $ unlines $ zipWith f [1..] items
    where
        f num s = show num ++ ". " ++ s
```

Problem 4: (4 points) (mean = 3.3, median = 4)

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.

It's impossible. Functions always produce the same output for a given input but in the first case, `count 10` produces `1` and in the second case, `count 10` produces `4`.

Problem 5: (8 points) (mean = 5.7, median = 7)

*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.*

```
1.    revtups :: [(a,b)] -> [(b,a)]
2.    ff acm (a,b) = (b,a):acm
```

Problem 6: (12 points) (mean = 8.2, median = 10)

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
...
```

My first solution was this:

```
co chars str = sum $ map (f str) chars
    where
       f str c = length $ filter (\e -> e == c) str
```

Of the non-recursive solutions I saw, I felt that Ms. McCabe's was the best:

```
co chars string = length $ filter (\x -> x `elem` chars) string
```

Several students wrote a non-recursive solution that was something like this:

```
co chars (c:cs)
    | c `elem` chars = 1 + co chars cs
    | otherwise = co chars cs
co _ _ = 0
```

Problem 7: (22 points) (mean = 14.4, median = 18)

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`

This is my solution:

```
puts STDIN.gets

while line = STDIN.gets
```

```
        parts = line.split ","

        for adj in ARGV
            col, delta = adj.split ":"
            col = col.to_i - 1
            if delta[0] == "="
                newval = delta[1..-1].to_i
            else
                newval = parts[col].to_i + delta.to_i
            end
            parts[col] = newval.to_s
        end

        puts parts * ","

    end
```

I'd hoped that it would be quickly observed that lines could processed one at a time, but a number of students read all the lines into an array and then iterated over the elements of that array. An attribute that I've found to be common among many long and/or wrong solutions on assignments and exams is this: creation of unnecessary data structures. It takes code to build those unnecessary structures and then code to get data out of those structures. For this problem it also turns a program that could process a file of unlimited size to one that's limited by available memory.

A minor thing that made me groan during grading was the use of a flag to indicate whether the input line at hand was the first line. It seems far simpler to just read and write that first line.

Problem 8: (5 points) (one point each unless otherwise indicated) (mean = 2.9, median = 3)

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]

[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 ]

(((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)

f1 = f . g
```

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

Patterns let us bind names to various elements of data structures, so that instead of writing code that digs into data structures, like

```
f x = g (head x) ++ h (tail x)
```

we can write

```
f (h:t) = g h ++ h t
```

(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.")

The junk I typed turned the two-argument function `f` into a seven-argument function named `asdsf`. Three of the arguments are literal pattern matches for the number 3.

Problem 9: (5 points) (one point each unless otherwise indicated) (mean = 2.3, median = 2)

The following questions and problems are related to Ruby.

(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?

Ruby lets us specify portions of strings with `s[n]`, `s[start,length]`, and `s[Range]`. Additionally, those expressions can be targets of an assignment. Many other answers are valid, too.

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

`mudge` has both an applicative and an imperative form. `mudge`, the applicative form returns a result but doesn't modify its receiver. `mudge!`, the imperative form modifies its receiver.

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

`if-else` is a statement in Java but in Ruby it's an expression. In Ruby I can say something like `val = if a > b then c end`.

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

`:load` is implemented by the REPL that `ghci` provides; it's not Haskell code. Ruby's `load` is a method; `load "x.rb"` is a Ruby expression that has a side effect of loading the code in `x.rb`.

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

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

0 || 1
      0
```

I was surprised by the number of students that missed this one.

Problem 10: (10 points) (one point each unless otherwise indicated) (mean = 6.1, median = 7)

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 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!)*
See Haskell slide 4.

(3) *What's a fundamental characteristic of a language that uses static typing?*
Type errors can be detected without running any code or knowing what specific input values are.

(4) *What's a fundamental characteristic of a language that uses dynamic typing?*
In the general case, type errors can't be detected until code is run.

(5) *Show an example of syntactic sugar in any language you know.*
In Haskell, `"abc"` is syntatic sugar for `['a', 'b', 'c']`.

(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?*
See intro slide 34.

(7) *whm often says "In Haskell we never change anything; we only make new things." What's an example of that?*
If the first element of a list of integers was to be changed to zero, we'd do that by making a new list whose head is zero and whose tail is the of the list at hand.

(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?*
Ralph said I could all the features I wanted but for every feature I wanted to add, I first had to find a feature to remove.

(9) *What's one way in which having a REPL available makes it easier to learn a programming language?*
It makes it easy to experiment with things.

Extra Credit Section (½ point each unless otherwise noted) (mean = 1.9, median = 2)

(1) *whm believes the abbreviation LHtLaL appears nowhere on the web except in his slides. What does it stand for?*
Learning How to Learn a Language

(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."
Dr. Proebsting

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

(4) *Double tricky: What is the exact type of the Haskell list* `[foldl, foldr]`*?*
My first thought was that it would be a type error because the functions don't have the same type but when I saw that it worked, I realized that the types of the two can be unified:
`[(b -> b -> b) -> b -> [b] -> b]`

(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]`.

```
range from to
  | from <= to = [from..to]
  | otherwise = reverse [to..from]
```

(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?
It would now be possible to implement `count`.

Note: This question was ill-conceived—virtually any answer could be argued as being correct but it was only

marked as correct if you said that `count` could now be implemented. Consider a half-point of the 8-point adjustment as being compensation for any injustice on this question.

(7) If you only remember one thing about the Haskell segment of 372, what will it be? (Ok to be funny!) Presented anonymously and in random order, here's what was said:

Haskell has no debugger, and thus sucks.
functional programming can be very useful
That recursion is occasionally ok
maps & folds are useful!
Partial application
The power of cons.
I need to watch my types...
unhelpful error messages :(
It was (fun)ctional!
recursion is easier with patterns
Anonymous functions are beautiful.
The Friday Night Club isn't as fun as it sounds
Don't over think it!
I'm not good at it.
Patterns.
pancakes on the stack
Haskell functions seem to be redundant to death (having to write the function name again for every possibility of input)
I can't eat pancakes any more
putting the fun in FUNctional programming
So much recursion
mapping
You never change anything. Only make new things.
Haskell is short for HaskHell
higher order functions
expected [Char], was instead [[Char]]
A whole lot of time spent to find a solution with very little code
Iteration is for the weak!
Recursion isn't complete hell
Pancake
Crying tears of blood when forgetting to add a base case in recursion problems
Type error everywhere
functions are values
higher order function
Matt Gautreau was right. Don't tell him I said that.
Patterns over guards over if-else.
Pancakes Toyota
There's probably a Prelude function to do something for you if you look hard enough
f this f that f it
variable state, why don't you change!
maps are crazy useful
Recursion is now my best friend haha
Know your recursion!
anonymous functions
recursion!

(8) What is "duck typing"?

A style of programming/mindset for typing where we're only concerned about the operations provided by objects rather than their types.

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

An iterator is a method that can invoke a block.

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

See intro slide 38.

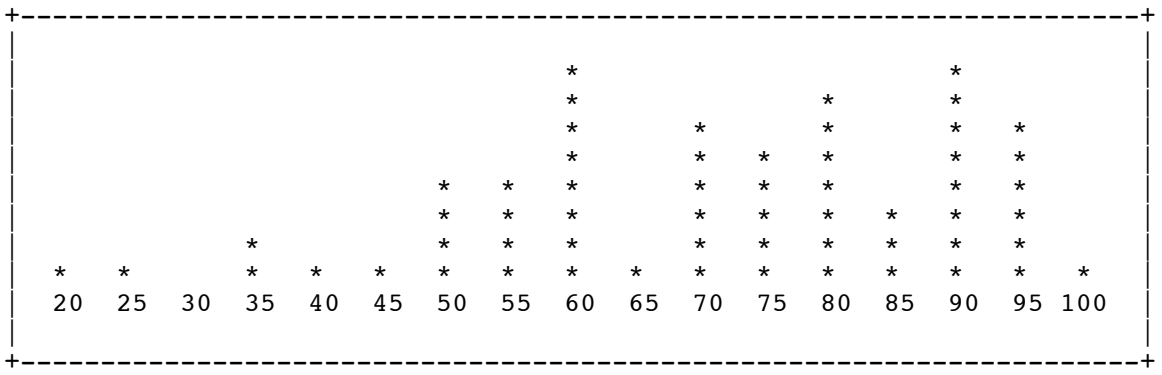
Statistics

Here are all scores:

```
98.5, 95.5, 94.5, 94, 94, 93, 92.5, 91, 91, 90, 89.5, 89.5, 88, 87.5, 87.5,
84.5, 84, 82.5, 81.5, 81, 81, 79.5, 79.5, 78, 77.5, 76, 75, 74.5, 73.5, 73,
71.5, 71.5, 69.5, 68.5, 68, 67.5, 63, 61.5, 61, 60.5, 60, 59, 58.5, 58.5, 58.5,
53.5, 53.5, 53, 53, 49.5, 48.5, 48, 47.5, 46, 40.5, 37, 34.5, 22.5, 18.5

n = 59
mean = 69.8
median = 73
```

Here's a histogram of all scores. The five-point wide bins cover the interval `[X-2.5,X+2.5)`, where `X` is the label for a bin.



The mean and median were lower than I expected, so eight points were added to all scores before loading them on D2L.