Solutions for
CSC 372 Haskell Mid-term Exam
Feburary 28, 2014

Problem 1: (15 points) Mean and median: 10.8, 10.5

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

Assume numbers have the type `Int`. *Remember that the type of* `True` *is* `Bool`.

```
"x"
```
`[Char]` or `String`

```
(1,'x',[True])
```
`(Int, Char, [Bool])`

```
["x":[]]
```
`[[[Char]]]`

A little tricky, and gotten wrong by many, typically one set of brackets short. Think about turning `"x":[]` into `["x"]`, then `[['x']]` and then adding the outer brackets for `[[['x']]]`, to get `[[[Char]]]`.

```
head
```
`[a] -> a`

```
not . not . isLetter
```
`Char -> Bool`

```
map not
```
`[Bool] -> [Bool]`

```
(+1) . (0<)
```
Invalid, since `(0<)` produces a `Bool`, and `Bool`s can't be added.

```
\x -> x + 1
```
`Int -> Int`

```
[1,'2',"3"]
```
Invalid, due to multiple types.

```
isDigit . chr . head . (:[]) . (+3) . ord . chr
```
`Int -> Bool`

As I mentioned at least a couple of times in class, if a composition is valid you only need to look at the input of the rightmost function and the output of the leftmost function to know the type of the full composition.

Problem 2: (12 points) (two points each) **Mean and median: 10.7, 11**

Using no functions other than helper functions you write yourself*, implement the following Prelude functions.*

Note*: There will be a ½-point penalty for each case of not using the wildcard pattern (the underscore) where it would be appropriate.*

I forgot to specify handling for empty lists for `head` and `tail` so we didn't take that case into account when grading.

```
head [] = error "empty"
head (h:_) = h


tail [] = error "empty"
tail (_:t) = t


length [] = 0
length (_:t) = 1 + length t


sum [] = 0
sum (h:t) = h + sum t


last [] = error "shortList"
last [x] = x
last (_:t) = last t


snd (_,x) = x
```

Problem 3: (10 points) Mean and median: 9.3, 10

Write a function `prswap` *that swaps the elements of a list on a pair-wise basis. That is, the first and second elements are swapped, the third and fourth are swapped, etc.*

ASSUME the list has an even number of elements but is possibly empty.

There are ***no restrictions*** *on this problem.*

A few approached this with a helper function for tracking the position and I think there were even a handful that used folding but my intention was a simple recursive solution:

```
prswap [] = []
prswap (a:b:xs) = b:a:prswap xs
```

Several students used a take 2/drop 2 approach, like this:

```
prswap [] = []
prswap list = reverse (take 2 list) ++ prswap (drop 2 list)
```

One of the general wisdoms I've learned for languages that support cons lists is that cons is always preferred over concatenation. Cons simply makes a new node but concatentation is done by making a series of cons nodes with each of the elements in the left operand. Here's the source for ++ from the Haskell Prelude:

```
(++) []     ys = ys
(++) (x:xs) ys = x : xs ++ ys
```

Something like `[1,2,3] ++ [4,5,6]` is effectively `1:2:3:[4,5,6]`

With all that in mind I'd think that the take 2, drop 2, concat version would be much slower and take far more memory but with `:set +s` in my `.ghci` I see the following.

```
% cat prswap.hs
prswap1 [] = []
prswap1 (a:b:xs) = b:a:prswap1 xs

prswap2 [] = []
prswap2 list = reverse (take 2 list) ++ prswap2 (drop 2 list)

m = [1..10000000]

run1 = length $ prswap1 m

run2 = length $ prswap2 m

% ghci prswap.hs
...
> run1
10000000
(3.87 secs, 1765864896 bytes)

% ghci prswap.hs  (Fresh run of ghci, to get a clean slate.)
...
> run2
10000000
(4.97 secs, 3325445008 bytes)
```

The cons version is faster but only by about 20%. The cons version has half as much memory throughput but I imagined it'd be a smaller fraction of the concat version. Results were similar with repeated testing.

I've clearly got a lot to learn about Haskell!

This is also a good example of a bad assumption about performance. I've seen plenty of those in my career, both by myself and others, and I've learned that nothing takes the place of running cases and looking at the numbers. But even then, pitfalls abound!

Problem 4: (10 points) Mean and median: 9.5, 10

Write a function `rotabc` that changes `a`'s to `b`'s, `b`'s to `c`'s and `c`'s to `a`'s in a string. Only lowercase letters are affected.

*There are **no restrictions** on this problem but it must be written out in full detail—no ditto marks, abbreviations, etc. It must be ready for an ASU or UNC-CH CS graduate to type in.*

Here's a solution written by an N. C. State University grad:

```
rotabc = map f
    where
        f 'a' = 'b'
        f 'b' = 'c'
        f 'c' = 'a'
        f c = c
```

I'd hoped that most students would immediately see this as a mapping problem. Also, my hope in requiring the solution to be written out in full detail was to make people think in terms of low repetition but I suppose that once a solution is mind there's a strong temptation to just go with it.

Lots of solutions were recursive and repetitious, like this:

```
rotabc [] = []
rotabc (c:cs)
    | c == 'a' = 'b': rotabc cs
    | c == 'b' = 'c': rotabc cs
    | c == 'c' = 'a': rotabc cs
    | otherwise = c : rotabc cs
```

Problem 5: (20 points) (ten points each) **Mean and median: 9.6, 10 for recursive; 8.3, 10 for non-recursive**

*For this problem you are to write **two separate versions** of a function named `bigTuples` (call it `bt`).*

One version must not use any higher order functions (like assignment 1); the other version must not use any explicit recursion (like assignment 2, minus `warmup.hs`).

`bigTuples max tuples` produces a list of the 2-tuples in `tuples` whose sum is larger than `max`, i.e., for a tuple `(a,b)`, `a + b > max`.

```
bigTuples _ [] = []
bigTuples n ((x,y):ts)
    | x+y > n = (x,y):rest
    | otherwise = rest
    where
        rest = bigTuples n ts

bigTuples n tuples = filter (\(x,y) -> (x+y) > n) tuples
```

Problem 6: (6 points) (5 for function, 1 for type) **Mean and median: 4.9, 6**

Write a function `fml list` that returns a 3-tuple with the first, middle and last elements of a list. Assume the list is non-empty and has at least one element.

Restriction: You may not use the !! list indexing operator.

***And, answer this question, too**: What is the type of `fml`?*

```
fml::[a] -> (a,a,a)
fml x = (head x, middle, last x)
    where
        middle = last (take ((length x `div` 2) + 1) x)
```

Some students made this a little harder than necessary by writing a recursive function to find the middle element of a list, some with a counter and some with an `init` and `tail` combo to repeatedly take an element off both ends. I'll be honest and say that when grading we didn't take too close a look for off-by-one errors on the counter-based functions, figuring they were already self-penalizing by the amount of time they took.

Problem 7: (10 points) **Mean and median: 6.9, 9.5**

Consider a function `separate s` that returns a 2-tuple with the digits and non-digits in the string `s` separated, with the initial order maintained.

```
> separate "July 4, 1776"
("41776","July , ")

> separate "Problem 7: (10 points)"
("710","Problem : ( points)")
```

Here is a partial implementation of `separate`*, using* `foldr`*:*

```
separate s = foldr f ([],[]) s
```

Your task on this problem is to write the folding function `f`*. Remember that for* `foldr`*, the type of the folding function can be described as* `elem -> acm -> ~~elem~~ acm`*. Use* `isDigit` *to test for a digit.*

```
f elem (digs, non)
    | isDigit elem = (elem:digs, non)
    | otherwise = (digs, elem:non)
```

Here's a Thanks! and a Bug Bounty Point to Mr. Garcia for being the first to point out that I'd specified the wrong type for the folding function, ending with `elem` instead of `acm`! What a whopper!!

Problem 8: (5 points) Mean and median: 2.6, 3.5

Implement `map` *in terms of a fold.*

Your solution must look like this*:* `map f list = fold...`

```
map f list = foldr (\e acm -> f e:acm) [] list
```

Problem 9: (2 points) Mean and median: 0.6, 0.5

Without using explicit recursion, write `last` *in point-free style. Hint: Tough, and easy to get backwards! Don't worry about handling empty lists.*

Note: Wording changing at start of exam to "Write `last` in terms of a fold: `last = fold...`"

```
last = foldl1 (\_ e -> e)
```

Without that last minute wording change the problem had a trivial solution, but worth a ½ point of extra credit:

```
last = head . reverse
```

Problem 10: (10 points) (one point each unless otherwise indicated) **Mean and median: 8.0, 8**

Answer the following questions. Keep your answers brief for questions 4-10—assume that the reader is a 372 classmate who just needs a quick reminder.

(1) Who founded the University of Arizona Computer Science department and when? (2 points)
Ralph Griswold, in 1971.

If you see this question again it might be worth zero points if correct but -2 points if wrong! Feel free to write this information on the back of your hand before future exams in 372.

(2) Name two programming languages created ***before 1990****.*
See intro slide 20, but the first two that come to mind for me are FORTRAN and COBOL.

(3) Name two programming languages created ***after 1989****.*
Lots on slide 20 but Java and Ruby are two I'd hope you know. Haskell was forming around this time, and counted as correct for this question. Another way to solve this would be "create" two languages on the spot, perhaps named after your goldfish. They'd be vaporware, but good enough!

(4) Name one language feature you would expect to find in a language that supports imperative programming.
Looping control structures, variables.

(5) *whm often says "In Haskell we never change anything; we only make new things." What's an example of that?*
A function like reverse builds a new list from the elements of an existing list. Another: A function to remove a value from a list builds a copy of the list, not including occurrence(s) of that value.

(6) *Things like cons lists, recursion, curried functions, and pattern matching on data structures are commonly used in functional programming but there's another capability that without which it's hard to do anything that resembles functional programming. What's that capability?*
The ability to treat functions as values.

(7) *What is relatively unique among programming languages about the way that Haskell handles strings and lists?*
Strings are simply lists of characters.

(8) *What is a "partial application"?*
A call to a multiple-argument function that's given fewer parameters than it needs.

(9) *What is "syntactic sugar"?*
A syntactic construct that makes a language more pleasant to use but that doesn't add a new capability.

(10) *In general, what's something we can represent with a tuple that we can't represent with a list?*
A heterogenous collection of values, like a string and an integer.

Note: I originally wrote this with nine questions, allowing two points for the first question. Then while "proofreading" I noticed I was a question short, and added a question, for a total of 11 points instead of 10 points. Mr. Srivastava caught this during the exam. Let's think of the extra point as possible extra credit—we'll count the exam as 100 points for purposes of average.

Extra Credit Section (½ point each unless otherwise noted) **Mean and median: 1.6, 1.5**

(1) *What is the type of* `foldr`*?*
I like to describe it as `foldr :: (val -> acm -> acm) -> acm -> [val] -> acm`

(2) *What is the type of the composition operator?*
`(b -> c) -> (a -> b) -> a -> c`

(3) *What's meant by "lexicographic comparison"? (Hint: Don't just say "dictionary order.")*
Pair-wise comparison of a sequence of values until a pair differs, with the difference deciding the ordering.

(4) *Name a programming language created at the University of Arizona.*
Intro slide 31 names a number of them but Icon is one I'd hope you know of. A presentation on Icon that Griswold gave at N. C. State led me to come to graduate school here to work on Icon. (I got on their radar here, ultimately getting a research assistantship, by being the first person to get Icon working on IBM mainframes.)

Lots of people cited SNOBOL. Griswold continued work on SNOBOL4 here at The University of Arizona but SNOBOL through SNOBOL4 were created at Bell Labs. There was also SL5 (SNOBOL Language 5) but Griswold ultimately was not happy with it. It was never released to the public although many papers about it were published while it was being developed.

Mr. Garcia used the approach suggested on Problem 10(3) with "Grant++. I made it just now." That's good for an Original Thought!

(5) *Haskell functions like* `getLine` *and* `putStr` *return an action. What does an action represent?*
An interaction with the outside world that happens when the action is evaluated.

(6) *What is the exact type of the list* `[head, tail]`*?*
`[head, tail]` is an invalid list, since the types differ.

(7) *If you only remember one thing about the Haskell segment of 372, what will it be? (Ok to be funny!)*
If time permits I'll issue a revised version with some of the highlights of these but I'm short on time at the moment!

(8) *With Ruby in mind, cite an example of an imperative method and an example of an applicative method.*
`reverse!` and `reverse` are the first two that came to mind for me.

(9) *Write a* ***Ruby*** *method* `printN` *that prints the numbers from 1 to N on a single line, separated by commas. (1 point)*

```
def printN n
   (1..n).to_a * ","
end
```

(10) *Predict the median score on this test. (The median is the "middle" value in a range of values.)*

I was wondering if *The Wisdom of Crowds* (see Wikipedia) would apply here but not so. These were the predictions.

```
100, 100, 88, 86, 84, 84, 83.2, 83, 83, 82, 81, 81, 81, 80, 79, 79, 78, 78,
76, 76, 75, 75, 75, 75, 74, 74, 72, 72, 72, 72, 71, 71, 70, 70, 70, 68, 65,
65, 60, 50, 50, 50
```

The mean of those predicted medians is 75.2. Any prediction at all earned the ½ point.

Here are the actual scores:

```
102, 102, 101.5, 100.5, 100.5, 100, 99, 97.5, 97, 95.5, 95.5, 95, 95, 94.5,
94, 93, 92.5, 92, 92, 91, 91, 91, 90.5, 90, 90, 90, 89, 88.5, 88, 87, 86,
83.5, 83, 82.5, 81, 80.5, 79.5, 79, 78.5, 78, 78, 78, 77.5, 76.5, 76.5, 76,
75, 74.5, 73.5, 69.5, 68, 66, 64.5, 59, 58, 52, 49, 45, 29.5
```

Simple statistics:

```
n = 59
mean = 82.7542
median = 87
```

For the record, here are the changes/updates accumulated during the exam and shown on the screens:

```
Exam done: 10:52

Prob 2: head, tail: assume not empty

Prob 6 (fml): Assume odd number of elements

Prob 7: foldr is elem -> acm -> acm (not elem!!)

Problem 9 change:
Write last in terms of a fold, just like problem 8, but for last

Problem 10 (8) (9) Ok to just cite an example.

New EC: Answer Problem 9 in a different way, just following the
instructions as written, which make it trivial! :(
```