Solutions for
CSC 372 Mid-term Exam
Thursday, March 12, 2015

A note about grading mistakes

I wish I was perfect but I'm not. I do make mistakes when grading. I encourage you to double-check my work and let me know if you find any mistakes or disagree with any of my grading decisions, including deduction amounts.

I do regrading on a problem by problem basis; a regrade request for a single problem does not trigger an audit of the full exam.

There's no deadline for reporting grading mistakes aside from the filing deadline for final grades.

Unless there are exceptional circumstances I require regrading requests to be in person, not via email or IM.

Before asking me to revisit any problem involving code, first type in your code and see what it does.

I hate to have to mention it but note that all completed exams were scanned into a PDF, making it easy to catch post-grading alteration/addition of answers. Don't be tempted to add a couple of characters to get a point or two back!

Problem 1: (15 points) mean = 10.18, median = 11.00

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

```
("len", "", "")
```
([Char], [Char], [Char])

```
(:)
```
```
a -> [a] -> [a]
```

```
[(1,'a'),('b',2)]
```
Invalid. The two tuples have different types.

```
map length
```
```
[[a]] -> [Int]
```

```
(+1) . (0<)
```
The composition is not valid. `(0<)` produces a Bool but `(+1)` requires a number.

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

```
(True, (False, "x"))
```
```
(Bool, (Bool, [Char]))
```

```
length . (\x -> [(x,x)]) . head . init . tail
```
Hint: the composition is valid.
```
[a] -> Int
```
If you wrote `... -> ... -> ... -> ... -> ...`, see slide 252.

```
[1..] < [2..]        (Remember: I want the type!)
      Bool
```

Problem 2: (12 points) (two points each) **mean = 9.92, median = 10.50**

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

Each instance of poor style or needlessly using other Prelude or helper functions will result in a deduction.

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

IGNORE empty list cases for `head`, `tail`, and `maximum`. There's no need to specify function types.

head

```
head (h:_) = h
```

tail

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

length

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

max *(Hint: `max 2 3` is 3)*

```
max x y
  | x > y = x
  | otherwise = x
```

maximum *(`maximum [5,2,3]` is 5. Ok to abbreviate as `mm`.)*

```
mm [x] = x
mm (x1:x2:xs)
  | x1 > x2 = mm (x1:xs)
  | otherwise = mm (x2:xs)
```

Another way. The explicit type specification avoids a "monomorphism restriction" error.

```
mm::Ord a => [a] -> a
mm = foldr1 max
```

A third: `mm [x] = x; mm (x:xs) = max x $ mm xs`

filter *(`filter odd [1,2,3]` is `[1,3]`. Ok to abbreviate as `flt`.)*

```
filter _ [] = []
filter f (x:xs)
    | f x = x : filter f xs
    | otherwise = filter f xs
```

Problem 3: (14 points) (7 points each) **recursive: mean = 6.05, median = 6.00; non-recursive: mean = 4.64, median = 6.50**

*For this problem you are to **write both recursive and non-recursive versions** of a function `tupruns` of type `[(Int, a)] -> [a]` that behaves like this:*

```
> tupruns [(3, 'a'), (2, 'b'), (4, 'c')]
"aaabbcccc"
```

```
> tupruns [(1,'#'),(0,'$')]
"#"

> tupruns  (zip [1..8] "abcdefgh")
"abbcccddddeeeeeffffffgggggggghhhhhhhh"
```

Solutions:

```
tupruns [] = ""
tupruns ((n,c):ts) = replicate n c ++ tupruns ts

tupruns = concat . map (uncurry replicate)
```

A number of non-recursive solutions looked like this:

```
tupruns list = foldr (\(n,ch) acm -> r n ch ++ acm) "" list
```

It'd be in point-free style if only `list` weren't present on both sides.

A common mistake was mapping or folding with a non-function value, like this:

```
... concat . map (replicate x y) ...
```

Problem 4: (3 points) <u>mean = 2.32, median = 3.00</u>

Write a function `flattups` that "flattens" a list of 3-tuples into a list of the values in those tuples.

```
> :t flattups
f :: [(a, a, a)] -> [a]

> flattups [(10,20,30),(3,4,5),(1,2,3)]
[10,20,30,3,4,5,1,2,3]
```

Solution:

```
flattups = foldr (\(a,b,c) acm -> a:b:c:acm) []
```

A number of students had not wrong but long answers due to writing three helper functions to extract the first, middle, and last element from three tuples.

Problem 5: (7 points) <u>mean = 4.18, median = 5.50</u>

The following function removes <u>consecutive duplicates</u> from a list:

```
dropConsecDups list = foldr ff [] list
```

Usage:

```
> dropConsecDups [3,3,1,5,7,5,5,7,7,7]
[3,1,5,7,5,7]

> dropConsecDups "a looooooooooooonnnnnnnnnng one!"
"a long one!"
```

<u>For this problem you are to write a folding function `ff` that will work with `dropConsecDups` as shown above.</u>

```
ff val [] = [val]
ff val all@(a:_)
    | val == a = all
    | otherwise = val:all
```

Problem 6: (6 points) mean = 4.44, median = 5.50

Write a Haskell program that reads a file named on the command line and prints the mean (average) length of the lines in the file.

```
avglen bytes = (show $ totalChars / numLines) ++ "\n"
    where
        lineLengths = map length $ lines bytes
        totalChars = fromIntegral $ sum lineLengths
        numLines = fromIntegral $ length lineLengths
```

The version above breaks the input into a list of `lines`. I wondered how much faster it would be to just count newlines by filtering on them. The version below takes that approach.

```
avglen bytes = (show $ totalChars / numLines) ++ "\n"
   where
      numLines = fromIntegral $ length $ filter (=='\n') bytes
      totalChars = (fromIntegral $ length bytes) - numLines
```

I then tried simply counting newlines:

```
avglen bytes = (show $ totalChars / numLines) ++ "\n"
    where
        numLines = fromIntegral $ count '\n' bytes
        totalChars = (fromIntegral $ length bytes) - numLines
        count c s = foldr
          (\elem acm -> if elem == c then acm + 1 else acm) 0 s
```

The times for the three surprised me:

Use `lines` to break `bytes` into lines:	0.817s
`length $ filter (=='\n') bytes`:	0.644s
Count with `foldr`:	1.160s

These solutions handle the integer division properly but that was not required of student solutions.

Very few students remembered to append a newline to the output. Those who did got a ¼-point bonus.

I wish I'd mentioned in the write-up that the value `bytes` produced by `readFile` is a string like `"xxx\ny\nzz\n"` but some assumed it was a list of lines. I also wish I'd reminded about the `lines` function in the Prelude; some students wrote their own version, and that wasn't my intention.

Problem 7: (7 points) mean = 6.38, median = 6.50

Write a Ruby version of the Haskell program in the previous problem. The Ruby version reads from standard input rather than opening a file specified on the command line. ... Unlike the Haskell version this Ruby version must properly handle the math to get a `Float` result, not a truncated value like `3/2 == 1`.

```
sum = n = 0
while line = gets
    line.chomp!
    sum += line.size
    n += 1
end

printf("%.1f\n", sum / n.to_f)
```

Several students approached the problem by building an array with line lengths and then summing the lengths. That seems like a Haskell solution written in Ruby!

The only common mistake was to not `chomp` or otherwise account for the line terminator(s) that `gets` leaves in place.

Problem 8: (7 points) mean = 5.51, median = 6.00

Write a Ruby program `nwords.rb` that reads lines from standard input and writes the first N words on each line to standard output. N is specified by a `-N` command line argument that is assumed to be present. If N is larger than the number of words on a line, all the words on that line are output.

```
n = -ARGV[0].to_i

while line = STDIN.gets
    line.chomp!
    puts line.split[0,n] * " "
end
```

A common assumption was an analog to Haskell's `concat` to join an array of strings together into a string but `concat` wouldn't put spaces between them. Something like `array * sep` (as shown above), or a loop was needed. Assuming something like Haskell's `unwords` was reasonable, if explicitly stated.

A common implied assumption was that `puts array[0,n]` would output blank-separated words but it actually outputs one array element per line.

A number of students assumed a single-digit N even though there is a `-100` example.

Lots of students wrote mysterious code to process multiple arguments and/or error checking code. None of that resulted in deductions but I imagine it ate up a lot of time. Avoid that on the final by taking advantage of what can be assumed.

Problem 9: (6 points) mean = 4.07, median = 5.00

Write Ruby method `chunkstr(s, n)` that returns an array of consecutive `n`-long substrings of the string `s`. Partial substrings are not included. Assume `n > 0`. `chunkstr` does not change `s`! (Maybe use `String#dup`.)

```
>> chunkstr("abcdef",3)
=> ["abc", "def"]
```

Here was my first solution. I forgot the return on the first run but it worked on the second run.

```
def chunkstr(s,n)
    r = []
    s = s.dup
    while s.size >= n
        r << s[0,n]
        s[0,n] = ""
    end
    r
end
```

I then wrote a more numeric solution:

```
def chunkstr(s,n)
    r = []
    pos = 0
    while pos + n <= s.size
        r << s[pos,n]
        pos += n
    end
    r
end
```

It took three runs to get right: (1) forgot `pos += n`, (2) used `s[0,n]`, (3) used `s.size + 1`. I might have also forgotten to return `r` on the first run if I'd started from scratch instead of hacking up my first solution.

Lots of student solutions did something like

```
a = []; apos = 0
...
a[apos] = str
apos += 1
```

but `a << str` or `a.push(str)` is simpler.

Problem 10: (6 points) (one point each unless otherwise indicated) **mean = 2.70, median = 2.50**

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) -> Int -> Int
```

It seems that only TODO students know that the type operator `->` is right associative.

```
(Int -> Int) -> (Int -> Int)
```

Expect to see a question like this on the final.

(2) Fully parenthesize the following expression.

```
f  g  a  +  x  y  z  +  f1  (a, b)  c
```

Quite a few students got tripped up on this one but only two rules come into play: (1) two expressions side-by-side is function call. (2) function call has higher precedence than any operator.

```
((f  g) a) + ((x y) z) + ((f1  (a, b))  c)
```

Expect to see a question like this on the final!

(3) Rewrite the following expression to use as few parentheses as possible.

```
len ((map f) (head (lines bs)))
```

Solution: `len $ map f $ head $ lines bs`

(4) Describe in English the data structure matched by this pattern: `[t:h]`
A list containing a single non-empty list.

A number of students got tripped by a head named `t` and a tail named `h` but those names mean no more to Haskell than do `ping` and `pong`!

(5) Consider the following definitions for a function that tests whether a list is empty. Each is shown with the types that Haskell infers for them:

```
empty1 :: [t] -> Bool
empty1 [] = True
empty1 _  = False

empty2 :: Eq a => [a] -> Bool
empty2 x
```

```
    | x == [] = True
    | otherwise = False
```

What's causing the type of the functions to differ? What's an example of a list that would work with `empty1` *but not* `empty2`*? (Two points.)*

The use of the `==` operator in `empty2` produces the added requirement that the type `a` must be in the `Eq` type class. `empty1 [length]` works but `empty2 [length]` produces a type error because function types aren't members of the `Eq` type class.

I'd initially planned each of the two questions to be worth a point each but few students got the second question right. I decided to score the first question as two points and the second question as a one-point bonus question.

An answer that was as little as "`==`" earned full credit on this problem.

Problem 11: (7 points) (one point each unless otherwise indicated) **mean = 3.07, median = 3.00**

The following questions and problems are related to Ruby.

(1) *Write a Ruby iterator* `itr` *that behaves like this:*

```
>> itr(3) {|x| puts x}
6
```

Solution:

```
def itr x
    yield x * 2
end
```

(2) *In the Ruby code* `$x = N * 5`*, we know that* `$x` *is a* global variable *and* `N` *is a* constant.

(3) *Given* `h = {}`*, write an expression such that after it is evaluated,* `h["x"]` *is* `10`*.*

```
h["x"] = 10
```

(4) *What's a big mistake in the following statement?*
"The `if-else`*,* `while`*, and* `for` *statements are examples of Ruby control structures."*

`if-else`, `while`, and `for` are expressions—they all produce a value!

(5) *A method named* `show_match` *is used extensively on the regular expression slides. Briefly, what does it do?*

See slide 192!

`show_match` is in my `.irbrc`. To get a copy of my `.irbrc`, do the following `cp`, assuming you're in your `a4` directory:

```
$ cp a4/.irbrc ~
```

After copying it, browse it with `less ~/.irbrc` and see if you learn anything new.

(6) *What are the minimum and maximum lengths of strings that can be matched by the following regular expression? (Be careful!)*

```
[Aa]..[Zz]0x9?
```

Six or seven characters are matched: 'A' or 'a', then any two characters, then 'Z' or 'z', '0', 'x', and maybe a '9'.

(7) Write a Ruby method that exemplifies duck typing. There's no need for any explanation!

I like the `double` example on slide 126 but most any method that invokes a method on an argument or applies an operator to an argument provides an example of duck typing.

Problem 12: (10 points) (one point each unless otherwise indicated) **mean = 6.48, median = 7.00**

Answer the following general questions. Keep your answers brief—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?

Ralph Griswold, in 1971. There was no partial credit on this one—you had get the year right, too. Here are two ways to remember 1971: (1) The decimal value for ASCII 'G' is 71. (2) There are 7+1 letters in "Griswold". Note that it's Griswold, not Griswald. (Remember the 'O's in SNOBOL.)

Good news: you'll have one more chance to get this right!

(2) Name a language that was created before Ruby. Name a language that was created after Ruby.

Popular pre-Ruby examples were FORTRAN, COBOL, C, and Icon. Common post-Ruby examples were Java and Swift. Python was a common wrong answer for a language that came after Ruby.

Two students created a language on the spot.

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

Learn(ing) How to Learn a Language

*(4) 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.*

See Haskell slide 3 and following.

(5) Perhaps the most fundamental characteristic of a functional programming language is that it permits higher-order functions to be written. What's the language feature that's required in order to write a higher-order function?

The ability to pass a function to a function as a parameter.

(6) What's an important difference between an imperative method and an applicative method? (Hint: it doesn't concern the method name!)

An imperative method changes state in an object. An applicative method produces a new object.

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

To reverse a list we'd make a new list.

(8) In general, every expression in a programming language has three aspects. What are those three? Hint: one of them starts with a "v". (1.5 points)

Type, value, side-effects

(9) *Briefly define the term "programming language". (1.5 points)*

Here are some of the answers I especially liked:

"A language that is used to communicate with a computer."—J. Naranjo

"A group of syntactic and semantic rules to tell a computer how to process input."—R. Storck

"A grammar and vocabulary for providing instruction in math and logic."—B. Whitely

"A collection of syntax and words which can be read by the machine."—J. Wolfe

"A compilation of syntax and semantics that supports a variety of computations."—A. Zarbock

Extra Credit Section (½ point each unless otherwise noted) <u>mean = 2.03, median = 2.00</u>

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

All predictions:

```
90, 88, 86, 85, 84, 84, 84, 83, 83, 83, 83, 83, 83, 82, 82, 82, 82, 82, 82, 82,
82, 82, 82, 81, 81, 81, 81, 81, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 78, 78,
78, 77, 77, 76, 76, 76, 76, 75, 75, 75, 74, 72, 72, 70, 70, 67, 65, 64

n = 58
mean = 79.22
median = 80
```

(2) *Add a dunsel to the following function definition*

```
f x = take 3 x ++ drop 3 x
```

Two common ones were `++ []` and `++ take 0 x`

(3) *Why was `\` chosen for use in Haskell's lamba abstraction syntax?*

It resembles λ.

(4) *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

(5) *What's the basic idea of whm's so-called "O(1) navigation"?*

Being able to hit a page with a short, fixed set of keystrokes.

(6) *What is the exact type of the list `[head, tail]`?*

The expressions is invalid—`head` and `tail` have different types.

(7) *What's interesting about the ASCII code sequence from 60 through 62?*

They form the "spaceship" operator: `<=>`

(8) What Ralph say when a young and eager graduate student put forth a list of potential new features for Icon?

Roughly quoting, "Add all the features you want but for every feature you want to add, first find one to remove."

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

"The compiler is always right but it will never tell you why."—B. Streeter

"Jeremiah going over my head in six words."—S. Stephens

"Haskell is really spelled 'Haskhell'."—J. Scarim

"Higher-order functions are the bees knees."—C. Macdonald

"Pancakes"—A. Rane

"Typing errors, typing errors everywhere."—J. Naranjo

"Watch the types!!!"—J. Knott

"Pancakes are evil!"—K. Enami

"It's a (fun)ctional language"—S. Desai

(10) What is Matz' full name?

Yukihiro Matsumoto

(11) Write a good extra credit question and answer it.

Lots of good ones but I'm out of time to cite them here! :(

Statistics and an adjustment

Here are all scores, in order:

```
103.75, 101.25, 97.25, 97.25, 95.75, 94.00, 92.25, 92.00, 91.00, 90.50, 89.75,
88.75, 88.25, 88.25, 88.25, 87.00, 86.25, 83.00, 82.75, 82.50, 82.25, 81.75,
80.75, 80.50, 79.75, 78.25, 78.25, 78.25, 78.00, 77.75, 76.75, 76.25, 76.00,
75.25, 75.00, 74.50, 74.00, 73.50, 72.00, 71.25, 70.25, 69.50, 67.75, 67.25,
66.75, 66.75, 65.75, 65.50, 65.00, 64.50, 63.75, 63.75, 63.00, 61.00, 60.00,
59.75, 58.75, 57.75, 57.75, 57.50, 56.75, 56.25, 54.75, 54.00, 53.50, 53.25,
50.75, 50.50, 40.50, 40.25, 39.00, 36.25, 35.75

n = 73
mean = 71.9623
median = 74
```

The scores ran a little lower than I would have liked. After taking all factors into consideration **I've decided to add six points to all scores**. The score you'll see on D2L will be six points higher than the score written on the cover page of your exam.