# CSC 372 Mid-term Exam Solutions
# Monday, March 19, 2018

**Problem 1: (5 points)** (one point each) (mean: 3.72, median: 4, 3rd quartile: 4)

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

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

```
'a'
     Char


("a", ['a'])
      ([Char], [Char])


snd
     (a, b) -> b

map snd
     [(a, b)] -> [b]


fold . length (3 + 4)
```

*`snd` (Reminder: `snd` produces the second element of a 2-tuple.)*

Error! `length` wants a list (actually a `Foldable`).

**Problem 2: (3 points)** (mean: 1.7, median: 2, 3rd quartile: 2)

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 an explicit type declaration. (e.g. `f1::...`)

```
f1 :: (a,b) -> (b,a,b)
     f1 (a,b) = (b,a,b)

f2 :: [a] -> [b] -> (b,a)
     f2 [a] [b] = (b,a)

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

**Problem 3: (7 points, as indicated)** (mean: 4.93, median: 5.5, 3rd quartile: 6.5)

*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>***.

*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) [1 point]
     snd (_,b) = b

not            [1 point]
     not False = True
     not   _   = False

at list n      (Like list!!n.  Assume n is in-bounds.) [2 points]
     at (h:_) 0 = h
     at (_:t) n = at t (n-1)

map            [2 points]
     map _ [] = []
     map f (h:t) = f h : map f t

flip           (Reminder: flip take "abc" 2 == "ab") [1 point]
     flip f x y = f y x
```

**Problem 4: (12 points)** (6 points each) (recursive—mean: 5.55, median: 6, 3rd quartile: 6)
(non-recursive—mean: 4.82, median: 6, 3rd quartile: 6)

*For this problem you are to* ***<u>write both recursive and non-recursive versions</u>*** *of a Haskell function* `sumexcl nums exclude`*, having type* `[Int] -> Int -> Int`*, that returns the sum of the values in* `nums`*, excluding all occurrences of the value* `exclude`*.*

```
> sumexcl [1,10,2,3,10] 10
6
```

Recursive

```
sumexcl :: [Int] -> Int -> Int
sumexcl [] _ = 0
sumexcl (x:xs) e
    | x == e = sumexcl xs e
    | otherwise = x + sumexcl xs e
```

Non-recursive:

```
sumexcl :: [Int] -> Int -> Int
sumexcl list e = sum $ filter (e /=) list
```

**Problem 5: (6 points)** (mean: 5.40, median: 6, 3rd quartile: 6)

*Certain conditional code in Haskell can be written in three different ways:*

- *with* `if-else`
- *with guards*
- *with patterns*

*Imagine a function* `f` *that returns* `'a'` *if called with* `1` *or* `2` *but if called with any other number, it returns* `'b'`*.*

*<u>For this problem you are to write three versions of f</u>: one using* `if-else`*, one using guards, and one using patterns.*

With `if-else`:
```
    f x = if x == 1 || x == 2 then 'a' else 'b'
```

With guards:
```
    f2 x
        | x == 1 || x == 2 = 'a'
        | otherwise = 'b'
```

With patterns:
```
    f3 1 = 'a'
    f3 2 = 'a'
    f3 _ = 'b'
```

**Problem 6: (6 points)** (mean: 4.18, median: 5, 3rd quartile: 5.5)

*The Haskell function `evenwords s` returns a string containing only the even-length words in the string `s`. Here is an incomplete solution for `evenwords`:*

```
        evenwords s = unwords (foldr ff [] (words s))
```

*What's missing is the folding function `ff`. Your task for this problem is to write the function `ff`.*

```
        ff w acm
            | even $ length w = w : acm
            | otherwise = acm
```

**Problem 7: (18 points)** (7 and 11 points, respectively) (recursive—mean: 6.19, median: 6.5, 3rd quartile: 7)
(non-recursive—mean: 5.21, median: 5, 3rd quartile: 10.5)

*For this problem you are to* ***write both recursive and non-recursive versions*** *of a function `deo` having type `[Int] -> [Int]`, that doubles every other number in a list.*

Recursive:
```
    deo [] = []
    deo [x] = [x]
    deo (x1:x2:xs) = x1:(x2 * 2):deo xs
```

This is the non-recursive solution I initially wrote:

```
    deo list = result
        where
            double = cycle [False,True]
            result = map f $ zip double list
            f (True,x) = x * 2
            f (_,x)    = x
```

But Mr. McClean showed me the right way to do it:

```
    deo_mc nums = zipWith (*) nums $ cycle [1,2]
```

**Problem 8: (5 points)** (one point each unless otherwise indicated) (mean: 3.54, median: 3.5, 3rd quartile: 4.5)

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

*(1) In Haskell, as in many functional languages, the list type is a "cons list". Using the style shown in the slides, draw a diagram of the list `[10,20,30]`.*

I'm lazy! See `b` on slide 140.

*(2) What's wrong with the following statement regarding a Haskell function named `f`?*
*"The type of `f` is `(Int, Char)`."*

If `f` is a function, there should be a `->` in the type.

*(3) What is the essential idea of lazy evaluation?*

Expressions are not evaluated until their value is needed.

*(4) In Haskell, what's the difference between a <u>type</u> and a <u>type class</u>?*

A type is a set of values. A type class is a set of types.

*(5) If a Haskell function requires three parameters but is called with only two parameters, what happens?*

A partial application is produced.

**Problem 9: (15 points)** (mean: 12.23, median: 14.5, 3rd quartile: 15)

*Write a Ruby program that reads lines from standard input and upon end of file writes the longest line and its line number to standard output. If there are ties for the longest line, the program writes out all the lines that tie. If there is no input, the program produces no output.*

```
longest = -1
num = 1
result = []

while line = gets do
    line.chomp!
    len = line.size
    if len > longest then
        result = [[num,line]]
        longest = len
    elsif len == longest then
        result << [num,line]
    end
    num += 1
end

for num, line in result do
    puts "#{num}: '#{line}'"
end
```

I have to admit there was some fantasy in the assignment 5 solutions. In the assignment 5 write-up I mentioned that I'd used `longest.rb` a number of times but over time enough students had become unduly frustrated with it that I decided to make it a zero-pointer this semester. Nobody turned in a solution for it this semester, but nonetheless I recycled the detailed story about student encounters from the Spring 2016 solutions since I felt it had some good lessons. But since nobody turned in a solution for it, I felt ok about putting a simple version of it on the exam.

Quite a few exam solutions stored the full input in an array, noting the length of the longest line in the process, and then making a second pass over the array, printing lines of that length. Memory is really cheap these days but I think there's still merit in algorithms that are conservative with memory.

Here's a slightly different version of the main loop, based on several solutions:

```
while line = gets do
    line.chomp!
    len = line.size
    if len > longest then
        result = []
        longest = len
    end
    if len == longest then
        result << [num,line]
    end
    num += 1
end
```

**<u>BEFORE READING FURTHER</u>**, see if you can find the bug in the loop above.

At first glance I thought that the code above was buggy but in fact it works just fine. It took me a second look to realize that the first `if` sets `longest` to len, which makes the second `if`'s condition be true. Which of the two forms do you prefer?

**Problem 10: (9 points)** (mean: 6.8, median: 8, 3rd quartile: 8.5)

*Write a Ruby program whose command-line arguments are pairs of counts and strings. For each pair, the string is replicated by the associated count and written to standard output. All strings are output on a single line with no separation, but that line does end with a newline.*

*If run with no arguments or an odd number of arguments, the program prints a question mark on a line by itself.* <u>*Assume counts are always non-negative integers.*</u>

This was my first solution:

```
a = ARGV

if a.empty? || a.size.odd? then
    puts "?"
    exit
end

while !a.empty? do
    n, s = a[0,2]
    print s * n.to_i
    a.slice!(0,2) # Why not: drop!(2) would be nice here.
end
```

```
        puts
```

I generally avoid indexing when I can, but in the process of grading this problem I came to like better a loop that uses indexing:

```
        i = 0
        while a[i]
            n, s = a[i,2]
            print s * n.to_i
            i += 2
        end
```

However, `Enumerable.each_slice` lets us replace the loop with a one-liner:

```
        a.each_slice(2) {|n,s| print s * n.to_i}
```

**Problem 11: (5 points)** (one point each unless otherwise indicated) (mean: 2.75, median: 3, 3rd quartile: 3)

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

(1) *In class it's been said that some Ruby methods are applicative and others are imperative. What's the difference between the two?*

Applicative methods never modify their receiver but imperative methods <u>might</u> in some cases.

(2) *What is the value of the following Ruby expression?* `""[0] || [""][0]`

"" *(empty string)*

(3) *What is a major way in which Ruby arrays differ from Java arrays? (OR, What is a major way in which Ruby arrays differ from Java lists?)*

Ruby arrays differ from Java arrays in many ways. Ruby arrays...

- Are heterogeneous
- Allow negative indexing
- Can be concatenated
- Can have elements inserted or deleted
- And more!

A fundamental difference between Ruby arrays and Java's `List` hierarchy is that Ruby arrays can be accessed with symbolic syntax like `a[n]`, `a[start,len]` and more.

(4) *What is the value of the following Ruby expression?* `[1,2,3,3] <=> [1,2,3,5]`

**<u>(a)</u>** **<u>`-1`</u>**
(b) `0`
(c) `1`
(d) `2`

(5) *With duck typing in mind, what are two things that must be true for* `x[y]` *to be a valid expression?*

`x` must have a `.[]` method for which `y` is a valid argument.

As mentioned at the start of the exam, this question was out of bounds wrt. the material I said the exam would cover. To compensate for that, the score written on the front of your exam is your actual score plus one point. If you went ahead and correctly answered this question, you got a point for that, too.

**Problem 12: (9 points)** (one point each unless otherwise indicated) (mean: 4.63, median: 5, 3rd quartile: 6.5)

*Briefly answer the following general questions.*

(1) *Who founded The University of Arizona's Computer Science department and when? (Hint: the ASCII code for 'D' is 68.)*

One student wrote, "1791". Dr. Ralph Griswold was certainly a pioneer but he didn't found our department until 1971.

For years I had trouble remembering whether it was 1971 or 1972 but when I learned the ASCII codes I noticed that the code for 'G', like in **G**riswold, was 71, and I never had trouble with that again.

I hinted that 'D' was 68 to help those who might have remembered the connection to ASCII from intro slide 39 but didn't know the code for any uppercase letter.

(2) *Circle the best answer:*
*Ralph Griswold said that if you're going to design a language it should be...*
***(a) A language you want to use***
*(b) A language that is easy to use*
*(c) A language that many people will want to use*
*(d) A language that professional developers will want to use*

(3) *In general, which is larger and why, the mental footprint required to be able to write code in a language or the mental footprint required to be able to read code in a language?*

As discussed in class, the reading footprint is bigger because while you can solve any problem using only a small, *Turing complete* subset of a language, to read an arbitrary program, you might need to any or all parts of the language.

(4) *In class it was said that some language designers have an attitude of "Why?" and others have an attitude of "Why not?" What's the difference in those two attitudes? Support your answer with a brief example.*

I felt Mr. Chipman said it best:
"Why?—you need a good reason to add a feature.
"Why not?—you need a good reason to not add a feature.

We've seen lots of examples of how Matz seems to tend toward "Why not?" with Ruby.

(5) *As it relates to this class, what's a dunsel?*

The answer to this can be found in any of the assignment solutions that have been distributed.

(6) *Specifications for programming languages generally focus on two areas, syntax and semantics. With respect to programming languages what does "semantics" mean?*

The meaning of a sequence of symbols.

(7) We've talked about three aspects of expressions: value, type, and side effect. For each of the three, write a

sentence or two that describes the concept (i.e, value, type, or side effect), being sure to cite an example. (1 point each)

Extra Credit Section (½ point each unless otherwise noted) (mean: 0.67, median: 0.25, 3rd quartile: 1)

(1) *What's the value of the following Ruby expression?* `[1,2,3].each { puts 1 }`
`[1,2,3]`

(2) *In Ruby, what is the output produced by* `"abc".each { |s| puts s[0].size.class }`*?*
It's an error. The `String` class doesn't have an `each` method, to my dismay.

(3) Write a Haskell function with the following type: `(a,b,a) -> (b,a,b)`

`f (a,b,a) = (b,a,b)` was a common answer but you'll need to wait for Prolog if you want to name a parameter more than once in the argument list. Meanwhile, you can do this:

```
  f (a,b,_) = (b,a,b)
  f (_,b,a) = (b,a,b)
```

(4) *What is the name of the Ruby operation performed by* `[1,2,3] * "x"`*?*
Intercalation

(5) *Fill in the blanks below to produce a Haskell expression whose value is a function.*
__take__ $ __5__

(6) *In* `ghci`*, what's something* `:info` *will show me that* `:type` *won't?*
For one thing, `:info +` will show the precedence of that operator.

(7) *What's wrong with the following statement? "Java is an interpreted language."*
Slide 76: "Interpretation and compilation are attributes of an implementation of a language, not the language itself!"

(8) *Fill in the blanks: Cells in a cons list comprise a/an* __head__ *and a/an* __tail__.

(9) *As a Ruby programmer, tell me what MRI is.*
Matz Ruby Implementation

(10) *Honor system: If you've created at least two Chrome (or Firefox, etc.) search engines, as described in* `cs.arizona.edu/~whm/o1nav.pdf` *and suggested on on Ruby slide 9, what are their keywords? (i.e., What do you type in the address bar to invoke them?)*

I was disappointed that only six students earned this easy half-point. I wonder if the class will fare any better on the final exam.

Here are all scores, in descending order:

```
100.25, 98.75, 95, 92.25, 90.5, 90, 89.5, 88.75, 88, 85.75, 83.5, 83, 82.5,
82.5, 81.5, 80, 79.75, 79, 79, 79, 78.75, 78.25, 78, 77.25, 76, 75, 74.5, 73.5,
73, 71.75, 71.25, 70.5, 67.25, 64.75, 64.25, 63.5, 62, 60, 59.5, 58.5, 58.5, 57,
56, 55, 53.5, 52, 49.25, 49, 41.5, 39.5
```

To see your rank, put the lines above in a file `x` and do this: `tr , \\n < x | cat -n`

N = 50, mean = 72.7, median = 75.5, 3rd quartile: 82.5