Last name

____

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

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.

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

The last five minutes of the exam is a "seatbelts required" period, 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.

You are allowed no reference materials whatsoever.

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 arugments, state the assumption and proceed.

Feel free to use abbreviations, like "otw" for "otherwise", "lt" for "longerThan". **Use `S` as a synonym for `[Char]`. Use `I` for `Int`**.

Don't make problems hard 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."

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

It is better to put forth a solution that violates stated restrictions than to leave it blank—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 **put your initials in the lower right hand corner of the top side of each sheet, checking to be sure you have all seven sheets.**

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

Problem 1: (15 points)

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'

("len", "", "")

(:)

[(1,'a'),('b',2)]

map length

(+1) . (0<)

head

(True, (False, "x"))

length . (\x -> [(x,x)]) . head . init . tail
```

<u>Hint: the composition is valid</u>.

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

Problem 2: (12 points) (two points each)

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

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

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

`head`

`tail`

`length`

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

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

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

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

Problem 3: (14 points) (7 points each)

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")
"abbcccddddeeeeefffffggggggghhhhhhhh"
```

Just like on assignment 2, the recursive version must not use any higher-order functions. Just like on assignment 3, write the non-recursive version imagining that you just don't know to how to write a recursive function.

For one bonus point each on the non-recursive version:
(1) Use point-free style.
(2) Do not use any helper functions or anonymous functions.

`concat` and `replicate` may be useful:

```
> concat ["a", "...", "c"]
"a...c"

> replicate 3 5
[5,5,5]
```

Problem 4: (3 points)

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]


> flattups [('H','i','!')]
"Hi!"
```

There are no restrictions on this problem. You may call it just "`ft`".

Problem 5: (7 points)

The following function removes consecutive duplicates 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!"
```

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

Problem 6: (6 points)

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.

Usage:

```
$ cat avglen.1
xxx
y
zz

$ runghc avglen.hs avglen.1
2.0
```

Note the computation: *`(3+1+2)/3 = 2` characters per line*

DO NOT WORRY about integer/float division issues. **Assume** that `3/2` produces `1.5`!

Assume there's at least one line in the file.

Be sure that a newline follows the value that's output. Recall that `show 3.2` is `"3.2"`. Here's a main program, similar to the ones supplied for `group.hs` and `avg.hs`. Your job is to write `avglen`.

```
main =
    do
       args <- getArgs
       bytes <- readFile (head args)
       putStr (avglen bytes)
```

Problem 7: (7 points)

Now it's time for some Ruby problems.

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:

```
$ cat avglen.1
xxx
y
zz

$ ruby avglen.rb < avglen.1
2.0
```

Use `printf("%.1f\n",  ...)` to produce the final output.

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`.

Problem 8: (7 points)

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.

```
$ cat nwords.1
a few words to
test the
operation
of nwords.rb on this exam.

$ ruby nwords.rb -3 < nwords.1
a few words
test the
operation
of nwords.rb on

$ cat nwords.2
aa bb cc dd ee ff gg
hh ii jj
kk
ll mm nn oo pp

$ ruby nwords.rb -2 < nwords.2
aa bb
hh ii
kk
ll mm

$ ruby nwords.rb -100 < nwords.2
aa bb cc dd ee ff gg
hh ii jj
kk
ll mm nn oo pp
```

Note the following behavior for *array*`[n, m]`:

```
>> w = "a b c".split
=> ["a", "b", "c"]

>> w[0,2]
=> ["a", "b"]

>> w[0,10]
=> ["a", "b", "c"]
```

SPACE FOR SOLUTION ON NEXT PAGE

(space for solution for `nwords.rb`)

For reference:

```
$ cat nwords.2
aa bb cc dd ee ff gg
hh ii jj
kk
ll mm nn oo pp

$ ruby nwords.rb -2 < nwords.2
aa bb
hh ii
kk
ll mm

$ irb
>> w = "a b c".split      => ["a", "b", "c"]

>> w[0,2]                 => ["a", "b"]

>> w[0,10]                => ["a", "b", "c"]
```

Problem 9: (6 points)

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"]

>> chunkstr("abcdef",1)
=> ["a", "b", "c", "d", "e", "f"]

>> chunkstr("abcdef",5)
=> ["abcde"]
```

Problem 10: (6 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) -> Int -> Int
```

(2) Fully parenthesize the following expression.

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

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

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

(4) Describe in English the data structure matched by this pattern: `[t:h]`

(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.)

Problem 11: (7 points) (one point each unless otherwise indicated)

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

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

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

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

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

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

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

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

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

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

Problem 12: (10 points) (one point each unless otherwise indicated)

Answer the following general questions. Keep your answers brief for questions—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) Name a language that was created before Ruby. Name a language that was created after Ruby.

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

(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.

(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?

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

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

(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)

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

Extra Credit Section (½ point each unless otherwise noted)

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

(2) Add a dunsel to the following function definition

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

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

(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."

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

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

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

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

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

(10) What is Matz' full name?

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