Last name, NetID

________________________________

# CSC 372 Haskell Mid-term Exam
Feburary 28, 2014

**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 and NetID in the box above.

<u>This is a 45-minute exam</u> 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.

<u>You are allowed no reference materials whatsoever.</u>

If you have a question, raise your hand. We 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".

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

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

**<u>BE SURE to enter your NetID on the sign-out log when turning in your completed exam.</u>**

**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"

(1,'x',[True])

["x":[]]

head

not . not . isLetter

map not

(+1) . (0<)

\x -> x + 1

[1,'2',"3"]

isDigit . chr . head . (:[]) . (+3) . ord . chr
```

Hint: the composition is valid. Here are some types:

```
ord :: Char -> Int
chr :: Int -> Char
isDigit :: Char -> Bool
```

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

**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.

`head`

`tail`

`length`

`sum`

`last` (Return `error "empty"` if the list is empty.)

`snd` (Returns second element of a 2-tuple)

**Problem 3: (10 points)**

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.

```
> prswap [1..6]
[2,1,4,3,6,5]

> prswap "abcd"
"badc"

> prswap [False,True,True,False]
[True,False,False,True]

> prswap []
[]

> :t prswap
prswap :: [a] -> [a]
```

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

**Problem 4: (10 points)**

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.

```
> rotabc "abc"
"bca"

> rotabc "test"
"test"

> rotabc "aaaaa"
"bbbbb"

> rotabc "ababab"
"bcbcbc"

> rotabc "cabinet"
"abcinet"

> :t rotabc
rotabc :: [Char] -> [Char]
```

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.

This is essentially like `xlt  "abc"  "bca"` with `editstr` from assignment 1 but don't be tempted to use that!

Hint: My first "solution" for this one got a non-exhaustive match error on one of the tests.

**Problem 5: (20 points)** (ten points each)

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 10 [(9,3),(3,4),(10,20)]
[(9,3),(10,20)]

> bigTuples 25 [(9,3),(3,4),(10,20)]
[(10,20)]

> bigTuples 100 [(9,3),(3,4),(10,20)]
[]

> bigTuples 1 []
[]

> take 5 $ bigTuples 1000 $ zip [1..] [1..]
[(501,501),(502,502),(503,503),(504,504),(505,505)]

> :t bigTuples
bigTuples :: (Num t, Ord t) => t -> [(t, t)] -> [(t, t)]
```

**REMEMBER**: You've got to write **two versions** of `bigTuples`!

**Problem 6: (6 points)** (5 for function, 1 for type)

Write a function `fml list` that returns a 3-tuple with the first, middle and last elements of a list.

```
> fml [1,2,3,4,5]
(1,3,5)

> fml [10]
(10,10,10)

> fml [1..101]
(1,51,101)

> fml "Haskell"
('H','k','l')
```

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`?

You didn't forget to state the type of `fml`, did you?!

**Problem 7: (10 points)**

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`. Use `isDigit` to test for a digit.

**Problem 8: (5 points)**

Implement `map` in terms of a fold.

**Your solution must look like this**:

```
map f list = fold...
```

Note that the ellipsis starts right after the "d"—you'll need to decide which of `foldl1`, `foldr1`, `foldl`, or `foldr` you need to use!

It's ok to use an anonymous function but that is not required.

**Problem 9: (2 points)**

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

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

(1) Who founded the University of Arizona Computer Science department and when? (2 points)

(2) Name two programming languages created **before 1990**.

(3) Name two programming languages created **after 1989**.

(4) Name one language feature you would expect to find in a language that supports imperative programming.

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

(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?

(7) What is relatively unique among programming languages about the way that Haskell handles strings and lists?

(8) What is a "partial application"?

(9) What is "syntactic sugar"?

(10) What's something we can represent with a tuple that we can't represent with a list?

**Extra Credit Section (½ point each unless otherwise noted)**

(1) What is the type of the composition operator?

(2) What's meant by "lexicographic comparison"?

(3) Name a programming language created at the University of Arizona.

(4) Haskell functions like `getLine` and `putStr` return an action. What does an action represent?

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

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

(7) With Ruby in mind, cite an example of an imperative method and an example of an applicative method.

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