Your last name	Last name of person sitting beside you
______________________________	______________________________

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

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. 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 arguments, state the assumption and proceed.

Feel free to use abbreviations, like "`otw`" for "`otherwise`".

It's fine to use helper functions unless they are specifically prohibited or a specific form for the function is specified.

Don't make a programming problem hard by assuming that it needs 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: (5 points) (one point each)

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.

Important: Remember that the type of a function includes the type of the arguments as well as the type of the value returned.

```
'a'
```

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

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

```
map snd
```

```
fold . length (3 + 4)
```

Problem 2: (3 points)

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)
```

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

`f3 :: (a -> b) -> a -> b -> b` *(Hint: don't get stuck on this one!)*

Problem 3: (7 points, as indicated)

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

Instances of poor style or needlessly using other Prelude or helper functions will result in deductions.

Be sure to use the wildcard pattern (underscore) when appropriate.

There's no need to specify function types.

`snd` (Returns second element of a two-tuple) [1 point]

`not` [1 point]

`at list n` (Like `list!!n`. Assume `n` is in-bounds.) [2 points]

`map` [2 points]

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

DID YOU REMEMBER TO USE WILDCARDS WHEN APPROPRIATE?

Problem 4: (12 points) (6 points each)

For this problem you are to **write both recursive and non-recursive versions** 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

> sumexcl [3,3,3,3,1] 3
1

> sumexcl [] 5
0
```

Just like on assignment 3, the recursive version must not use any higher-order functions.

Just like on assignment 4, write the non-recursive version imagining that you just don't know to how to write a recursive function.

Problem 5: (6 points)

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'`. Usage:

```
> f 1
'a'

> f 2
'a'

> f 10
'b'
```

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

With `if-else`:

With guards:

With patterns:

Problem 6: (6 points)

The Haskell function `evenwords s` returns a string containing only the even-length words in the string `s`. Example:

```
> evenwords "now this is an exam for you right here"
"this is an exam here"

> evenwords "all odd len words"
""
```

Here is an incomplete solution for `evenwords`:

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

What's missing is the folding function `ff`. <u>Your task for this problem is to write the function `ff`.</u>

Problem 7: (18 points) (7 and 11 points, respectively)

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.

Usage:

```
> deo [10,20,30]
[10,40,30]

> deo [3]
[3]

> deo $ take 10 [1..]
[1,4,3,8,5,12,7,16,9,20]

> deo []
[]
```

Just like on assignment 3, the recursive version must not use any higher-order functions.

Just like on assignment 4, write the non-recursive version imagining that you just don't know to how to write a recursive function.

Hint: One of the clauses in the binding for my recursive solution starts like this: `deo (x1:x2:xs)`

Problem 8: (5 points) (one point each unless otherwise indicated)

The following questions and problems are related to Haskell.

(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]`.

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

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

(4) In Haskell, what's the difference between a type and a type class?

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

Problem 9: (15 points)

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.

Usage: (blank lines added between invocations below for clarity)

```
$ cat lg.1
a test
for
the program
here

$ ruby longest.rb < lg.1
3: 'the program'

$ cat lg.2
xx
a
yy
b
zz

$ ruby longest.rb < lg.2
1: 'xx'
3: 'yy'
5: 'zz'
```

Be sure to produce output that is exactly as shown above.

Problem 10: (9 points)

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. Assume counts are always non-negative integers.

Examples:

```
$ ruby repstrs.rb 3 x 4 ab 0 testing 1 .
xxxabababab.

$ ruby repstrs.rb 1 2 3 4 5 6
244466666

$ ruby repstrs.rb 1 "*" 10000000 "" 1 "*"
**

$ ruby repstrs.rb 3 a 4
?
```

Hint/warning: Remember that `"ab" * 3` produces `"ababab"`, but `3 * "ab"` produces an error.

Problem 11: (5 points) (one point each unless otherwise indicated)

<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?

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

(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?)

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

Problem 12: (9 points) (one point each unless otherwise indicated)

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.)

(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?

(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.

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

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

(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)

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

(2) In Ruby, what is the output produced by `"abc".each { |s| puts s[0].size.class }`?

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

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

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

_________ `$` __________

(6) In `ghci`, what's something `:info` will show me that `:type` won't?

(7) What's wrong with the following statement? "Java is an interpreted language."

(8) Fill in the blanks: Cells in a cons list comprise a/an __________ and a/an ___________.

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

(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?)