Assignment 1

Haskell

Due Date: Monday, January 30th 2017, 11:59pm

Directions

Answers to English questions should be in your own words; don't just quote from articles or books. We will take some points off for: code with the wrong type or wrong name, duplicate code, code with extra unnecessary cases, or code that is excessively hard to follow. You should always assume that the inputs given to each function will be well-typed, thus your code should not have extra cases for inputs that are not of the proper type. (Assume that any human inputs are error checked before they reach your code. Avoid duplicating code by using helping functions, library functions (when not prohibited in the problems), or by using syntactic sugars and local definitions (using **let** and **where**). It is a good idea to check your code for these problems before submitting. Be sure to test your programs on Eustis to ensure they work on a Unix system.

This is a group assignment. Make a group of 3-4 in WebCourses and use that to turn in the assignment.

Deliverables

For this assignment, turn in the following files:

- Average3.hs
- Add10List.hs
- CubeOdds.hs
- DeleteNth.hs
- Solution.pdf

For problems that require an English answer, put your solution in Solution.pdf. For coding problems, put your solution in the appropriate Haskell source file (named with ".hs" or ".lhs" suffix).

Problem 1 (5 pts)

The function you are to write in Haskell is

```
average3 :: (Double, Double, Double) -> Double
```

which takes in a triplet of Doubles, (x, y, z), and returns a Double that is the average (i.e., the arithmetic mean) of x , y, and z. The following are examples, written using the Testing and FloatTesting modules which are included in the hw2-tests.zip file.

To run our tests, use the Average3Tests.hs file. To make that work, you have to put your code in a module Average3, which will need to be in a file named Average3.hs (or Average3.lhs), in the same directory as Average3Tests.hs. Your file Average3.hs should thus start as follows:

```
module Average3 where
average3 :: (Double, Double, Double) -> Double
```

Then run our tests by running the main function in Average3Tests.hs. Our tests are written using the Testing.lhs and FloatTesting.hs files, which are included on WebCourses.

Problem 2 (22 pts)

These questions will test your conceptual knowledge of Haskell and programming languages.

(a) (5 pts) In Haskell, which of the following is equivalent to the list $[3, 5, 3]$?

```
1. (3,5,3)
2. 3:(5:3)
3. (3:5):3
4. (((3:5):3):[])
5. 3:(5:(3:[]))
```

(b) (12 pts) Suppose that ohno is the list ['o', 'h', 'n', 'o'] and that yikes is the list "yikes". For each of the following, say whether it is legal or illegal in Haskell, and if it is illegal, say why it is illegal.

1. ohno:'y'
2. ohno ++ yikes
3. ohno:yikes
4. ['o']:yikes
5. 'o':yikes
6. 1:yikes

(c) (5 pts) Haskell has built in functions **head** and **tail** defined as follows.

```
head          :: [a] -> a
head (x:_)    = x
head []       = error "Prelude.head: empty list"

tail          :: [a] -> [a]
tail (_:xs)   = xs
tail []       = error "Prelude.tail: empty list"
```

For example, `head [1 ..]` equals 1 and `tail [1 ..]` equals `[2 ..]`. Consider the following function.

```
rip lst =
    let back = tail lst
    in let front = head lst
       in (lst, front:back)
```

What is the result of the call `rip [3,4,7,5,8]`?
(We suggest that you think about it first, and only use the Haskell system to check the answer.)

Problem 3 (10 pts)

This problem will have you write a solution in 2 ways. The problem is to write a function that takes a list of Integers and returns a list that is just like the argument but in which every element is 10 greater than the corresponding element in the argument list.

(a) (5 pts) Write the function

```
add10_list_comp :: [Integer] -> [Integer]
```

that solves the above problem by using a list comprehension.

(b) (5 pts) Write the function

```
add10_list_rec :: [Integer] -> [Integer]
```

that solves the above problem by writing out the recursion yourself; that is, without using a list comprehension and without using map or any other higher-order library function.

There are test cases contained in Add10ListTests.hs. To make that work, you have to put your code in a module Add10List, which will need to be in a file named Add10List.hs (or Add10List.lhs), in the same directory as Add10ListTest.hs. Your file Add10List.hs should thus start as follows.

```
module Add10List where
add10_list_rec :: [Integer] -> [Integer]
add10_list_comp :: [Integer] -> [Integer]
```

Then run our tests by running the main function in Add10ListTests.hs.

As specified on the first page of this homework, turn in both your code file and the output of your testing. (The code file should be uploaded to Webcourses, and the test output should be pasted in to the Comments box for that assignment.)

Problem 4 (10 pts)

In Haskell, write function:

```
cubeOdds :: [Integer] -> [Integer]
```

that takes a list of Integers, `lst`, and returns a list of Integers that is just like `lst`, except that each odd element of `lst` is replaced by the cube of that element. In your solution, you might find it helpful to use the build-in predicate odd.

Tests are in CubeOddsTests.hs.

Problem 5 (10 pts)

Write the function

```
deleteNth :: (Eq a) => Int -> a -> [a] -> [a]
```

that takes a positive Int, n, an element, toDelete, of some equality type a, and a list, as, of type [a], and returns a list that is just like as, but which does not contain the nth occurrence (in as) of the element toDelete.

Your solution must *not* use any Haskell library functions. You may assume that n is strictly greater than 0.

There are test cases contained in DeleteNthTests.hs.

Problem 6 (5 pts)

Is it possible to use a list comprehension to solve problem 5 in an easy, direct way? Briefly explain.