CSCE 314 [Sections 500, 501] Programming Languages – Spring 2020

Hyunyoung Lee

Homework Assignment 2

Assigned on Thursday, January 30, 2020

Electronic submission to eCampus due **at 10:00 p.m., Monday, February 10, 2020**
By electronically submitting this assignment to eCampus by logging in to your account, you are signing electronically on the following Aggie Honor Code:

"On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment."

In this assignment, you will earn total 100 points. Here are some general instructions.

1. Read the problem descriptions and requirements carefully! There may be significant penalties for not fulfilling the requirements.

2. Some problems ask you to explain the working of your function with the given input. Your explanation must be consistent with your definition of the function. Your work will be graded not only on the correctness of your answer, but also on the consistency and clarity with which you express it.

3. This homework set is an *individual* homework, not a team-based effort. Discussion of the concept is encouraged, but actual write-up of the solutions must be done individually and your final product – the code as well as the comments and explanations – should never be shared.

4. Submit electronically exactly one file named *YourLastName*-*YourFirstName*-**hw2**.hs, and nothing else, on eCampus.tamu.edu.

5. Make sure that the Haskell script (the .hs file) you submit compiles without any error when compiled using the Glasgow Haskell Compilation System (ghc or ghci) version 7 and above[1].

   If your program does not compile, you will receive very few points (more likely zero) for this assignment. To avoid receiving zero for the entire assignment, if you cannot complete defining a function correctly without compile error, you can set the function definition `undefined`, see the skeleton code provided.

6. Remember to put the head comment in your file, including your name, UIN, and *acknowledgements of any help received* in doing this assignment. Again, remember the Honor Code!

[1] Version 7 is installed in the departmental servers (linux.cse.tamu.edu and compute.cse.tamu.edu), and version 8 is what you will get if you install the Haskell system in your computer.

Below, the exercise problems are from the Haskell Textbook: "Programming in Haskell, 2nd Ed.", by Graham Hutton. Some problems are modified (with additional requirements) by the instructor. Please read corresponding textbook chapters and the problem statements carefully, paying attention to the requirements. For example, "using `foldr`, define ..." means that using the `foldr` function when you define the assigned function is required. There may be significant penalties for not fulfilling such requirements. Keep the name and type of each function exactly the same as given in the problem statement and the skeleton code.

Problem 1. (5 points) Put your full name, UIN, and *acknowledgements of any help received* in the head comment in your .hs file for this assignment.

Problem 2. (5 points) Chapter 4, Exercise 5. Using two nested conditional expressions in the definition is a requirement.

Problem 3. (20 points) Chapter 4, Exercise 8.

Problem 4. (10 points) Chapter 5, Exercise 6. Using a list comprehension and `factors` in the definition is a requirement. Include the definition of `factors` in your hw2.hs file (the definition is in the text as well as in my lecture slides).

Problem 5. ($7 + 7 + 6 = 20$ points) Chapter 6, Exercise 5. Your answer should follow the style of examples such as `reverse`, `(++)`, `insert`, and `zip` in pages 62–64 in the text. Write your answer neatly and clearly within a block comment `{- ··· -}`.

Problem 6. (15 points) This problem has two subproblems. In Assignment 1, Problems 5 and 6, you implemented merge sort that sorts a list in an ascending order.

1. (8 points) Define a recursive function `mergeBy` that merges two sorted lists by the given criterion, for example, in an ascending order or in a descending order (so that the resulting list is also sorted). The type signature of `mergeBy` is as follows.

   ```
   mergeBy :: (a -> a -> Bool) -> [a] -> [a] -> [a]
   ```

   Notice the difference from `merge :: Ord a => [a] -> [a] -> [a]` in Ch. 6 Exercise 7 such that `mergeBy` accepts three arguments, the first of which is a comparison function of type `(a -> a -> Bool)` that determines in which way the list is to be sorted. Such comparison function that returns a Boolean value (true or false) is called a *predicate*.

2. (7 points) Using `mergeBy` that you wrote above and `halve` that you wrote for Problem 6 in Assignment 1, define a recursive function `msortBy`. The problem specification stays the same as that for *msort* in Ch. 6 Exercise 8, except the additional requirement of the first argument being a predicate. Thus, the type of `msortBy` is:

   ```
   msortBy :: (a -> a -> Bool) -> [a] -> [a]
   ```

Problem 7. (15 points) Chapter 7. Exercise 9.

1. (10 points) Define `altMap`.

   ```
   altMap :: (a -> b) -> (a -> b) -> [a] -> [b]
   ```

2. (5 points) Explain how your `altMap` works when it is applied as below.

   ```
   > altMap (*2) (`div` 2) [0..6]
   ```

Problem 8. (10 points) Using `map`, `filter`, and `(.)` (function composition operator), define a function that examines a list of strings, keeping only those whose length is odd, converts them to upper case letters, and concatenates the results to produce a single string.

```
concatenateAndUpcaseOddLengthStrings :: [String] -> String
```

You need to `import Data.Char` in order to use the `toUpper` function (see the skeleton code).

Have fun!