# CSCE 314 [Sections 595, 596, 597] Programming Languages – Spring 2024

**Hyunyoung Lee**

**Homework Assignment 3**

Assigned on Monday, February 19, 2024

Electronic submission to Canvas due **at 11:59 p.m., Friday, March 1, 2024**
*By electronically submitting this assignment to Canvas 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 120** points. Here are some general instructions.

1. 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 by yourself. Your final product – the code as well as comments and explanations – should never be shared.

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

3. **Explain each function definition line-by-line in your own words.** In addition, 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.

4. Submit electronically exactly one file named *YourFirstName-YourLastName*-hw3**.txt**, and nothing else, on canvas.tamu.edu.

5. Make sure that the Haskell script you submit compiles without any error when compiled using the Glasgow Haskell Compilation System (ghci) version 8 or 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 8.10.7 is installed in the servers maintained by the college of engineering (linux2.engr.tamu.edu and compute.engr.tamu.edu), and version 9 is the most recent version.

Below, the exercise problems are from the required 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. There will be significant penalties for not fulfilling such requirements. Keep the name and type of each function exactly the same as given in the skeleton code.

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

Use the following data type for Problems 2 and 3.

```
data Tree a b = Leaf a | Branch b (Tree a b) (Tree a b)
```

**Problem 2.** (20 points) Make `Tree` an instance of `Show`. Do not use `deriving`; define the instance yourself. Make the output look somewhat nice (e.g., indent nested branches). See the supplementary PDF document for more information.

**Problem 3.** $(15 + 20 = 35$ points)

1. $(3 \times 5 = 15$ points) Implement the three functions that traverse the tree in the given order collecting the values from the tree nodes into a list:

   ```
   preorder  :: (a -> c) -> (b -> c) -> Tree a b -> [c]
   inorder   :: (a -> c) -> (b -> c) -> Tree a b -> [c]
   postorder :: (a -> c) -> (b -> c) -> Tree a b -> [c]
   ```

   Notice that the data type `Tree` can store different types of values in the leaves than on the branching nodes. Thus, each of these functions takes two functions as arguments: The first function maps the values stored in the leaves to some common type `c`, and the second function maps the values stored in the branching nodes to type `c`, thus, resulting in a list of type `[c]`.

2. $(10 + 10 = 20$ points) Also, explain how each of your `preorder` and `inorder` works in detail step-by-step with the following tree object (given in the skeleton code).

   ```
   tree1 :: Tree Int String
   tree1 = Branch "*"
               (Branch "+"
                  (Branch "*" (Leaf 2) (Leaf 6))
                  (Branch "+" (Leaf 3) (Leaf 4)))
               (Branch "*"
                  (Branch "+"
                     (Branch "*" (Leaf 8) (Leaf 2))
   ```

```
                       (Leaf 7))
                    (Branch "+" (Leaf 5) (Leaf 4)))

> preorder show id tree1

> inorder show id tree1
```

**Problem 4.** (40 points) Chapter 8, Exercise 9 (Modified). Study Section 8.7 carefully before attempting this problem. The skeleton code contains more explanation on what you are supposed to do.

**Problem 5.** (20 points) Explain how the functions you defined in the previous problem work in detail step-by-step when `value` is applied to `e9` as below (given in the skeleton code).

```
e9 = (Subt (Mult (Val 2) (Val 3)) (Add (Val 4) (Val 1)))

> value e9
```

Have fun!