CSci 450: Org. of Programming Languages
CSci 503: Fundamental Concepts in Languages
Assignment #3, Fall 2017

H. Conrad Cunningham

9 October 2017 (Revised 10 Oct)

Assignment #3
Due Monday, 16 October, 11:59 p.m.

General Instructions

All homework and programming exercises must be prepared in accordance with the instructions given in the Syllabus. Each assignment must be submitted to your instructor by its stated deadline.

In accordance with expected scholarly and academic standards, if you reference outside textbooks, reference books, articles, websites, etc., or discuss this assignment with individuals inside or outside the class, you must document these by including appropriate citations or comments in the header of the primary source file or in another prominent place in the program documentation.

Assignment Description

1. This is an individual assignment. When complete, submit your Haskell source and testing modules to the course Blackboard site.
2. **CSci 450 students** do all parts of exercise 3 and five of the six problems 4-9 from Chapter 5 of *Introduction to Functional Programming*

Using Haskell (shown below).

Put your functions in a module named `HW03` (in file `HW03.hs`).

Test your functions thoroughly. Put your testing code in a module `HW03_Test` (in file `HW03_Test.hs`).

CSci 503 students do all ten of the exercises listed above.

I encourage all students to work on all exercises.

Chapter 5 Exercises

3. Suppose you need a Haskell function `sumSqNeg` that takes a list of integers (type `Integer`) and returns the sum of the squares of the negative values in the list.

   Define the following Haskell functions. Use the higher order library functions (from this chapter) such as `map`, `filter`, `foldr`, and `foldl` as appropriate.

   a. Function `sumSqNeg1` that is backward recursive. (Use recursion directly. Do not use the list-folding Prelude functions such as `foldr` or `sum`.)

   b. Function `sumSqNeg2` that is tail recursive. (Use recursion directly. Do not use the list-folding Prelude functions such as `foldr` or `sum`.)

   c. Function `sumSqNeg3` that uses standard prelude functions such as `map`, `filter`, `foldr`, and `foldl`.

   d. Function `sumSqNeg4` that uses list comprehensions (Chapter 7).

4. Define a Haskell function

```
total :: (Integer -> Integer) -> Integer -> Integer
```

   so that `total f n` gives `f 0 + f 1 + f 2 + ... + f n`.

5. Define a Haskell function

```
removeFirst :: (a -> Bool) -> [a] -> [a}
```

   so that `removeFirst p xs` removes the first element of `xs` that does has the property `p`.

6. Define a Haskell function

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

   so that `removeLast p xs` removes the last occurrence of element of `xs` that has the property `p`.

   How could you define it using `removeFirst`?

7. Define a Haskell function `map2` that takes a list of functions and a list of values and returns the list of results of applying each function in the first list to the corresponding value in the second list.

8. Define a Haskell function `fmap` that takes a value and a list of functions and returns the list of results from applying each function to the argument value. (For example, `fmap 3 [((*) 2), ((+) 2)]` yields `[6,5]`.)

9. Define a Haskell function `composeList` that takes a list of functions and composes them into a single function. (Be sure to give the type signature.)