CSCE 314 Programming Languages – Spring 2018

Anandi Dutta

Assignment 2

Assigned on Saturday, January 27, 2018

Electronic submission to eCampus due at 23:59, Friday, February 9, 2018
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 practice the basics of functional programming in Haskell. Below, you will find problem descriptions with specific requirements. Read the descriptions and requirements carefully! There may be significant penalties for not fulfilling the requirements. You will earn total 100 points.

Note 1: This homework set is *individual* homework, not a team-based effort. Discussion of the concept is encouraged, but actual write-up of the solutions must be done individually.

Note 2: Submit electronically exactly one file, namely, *yourLastName-yourFirstName*-a2.hs, and nothing else, on eCampus.tamu.edu.

Note 3: Do not use functions in the standard prelude such as `sum`, `product`, `map`, etc. for problem 2,3 and 5. You can use operators such as +, ∗, ==, :, ++, etc.You are allowed to use prelude functions for other problems.

Note 4: Make sure that the Haskell script (the .hs file) you submit compiles without any error. If your program does not compile, you will likely receive zero points for this assignment. To avoid receiving zero for the entire assignment, if you cannot complete defining a function correctly, you can set it `undefined`, see the skeleton code provided.

Note 5: Remember to put the head comment in your file, including your name, UIN, and *acknowledgements of any help received* in doing this assignment. You will get points deducted if you do not put the head comment. Again, remember the honor code.

Keep the name and type of each function exactly as given.

Problem 1. (5 points) Put your name, UIN, and *acknowledgements of any help received* in the head comment.

Problem 2. (5 points) Write a recursive function `fibonacci` that computes the n-th Fibonacci number.

```
fibonacci ::  Int -> Int
```

Problem 3. (5 points) Write a recursive function `myProduct` that multiplies all the numbers in a list.

```
myProduct ::  [Integer] -> Integer
```

Problem 4. (10 points) Write a recursive function `flatten` that flattens a list of lists to a single list formed by concatenation.

```
flatten ::  [[a]] -> [a]
```

Problem 5. (5 points) Write a recursive function `myLength` that counts the number of elements in a list.

```
myLength ::  [a] -> Int
```

Problem 6. (10 points) Write a recursive function `quicksort` that sorts a list of elements in an *ascending* order.

```
quicksort ::  Ord t => [t] -> [t]
```

Problem 7. (10 points) Write a recursive function that returns `True` if a given value is an element of a list or `False` otherwise.

```
isElement ::  Eq a => a -> [a] -> Bool
```

Problem 8. (10 points) Using `insert`, define a recursive function `isort :: [Int] -> [Int]` that implements insertion sort, in which the empty list is already sorted, and any non-empty list is sorted by inserting the head of the list into the result of recursively sorting the tail of the list.

Problem 9. (10 points) Using recursion, define a function `riffle :: [a] -> [a] -> [a]` that takes two lists of the same length, and interleaves their elements. For example, `riffle [1,2,3] [4,5,6] = [1,4,2,5,3,6]`.

Problem 10. (10 points) Define a function `shuffle :: Int -> [a] -> [a]` that takes a natural number n and an even-lengthed list, and splits and then riffles the list n times. For example, `shuffle 2 [1,2,3,4,5,6] = [1,5,4,3,2,6]`.

Problem 11. (10 points) A positive integer is perfect if it equals the sum of its factors, excluding the number itself. Using a list comprehension and the function factors, define a function `perfects :: Int -> [Int]` that returns the list of all perfect numbers up to a given limit. For example: `perfects 500 [6,28,496]`

Problem 12. (10 points) Show how the library function `replicate :: Int -> a-> [a]` that produces a list of identical elements can be defined using list comprehension. For example: `replicate 3 True [True, True, True]`

Skeleton code: The function bodies are initially `undefined`, a special Haskell value that has all possible types. Also in that file you find a test suite that test-evaluates some of the functions. Initially, all tests fail – until you provide correct implementation for the function. Some tests are written using the HUnit library. Feel free to add more tests to the test suite. Please follow the instructions attached to load the skeleton code (a2.hs file) to the interpreter. Evaluating the function main will run the tests.

Loading your Haskell script in ghci

1. At first open your choice of editor such as Notepad++ / Sublime etc. Then create a new file, write your Haskell code on that file and save that file with .hs extension. For example, I am writing this code:

```
double x = x+x
```

Then I am saving this file as double.hs

2. Open your ghci from your terminal/command prompt by typing `ghci`.

```
C:\Users\Anandi>ghci
GHCi, version 8.0.1: http://www.haskell.org/ghc/  :? for help
Prelude>
```

3. Now, you can run all the built-in functions from here. Such as:

```
C:\Users\Anandi>ghci
GHCi, version 8.0.1: http://www.haskell.org/ghc/  :? for help
Prelude> length [1,2,3,4,9]
5
Prelude> drop 2 [4,7,8,9]
[8,9]
Prelude>
```

4. The next step is to load your Haskell script such as double.hs script on ghci. You have to specify the directory first to load your double.hs file. For example, I saved my double.hs file in Desktop. Therefore, I am specifying the directory by using `cd` command first. Then you can load your script by using the `load` command. Below is an example for Windows:

```
Prelude> :cd Desktop
Prelude> :load double
[1 of 1] Compiling Main             ( double.hs, interpreted )
Ok, modules loaded: Main.
*Main> double 6
12
*Main>
```

The double.hs script has been loaded to the ghci now. We can use the double function just like other built-in functions like length, drop etc.

Below is an example for Mac users: