CSCE 314 Programming Languages – Fall 2016
Dylan Shell
Assignment 2
Assigned on Wednesday, September 7, 2016

Electronic submission to eCampus due at 23:59, Wednesday, Sep. 21, 2016
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: 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 4: 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.

Note 5: We will use the Glasgow Haskell Compiler (ghc), version 7.4.2, `not` the newer version. The departmental servers (linux.cse.tamu.edu and compute.cse.tamu.edu) have this version. Please make sure that you install this version on your computer.

Keep the name and type of each function exactly as given. Do not use functions in the standard prelude such as `sum`, `product`, `map`, etc. You can use operators such as +, ∗, ==, :, ++, etc.

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. (5 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 a *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 the technique of List Comprehension write a function that would remove even numbers from a list of lists.
For Example: given input: [[1,2,3,4], [6,3,45,8], [4,9,23,8]] expected output: [[1,3], [3,45],[9,23]].

Problem 9. (20 points) Using Pattern matching and Guards, write a function that would take total GPA of some courses for a semester and the number of courses as input. Then it would calculate the CGPA for that semester and print a message for the student. You can choose messages according to your wish. If CGPA is greater than 3.5, you need to print a message, If CGPA is less than 2.5, you need to print a message,
For Example: given input: 14 4 expected output: " You are doing well!"

Problem 10. (15 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 11. (10 points) Show how the library function `replicate :: Int -> a-> [a]` that produces a list of identical elements can be defined using a list comprehension.For example:
`> replicate 3 True [True, True, True]`

Skeleton code: The file a2-skeleton.hs contains "stubs" for the functions problems (1-7) you are going to implement. 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 the functions. Initially, all tests fail – until you provide correct implementation for the function. The tests are written using the HUnit library. Feel free to add more tests to the test suite. The skeleton code can be loaded to the interpreter (`> ghci a2-skeleton.hs`). Evaluating the function main will run the tests. Alternatively, one can compile the code and execute it:
`> ghc a2-skeleton.hs` and then `> ./a2-skeleton`