CSci 450: Org. of Programming Languages
CSci 503: Fundamental Concepts in Languages
Assignment #1, Fall 2017

H. Conrad Cunningham

1 September 2017

Assignment #1
Due Monday, 11 September, 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.

Assignment Description

1. When complete, submit your Haskell source and testing files to the course Blackboard site. This is an individual assignment.

2. Develop Haskell functions to solve exercises 1, 3, 4, 5, 6, 7, and 8 from Chapter 2, Basic Haskell Functional Programming. You may put all these functions in the same module and include functions to test them appropriately.

Copy of Exercise Descriptions

1. Develop a Haskell function `sumsqbig` that takes three numbers as arguments and returns the sum of the squares of the two larger numbers. That is `sumsqbig 2.0 1.0 3.0` yields `13`.

2. OMITTED

3. Develop a Haskell Boolean function `div23n5` such that `div23n5 n` returns `True` if and only if `n` is divisible by 2 or divisible by 3 but not divisible by 5. That is, `div23n5 6` yields `True` and `div23n5 30` yields `False`.

4. Develop a Haskell function `notDiv` such that `notDiv n d` returns `True` if and only if integer `n` is not divisible by `d`. That is, `notDiv 10 5` yields `False` and `notDiv 11 5` yields `True`.

5. Develop a Haskell function `mult` that takes two *natural numbers* (i.e., nonnegative integers) and returns their product. The function must not use the multiplication (`*`) or division (`div`) operators. Hint: Multiplication can be done by repeated addition.

6. Develop a Haskell function `addTax` that takes two `Double` values such that `addTax c p` returns `c` with a sales tax of `p` *percent* added. For example, `addTax 2.0 9.0` returns `2.18`.

   Also develop a function `subTax` that is the inverse of `addTax`. That is, `subTax (addTax c p) p` yields `c`.

7. The time of day can be represented by a tuple `(hours,minutes,m)` where `m` indicates either "AM" or "PM". Develop a Boolean Haskell function `comesBefore` that takes two time-of-day tuples and determines whether the first is an earlier time than the second. (Note: Consider midnight and noon.)

8. Develop a Haskell function

   ```
   minf :: (Int -> Int) -> Int
   ```

   such that `minf g` returns the smallest integer `m` such that `0 <= m <= 10000000` and `g m == 0` (if such an integer exists).