CSci 450: Org. of Programming Languages
CSci 503: Fundamental Concepts in Languages
Assignment #1, Fall 2018

H. Conrad Cunningham

8 September 2018

8 Sept: Added link to module template file. Corrected a typo in exercise 9 (`subTax` replaced incorrect spelling `subtax`). Clarified the wording of exercise 11.

Assignment #1

Due 11:59 p.m., Thursday, 13 September, 2018

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.

Citations: In accordance with expected scholarly and academic standards, if you reference outside textbooks, reference books, articles, websites, etc., or discuss an assignment with individuals inside or outside the class, you must document these by including appropriate citations or comments at prominent places in your submission such as in the header of the primary source file.

Identification: Put your name, course name, and assignment number as comments in each file you submit.

Assignment Description

- This is an individual assignment.

- When complete, submit your Haskell source code file to the course Blackboard site for Assignment #1.

  Be sure to document your code appropriately using program comments. Give attention to the general instructions given above and in the Syllabus.

- **Create a Haskell module `HW01` in file `HW01.hs`.**

- **Include solutions for Exercises 2, 3, 4, 7, 9, 11, and 12 from Chapter 5 of textbook *Exploring Languages with Interpreters and Functional Programming*.** These are copied below.

  You may use the module file `HW01.hs` to develop your module.

- We may use a unit testing framework to partially automate grading. So it is important that you use the precise file name, module name, function names, and function signatures given in this assignment.

Exercises from ELIFP Chapter 5

1. OMIT

2. Develop a Haskell function `prodSqSmall` that takes three `Double` arguments and returns the product of the squares of the two smaller numbers.

   For example, `prodSqSmall 2.0 4.0 3.0` yields `36.0`.

3. Develop a Haskell function `xor` that takes two Booleans and returns the "exclusive-or" of the two values. An exclusive-or operation returns `True` when exactly one of its arguments is `True` and returns `False` otherwise.

4. Develop a Haskell Boolean function `implies` that takes two Booleans `p` and `q` and returns the Boolean result `p` ⇒ `q` (i.e. logical implication). That is, if `p` is `True` and `q` is `False`, then the result is `False`; otherwise, the result is `True`.

   Note: This function is sometimes called `nand`.

5. OMIT

6. OMIT

7. Develop a Haskell function `ccArea` that takes the *diameters* of two concentric circles (i.e. circles with the same center point) as `Double` values and returns the area of the space between the circles. That is, compute the area of the larger circle minus the area of the smaller circle. (Hint: Haskell has a builtin constant `pi`.)

   For example, `ccArea 2.0 4.0` yields `9.42477796076938` on the author's Mac.

8. OMIT

9. 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`. For example, `subTax 2.18 9.0` yields `2.0`.

10. OMIT

11. A day on the calendar (usual Gregorian calendar used in the USA) can be represented as a tuple with three `Int` values `(month,day,year)` where the `year` is a positive integer, `1 <= month <= 12`, and `1 <= day <= days_in_month`. Here `days_in_month` is the number of days in the the given `month` (i.e. 28, 29, 30, or 31) for the given `year`.

    Develop a Boolean Haskell function `validDay d` that takes a date tuple `d` and returns `True` if and only if `d` represents a valid date.

    For example, `validDay (8,20,2018)` and `validDay(2,29,2016}` yield `True` and `validDay (2,29,2017)` and `validDay(0,0,0)` yield `False`.

    Note: The Gregorian calendar was introduced by Pope Gregory of the Roman Catholic Church in October 1582. It replaced the Julian calendar system, which had been instituted in the Roman Empire by Julius Caesar in 46 BC. The goal of the change was to align the calendar year with the astronomical year.

    Some countries adopted the Gregorian calendar at that time. Other countries adopted it later. Some countries may never have adopted it officially.

    However, the Gregorian calendar system became the common calendar used worldwide for most civil matters. The *proleptic Gregorian calendar* extends the calendar backward in time from 1582. The year 1 BC becomes year 0, 2 BC becomes year -1, etc. The proleptic Gregorian calendar underlies the ISO 8601 standard used for dates and times in software systems.

12. Develop a Haskell function `roman` that takes an `Int`) in the range from 0 to 3999 (inclusive) and returns the corresponding Roman numeral as a string (using capital letters). The function should halt with an appropriate `error` messages if the argument is below or above the range. Roman numbers use the following symbols and are combined by addition or subtraction of symbols.

I	1
V	5
X	10
L	50
C	100

D	500
M	1000

For the purposes of this exercise, we represent the Roman numeral for 0 as the empty string. The Roman numbers for integers 1-20 are `I`, `II`, `III`, `IV`, `V`, `VI`, `VII`, `VIII`, `IX`, `X`, `XI`, `XII`, `XIII`, `XIV`, `XV`, `XVI`, `XVII`, `XVII`, `XIX`, and `XX`. Integers 40, 90, 400, and 900 are `XL`, `XC`, `CD`, and `CM`.

13. OMIT