CPSC 449 – Programming Paradigms

Assignment #2

Due Date: Friday, October 19, 11:59 PM

Individual Work:

All assignments in this course are individual work. Students are advised to read the guidelines for avoiding plagiarism located on the course outline. Students are also advised that electronic tools may be used to detect plagiarism.

Late Penalty:

Late submissions will not be accepted.

Submissions:

Submit `assignment02.hs` file that contains the required functions in `D2L`.

Problem Description:

1- Define a function called *`productFactorial`* which takes an Integer n and computes the product of all the factorials from 0 up to and including n. You may consider using a `let` or `where` expression to define a factorial functions to be used by this function. Both `factorial` and *`productFactorial`* will be recursive functions.

2- A prime number p has only two factors, 1 and p itself. A composite number has more than two factors. Define a function *`smallestFactor`* that returns the smallest factor of n larger than one. For example, `smallestFactor 14` would return 2, while `smallestFactor 15` would return 3.

3- A Game. Think of a whole number greater than one. If it's even, divide it by two, otherwise multiply it by three and add one. Stop if the resulting number is one, otherwise repeat the procedure.

   As example, we start with 10.

   10 is even, so the next number is: 10/2 = 5

   5 is odd so the next number is: 3*5+1=16

   16 is even, so the next number is: 16/2 = 8

   8 is even, so the next number is: 8/2 = 4

4 is even, so the next number is: 4/2 = 2

2 is even, so the next number is: 1.

If we start with 7 we get: (See below)

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Write a function *gameOddEven* that given a number n, generate a list of numbers there in the sequence?

Note that we include both the number n we start with and the final number 1.

4- We assume that a good password has the following characteristics:

a. Should be at least 8 characters long.
b. Should contain letter and numbers.
c. Should have both upper and lower case letter.

Write a function *isGoodPassword* that returns true if the password is good, false otherwise.

5- Define a function *isPrime* that determines whether a given number `n` is prime or not?

6- Define a function *allDivisors* that takes number `n` and returns all its divisors as a list.

7- Define the function *matches* such that
matches:: Integer -> [Integer] -> [Integer]
that picks out all occurrences of an integer n in a lists.

8- Write a function called *solveQuadraticEquation* which takes in three arguments (a, b, and c) which are coefficients to the quadratic equation ax2 + bx + c = 0. a, b, c, and x should have type Double. The output should be a tuple containing the two roots. Don't worry about complex roots; if you apply the sqrt function to a negative number you will get NaN (Not A Number). Use a let or a where expression to define the square root of the discriminant (sqrt(b ** 2 - 4 * a * c)).

9- Define a function `occursIn x xs,` which returns True if x is an element of xs.

10- `Define a function allOccurIn xs ys,` which returns True if all of the elements of xs are also elements of ys.

11- Define a function `sameElements xs ys,` which returns True if xs and ys have exactly the same elements.

12- Define a function `numOccurrences x xs,` which returns the number of times x occurs in xs.

13- Write a function *`allUrls`* that given a text as a string, and return a list with all the web addresses in the text. We assume that each web address will start from `http://`

14- Eratosthenes' sieve is an ancient method for finding prime numbers. Define a function *sieve* which applies Eratosthenes' sieve to the list of numbers it is given, and returns a list of all the prime numbers that it found. Use sieve to construct the list of primes from 2 to 100.

15- Pascal Triangle: Pascal triangle is a triangle of numbers computed as follows:

a. The first row contains 1
b. The following rows are computed by adding together adjacent numbers in the row above, and adding a 1 at the beginning and at the end.

Write a function `pascal::Int->[Int]` so that it computes nth row of a Pascal triangle

Grading:

A+: All functions (1 – 15) of the assignments are completed successfully and pass all the test cases.

A: All functions (1 - 13) of the assignment AND at least one of the function from (14 – 15) is completed successfully.

B: All functions (1 – 13) of the assignment are implemented successfully.

C: Any 10 functions are implemented successfully.

D: Any 7 functions are implemented successfully.

Credits:

The text in this assignment is adapted from the two courses on Haskell offered at California Institute of Technology (Caltech) and Chalmers Institute of Technology.