CSE 262: Programming Languages

Fall 2016

Homework 3: **Due on September 26th at 11pm on CourseSite.**

READINGS: Learn You a Haskell: Chapter 4. Real World Haskell: Chapter 2.

1. (20 points): Write a Haskell function that takes two lists of Num as arguments. Call the arguments x and y. Append the elements from y that are even to the end of list x. Call this appendEvens.

   ```
   appendEvens [1,2,3,4] [6,5,7,9,8,3,2] -> [1,2,3,4,6,8,2]
   ```

2. (20 points): Write a Haskell function that takes no arguments. It should return a list of elements that contains the multiples of 5 and 7. This list will be infinite in size. Remember that you will need use take to see results of this. Call this infiniteBuzz.

   ```
   take 10 infiniteBuzz -> [5,7,10,14,15,20,21,25,28,30]
   ```

3. (20 points): Write a Haskell function that takes two arguments. Using let to define an infinite list of even numbers starting at 2 (so [2,4,6...]), call the arguments x and y, return the list containing the elements of the infinite list within, and inclusive of, the range of x to y. Call this evenGrabber.

   ```
   evenGrabber 3 5 -> [8,10,12]
   evenGrabber 7 5 -> []
   evenGrabber 1 4 -> [4,6,8,10]
   ```

4. (20 points): Write a Haskell function that takes a single list argument and generates different results based on the first two elements of the list - do this with pattern matching of the argument where possible and guards otherwise. Follow the rules below, order matters. Call this patternChoice.

(a) If the first element is 0, return the remainder of the list.

(b) If the first element is odd, skip a number of elements equal to the second element and return the rest of the list.

(c) If the first element is even, and the second element is even, return the third element, if any (empty if it does not exist). As a list. Obviously.

(d) Otherwise, return a list that contains the elements of the input list decremented by the first element of that list.

```
patternChoice [0,2,3,4] -> [2,3,4]
patternChoice [2,2,3,4] -> [3]
patternChoice [1,2,3,4] -> [3,4]
patternChoice [2,2] -> []
patternChoice [2,1,5,6,7] -> [0,-1,3,4,5]
```

5. (20 points): Write a Haskell function takes a 'function that takes two parameters' as its first parameter and some value as a second parameter, call these x and y. Return function x partially evaluated with y as its first argument. Call this partialFunc.

```
partialFunc mod 5 3 -> 2
(partialFunc mod 5) 3 -> 2
(partialFunc take 3) [2,3,5,7,8,3] -> [2,3,5]
```