Discrete Math & Functional Programming— CSCI 054— Spring 2024
Instructor: Osborn
Homework 2 (31 point(s))
Due: 10:00PM on Sunday

- You will be working with 1 or 2 other people from your small group on this assignment; check canvas for the pairings. Only one of you should turn in the code on Gradescope, but that person should make sure to add everyone else as a collaborator.
- You may discuss ideas with other people in the class, but should **never** be looking at someone else's working code or showing someone else your working code (this includes, for example, looking up how to write the exact function on Google (or ChatGPT!), asking someone to tell you what they did, or reading your code out loud to someone else). If you are unclear about what is acceptable, you are expected to check with Prof. Osborn **in advance**.
- Make sure that each of your functions starts with a type declaration and a short description. We've provided the type for the first function in the `week02-code-template.hs` file but you'll need to add the rest. For this assignment the type declarations **only** can be discussed and agreed on by your entire small group during the small group meeting.
- Double check that your code exhibits good style: accurate header comment at the top of the file, a comment before each function describing what it does, consistent indenting and variable names, and so on.

1. **[4 point(s)] Functions**
   (a) Write a function `cube` that takes a number `n` and returns n^3.
   (b) The Takeuchi function was developed in the 1970's as a benchmark for Lisp systems. It makes a large number of recursive calls without generating large integers. Write a function `tak` to compute the Takeuchi function, which is defined as follows:

$$\text{tak}(x,y,z) = \begin{cases} y & \text{if } x \leq y, \text{ and} \\ \text{tak}(\text{tak}(x-1,y,z), & \\ \quad \text{tak}(y-1,z,x), & \\ \quad \text{tak}(z-1,x,y)) & \text{otherwise.} \end{cases}$$

   Note that x, y, and z are required to be integers.

2. **[6 point(s)] List comprehension** For this problem your functions **must** use a list comprehension.

(a) Write a function `cubeAll` that takes a list of numbers and returns a list containing the cube of each of the numbers. For example, `cubeAll [1, 3, -2]` should return `[1, 27, -8]`. You are encouraged to use the `cube` function that you wrote for an earlier problem in this assignment.

(b) Write a function `lengthPairs` that takes a list of strings and returns a list of tuples where the first element of each tuple is a string and the second element is the length of that string. For example, `lengthPairs ["apple", "banana"]` should evaluate to `[("apple", 5), ("banana", 6)]`

3. **[10 point(s)] List recursion** For this problem your functions **must** use list recursion with pattern matching.

(a) Write a function `myLength` that takes a list (containing elements of any type) and returns the number of elements in the list. For example, `myLength ['a','b']` should return `2`.

(b) Write a function `duplicate` that duplicates each element of a list. For example, `duplicate [1, 7, 3]` should return `[1, 1, 7, 7, 3, 3]`. Note that your function should work with lists containing any type of element.

(c) Write a function `lessThanAll` that takes a number and a list of numbers and returns `True` if the number is less than every number in the list and `False` otherwise. For example, `lessThanAll 2 [1, 7, 3]` should evaluate to `False`. **Hint: pay attention to the base case! The number 10, for example, is in fact less than every element of the empty list.**

4. **[6 point(s)] Functions on lists and lists of lists**

(a) Write a function called `myZip` that takes two lists and returns a list of pairs. If the lists are of unequal lengths, the trailing elements of the longer list are discarded. You may **not** use the built-in `zip` function! (Hint: the recursion will be carried out on both lists simultaneously. Think about the base cases!)

As an example `myZip [1,2,3] [4,5]` should return `[(1,4), (2,5)]`.

(b) Write a function `consAll` that takes a list of lists and an element, and prepends the element to every member of the list of lists.

As an example, `consAll [[1,2], [], [3]] 8` should return `[[8,1,2],[8],[8,3]]` and `consAll [[]] "a"` should return `[["a"]]`.

5. **[5 point(s)] Cycles**

Write a function `cycleK` that rotates the elements in a list by a given amount. For example, `cycleK 1` is a function that removes the first element of a list and places it at the end. The call `cycleK 2` is equivalent to two consecutive calls to `cycleK 1`.

If the argument `n` in `cycleK n` is zero or another multiple of the length of the list, then `cycleK n` returns the list unchanged. If the argument `n` is negative, then `cycleK n lst` and `cycleK (length lst + n) lst` give identical results.

As an example, `cycleK 3 [10,20,30,40]` should return `[40,10,20,30]` and `cycleK 4 "I love CS"` should return `"ve CSI lo"`

There are many ways to solve this problem, but one possibility is to use a helper function that cycles the list by 1. You could then use that function in your general `cycleK` function. Other observations: the recursion is on the integer, not on the list. And the paragraph about what happens when `n` is negative or 0 suggests that using guards might be a reasonable approach.