# CMPT 383: Vitamin #2

Anders Miltner
miltner@cs.sfu.ca

## Introduction

This Vitamin is here to get you accustomed to working with algebraic data types like product and sum types, and with higher order functions.

This submission will be autograded. There are some portions of the assignment that are ungraded, and some that will be graded. We provide a (partial) test suite for partial validation. You can run these tests by opening a terminal in the v2 directory, and running `stack test`.

We have included explicitly some imports, and have omitted some. If you import additional functions, you may get a zero on the assignment.

## 1 ListHOFs

The ListHOFs file contains some stubbed-out functions on lists. Some of these functions are higher order functions, some are not higher-order functions, and some are most easily implemented using higher-order functions.

- `zip`: One useful thing to do with lists is zip them together. Let's build the zip function, that does exactly this. However, lists don't always have the same length, so when they have differing lengths, we should return Nothing. If they have the same length, we should return Just l, where l is the zipped list.
- `alternatingMap`: You've seen map before, so it's time to try building an "alternating map". An alternating map does the same thing that a map does, but it alternates which function is used at a time. The first function used should be the first function, then the second element should have the second function applied, and so on.
- `sumEvens`: Now you should sum the even elements of a list up. Can you do this by combining two other functions together. Notice what you've been provided in your imports. For full credit, you must implement this using higher-order functions.
- `flatten`: The last function is the flatten a list of lists into a single list. You must implement this using higher-order functions.

Test based specifications are provided in the ListHOFsTests.hs file.

## 2 TreeHOFs

The TreeHOFs file contains an algebraic data type formalization for trees, and some stubbed-out functions on these trees.

Some of these are the tree analogues of list functions, like fold and map. Looking at the definitions of those list-based functions may be hepful for defining these functions on trees.

You must implement:

- `treeMap`: First you should implement tree map. `treeMap` takes a tree, and updates all the values at the internal nodes according to the provided function. The structure of the tree should not change, but simply the values contained within the tree.

- `treeFold`: Next, we want to be able to fold our trees. `treeFold` is the tree analogue's of List's foldr. In particular, when we hit leaves, we return the provided initial "b". On nodes, we fold the subtrees, and use the provided higher- order function to fold the values computed by those trees with the value at the node.
- `treeHeight`: Now we would like to find the height of the tree. You must implement this with a higher-order function for full credit.
- `treeSum`: Next we would like to find the total sum of all the values in a tree of integers. For full credit, you must implement this using a higher-order function.
- `treeSizer`: Finally, we would like to turn the trees into a "sized tree." On the nodes of the sized tree are pairs of initial values and sizes of the trees. The size of a Leaf is 0, and the size of a Node is the sum of the sizes of the subtrees plus 1. This one is trickier, but still possible to do with a fold. Hint: you may need to use a higher order function that returns a pair of values, then project the relevant portion of the pair. In a way, this is analogous to strengthening an inductive hypothesis. For full credit, you must implement this using a higher-order function.

You can implement these any way you wish. However, doing everything with pattern matching and recursion, rather than instantiating higher-order functions will become harder and harder as the class goes on – it is better to try using a relevant higher-order function when one is available.

Test based specifications are provided in the TreeHOFsTests.hs file.