Problem 1. Combining the functional programming techniques from Lab 4, write functions to solve the following problems:

1. Given an amount of money which needs to be counted into change, and a finite list of currency denominations, count the number of unique ways of making change. (e.g. In GYD, we can make change for $10 in 4 ways - one 10-dollar coin, two 5 dollar coins, one 5-dollar coin and 5 1-dollar coins, and ten 1-dollar coins).

2. Given a document, write a function to lowercase it, remove x all punctuation and split it into individual words. Then write another function to build an index, associating each word with all its positions in the document.

3. Implement the RSA Cryptosytem. Assume that messages are small integers (padding is not required) and implement the following three functions: one that generates a new public/private key-pair based on given parameters(You do not need to randomly generate the primes, but may choose to do so as an optional improvement), one that encrypts a message with a public key and a corresponding function to decrypt a cyphertext using the public key and the corresponding private key.

Problem 2. You will spend a large amount of your time as a programmer reading code - often very bad code. The file *reading_code.hs* contains code which has had all meaningful variable and function names removed. It is your task to work out what each bit of code does, and rewrite it so that it is apparent at a glance. Bonus points if you can do so without running the code. Finally, refactor the code to fix any glaring issues you find and introduce a more readable style.

Problem 3. It is important to be able to reason about properties of programs. We may wish to prove that a function we have written correctly represents a given computation, or that a function we are using has certain properties we can take advantage of. You have seen this before with proofs of runtime complexity. Using the tools of reasoning described in lectures:

1. Prove that `(++)` is associative if both its arguments are finite lists.

2. Show that `zip` always terminates so long as at least one of its arguments is finite.

3. Show that reverse `(reverse xs) = xs` holds true for any arbitrary finite list xs.

4. Prove the correctness of q4 in `reading_code.hs`

5. Prove the correctness of your solution to Problem 1.1. Prove that your solution terminates for any valid input.