Section Handout 1

Based on a handout by Eric Roberts

Sections will meet once a week to give you a more intimate environment to discuss course material, work through problems, and raise any questions you have. Each week we will hand out a set of section exercises, which are the problems proposed for section that week. While we will not be collecting or grading these problems, you will receive greater benefit from section if you've looked over the problems in advance and tried to sketch solutions. Your section leader won't necessarily cover every exercise in depth, so be sure to speak up if there are particular problems you'd like to focus on. Solutions to all problems will be given in section so you can work through any remaining exercises on your own. Many of the section problems have been taken from old exams, so they are also an excellent source of study questions when exam time rolls around.

Problem 1: Random Shuffling

How might the computer shuffle a deck of cards? It turns out that this problem is a bit more complex than it might seem, and while it's easy to come up with algorithms that randomize the order of the cards, only a few algorithms will do so in a way that ends up generating a uniformly-random reordering of the cards.

One simple algorithm for shuffling a deck of cards is based on the following idea:

- Choose a random card from the deck and remove it.
- Shuffle the rest of the deck.
- Place the randomly-chosen card on top of the deck.

Assuming that we choose the card that we put on top randomly, this ends up producing a random shuffle of the deck.

Write a function

```
string randomShuffle(string input)
```

that accepts as input a string, then returns a random permutation of the elements of the string using the above algorithm. Your algorithm should be recursive and not use any loops (**`for`**, **`while`**, etc.).

Interesting note: this shuffling algorithm is a variant of the *Fisher-Yates Shuffle*. For more information on why it works correctly, take CS109.

Problem 2: Computing Averages

Your job is to write a function

```
void averageValueInFile(string filename, double& result);
```

that accepts two parameters. The first parameter, **`filename`**, gives the name of a file that contains a list of real numbers, one on each line. The second parameter, **`result`**, is an outparameter (that is, a parameter passed by reference that should be updated to hold the result) and should be updated to hold the average value of all the numbers in the file.

You can assume that the file exists and that it contains at least one line. As mentioned in Chapter 4 of the course reader, to create an **`ifstream`** that reads from a filename held in a C++ **`string`**, write

```
ifstream input(str.c_str());
```

Problem 3: Haiku Detection

A *haiku* is a three-line poem where the first line has five syllables, the second has seven syllables, and the final line has five syllables. For example, the following is a haiku:

An observation:
Haikus are concise, but they
Don't always say much.

while this is not:

One two three four five
Six seven eight nine ten, then
eleven, twelve, thirteen!

Because the last line has six syllables.

Your job is to write a program that reads three lines of text from the user, then checks whether those lines form a haiku. You can assume that you have access to a function

```
int syllablesIn(string word);
```

which returns the total number of syllables in a word. You can assume that the input text consists solely of words, spaces, and punctuation marks (e.g. commas and exclamation points).