Section Handout 1

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: Censorship

Your first problem is to design and implement a function `censorString` that takes two strings as input and eliminates from the first string all characters that are present in the second. Thus

> `"Stanford University"` with `"nt"` removed becomes `"Saford Uiversiy"`
> `"Llamas like to laugh"` with `"la"` removed becomes `"Lms ike to ugh"`
> and so on . . .

Note that the function is case sensitive, so that the uppercase `L` in `Llamas` stays there.

You could design this function to operate in two different ways. One way is to have the function return a new string, leaving the original string unchanged. A second way is to write a procedure that modifies the original string. For this section, try writing both of these versions. First, write a function that returns a completely new string with the following prototype:

```
string censorString1(string text, string remove);
```

When you complete the first version, write a function that modifies the original string instead. This version should have the following prototype:

```
void censorString2(string & text, string remove);
```

Problem 2. How Did We Do?

When we grade your exams, we typically keep track of various statistics like the minimum, maximum, and mean (i.e., the traditional average) scores. Write a function

```
void readStats(string fname, int & min, int & max, double & mean);
```

that takes the name of an input file and three reference parameters that will hold this statistical information. Your implementation should open the file, read the scores one line at a time, close the file, and then return to the client with the values of `min`, `max`, and `mean` correctly set. For efficiency's sake, your function should make only a single pass over the file. Your implementation should also call `error` with an appropriate message if the file does not exist or if any of the scores are not in the range 0 to 100.

Problem 3. Stacking Cannonballs (adapted from Chapter 7, exercise 1, page 346)

Suppose that you have somehow been transported back to 1777 and the Revolutionary War. You have been assigned a dangerous reconnaissance mission: evaluate the amount of ammunition available to the British for use with their large cannon, which has been shelling the Revolutionary forces. Fortunately for you, the British—being neat and orderly—have stacked the cannonballs into a single pyramid-shaped stack with one cannonball at the top, sitting on top of a square composed of four cannonballs, sitting on top of a square composed of nine cannonballs, and so forth. Unfortunately, however, the Redcoats are also vigilant, and you only have time to count the number of layers in the pyramid before you are able to escape back to your own troops. To make matters worse, computers will not be invented for at least 150 years, but you should not let that detail get in your way. Your mission is to write a recursive function `numCannonballsInStack` that takes as its argument the height of the pyramid and returns the number of cannonballs therein.

Problem 4: Xzibit Words

Some words contain other words as substrings. For example, the word "pirates" contains a huge number of words as substrings:

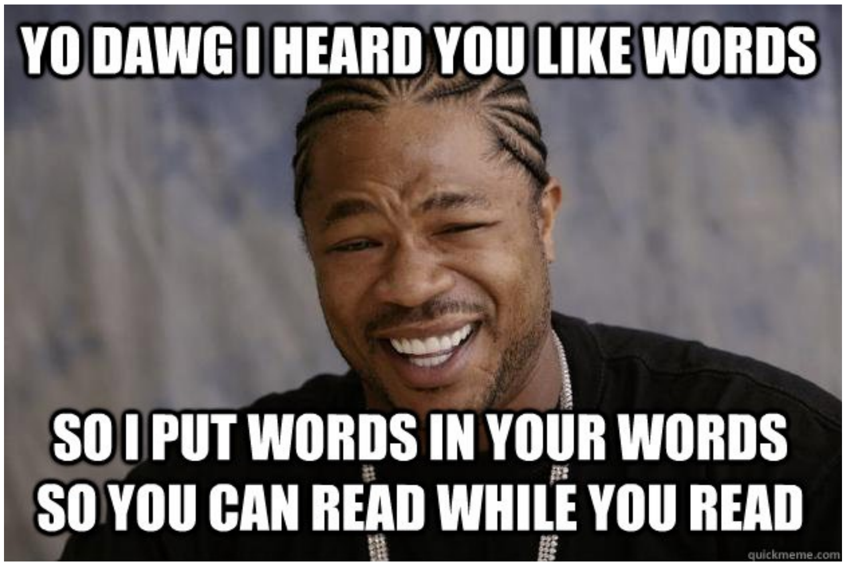
a	I	rat
at	irate	rate
ate	pi	rates
ates	pirate	
es	pirates	

Note that "pirates" is a substring of itself. The word "pat" is not considered a substring of "pirates," since even though all the letters of "pat" are present in "pirate" in the right order, they aren't adjacent to one another.

Let's call a word an "Xzibit word" if it contains a large number of words as substrings. Write a function

```
string mostXzibitWord(Lexicon& words);
```

that accepts as input a `Lexicon` of all words in English, then returns the word with the largest number of words contained as substrings.


YO DAWG I HEARD YOU LIKE WORDS
SO I PUT WORDS IN YOUR WORDS
SO YOU CAN READ WHILE YOU READ
quickmeme.com

Problem 5: RNA Protein Codes

RNA (ribonucleic acid) is a chemical that can encode genetic information. Plant and animal cells use RNA for a variety of cell functions, while viruses often use RNA as their primary genetic storage.

Each strand of RNA consists of a series of nucleotides – adenine (A), guanine (G), uracil (U), and cytosine (C) – and a strand of RNA can be thought of as a string over those four letters. Because of this, computational geneticists often treat DNA and RNA as strings and run algorithms on those strings to deduce their properties.

RNA is used by a cell to encode **proteins**, biological molecules essential to cell function. Each protein consists of a series of **amino acids** that, strung together, serve some complex purpose. The actual letters in an RNA strand spell out what amino acids need to be combined together to produce the overall protein. So how exactly are these amino acids represented? In RNA, letters are grouped into clusters of three letters called **codons**. Each codon encodes a specific choice of amino acid. When decoding RNA, the cell reads these codons in sequence and assembles the protein one amino acid at a time. For example, the RNA strand

```
GGGAUGAAUAUCUCGGCG
```

would be treated at this sequence of three-letter codons:

```
GGG   AUG   AAU   AUC   UCG   GCG
```

The cell would then use the codons to determine what amino acids to string together into a protein. In this case, these codons represent the following sequence of amino acids:

GGG	AUG	AAU	AUC	UCG	GCG
Glycine	Methionine	Asparagine	Isoleucine	Serine	Alanine

So the generated protein would have amino acids ordered as glycine, then methionine, then aspargine, then isoleucine, then serine, and finally alanine.

An important detail is that each strand of RNA does not encode just one protein; typically, a single strand of RNA encodes many different proteins. How, then, does the cell know where each protein starts and stops? There is an ingenious system set up for just this purpose. In an RNA strand, the codon `AUG` has two meanings – it can code for methionine (as shown above), but it also acts as a special "start of protein" marker. As a cell scans across an RNA strand, if it encounters the codon `AUG`, it begins assembling a protein starting at that location. It then continues to assemble the protein one amino acid at a time by decoding codons in sequence. The cell stops assembling base pairs when it encounters one of three "stop" codons (`UAA`, `UAG`, or `UGA`) that act as an "end-of-protein" marker. The cell can then keep scanning across the RNA until it finds another `AUG` start codon, at which point the process repeats. For example, consider this RNA strand:

```
GCAUGGAUUAAUAUGAGACGACUAAUAGGAUAGUUACAACCCUUAUGUCACCGCCUUGA
```

This would be decoded as follows:

GC	Skip letters until we find AUG.
AUGGAU**UAA**	Read from AUG until we hit a stop codon (here, UAA)
U	Skip letters until we find AUG.
AUGAGACGACUAAUAGGA**UAG**	Read from AUG until we hit a stop codon (here, UAG)
UUACAACCCUU	Skip letters until we find AUG.

<u>**AUG**</u>`CACCGCCU`<u>**UGA**</u>	Read from AUG until we hit a stop codon (here, UGA)

Your job is to write a function

```
Vector<string> findProteins(string& rna, Map<string, string> codons);
```

that accepts as input a string of RNA and returns a **`Vector`** of all the proteins that would be formed from that RNA. The first parameter represents the actual string of RNA, and the second parameter is a **`Map`** from three-letter codons to the name of the amino acid they represent (or the special string **`"stop"`** if the codon is a stop codon). For example, running this function on the above RNA strand would return the **`Vector`** holding the stings

methionine, asparitic acid	(encoded by `AUGGAU`**`UAA`**)
methionine, arginine, arginine, leucine, isoleucine, glycine	(encoded by `AUGAGACGACUAAUAGGA`**`UAG`**)
methionine, serine, proline, proline	(encoded by `AUGUCACCGCCU`**`UGA`**)

You can assume that all the proteins are properly terminated, which means that if you find an `AUG` codon then there will be a matching stop codon before the end of the protein.