CS 1428
Fall 2018
Gentry Atkinson
Lab 8

Introduction:

Last week we started an investigation of functions and their uses in programming. Functions though, are much broader than what we were able to cover in a single lab. This week we will focus of refining our understanding of functions.

Lab 7 mentioned that functions do not need to have a unique name, only a unique "signature" which is a combination of the functions name and its parameters. So for instance **void foo (int a)** and **void foo (char c)** are recognized by the compiler as different functions. Utilizing this feature is called "overloading" and it allows us to right several versions of the same function in order to make it more useful and general-purpose.

Another refinement we can apply to our understanding of functions is to further analyze how values are passed into functions as parameters. Specifically we need to understand when a function is able to alter a variable which has been passed to it. Additionally we need to understand the "life span" of variables in a computing environment and where within a program a variable might or might not be visible.

The purpose of today's lab is to introduce you to function overloading and variable scope.

Directions:

1- Launch Code::Blocks and start a new file. Name it your_last_name_lab8.cpp.

2- Include the standard header for this lab:

```
//Your Name
//CS1428 Fall 2018
//Lab 8
```

3- Include the iostream, string, and fstream standard libraries and declare some functions that we will be using in this lab. Start your main function:

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
```

4- You are working for a client that has come into possession of of an encrypted message that they need to be able to read. The document appears to be a string of numbers rather than letters. The first step towards cracking the code is to count how frequently each number appears in the document. Fill in the following two functions to accomplish this:

> **Parameters:** either an open filestream or the name of a valid input file, countArray should be an integer array which will hold the count performed by the function.
> **Precondition**: a file containing a string of integers is accessible by the function. The range of values in the file should be the same as the size of countArray.
> **Postcondition:** countArray will be loaded with counts of each different integer that appears in the input file. So if the number 15 appears 20 times then countArray[15] should equal 20. If a number does not appear in the file then the value at its index should be 0.
> **Return:** false if the file cannot be read, if improper parameters have been passed to the function, or if the count cannot be performed. True otherwise.
>
> **bool countIntegers(string fileName, int * countArray);**
> **bool countIntegers(filestream file, int * countArray);**

5- After the countIntegers functions has run, we will have an array loaded with the total occurences of each number in the file. The next step in our "frequency analysis" of this file is to determine which number occurs the most frequently. To do this we need a function that looks at an array and determines **the index** of the biggest value in the array. **Select one of the following functions and implement it in order to accomplish this:**

> **//this function takes an array as input and returns the index of the largest value**
> **int findMostCommon(int * countArray, int size);**
>
> **//this function takes an array and an integer as input then sets the integer to the index of the**
> **//largest value in the array.**
> **void findMostCommon(int index, int * countArray, int size);**

6- After reviewing your analysis your client believes that the document was written with a "Caesar Shift" cipher where the position of each letter is recorded as a number and shifted by a fixed amount. So with a shift of 0 A = 0, B = 1, C = 2… and with a shift of 8 then A = 8, B = 9, C = 10… and so on. We also now that E is the most commonly used letter in the English language. So a document written with an unshifted alphabet would have more 4s than any other number since E is the 5th letter of the alphabet. Implement the following function to determine the shift used to write your mystery document:

> **Parameters:** mostFrequent is an integer which represents the index of the highest value in some array.
> **Precondition:** none
> **Postcondition:** none

Return: the shift required to produce a readable document from the cipher being analyzed. Can be positive or negative. So for example if text is coded with a shift of 7 then 11 should be the most common number in the document and this function will return a shift of -7 in order to break the encipherment.

int determineShift(int mostFrequent);

7- Finally, you need to fill in your **main** function in order to make everything work. Start by declaring the following variables:

int countArray[26];
int mostFrequent;
ofstream outFile;
ifstream inFile;
int input;
int shift;

After this you should run countIntegers, findMostCommon, and determineShift in order to load the correct value into your shift variable. Now copy the following code in order to write the broken code into its own file:

```
outFile.open("lab8_plainText.txt");
inFile.open("lab8_cipherText.txt");

while (!inFile.eof()){
        inFile >> input;
        input += shift;
        outFile << static_cast<char>(input + 'A');
}
```

8- Compile and run the program you've written so far. Open your new plain text file. There are no spaces between any of the characters but you should be able to make out what it says. Print the author's name by adding a **cout** statement to your main function.

9- Build and run your code again. Correct any errors. Your output file should look something like this: