CS 1428
Fall 2018
Gentry Atkinson
Lab 6

Introduction:

Up to now the only input we've had for our programs has come from the console. But often times we will want to input so many values that it would be tedious to type them all out. Or we will want to use the same inputs over and over without having to retype them. In these cases, we can take inputs from files rather than rather than from the console.

File I/O in C++ can be accomplished using the **<fstream>** library, which works much like the **<iostream>** library that you're used to. Values are "extracted" from the file stream with the >> operator and and "inserted" into the stream with the << operator. The main difference is that we must explicitly open a file I/O stream unlike **cout** and **cin** which are already created for us.

The purpose of today's lab is to introduce you to file operations and to integrate those operations with the techniques we've already learned so far.

Directions:

1- Launch Code::Blocks and start a new file. Name it your_last_name_lab6.cpp. Save lab6_data.txt from TRACS into the same directory as your program.

2- Include the standard header for this lab:

```
//Your Name
//CS1428 Fall 2018
//Lab 6
```

3- Include the iostream and fstream standard libraries and start your main function:

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

4- You've been hired by a small business to do some simple data analysis. They've been counting the daily page views for their website and want to know how the numbers are trending. They have given you a file with all of the numbers they've collected seperated with a single blank space. **The first number in this file is the number of entries in the set!** Everything after is a data point that you need to analyze. Debug the following code written by their in-house developer to load the data into an array:

```
// This part is correct!

fstream dataFile;
dataFile.open("lab6_data.txt", fstream::in);
int setSize =  0;
dataFile >> setSize;

//After this is broken!

int[setSize] data;
int counter;
while (conter <= setSize) {
        dataFile << data[counter];
        counter+;
}
```

5- Implement the following pseudo-code to start your analysis of your client's data:

```
//1. DECLARE ANOTHER FILESTREAM TO HANDLE YOUR PROGRAMS OUTPUT
//2. OPEN THE FILESTREAM FOR OUTPUT WITH THE FILENAME "lab6_output.txt"
//3. DECLARE TWO VARIABLES CALLED maxViews AND minViews
//4. DECLARE ANOTHER VARIABLE TO TEMPORARILY HOLD VALUES FROM data
//5. INITIALIZE minViews WITH A VERY LARGE VALUE. INITIALIZE maxViews
// WITH A VERY SMALL VALUE
//6. ENTER A LOOP WHICH WILL COUNT THROUGH THE INDEXES OF data
        //7. ASSIGN YOUR TEMP VAR TO EQUAL THE VALUE IN THIS INDEX OF
        //data
        //8. IF THE VALUE IN data AT THIS INDEX IS GREATER THAN maxViews,
        //UPDATE maxViews
        //9. IF THE VALUE IN data AT THIS INDEX IS SMALLER THAN minViews,
        //UPDATE minViews
//10. EXIT LOOP
//11. WRITE maxViews AND minViews INTO THE OUTPUT FILE WITH AN
//APPROPRIATE LABEL
```

6- Your client can see from your output file in the last problem that their maximum number of page views is nearly twice as big as their minimum. Now they want to know if they usually have more views on one day than they did on the previous. They would like you to write a loop which will step through the whole data array and compare each 2 consecutive values (i.e. your first comparison will be data[0] and data[1] and your last comparison will be data[size-2] and data[size-1]. If the bigger index has a larger value, increment a counter for the days that the page views increased. If the bigger index has has a smaller value, increment a counter for the days that page views decreased.

After this loop has executed you should write the string **“This data is generally increasing.”** to your output file if the the increasing counter is larger than the decreasing index. Write the string **“This data is generally decreasing”** if your decreasing counter is larger.

Your code for this problem should include **comments** for each action it performs to help your client maintain the code once you’ve turned it over to them. You should write a comment for each **action**, not necessarily each line.

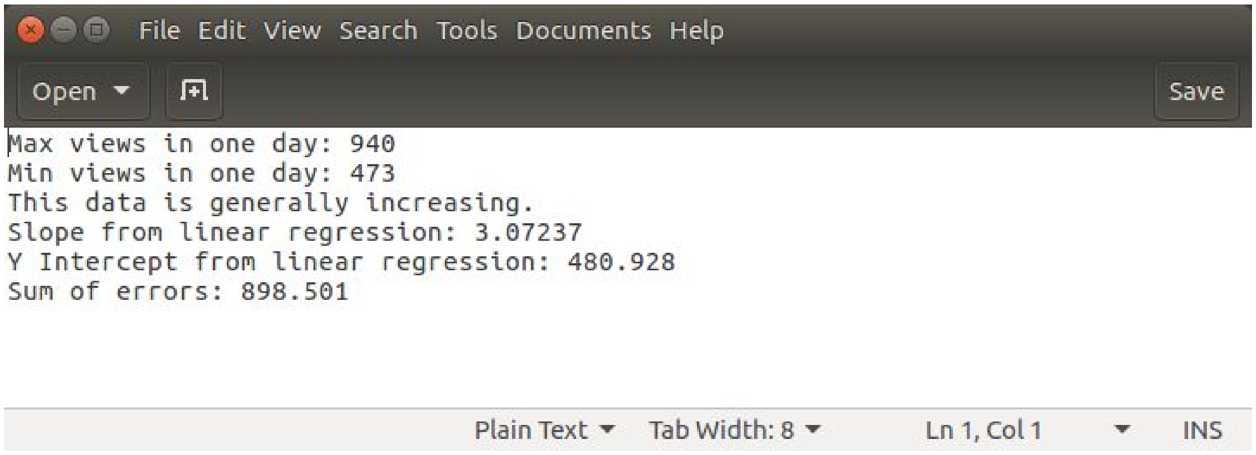
7- The final piece of you data analysis will be to apply a technique called Linear Regression. This means creating a line which we think will lay neatly over the data. To do this we will calculate 3 things:

1. The slope of this line using the equation: $(y_{max} - y_{min}) / (x_{max} - x_{min})$.
2. The y-intercept of this line using the equation: $y_0 = y_1$ - slope.
3. The sum of errors, which will tell us the sum of all the differences between our line and the actual data. On a well fit line this number will be near zero. **Note: in an actual analysis, we would use the sum of squared errors rather than the sum of errors. Can you think of why?**

Implement the following pseudo-code to perform your linear regression:

```
//1. DECLARE VARIABLES OF AN APPROPRIATE TYPE TO HOLD THE SLOPE,
//Y-INTERCEPT, AND SUM OF ERRORS
//2. ASSIGN SLOPE VALUE EQUAL TO maxViews - minViews OVER setSize
//3. USE THE SLOPE AND data[0] TO CALCULATE THE Y-INTERCEPT
//4. ENTER A LOOP WHICH WILL COUNT THROUGH ALL INDEXES OF data
        //5. CALCULATE THE DIFFERENCE BETWEEN slope*i + y-intercept AND data[i]
        //6. ADD THIS NUMBER INTO YOUR SUM OF ERRORS.
//5. END LOOP
//6. WRITE THE SLOPE, Y-INTERCEPT, AND SUM OF ERRORS INTO THE OUTPUT
//FILE WITH APPROPRIATE LABELS.
```

8. Close your file streams. Build and run your code. Fix and errors. Your output file should look something like this:



```
Max views in one day: 940
Min views in one day: 473
This data is generally increasing.
Slope from linear regression: 3.07237
Y Intercept from linear regression: 480.928
Sum of errors: 898.501
```

9. Submit your .cpp file through TRACS. You do not need to attach the output file. You can leave when you're done.