Programming Assignment #2
CS253, Fall 2016
Due: Tuesday, Sept. 13th

"*Comparing Histograms*"

Motivation

Last week you read in and generated a histogram. This week you will read in, generate and compare two histograms. I realize that for many of you, this may seem a little opaque. *Why* are we making histograms? *What purpose* do they serve?[1] Trust me, that will start to make more sense next week (Assignment #3). In the meantime, hang in there.

What should be clear from this assignment is that it builds on the 1st assignment, and that as a result, modularity is rewarded. If you wrote Assignment #1 by putting a lot of code in main and/or using global variables, Assignment #2 will be harder. (Not to mention subsequent assignments.) After all, you now need to open and read from two files, producing two histograms, etc.

As always, remember that generating test cases and testing code is your responsibility, and that you should never trust a user or a file.

Task

Your program will take two file names as arguments. (In other words, the argc parameter in main should now be 3, and argv[1] and argv[2] will hold file names.) The input files are in the same format as Assignment #1: each file should contain 1 or more values, where a value is an integer between 0 and 255. If either file does not conform to this specification, you should print an error message to std::cerr and return -1 from main.

This time, your program will actually do something with the histograms. First, it will normalize each histogram by summing up the bins counts in the histogram (this should be the same as the number of integers in the file), and then dividing the counts by this sum. Note: histogram counts should now be doubles or floats, otherwise when you divide by the total count, most values become zero. The result of histogram normalization is that the bucket counts are now percentages. For example, if the first count in the first histogram is now 0.05, it suggests that 5% of all the values in the first file are in the range 0-3.

Note that you normalize each histogram independently. The goal is to make is possible to compare histograms created from different lengths of files.

After you have normalized both histograms, you will multiply them together element-wise. In other words, you multiply the first count in the first normalized histogram by the first count in the second normalized histogram, the 2nd element in the first normalized histogram by the second

[1] For those of you who have had statistics, a histogram can be viewed as a discrete sampling from an underlying, unknown probability distribution. Although this factoid will not help you directly, it gives you another way to look at the first several assignments in this class.

element in the second normalized histogram, and so forth. This will result in 64 values. Sum these 64 values, print the result to std::cout, and return 0 from main.[2]

Submitting Your Work

To submit your program, first create a single tar file containing all of your source files (i.e. your .cpp and .h files) but not your compiled files (no executable or .o files, please). This file should be "flat", i.e. it should not contain directories. (Think about the compile command below; it will not work for subdirectories. Once we start including make files, then you can use directories to your heart's content.) Then submit the tar file as PA2 on Canvas.

Grading Your Homework

To grade your assignment, the GTAs will unpack your archive in an empty directory and compile it using "g++ -I. –Wall *.cpp –o exec2". If your program does not compile, you will be awarded no points for the assignment. If the compiler produces any warning messages, points will be deducted (up to 10% of the total, depending on the number and severity of the messages). Assuming your program compiles, the remaining 80 points will be determined by how your program performs on test cases, including test cases with errors in the input files. When the input files do not contain errors, you will receive full credit for a test if the value printed to std::cout are correct. If an input file does contain errors, you will receive full credit if your program produces a meaningful error message and returns -1. Note that unless there is an error, the ONLY thing you output to std::cout or std::cerr should be the computed value.

Hints

- Once again, never trust a file or a user. You can be sure that we will include test cases with illegally formatted input, illegal arguments, etc.
- Use internal consistency checks. For example, after normalization, the sum of all the buckets in a histogram should be 1.0. After normalization, every bucket count should be between 0.0 and 1.0. The final value you print to std::cout should always be between 0 and 1.0.
- Do not be afraid to go back and fix Assignment #1 if its design isn't good. Better to fix it now than to nurse it along for another week or two, only to have to redesign it when the system is bigger.
- Once again, test your program on the department machines in CSB120. That is where we will evaluate it. Your grade depends on how your program performs on those machines. (Warning: C++ is not as portable across platforms as Java.)

Policies

All work you submit must be your own. You may not submit code written by a peer, a former student, or anyone else. You may not copy or buy code from the web. The department academic integrity policies apply.

You may not submit your program late. To receive credit, it must be submitted on Tuesday, Sept. 13th. There is no late period. The exception is an unforeseeable emergency, for example a medical crisis or a death in the immediate family. If an unforeseeable emergency arises, talk to the instructor.

---

[2] For those of you who have taken M229 or higher, you are taking the dot/inner product of the two normalized histograms (which are vectors).