Lab 7: Array processing

Overview

This lab will give you practice with arrays, sorting and searching, and file I/O. The program will read input data from a file and produce its output in a file. The only interaction with the user will be limited to asking for the names of the files[1].

More About Opening Files

So far you have only used the `open()` member function of the `fstream` class to open files with fixed names, as in

```
ifstream fin;
fin.open(“dna.txt”);
```

which will attempt to open a file named `dna.txt` in the program's current working directory when it runs. If you want to ask the user for the name of a file to open, store it into a `string` variable, and then use it to open that file, you cannot write this:

```
ifstream fin;
string filename;
cin >> filename;
fin.open(filename); // error -- this will not work
```

because the `open()` function expects a different type of string called a ***C string***. C strings are the progenitors of C++ strings – they were defined in the C language originally and became part of C++ by the fact that they are part of C. People call them C strings to distinguish them from C++'s `string` class. You are not going to learn about C strings here[2], but you will learn how to circumvent this problem.

The `string` class has several functions that can be called on its `string` objects. You know about the `size()` function already. It also has a `c_str()` function. This function, whose name is pronounced "C string", does exactly what the doctor ordered for this problem – it returns the C string equivalent of the string itself.

For example, if `filename` is declared to be a `string`:

```
string filename;
```

and if the string “`../../data/mydata.txt`” is stored into it via an input instruction such as

```
cin >> filename;
```

then `filename.c_str()` will be the C string “`../../data/mydata.txt`”. This means you can use it as the argument of `open()` as in the following:

```
ifstream fin;
string filename;
cin >> filename;
fin.open(filename.c_str() ); // this will open the file whose name the user entered
```

This is how you can ask the user to enter a file name and then try to open that file. Of course you need to check that the `open()` succeeded. This expands your programming toolkit for opening files.

[1]In a future lab exercise, you will learn how to write programs that can get these file names directly from the command line.
[2]You first need to learn about pointers.

Exercises

Problem 1. You will write a program that prompts the user to enter the names of an input file and an output file, and then reads input from the input file, processes the input, and sends its output to the output file. If the input files cannot be opened, or if the output file cannot be opened, the program returns with an error value (1). The input file will consist of real numbers separated by white space (blanks, tabs, newlines). These numbers should be thought of as sample values from some experiment. The numbers in the file are not in any particular order. The program will compute various statistics about the file's data and write these one per line into the output file in the order listed below:

1. the *smallest* number in the file,
2. the *largest* number in the file,
3. the *median* of the numbers in the file,
4. the *mean* of the numbers in the file, and
5. the *standard deviation* of the numbers in the file.

The *median* of a set of sample values (i.e., numbers) is the numerical value for which half of the numbers in the set are larger and half of the numbers in the set are lower. If there is an odd number of samples in the set, the median is one of the values. For example, if the list of numbers is 2.3, 8.4, 9.3, 10.0, 12.0, then the median is 9.3, since there are two values below 9.3 and two above it. If the set has an even number of values, the median is halfway between the two middle values. For example, if the list is 2.3, 8.4, 9.3, 10.0, then the median is halfway between 8.4 and 9.3, i.e., the average of 8.4 and 9.3, which is 8.85.

The *mea*n is the sum of the values divided by the number of values. (It is the same as the average value.)

The *standard deviation* is defined as $\sqrt{\left(\sum_{i=0}^{N-1}(x_i - a)^2\right)/N}$, where $x_0, \ldots, x_{N-1}$ are the N values, and a is their mean. In order to compute the standard deviation, it is easiest to first compute the mean. If you have somehow made it into this class never having seen the summation symbol $\sum_{i=0}^{N-1}(x_i)$ it means the sum of the values $x_0, x_1, ..., x_{N-1}$.

You can assume that the file contains at most 1000 numbers. It might contain fewer, but never more, than that. It is possible that the file is empty, in which case the output file should be created and should be empty as well, but an error message should be written to the standard error stream (`cerr`).

Advice. Although there are algorithms that can find the median of a list of unordered numbers, the simplest solution is to first sort the numbers. Your program should sort the numbers before doing anything else. This will make the other computations much easier.

What to Submit

Submit your program, however complete it is, by the end of today's lab, i.e., before the end of the class at 2:00 P.M. ***There is no grace period for this. Programs submitted after 2:00 PM will not be accepted.*** The instructions for submitting are:

1. Create a directory in
   `/data/biocs/b/student.accounts/cs135_sw/cs136labs/lab07/submissions`
   whose name is your *username*. For example, I would create the directory `sweiss`.
2. Copy your program, which should be named `*username*_lab07.cpp`. You will lose 5% of the grade if you misname the file!
3. Change the permission on the directory that you created so that no one else can read or modify it. You do this with the commands

```
$ cd /data/biocs/b/student.accounts/cs135_sw/cs136labs/lab07/submissions
$ chmod 700 username
```

Do not submit executable files. Remember to document your code (both preamble and comments in the code) and make it easy to read. Your work will be graded based on the rubric outlined in the Programming Rules document.

There are absolutely no extensions to the deadline. You can submit a revised version of the program by 10:00 P.M. on Thursday, Oct. 11 for partial credit. If you do, you must name it with a different name than what you first posted, e.g., `username_lab07_v2.cpp`. and put it in the same directory as the original. It must be a *revision* of the original file, with only fixes to things that were not working before. I will adjust the grade based on how well the first version worked and how many changes you made.