Dephy Coding Assignment

## Design Rationale

My first consideration was deciding on a data structure for each data point. My first thought was to break each field into its own variable in the structure. This provides good “self-documenting” code by explicitly accessing data members by their name. This is fine, but it means I have to write a lot of duplicated code to access each member of the struct and prevents me from easily modifying the field since I would need to modify the struct and any code referencing it. To improve the maintainability and simply future modification of the data structure I decided to put sensor data into its own vector so I could iterate through any number of int data fields in my code. I see this as a slight tradeoff from readability for maintainability, but the benefit in future modifications justified this decision.

Another decision in this design was deciding how to output data to a csv file. My initial thought was to write a function that takes parameters for a pointer to the data collection and a string for the file name and open a stream within the function to output the data. My problem with this was the lack of portability to output the data through other means such as through a terminal. For this reason, and in the spirit of OOP, I decided to overload the insertion operator so I could feed the data to whatever stream I wanted down the line and output the data to the terminal to simply debugging. I considered overloading the extraction operator to input data into the data object, but since I wanted to retain the file name in the datafile structure, and I didn’t see much functional benefit other than symmetric function overloading, I ended up just writing a function for input.

Another consideration was the data and time efficiency of parsing data from files of different lengths. An example of this problem is brought to light when I considered the following use case: if a user imported a file of length 5 then a file of length 10 the program would have to insert dummy values into the csv to offset the data to the correct column for the second file. Functionally this is probably fine, but it is less efficient than if the data were ordered greatest to smallest such that once all the data is added from the shortest file, it is removed from the output loop and no offsets need to be added to the output stream to index the data to the correct set.

The next problem I confronted was deciding how I wanted to calculate the different statistics for each data file. I first considered writing a class that would inherit from the Data class, but I didn’t see inheritance as the best solution because it linked the two classes that I wanted to keep as distinct and generic as possible. I decided to have the statistics structure in a static class that would contain templated functions to calculate statistics. This pushes knowledge of the Data class to the implementing function “main” thus preserving some level of encapsulation between the Data and DataProcessor class.

An ideal solution for detecting peaks would dynamically tune its parameters to find peaks in data with different characteristics than the samples I was given. For the purposes of this assignment, I’ve kept the peak detection algorithm simple, using an algorithm which I’ve referenced in my code. An additional simplification was locating corrupt data. I used my peak detection function with different parameters to better detect what I considered to most likely be corrupt data.

I am aligning my data off the accelerometer X-axis data for each file and verified the success of my program by generating the following scatter plots.

The plot on the left shows the data before alignment, while the data on the right shows it after being processed through my program. I also checked the peak array to verify it was detecting other peaks in the files.

## Build Instructions

This program is configured with CMake. For instruction configuring the program for your operating system check out the instructions in this link: <https://preshing.com/20170511/how-to-build-a-cmake-based-project/>

CMake will configure the software for your system so you can use the build tools on your OS to build the application. After building the application the executable is located under “csv\_aligner\Debug” in the build directory you selected in CMake.