Justin Kleiber

EclipseR 1.0 Design Document

In order to maintain the Eclipse data in an array, I stuck with my ResizableArray class from Lab4. However, I made some modifications and upgrades. I added a swap function to facilitate the swapping of elements at 2 particular indices. Additionally, I created a string variable to provide more information on what is being contained in the array, and used this string to store the header from the data files.

Furthermore, I updated my Eclipse object from Lab 4 to hold an array of an object called Cell instead of the strings it held in an array. Cell is a new object that I created that keeps track of its own data type (int, double, or string) and has overloaded comparison operators to make sorting and searching for values much easier. Additionally, the Eclipse object now has an array of type bool to signal which of its columns have errors (if any), so the validation checker does not have to do as much work to print the errors. Even more, the Eclipse object now contains the raw input used to create the object in order to output to the file or stdout easily.

All of my sorting and searching are in the EclipseOperations class. This class provides two functions on public access: find() and sort(). These serve as jumping off points for search algorithms and quicksort respectively. In search, if the current column to search is not sorted, my program performs linear search. If the sorted column and searched column match however, I perform a modified binary search to find all relevant values. My modified binary search calls recursively until the value at its midpoint is found to be the searched value. Upon finding the correct value, my binary search linearly works backwards to find the first value in the list that is equal to the searched value. From there, all eclipses are recorded in a ResizableArray until one of the eclipses is not equal to the search requirements. Then the search pattern ends.

With sorting, I implemented quicksort with a central pivot to avoid worst case scenarios. I found that my implementation of Cell made it extremely easy to compare column values of the same type during the sorting process, which resulted in much cleaner code.

Another item of note is that I moved all my file input operations to a FileInput class. Two new functions were created to convert strings to integers and doubles, which made it easy to manipulate data. This was relatively annoying since I am not allowed to use stringstream, but I stuck to the spirit of the assignment and converted the strings to numbers manually. The FileInput class was used to track the total number and number of valid Eclipse objects read into EclipseR.

Finally, my EclipseR driver files got way smaller with the increased encapsulation. There are three functions in EclipseR.cpp: main(), dataInputLoop() and dataManipLoop(). Each of these are responsible for the two stages of the program, the transition betweeen the two, and the graceful exit. These loops ultimately use the above described objects to make EclipseR 1.0 work effectively.