CS214 Project 1: Multiprocess CSV Sorter Documentation
Due Date: 10/25/2018

Revanth Korrapolu
Andrew Russomagno

Design:

Part 1 (search):

Built and created two functions: search() and sort(). Search() will recursively look for all .csv files in the directories provided as an argument. If Search() finds a subdirectory, it will call itself, but if Search() finds a .csv file it then forks a process that then sends it to the sort() function. The search() runs continuously until all files have been found and all subdirectories are explored. The sort() is essentially the same as we had it in the first project, however, we made it so it works on a more universal basis so no matter what file is given to it works.

Part 2 (sort):

The sort() function then reads the input file path, gets the file needed, and makes a 2-dimensional Row size x Column size database array to hold each cell of information in it's own cell in the array and simulate the table. We then made a second one-dimensional array of structs that holds a int rowid value and a "sorting value" which is the value of the movie in the sorting column. Those structs have two fields, the processed sorting value (removing leading/trailing whitespace) and their rowID. We then send that array into the mergesort and receive the sorted array back. As we traverse this sorted struct array, we get each's original rowID and output all of that row's values from the database array. Our mergesort sort also accounts for comparisons from both string and numeric values. When deciding how to rank the columns, we used the default ASCII numbers to compare strings and floats to compare numeric values. Also, we decided to place null values at the top during an ascending order.

Assumptions:

We worked on the assumption that the sorting and everything works the exact same as it did in the first project and based on that modularized our first projects code to be able to use it for each file in this one. We also used the assumption that all the files will have the same set of headers or close to it since we are only given 1 filter column and the assignment page contained the same set of headers.

Difficulties:

Difficulties came when we were trying to maneuver how to use the program we already created for the first project and adapt it to the needs of the second. We soon realized we should modify our original code to work on a more open ended basis such as getting any CSV file path and the the filter column and allow it to traverse and sort the file appropriately. Luckily we had our code already be fairly open ended like that and changed a couple things and left the file reading and sorting as seperate functions. Once we got those to be universal and work we then had to adjust our main function to handle the new arguments being passed and create a new search function to do the proper directory traversal. The main adjustment was a little speed bump because we had to make it adjust to the new arguments but also allow it to handle a lack of arguments inputted. The most trouble came from the search function we created. The trouble was originally how we wanted to read to the directory given to get the files to sort. We went with a while loop that kept running until all files were read and then within that is an if statement that checked the files or saw that it was a subdirectory then checked inside that and so on. We then realized that there may be other files that are not CSV files so when checking files in directories we then set a check for only CSV files. The forking for child processes was another more major difficulty because it was deciding how to use it properly and where in the program to use it. We implemented it with the bulk of our new code in the search function and we had it fork every time a new file to sort is found so the child sorts and the parent continues the search.

How to Run command:

To compile code: “make simpleCSVsorter”
To run code: “./simpleCSVsorter -c color -d ./testdata/innerFolder/ ”

Testing Procedure:

We tested this project in a modularized way by testing each section separately. We would test the search() function by sending it a directory path and having it search through and just print all the files and subdirectories and their files repeatedly until it found everything properly. We then made different tests by adding files and several layers of subdirectories to make sure it works with different combinations of them. Then we tested the sort by sending it different file paths to make sure it reads it right, finds the file, and sorts it properly. After we felt each part was sufficient we tested the whole thing together with filter columns and different input and output file directories.

We used test csv's of multiple sizes and placed them in different directories. We handled edge cases like empty folders and empty csv's. We also handled exceptions such as missing columns.

Header File:

Our header file consists of the declaration of our additional methods and functions from our mergesort and simpleCSVsorter files. It also contained the declaration of our Node struct that held our sorting column's information.