Andrew Hernandez & Malek Saadeh

Project Part 0: The Sorter

Design, Difficulties, and Assumptions

When starting this project, I thought it would be best to store records as structures that contained fields for each of the 28 columns in the movie database. This quickly proved to be a bad decision once I started tokenizing each line of the database because it was too tedious to match each token with its corresponding struct member variable. Instead, I changed the structure to contain a char ** that would be an array of all the tokens in a record.

This program requires two parameters: -c (to specify sorting by columns) and a column name.

If the provided column name is not present in the CSV read by STDIN, the program will stop and print an error message asking for a correct column name.

Assuming proper parameters, the program reads the CSV from STDIN line by line using getline() until there are no more lines to read. This is under the assumption that there are no empty lines in the middle of the CSV. It will ignore empty lines at the end of a file, but there must not be empty lines between records.

After reading a line from the CSV, the line is then tokenized using strsep() and separated by commas. Leading and trailing whitespace are removed from each token. If a token is in quotation marks, it will accommodate for this by separating by quotation marks and keeping any commas in between them. After the array is sorted, these quotation marks are re-added to preserve the CSV's readability.

Once all the lines have been read into an array of record structures, we get the index of the column that is being sorted. Then, we call mergeSort() on this array of records.

mergeSort() uses the mergesort algorithm to recursively sort array of records. The array gets broken down into left and right sub-arrays, then is sorted upon being merged back together. Within mergesort.c, the program checks the type of the column that is being sorted on and makes comparisons accordingly. If a column is of type int, atoi() is used to make comparisons. If a column is of type double, atof() is used. Otherwise, the column is treated as a string and strcmp() is used.

After every comparison between the left and right sub-arrays, the record with the lesser value is copied to the original array of records. To accomplish the copying of records, I swap the line members (array containing all the tokens of a record) of the sub-arrays and the original array.

Once the array of records is sorted, I go back through it and add any quotation marks that may have been removed (this applies to tokens that have commas in them, such as movie titles).

The sorted array of records is then printed to STDOUT and the program exits successfully.

Andrew Hernandez & Malek Saadeh

Project Part 0: The Sorter

Running the Code

Compile using: *gcc -g -Wall -Werror -fsanitize=address -sorter.c -mergesort.c -o sorter*

Then, run with: *cat movie_metadata.csv | ./sorter -c columnName*

If you want to output the sorted array to a file, type the following:

cat movie_metadata.csv | ./sorter -c columnName > outputFileName.csv

Note columnName must be a column name in the given CSV

Extra Credit Part 1 – (contained in analyze.c)

For the data, we mined and analyzed the data to see if there was any relation between the amount of gross income a movie made and the number of genres for the movie. We split up movies by two categories: more genres and less genres. More genres had genres of 4 or more types, while less had genres of 3 or less.

The algorithm for this was just to count up all of the movies into two categories (more and less genres) then average out how much each category grossed. We did this by finding a column with "|". If there were 0 to 2 of the symbol, then the movie has 1-3 genres. Surprisingly to me, there was no significant difference in the amount of money grossed in the two categories.

Extra Credit Part 2

My sorter should work for any given CSV because of the way I made my record structure. Since the tokens are stored in a dynamically allocated array of strings, there can be any number of columns. Additionally, the array of records is dynamically allocated too, so there can be any number of rows. As long as the data in a given CSV is delimited by commas, and any entries that contain commas are delimited by quotation marks, then my sorter should be able to properly sort it. Additionally, the first row needs to contain the names of the columns because that's how it checks that the given sort parameter is legal. Also, I am under the assumption that any given CSV would not contain random empty lines in between records. As long as these assumptions are satisfied, no additional metadata should be required. There is no need to specify the types of columns since checkType() automatically detects the data types of columns. If the all the characters are digits, it is treated as an int. If they're all digits and the string contains a decimal point, it is treated as a double. Otherwise, the type is considered as a string.