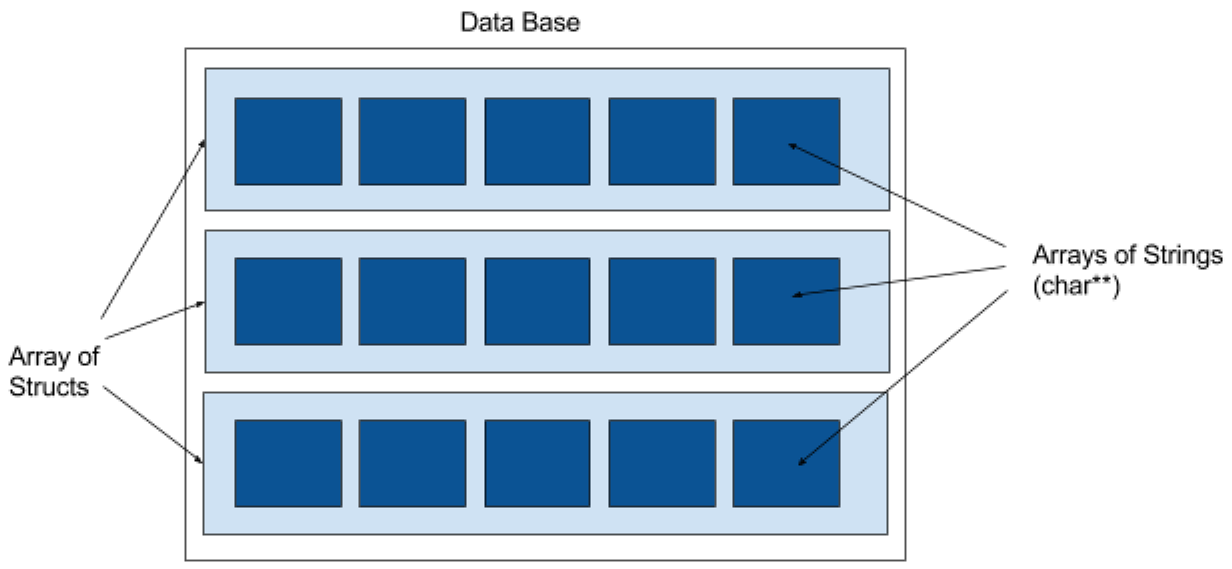
Project 0 - Basic Data Sorter

Neel Patel - 163007037 - nap166
Fareen Pourmoussavian - 165007750 - fp183

Design:



- We first planned on making an array of structs, with 28 variables inside for fields. We soon realized that it had a lot of cons such as being harder to loop through each field, and it being a static value that we cannot edit during execution. So instead we changed the fields to an array of strings, where each string would represent each field. This layout had a few perks too, where we could call the data of any particular field with var[row].data[col], and we can design the char** array to hold as many columns as determined by the csv file. Thus, allowing us to make dynamic number of rows and columns

.

- Methods

indexOf	To check if a character exists in the string or not
hasLetter	Checks if a string has atleast one letter
hasDecimal	Checks if a string has a decimal point
hasNumbers	Checks if a string has atleast one number
determineTypeOfSort	Determines what type of sorting/comparison to carryout from Strings vs Double vs Long
isEmpty	Checks if a string is all whitespace

Merge	Joins two struct arrays together and swaps if needed
MergeSort	Recursively splits struct array into left and right subarray
trimWord	Trims whitespace from every field
getNumOfColumns	Returns the number of columns detected from the header row
getSortingColumnIndex	Returns the index of the column requested by user by traversing the names of headers
isAtEndOfTable	Checks if the position is at the end of the table
main	Main program, divded into parts Part 1: Setup - Creates an array of header names to get the number of columns and index of the column targetted. Part 2: Initialize stdin - Creates an array of strings to store all the stdin rows to get number of rows Part 3: Create/Store/Sort - Creates an array of structs to store the csv metadata and uses Mergesort on the targeted column.

Compilation:

- Inside the zip file you will find sorter.c, sorter.h, Mergesort.c, and a fourth file sorterSolo.c. The main official project is sorter.c, but if it does not work, please use sorterSolo instead as it contains the Merge and MergeSort methods inside the file itself.
- Use this to compile sorter
  - `gcc -Wall -Werror -c -o sorter.o sorter.c`
  - `gcc -Wall -Werror -c -o Mergesort.o Mergesort.c`
  - `gcc -Wall -Werror -o sorter sorter.o Mergesort.o`
- Use this to compile sorterSolo
  - `gcc -g -Wall -lm -Werror -fsanitize=address sorterSolo.c -o sorterSolo`

Assumptions:

- We wrote a rather detailed outline of all the sections of the code, to avoid any surprises. The only surprises were how many extra methods were needed to handle extraneous cases.

Difficulties:

- Coding wise, it took us a long time to understand how to use malloc and realloc properly, to establish space for our data. Hence we ran into many segmentation faults and overflows. One of the more hard to resolve problems was lexicographical order. We were confused by how to exactly program it and stumbled upon the method, strcasecmp(). Upon testing, we realized its what we needed as it converts it to lowercase and compares the strings so it appears in dictionary order.
- Neel: The only major difficulty I had was that my partner, Fareen, did not know C at all, yet he managed to contribute a significant amount to the project with his logical thinking and a few methods. He was able to learn a lot about C programming from this project and should be able to keep up in the future. Other than that, it was my rusty knowledge of Computer Architecture that slowed me down initially, but after a few days of coding I was well versed again.
- Fareen: Coming into this, I thought C was close to C++ but it's a bit different. A few new problems i had to deal with were allocating new memory using malloc to prevent memory errors and coming across seg-faults. I would ask my partner, neel, all of my questions that I had who tried his best to give me answers that made logical sense. If his explanations weren't suffice, i had to do a lot of self-learning to be able to contribute a decent amount to this project. This project definitely helped in encouraging self-teaching of C programming learning about memory and structs, and being able to apply some of the things we learned in lecture and in recitation.

Testing procedure:

- To test the durability of our program, we made a series of csvs to test out. The original movie_metadata.csv was too large to handle in the beginning so we made many small ones but as time progressed we started copying more rows into a test.csv file. This helped us establish easy to understand values and limits of our program. We wanted to focus on movie titles specifically, so we have a file just for movie titles as well, aptly named moviesOnly.csv. For the grand finale, we made a 10-row 8-column csv file to test sorting scenarios. The key ones to check are: col3 which sorts periods in shortest to longest, col5 to test lexicographical sort, and col7 and 8 which both test a variety of possible fields. This CompleteTest.csv will be included in the zip file.