David Tian (dt474) & Rahul Boppana (rsb172)

Design:

The first steps we took were to fix any issues with the program from Assignment 1. For example, we added a check to make sure that every row had the same number of columns as the header. We also added a way to verify the columns and format rows to make sure they contain the necessary number of slots for the output file. We did this by inputting the values of a given row into a struct and then taking the fields of the struct sequentially for output.

We used the dirent.h library to open DIR objects on the specified input and output directories. The DIR objects can be iterated with readdir() to generate a directory entity, or dirent, for each file object in the directory. The dirent struct provides all the information we need, including the type of file object and the file name.

The last main topic was threading. We were required to thread when we came across a nested folder within a directory and to thread to sort when we came across a valid csv. A lot of the functions were built into the linux system and c package but it was important to understand how to use them and how to take advantage of their return values as processes cannot communicate with each other. As a general description, every time we threaded, we kept in mind the data we were accessing and conducted the respective operations. To keep data consistent, we used mutexes appropriately and for sorting, we kept a large link list of all the data to be outputted to the final file.

Implementation Basic Overview:

Most of the sorting aspects of the program were carried over from assignment 1, except the process of sorting now includes the checking for columns and the proper formatting for output to the cumulative file. Our main method is still responsible for checking arguments and setting up initial directories and calling the threading functionality. The program uses getopt() to read in the flags in any order and set the column and directory values. Only the -c option is required; if -d or -o are not specified, the default is to set the directories to current for both. From there the first thread is initialized and traversal of the directories begins.

DirectoryThread() is responsible for all traversal threading operations and calling threads for file processing as well. We check each entry in a directory and create a thread to handle it. If the file object is a directory, the method calls a new thread and instance of directoryThread() and passes in the calculated directory path location. If it is a file, it checks if it is a CSV and calls fileThread() to process the file. FileThread() handles all the checks for format consistency and creates the output file in the correct location. It stores the formatted input into a temporary link list and if there were no errors as the temp link list was populated, it appends the list to the larger combined list existing in the heap. And this is done within a mutex.

Threads always wait for nested threads to finish and print the thread id's and on threading errors, the program gracefully halts execution for that thread.

Headerfile:

The headerfile contains a lot of sorting structures and methods carried over from assignment 1. The new additions include some global variables to keep track of the thread counts and thread arguments. The Listing structure now contains all of the movie data columns to preserve data integrity. FindI, getListingField, and setListingField let us work with the new struct and map columns accurately.

The globals LLMutex and stdoutMutex are simply mutexes for the editing the cumulative link list and printing to stdout respectively so entries do not get mixed up. We also included the global variable totalThreads to avoid having to pass it between functions and keep track of the number of threads.

The declarations for directoryThread() and fileThread() are also stored here and are defined in scannerCSVsorter.c. All other fields were present in the prior assignment.

Testing Process:

We created a directory called input for testing purposes which contained several layers of subdirectories and our previous test CSV files at different locations. We ran the program with the -d flag set to "input", and then with the full path name, which gave the same results. Running the program without the -d flag parses through the current directory as expected. We then repeated this with the -o flag set to the output folder. For every test we also wrote stdout to a file called stdout.txt to be able to differentiate between stdout and stderr. The current contents of output and stdout.txt are the result of running

```
./multiThreadSorter_thread -c color -d input -o output > stdout.txt
```

The program works correctly with the options in any order, and prints a status message and the filename for any errors it encounters.

Instructions for Running:

Please follow the same instructions outline in the Asst2 assignment description. It specifies to run the program using

```
./multiThreadSorter_thread -c column_name [-d input_directory] [-o
output_directory]
```

once compiled with the make file.