How to use and implement simpleCSVsorter.c.

Files included:

simpleCSVsorter.c

simpleCSVsorter.h

Makefile

mergesort.c

How to run our code:

Flags:

- -c : the column name that will be sorted
- -d : the directory that will start the search and sorting process
- -o : the place where the output csvfile will go

To run the file: call make and include the flags and the full path names of your input and output directory.

Our main:

Takes the input directory and iterates through the contents to find the properly formatted csv files and then stores it in the Movie Linked list and then call evaluateDir on the file or directory to do the actual sorting. Then we output the sorted CSV file to the indicated output directory. If there are any errors we send them to stderr.

Description of Files:

Makefile:

make: complies simpleCSVsorter.c to simpleCSVsorter.o

clean: removes simpleCSVsorter.o

mergesort.c:

struct movieNode * MergeSort (struct movieNode * start, int sortColumnIndex);

- Parameters
  - Start: the list of movies to be sorted
  - sortColumnIndex: the index of the column for the list of movies to be sorted on
- Result
  - Returns the sorted list(done with Mergesort Algorithm).

struct movieNode * Merge(struct movieNode * first, struct movieNode * second, int sortColumnIndex);

- Parameters
  - first: first half of sorted list of movies to merge
  - second: second half of sorted list of movies to merge
- Result
  - Merges the two sorted list to one struct to output.

int Compare(char value1[], char value2[]);

- Parameters
  - value1: first value to be compared as a string
  - value2: second value to be compared as a string
- Result
  - Returns an int (0 for if first Is larger or 1 if the second is larger) that indicates whether the first value or the second value is larger.

char* DeleteSpace(char * field, int length);

- Parameters
  - field: string that will be used to delete leading spaces.
  - length: length of field
- Result
  - Returns field with no leading spaces in order to sort correctly.

simpleCSVsorter.c:

void CreateCSV(char * inputPath, char * outputPath, char searchColumn[]);

- Parameters
  - inputPath: string that holds the location of a CSV file to be sorted.
  - outputPath: string that holds the location of where to output the sorted CSV file.
  - searchColumn: string that holds the name of the column to sort the CSV file on.
- Result
  - Reads in a CSV at the given location and sorts it by the given column and outputs it to the given output directory location.

int GetDataType(char input[]);

- Parameters
  - input: string that represents a column in a CSV.

- Result
    - Returns the enum value of what datatype it is(string, numeric or datetime).

int GetColumnIndex(char titleRow[28][200], char searchColumn[], int columnCount);

- Parameters
    - titleRow: array of strings that represent a row in a movie CSV file.
    - searchColumn: string that holds the column to be searched for.
    - columnCount: the integer value of how many columns are in the row.
- Result
    - Returns the index of the search column in the title row and returns -1 if the column was not found.

int GetMovie(char buffer[], char movie[28][200]);

- Parameters
    - buffer: a comma separated string that represents a row in the CSV file.
    - movie: the array of strings the buffer is being turned into.
- Result
    - The contents of buffer are put into the movie array. The amount of columns is returned as an integer.

void Add(struct movieNode** headptr, char movie[28][200], int columnCount);

- Parameters
    - Headptr: node to the linked list of movies.
    - searchColumn: an array of strings representing a row to be added to headptr.
    - columnCount: the integer value of how many columns are in the movie row.
- Result
    - Adds the movie array to the headptr linked list.

int getPathType(char * path, char searchColumn[]);

- Parameters
    - path: the path of the file or directory that needs to be checked..
    - searchColumn: string that holds the column to be searched for.
- Result

    - Returns the hardcoded type from the enum global variable, this can be used to check the different error cases.

void EvaluateDir(char * curDirPath, char *outputDirPath, char searchColumn[]);

- Parameters
    - curDirPath: path to the directory or file that is being currently sorted.
    - outputDirPath: The path to the location of where we outputted the sorted CSV file.
    - searchColumn: string that holds the column to be searched for.
- Result
    - Returns nothing, but populates the outputdirectory with the sorted CSV if it is of the correct format.

simpleCSVsorter.h:

This file contains the prototypes of all methods in the program as well as the following global variables:

- struct movieNode
    - fields: char movie[28][200], struct movieNode* next;
    - stores the fields of each movie in the CSV file in a node of a linked list.
- struct Column
    - fields: char name[50], int dataType;
    - stores the column and the contents of the column that need to be sorted.
- char buffer[1024];
    - this is what each line of the CSV is read into
- int numProcesses(char *path, char searchColumn[]);
    - holds the total number of processes that get forked.
- int searchColumnType;
    - stores the type of the column that needs to be searched
- int hasOutputPath;
    - used to check if there is a valid output path
- enum pathType{DIR_TYPE, DIR_NOT_FOUND, PERMISSION_DENIED, OTHER_FILE_TYPE, FILE_NOT_FOUND, CSV_TYPE, INV_TYPE, UNFOUND_TYPE};
    - enum that stores the various types our file or directory can have.
- enum columnType{STRING_TYPE, NUMERIC_TYPE, DATETIME_TYPE, NOTFOUND_TYPE};
    - enum that stores the different types of the search column.

Problems we encountered:

- The requirements were not fully explained, so we needed to search through piazza, class and office hour to fully realize the exact input and output parameters of this assignment.
- We created many zombie processes when we did not account for exiting if there was a directory within a directory.
- We created a fork bomb when we did not account for the directories, . And .. , in our code. We luckily fixed the number of processes that can run at one time so we did not crash the system.

To fix these issues we just accounted for all of the cases that were giving us problems.

Test Cases:

To test this program we first created a directory with files of different types and various directories.

Here is our file tree

```
-proj2
        -Makefile
        -simpleCSVsorter.c
        -simpleCSVsorter.h
        -mergesort.c
        -testdir
                -dir1
                        -dir2
                                -Test1.csv
                                -results.txt
                                -test4 -sorted- color.csv
                        -dir3
                                -movie_metadata1.csv
                                -movie_metadata3.csv
                        -test4.csv
                        -movie_metadata.csv
```

-test.c

-outputdir

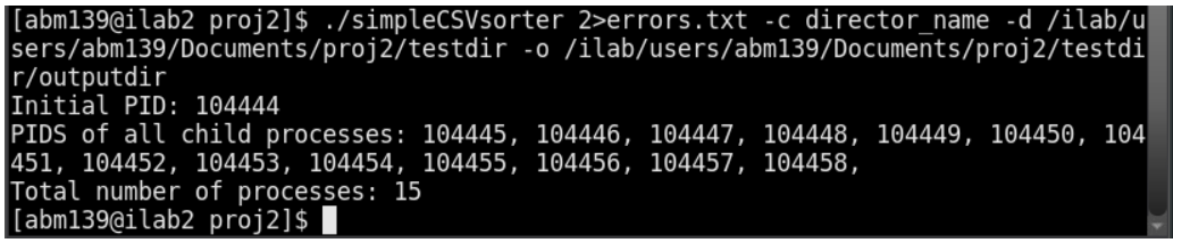
-test3.csv

-test1.csv

We included the above directory in our tar file so you can see our test cases.

The csv files contained the different testcases released for Asst0. We then ran this the code and counted the processes, which was 15 processes and compared that with how many files and directories we were searching. Next, we tested the -c, -d, and -o flags. We tested using various output directories and trying to sort only on directory dir1 or dir2 and with different amounts of arguments to check our error cases. We also checked the different errors and redirected them to errors.txt so that the output would not be readable. We checked all the errors possible such as: checking permissions of files, if a file exists, the filetype of the file, if the column exists in the CSV we are sorting, and if the CSV is initially in the correct format.

Output:



```
[abm139@ilab2 proj2]$ ./simpleCSVsorter 2>errors.txt -c director_name -d /ilab/u
sers/abm139/Documents/proj2/testdir -o /ilab/users/abm139/Documents/proj2/testdi
r/outputdir
Initial PID: 104444
PIDS of all child processes: 104445, 104446, 104447, 104448, 104449, 104450, 104
451, 104452, 104453, 104454, 104455, 104456, 104457, 104458,
Total number of processes: 15
[abm139@ilab2 proj2]$ 
```