Documentation for Project 1: Basic Data Sorter – multiproc/mass sorter

Design:
We wanted our code to be highly reusable in general and also for future projects. We tried to make functions for most things that would be used more than once. Our general design is the following. First, we would take in the input, analyze it and save it. In the analyzing process if the output directory doesn't exist then we create it in the current directory (if only a relative path is given. Else it would be created in the absolute path given.) Then we would call a function that would do the traversing, forking and sorting. For our traversing, we designed a recursive algorithm where we first check if the current entry is a DIRECTORY or a NOT A DIRECTORY. If it is a directory then we fork() and child calls back the function (recursive call) giving it the correct new parameters. The parent on the other hand continues looking for files at this level. When a file is found, we first check if it a CSV file. If it is then we fork(). The child first checks if the CSV is a valid csv file. It does this by calling a function which checks the first column and counts to make sure there are only 28 fields. If it is valid then we call file_sort. The job of this function is to read the CSV file and sort it using merge sort. If the file is not valid then we exit. The parent keeps going looking for file or directories (we print pids in each child function). After its done looking for files the loops ends and we wait for all the child created in this pass. We collect the exit status for all and this way we can count the total number processes. We return this process count to the main once at the end and main prints out the total number of created processes.

Assumptions:
1. The given directory actually exists.
2. The output directory actually exists (we create one otherwise).
3. The CSV file contains the correct order and correct data for the first line. (color,director_name…)

Difficulties:
Not many difficulties were faced, however, we did have some trouble figuring out the total process count.

How to use our files:
4 Files are needed to run the program. arg.c, is.c, sorter.c and sorter.h.
You can compile the program using **gcc -o arg arg.c** and then run our program using
**./arg -c sortingField -d Directory -o Directory**. Just replace sortingField and Directory to whatever you want.
In the files uploaded we are including the setup of directories and csv's we tested our program with.

Files:

Arg.c - Goes through each of the directories recursively, and creates a child everytime a csv file is encountered and everytime a new directory is discovered. It will then increment a global pointer in the parent function and print out the child's PID. After each parent call it waits for the child exits and continues going through files and sorting through directories. Afterwards it closes the directory. In this file it also checks the input flags and validates the csv to make sure it is worth sorting.

Is.c - Splits input for multi parameter sorting into a list of integers for the sorter.c file to use. It also gets the size of the number of parameters and provide other string computations and functions useful for directory sorting.

Sorter.c - Contains the sorting and comparison function. Used to call Split and merge as well as compare values

given multiple parameters

Sorter.h - Contains our structure and functions layouts

API:

changePos ( mdata **records, int left, int right)

Swap function that changes

compare(struct mData leftArr, struct mData rightArr, int comp_ptr)

Compares the two data types based on the column (comp_ptr)

mergeData(struct mData *array, int left, int middle, int right, int* comp_ptr, int comp_ptr_size)

Merge sort function does the testing, and the combination to get the final merge sorted list

split(struct mData *array, int left, int right, int* comp_ptr, int comp_ptr_size)

Recursively split the array and then perform merge sort

Print(FILE *wf, struct mData records[], int size)

Print the contents of records into the fie passed. Size is to know when to stop.

checkIf(char*p)

Checks if there is a space in the given char* array

getSize(const char* s)

Gets the size of a char *

getInputSize (const char *input)

Gets the amount of things in a char * separated by a comma

findInt(char* comp,char* cat[], int size)

Given a comparison string this program compares a list of cat against compare and returns a int array with the column numbers.

getInput(int* input, const char *directory,char *cat[],int size,int i)

Generates a int array based on findInt. Used for multi-parameter sorting

check_oDir(char* output)

Checks to see if directory is valid or not If it is we close it and continue on

get_cwd(char *var)

Gets the current working directory and returns it

int validate_csv(FILE *fp)

Checks the validity of the csv file and return 1 if it is valid 0 if it isn't.

int csv_line_count(FILE *fp)

Returns the number of line in the csv file for dynamic allocation

.

int print(const char * name, const char *output, char *sF, int* compare,

int compare_size)

Sorts through the directories recursively and create children in the appropriate places. Print the PIDs of the children and count how many exist. This checks the file and sorts them then prints them accordingly.