P2

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. We create a thread for starting the parsing process in the Main thread. This thread has the function to parse directories, and as more directories are found more threads are created. For each CSV file we take in file name and location and call our sorting function on in. We have used Quicksort and Bubble sort as our sorting algorithms. We decided to do this because we were able to achieve O(kn) and the combination of the two lead to better run times of our test cases. We have implemented error handling for each time we have to realloc for more threads or more struct space. To tackle race conditions we input 1 file sorted file into our struct at a time using locks. Our struct was finally sorted using quicksort and bubble. Finally printed the data in the struct to a file.

Difficulties:
For parsing each directory I forgot that threads share the same heap space, this caused many problems. I would send in my struct but the struct would be overwritten with the last found directory. Then I went over notes and came across the fact that threads share the same heap space. To fix this problem I segregated the data each thread received. This was the biggest difficulty I had which caused me to think over and over again about my implementation and took me 3 days to realize.

How to use our Code:
Analysis:
https://docs.google.com/document/d/1-F8P5-zVIicKB43X2tbxRTvsVSJLpRFggT2ZEpj3ALo/edit?usp=drivesdk
Note: Our program takes in input in any order as required by the project specifications.

Files:
We have two header files. Threads.h contains all the structs for threads.c and sorter.h contains structs for the sorting process.
threads.h : 1) struct file_dir : contains pointers for the working directory, output directory,
sorting field and the number value for it (between 0-27) based on the metadata,
current directory, and also the pointer to the main struct that holds all the data.
2) struct wdir: contains a field for the working.
3) struct Counter: the purpose of this is to keep track of the thread count.
Contains a counter and a lock.
4) struct Sorter: this is the struct that is created for passing to the sorting procedure.
contains pointers for the working directory, output directory, the int for the
Sorting field, a lock and a pointer to the global struct.
Sorter.h : 1) struct mData: struct to hold all the data of each CSV.
Is.c - We use this file to convert the given sorting field to a int between 0-27 based on the metadata.
sort_starter.c - Contains the sorting and comparison functions. Also adds data to the Main struct.
threads.h - Contains main functions that initiates everything. Parsing, calling, and taking input.
sorterQuickStruct.c - Contains the functions for Quicksort and Bubble sort.

P2

API:

int counter_get(struct Counter *st);

Return back the count of threads.

void counter_inc(struct Counter *st);

Increases the count of threads

void counter_init(struct Counter *st);

Initialize the counter lock

//Misc

void print_usage();

Prints usage incase wrong input was provided

void print_stat(struct file_dir *str);

Just for debug purposes to print the contents of a struct

char* get_cwd(char* var);

Return the current working directory

void print_cwd(char* var);

Prints the current working directory

int check_oDir(char* output); // 0 = Exists 1 = Does not exist 3 = Couldn't create

Checks if the provided output directory exists

int check_wDir(char* output); // 0 = Exists 1 = Does not exist

Checks if the provided input directory exists

char * checkIf(char * p);

Check if a given address is a space character

//Parsing

void *parse_dir(void *st);

For paring directories and CSV's

//printing

void print_final(struct mData *records, struct file_dir dir,int size);

Print final data to a file

P2

```
//sort
void *file_sort(void *str);
```

Reads data from a CSV and saves to a temp struct to be written into the final.

```
void Print(struct Sorter *st, struct mData records[], int size);
```

Saves the temp structs data into the main struct.