sort

Thomas Rathbun, Eric Rogers

1 NAME

sorter - sorts all .csv files inside of a directory tree

2 SYNOPSIS

./sorter [OPTION]... to compile: gcc mergesort.c main.c -lm -o sort.out

3 DESCRIPTION

The sorter program traverses the directory tree starting at current directory unless -d is specified.
OPTIONS

- -c
  - header to sort by
- -d
  - directory to sort
- -o
  - the output directory for all sorted files

4 FUNCTIONS

At the start the program checks for the options supplied. When a -c option is not supplied the program will return -1 and will give a message that the required option is missing.
Optional options -d and -o: The searchDir and outputDir variables are set respectively based on the supplied values. If the value of the -o option does not exist mkdir is called and the directory is created. If the directory does not exist the program returns -2 and gives a message that the supplied directory does not exist.
Default Behaviour: If -d is not supplied, searchDir is set to the current working directory. If -o is not supplied, the sorted file is left in the directory the unsorted file is found in. NOTE: If for example (./sort -c budget -d searchDir) is called and then called a second time, the only file will be foundfile-sorted-budget.csv.
After all external variables are set the intial call of dirTraversal is called with the supplied variables (searchDir, headerTitle, outputDir). The Function initialize an array of childStatus and Process which are used to keep track of the child processes. A processCounter is also initialized to 0 in order to keep

track of how many processes are forked. The process id is stored in myPID. A struct stat is created in order to check if the -o directory exists.
Dir* base is opened with the given filename struct dirent * entry is initialized to the first directory in base
A while loop is started checking if the entry is NULL. If it is NULL then the loop terminates and the process waits for its children before returning.
A switch statement checks if the current entry is a directory or file. If it is a directory the file name is concatanated to the filepath and the forked process exits on the completion of a recursive call to dirTraversal. In the parent process the processCounter is incremented and breaks from the switch.
If the entry is a file, the node for the head and node for the data are created. The file path is concatenated with the entry name and then the file is checked for the correct file extension. If the file does not have the correct extension. If it does not have the correct extension it will be output to stdout.
If the file is of the correct type the headers are read into node head. The chosen field is looked for. If the header does not exist then the process exists with -100 and "header DOES NOT EXIST" is printed to stdout.
If the header exists and the total number of headers is equal to the number in an appropriate moviedatabase.csv, the rest of the file is read into node row. The node row is then passed to the mergesort function and after a successful run, the node head-¿next is set to row. The data is then writen to the file in the chosen directory and then the nodes are freed and the process exits with 1.

5 TESTING

Tested with a single directory with a single csv inside. Tested with multiple directories, each with either a correct.csv file or an incorrect .csv file inside Tested with multiple directories consisting of files with .csv files and other file extionsions. Tested with on directories which we alread ran a sort on using multiple head names. NOTE: This created many, many csv files each time it was run. So we have tested with a multitude of files in one run.

6 REPORTING BUGS

https://github.com/ThomasJRathbun/Multi-ProcessSorter/issues