README.md

Lab 0

Quicksort File Text

Derek Prince

Code Organization

The code is in a single file save for the argument parser in the header. main() starts by setting up the argument parser and then parsing out the the options. Options are parsed as if-checking statments with `--help` and `--name` having first and second priority as they return with 0 after running. If neither of those options are selected then it pulls out the file names and attempts to open and read in data from the input file. If successful, it then sorts the data and prints out the result in a command line. Finally the program opens or creates the output file and dumps the corrected array into the file and exits main.

The code is almost entirely sequentially read with only the quicksort algorithm being broken out at the top of the file with it's associated partition function.

Why I chose Quicksort

I chose quicksort because I had not written a quicksort before and I know its fast. They're both good algorithms so thats about it.

There was some though put into how merge sort grows wider as the problem increases insted of quicksort's deeper (recursive) approach and how that could be done in parallel but as that was not a requirement this time around I did not worry too much about how each process would end, etc.

Compilation instructions

Unzip file and cd into top level with the Makefile. The file structure should look as such:

```
 --- Lab0/
| ------ Readme.md
| ------ Writeup.pdf
| ------ Makefile
| ------ Source/
|       |------ mysort.cpp
|       |------ Include/
|       |       |------ cxxopts.hpp
```

To build simply type `make` and it will compile with g++ and C++11 standards as well as -Wall -Werror.

`make clean` will remove object files.

Execution Instructions

The program operates as the lab directed with a few extra options that should be mostly transparent. The main difference being the help menu accessed by running `mysort --help`.

To have `mysort` print author name (me) run `mysort --name`.

Sorting files

To sort a file all that need be done is run `mysort input_file -o output_file`. As far as has been tested, relative pathing works in the arguments for the text files. For instance `./mysort Source/test.txt -o Source/output.txt` will source and create the files in the correct directories.

If no output file is specified but a proper input file is, the program will print the output file in the same directory under `output.txt`.

Terminal Output

During normal sorting operation the program prints each number to the command line once on their own lines.

Expected Input

A text file with one number per line up to the size of the range of an int.

Ex:

```
45
1
9
11
3456
6
9
0
```

Letters as fas as has been tested are ignored and removed from the input. See extant bugs section for more.

File Descriptions

- Makefile - makefile to automate build with gcc.
- Readme.md - Markdown version of this PDF
- source/mysort.cpp - Entry point of program containing all relevant code.
- source/include/cxxopts.hpp - CLI argument parser library

Credits

cxxopts for command line parsing because nobody but C++ needs to solve this issue again.

Extant Bugs

None *known* bugs but unsupported file formats are largely untested. Letters and leading 0's are have proven to be ignored for all basic testing cases.

A file of only 0s will crash on overflow as it does an infinite recursion downwards. I don't realistically see this being a problem so I didnt limit it.