Project 1: Sorting

Overview

You will write a simple sorting program. This program should be invoked as follows:

```
% ./fastsort inputfile outputfile
```

The above line means the users typed in the name of the sorting program `./fastsort` and gave it two inputs: an input file to sort called `inputfile` and an output file to put the sorted results into called `outputfile` .

Input files are generated by a program we give you called generate.c (good name, huh?).

After running `generate` , you will have a file that needs to be sorted. It will be filled with binary data, of the following form: a series of 100-byte records, the first four bytes of which are an unsigned integer key, and the remaining 96 bytes of which are integers that form the rest of the record. Something like this (where each letter represents two bytes):

```
kkRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Your goal: to build a sorting program called `fastsort` that takes in one of these generated files and sorts it based on the 4-byte key (the remainder of the record should of course be kept with the same key). The output is written to the specified output file.

Some Details

Using `generate` is easy. First you compile it as follows:

```
% gcc -o generate generate.c -Wall -Werror
```

Note: you will also need the header file sort.h to compile this program.

Then you run it:

```
% ./generate -s 0 -n 100 -o /tmp/outfile
```

There are three flags to `generate`. The `-s` flag specified a random number seed; this allows you to generate different files to test your sort on. The `-n` flag determines how many records to write to the output file, each of size 100 bytes. Finally, the `-o` flag determines the output file, which will be the input file for your sort.

The format of the file generated by the `generate.c` program is very simple: it is in binary form, and consists of those 100-byte records as described above. A common header file sort.h has the detailed description.

Another useful tool is dump.c. This program can be used to dump the contents of a file generated by `generate` or by your sorting program.

Hints

In your sorting program, you shall use system calls (`open()` , `read()` , `write()` , `close()` , etc.), rather than standard library functions(`fopen()` , `fclos()`). See `generate.c` or `dump.c` .

If you need to know the size of an input file, `stat()` or `fstat()` could help.

To sort the data, use any old sort that you'd like to use. An easy way to go is to use the library routine `qsort()` .

To exit, call `exit()` with a single argument. This argument to `exit()` is then available to the user to see if the program returned an error (i.e., return 1 by calling `exit(1)`) or exited cleanly (i.e., returned 0 by calling `exit(0)`).

The routine `malloc()` is useful for memory allocation. Make sure to exit cleanly if malloc fails!

If you don't know how to use these functions, use the man pages. For example, typing man

qsort at the command line will give you a lot of information on how to use the library sorting routine.

Assumptions and Errors

- *32-bit integer range*. You may assume that the keys are unsigned 32-bit integers.
- File length: May be pretty long! However, there is no need to implement a fancy two-pass sort or anything like that; the data set will fit into memory.
- *Invalid files*: If the user specifies an input or output file that you cannot open (for whatever reason), the sort should EXACTLY print: `Error: Cannot open file foo\n`, with no extra spaces (if the file was named `foo`) and then exit.
- *Too few or many arguments passed to program*: If the user runs fastsort without any arguments, or in some other way passes incorrect flags and such to fastsort, print `Usage: fastsort inputfile outputfile` and exit.

Important: On any error code, you should print the error to the screen using `fprintf()`, and send the error message to `stderr` (standard error) and **not** `stdout` (standard output). This is accomplished in your C code as follows:

```
fprintf(stderr, “whatever the error message is\n”);
```

Hand In

You should hand in only one source file: `fastsort.c` (without the ".o" file).

General Advice

Start small, and get things working incrementally. For example, first get a program that simply reads in the input file, one line at a time, and prints out what it reads in. Then, slowly add features and test them as you go.

Testing is critical. One great programmer I once knew said you have to write 5-10 lines of test code for every line of code you produce; testing your code to make sure it works is crucial. Write tests to see if your code handles all the cases you think it should. Be as comprehensive

as you can be. Of course, when grading your projects, we will be. Thus, it is better if you find your bugs first, before we do.

Keep old versions around. Keep copies of older versions of your program around, as you may introduce bugs and not be able to easily undo them. A simple way to do this is to keep copies around, by explicitly making copies of the file at various points during development. For example, let's say you get a simple version of `fastsort.c` working (say, that just reads in the file); type `cp fastsort.c fastsort.v1.c` to make a copy into the file `fastsort.v1.c`. More sophisticated developers use version control systems like CVS (old days) or mercurial or git (modern times), but we'll not get into that here (though you can, and perhaps should!).

Adapted from WISC CS537 by Remzi Arpaci-Dusseau