Project 1a: Text Sorting

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

```
     shell% ./fastsort [ -3] file
```

The above line means the users typed in the name of the sorting program `./fastsort` and gave it one or two inputs. If just one input is given, the input file `file` should be sorted and the sorted output printed to the screen; it is assumed that the file is a text file (full of ASCII characters) and, when no other arguments are given, the sorting should be over the first word in each line.

If the optional argument is included (`-3` in the example), the program should sort the text input file using the specified word as the key to sort upon (with `-3` , the program should find the 3rd word in each line and sort the lines based upon that).

Examples

Let's say you have the following file:

```
this line is first
but this line is second
finally there is this line
```

If you run your fastsort and give it this file as input, it should print:

```
but this line is second
finally there is this line
this line is first
```

because `but` is alphabetically before `finally` is before `this` .

If, however, you pass in a flag to sort a different key, you'll get a different output. For example, if you call `fastsort -2` on this file, you should get:

```
this line is first
finally there is this line
but this line is second
```

because `line` comes before `there` comes before `this` . Yes, we are assuming `-2` means the second word in each line (like most people would, except computer scientists who always want to start at 0).

Hints

In your sorting program, you should just use `fopen()` to open the input file, `fgets()` to read each line of the file, and `fclose()` when you are done with the input file.

If you want to figure out how big in the input file is before reading it in, use the `stat()` or `fstat()` calls.

To compare strings, use the `strcmp()` library call.

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

To chop lines into words, you could use `strtok()` . Be careful, though; it is destructive and will change the contents of the lines. Thus, if you use it, make sure to make a copy of the line to output.

The routine `atoi()` (or better yet, `strtol()`) can be used to transform a string into an integer.

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)`). You can also just return from the `main()` function and pass the return code that way where appropriate.

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

> **The return code upon success is zero.** When the program runs normally and no errors are encountered, you should return an error code of 0.
>
> **Only space characters (i.e., what you get when you hit spacebar) will be used to separate words in the input.** Thus, you don't have to worry about tabs or other whitespace. However, your program should correctly handle the case where there are two or more spaces between words, i.e., it should treat that as one big separator between the words.
>
> **Max line length will be 128.** If you get a line longer than this (detected by the lack of a newline character in the last position), please print `Line too long` to standard error and exit with return code 1.
>
> **You should check the arguments of fastsort carefully.** If more than two arguments are passed, or two are passed but the second does not fit the format of a dash followed by a number, you

should EXACTLY print (to standard error): `Error: Bad command line parameters` and exit with return code 1.

Key does not exist on one line of input file: If the specified key does not exist on a particular line of the input file, you should just use the last word of that line as the key. For example, if the user wants to sort on the 4th word (`-4`), and the sort encounters a line like this (`sample line`), the sort should use the word `line` to sort that line.

Empty line: You should use an empty string to sort any empty lines (i.e., lines that are just a newline or spaces and a newline character).

File length: May be pretty long! However, no need to implement a fancy two-pass sort or anything like that; the data set will fit into memory and you shouldn't have to do anything special to handle this. However, if malloc() does fail, please print `malloc failed` to standard error and exit with code 1.

Invalid files: If the user specifies an input file that you cannot open (for whatever reason), the sort should EXACTLY print (to standard error): `Error: Cannot open file foo` with no extra spaces (if the file was named `foo`) and then exit with return code 1.

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");
```

History and a Contest

This sorting assignment is reminiscent yearly competition to make the fastest disk-to-disk sort in the world. See the sort home page for details. If you look closely, you will see that your professor was once -- yes, wait for it -- the fastest sorter in the world.

To continue in this tradition, we will also be holding a **sorting competition.** Whoever turns in the fastest sorting program on a few different inputs will win a fancy 537 T-shirt. Read more about sorting, including perhaps the NOW-Sort paper , for some hints on how to make a sort run really fast. Or just use your common sense! Hint: you'll have to think a bit about hardware caches.

Restriction: No threads. You cannot implement a multi-threaded sort for this assignment or competition. Just make the fastest single-threaded sort that you can!

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 git or mercurial, but we'll not get into that here (yet).

Keep your source code in a private directory. An easy way to do this is to log into your account and first change directories into `private/` and then make a directory therein (say `p1` , by typing `mkdir p1` after you've typed `cd private/` to change into the private directory). However, you can always check who can read the contents of your AFS directory by using the `fs` command. For example, by typing in `fs listacl .` you will see who can access files in your current directory. If you see that `system:anyuser` can read (r) files, your directory contents are readable by anybody. To fix this, you would type `fs setacl . system:anyuser ""` in the directory you wish to make private. The dot "." referred to in both of these examples is just shorthand for the current working directory.