CSCI 1120 (Low-Level Computing), Fall 2016

Homework 6

Credit: 20 points.

1 Reading

Be sure you have read, or at least skimmed, the assigned readings for classes through 10/12.

2 Honor Code Statement

Please include with each part of the assignment the Honor Code pledge or just the word "pledged", plus one or more of the following about collaboration and help (as many as apply).[1] Text *in italics* is explanatory or something for you to fill in. For written assignments, it should go right after your name and the assignment number; for programming assignments, it should go in comments at the start of your program.

- This assignment is entirely my own work.
- This assignment is entirely my own work, except for portions I got from the assignment itself *(some programming assignments include "starter code")* or sample programs for the course *(from which you can borrow freely — that's what they're for).*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, the instructor, etc.*
- I got significant help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. ("Significant" here means more than just a little assistance with tools — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided significant help to *names of students* on this assignment. *("Significant" here means more than just a little assistance with tools — you don't need to tell me about helping other students decipher compiler error messages, but beyond that, do tell me.)*

3 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu` with each file as an attachment. Please use a subject line that mentions the course and the assignment (e.g., "csci 1120 hw 6" or "LL hw 6"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

[1] Credit where credit is due: I based the wording of this list on a posting to a SIGCSE mailing list. SIGCSE is the ACM's Special Interest Group on CS Education.

1. (20 points) Write a C program that sorts the lines in a text file (considering each line as a string) using the library function `qsort`. The program should take the name of the file to sort as a command-line argument (and print appropriate error messages if none is given or the one given cannot be opened) and write the result of the sort to standard output.

   To do this, I think you will need to read the whole file into memory. There are various ways to do this and perform the sort; the one I want you to use is somewhat involved but intended to give you more practice working with pointers. To get full credit you must use the approach described here.

   First, read the whole file into memory. To do this you will need to know its size, and interestingly enough there does not appear to be any truly portable and reliable way to find that out. My suggestion is therefore to just open the file, read it a character at a time, counting the number of characters but not trying to save them, and close it again. Reading the file twice (once only to find its size) is of course inefficient but will give the desired result (unless some other application is changing the file at the same time) using only standard and portable C functions, and coming up with a nicer way to accomplish this task is beyond the scope of this assignment.

   Once you have the (best estimate for) file size, you can allocate a single array for the file using `malloc`, something like this:

   ```
   char * data = malloc(size_in_bytes);
   ```

   You can now operate on `data` as if it had been declared as an array of `char`. (Check first that `malloc` succeeded.)

   Read in the contents of the file; a character at a time is probably simplest. Notice that as you do this you will get the newline characters at the ends of lines. (You might write this much of the program and check that it works before going on.)

   Once you have the whole file in memory, the objective is to sort it with `qsort`. The sample program sort-improved.c[2]. has an example of using `qsort`. It needs four parameters: an array to sort (of elements of fixed size), a count of elements, a size for each element, and a comparison function. Your first thought may be to wonder how this can work, since text strings aren't of fixed size. But we can play a trick ...

   The idea will be to build an array of pointers pointing to starts of lines, sort *the pointers* so the first one points to the first line to print, etc., and use them to print the lines in order. (If you think you know at this point how to proceed, you could try doing so, and then come back and read the rest of this description.)

   So the next step is to build the array of pointers to lines. How many do you need? Well, you could figure that out as you're reading the file into memory. Say you have that in a variable called `N`. Then you can allocate space for an array of pointers like this:

   ```
   char ** lines = malloc(sizeof(lines[0]) * N);
   ```

   The first one should point to data[0], which you can accomplish like this:

   ```
   lines[0] = &data[0];
   ```

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2016fall/SamplePrograms/Programs/sort-improved.c`

Then the idea is to go through the rest of the characters, and make `lines[1]` point to the character after the first newline, `lines[2]` point to the character after the second newline, etc.

Once you get this array built, you can check it by printing the file contents out again using the array of pointers; for example, to print the first line you can write

```
printf("%s\n", lines[0]);
```

(You'll need this code anyway, so might as well write it now and check that it works. You may get a surprise when you first run it, as a result of which you may decide you need to do more processing of your `data` array.)

Now the missing piece of the puzzle is to use `qsort` to actually sort the lines before printing them. Its parameters were described earlier; the only one you don't yet have is the comparison function. It can look a lot like the one in the sample program; all that's different is that you're sorting pointers-to-strings rather than integers. You will probably want to use the C library function `strcmp` for the actual comparison. Read its `man` page to find out what it does and what parameters it takes.

You can check your program's output by using the `sort` command to sort the input file and comparing its result (captured with I/O redirection!) with your result (also captured with I/O redirection).