CSCI 1120 (Low-Level Computing), Spring 2020

Homework 6

Credit: 15 points.

1 Reading

Be sure you have read, or at least skimmed, the assigned readings for classes through 10/02.

2 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 my TMail address 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. (5 points) In CS1 you probably learned about sorting algorithms and implemented one or more of them.[1] A simple way to test such an algorithm is to generate a sequence of "random" numbers, sort them, and check that the result is in ascending order. Sample program `sorter-start.c` shows how this might be done in C, all but the actual sorting. For this problem your mission is just to fill in code for the `sort` function so that it actually sorts. It's completely up to you which sorting algorithm to implement, though I'm inclined to recommend that you just do one of the simple-but-slow ones (e.g., bubble sort or selection sort). If you feel ambitious, you could try quicksort or mergesort, though mergesort is apt to be more trouble since it requires a work array. (If you're tempted to look up and use a library function: Don't; that misses the point of this assignment, and you won't get full credit.) *Note* that when you get your sort working you should edit the comments at the start of the program appropriately!

2. (10 points) For this problem your mission is to further revise the sort program from the previous problem so that rather than generating random data it reads the values to sort from a file and writes the sorted values to another file. The completed program should take two *command-line arguments* giving the names of the input and output files. (It should not prompt the user for anything.) The program should print appropriate error messages if not enough arguments are supplied, if it cannot open the input and output files, or if the input file contains anything but a sequence of integers. Since we have not yet talked about how to make arrays larger at runtime, just write the program with a fixed-size array for holding input, and have the program print an error message if the number of input values exceeds the size of the array. It's up to you whether you keep the part of the existing program that checks whether the sort succeeds (I say "might as well"); if you do, just have it print to standard output as before.

   Hints:

[1] If you didn't take CS1 and don't know about sorting, ask me about supplemental reading.

- Sample program `while-sum-fromfile.c` illustrates reading a sequence of integers from an input file. Note that the `while` loop to read integers stops when `fscanf` detects either an error or the end of the file. The `if` after the loop uses `feof` to find out which of these two things happened — `feof` returns a nonzero value ("true") when the previous attempt to read something detected end of file, zero ("false") otherwise (i.e., an error). Be advised that `ferror()` is useful only for detecting I/O errors and is not set if `fscanf()` can read input from the stream but can't convert it to the requested format.

3. (10 points) A very simple way to encrypt text is to rotate each alphabetic character N positions. For example, if N is 1, "abc XYZ 1234" becomes "bcd YZA 1234". (This is obviously not industrial-strength encryption but is good enough to somewhat obscure the plaintext.) Write a C program that implements this scheme. The program should take three command-line arguments: the number of positions to rotate (which for simplicity should be a positive integer), the name of the input file, and the name of the output file. It should print error messages as appropriate (not enough command-line arguments, non-numeric N, input or output file cannot be opened). For valid arguments, it should encrypt the input file and write the result to the output file.

   There are probably several ways you could approach encoding each character. *To get full credit, your program must encrypt using the following approach,* rather than other ways you may have seen for doing this sort of thing:

   First look up the character in a string representing the alphabet, then convert it also using the alphabet string. Starter code for such a scheme, to encode `int` variable `inchar`, is available here.

   (I ask you to do this partly for more practice working with strings and pointers, but also because it doesn't rely on characters being encoded in ASCII (which on most systems these days they are, but C doesn't require it).)

   Hints:

   - You don't need to try to read input a line at a time; you can just read and process it a character at a time using `fgetc`, `fputc`, and a function that encrypts a single character. Sample program `copy-file.c` illustrates how to process a file one character at a time.
   - You can use library function `strtol` to convert a command-line argument string into an integer. (You could also use `atoi`, which is simpler, but it doesn't provide a good way to check for errors.) There's an example of using `strtol` in sample program `echo-args.c`.

3 Pledge

Include the Honor Code pledge or just the word "pledged", plus *at least one of the following* about collaboration and help (as many as apply).[2] Text *in italics* is explanatory or something for you to fill in. For programming assignments, this should go in the body of the e-mail or in a plain-text file `pledge.txt` (no word-processor files please).

[2] 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.

- This assignment is entirely my own work. *(Here, "entirely my own work" means that it's your own work except for anything you got from the assignment itself — some programming assignments include "starter code", for example — or from the course Web site. In particular, for programming assignments you can copy freely from anything on the "sample programs page".)*
- 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. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*
- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — 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 help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

4 Essay

Include a brief essay (a sentence or two is fine, though you can write as much as you like) telling me what if anything you think you learned from the assignment, and what if anything you found found interesting, difficult, or otherwise noteworthy. For programming assignments, it should go in the body of the e-mail or in a plain-text file `essay.txt` (no word-processor files please).