CSCI 1120 (Low-Level Computing), Fall 2016

Homework 5

Credit: 20 points.

1 Reading

Be sure you have read the assigned readings for classes through 10/05.

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 5" or "LL hw 5"). 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. (10 points) In CSCI 1320 you probably learned about sorting algorithms and implemented one or more of them. 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 sort-starter.c[2] shows how this might be done in C (leaving out the actual sorting). For this problem you will turn in two revisions of this program:

   - First, 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.
   - Next, revise the program 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.

   To repeat: You will turn in two programs, one that just fills in the `sort` function but sorts the randomly-generated data, and one that gets input from a file and writes to another file.

   Hints:

   - Sample program whilesum-fromfile.c[3] illustrates reading a sequence of integers from an input file. Notice 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.

2. (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

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

[3] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2016fall/SamplePrograms/Programs/whilesum-fromfile.c`

write the result to the output file. *To get full credit, your program should encrypt characters as discussed in the hint below.*

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 countchars-fromfile.c[4] illustrates how to read a file one character at a time. You can use `fputc` to write 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 nice way to check for errors.) Example of using `strtol`:

```
char* endptr;
int N = strtol(argv[1], &endptr, 10);
if (*endptr != '\0') {
    /* error */
}
```

- There are probably several ways you could approach encoding each character. The one I want you to use here — partly for practice working with strings, 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) — begins by looking up the character in a string representing the alphabet. Starter code for such a scheme, to encode `int` variable `inchar`, is available here[5], and I strongly recommend that you use it as your starting point.

[4] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2016fall/SamplePrograms/Programs/countchars-fromfile.c`

[5] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2016fall/Problems/starter-encrypt.c`