CSCI 1120 (Low-Level Computing), Spring 2014

Homework 4

Credit: 20 points.

1 Reading

Be sure you have read the assigned readings for classes through 3/19.

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 `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course number and the assignment (e.g., "csci 1120 homework 4"). 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. (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 files cannot be opened). For valid arguments, it should encrypt the input file and write the result to the output file.

   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 your own function that encrypts a single character.
   - 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. One I like (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`:

```
char* lc_alphabet = "abcdefghijklmnopqrstuvwxyz";
char* in_lc_alphabet = strchr(lc_alphabet, inchar);
if (in_lc_alphabet != NULL) {
  /* lower-case character */
  int position_in_alphabet = in_lc_alphabet - lc_alphabet;
  /* more code goes here .... */
}
```

  `lc_alphabet[position_in_alphabet+1]` then gives you the next character in the alphabet. You could do something similar for uppercase characters, with a string `uc_alphabet`.

2. (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-example.c`[1] shows how this might be done in C (leaving out the actual sorting). For this problem you will do two things:

   - 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.
   - Once you have your `sort` function working, 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. The program should print appropriate error messages if it cannot open the input or output file 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:

   - Sample program `sum-from-file.c`[2] 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).

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2014spring/SamplePrograms/Programs/sort-example.c`

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2014spring/SamplePrograms/Programs/sum-from-file.c`