# CSCI 1120 (Low-Level Computing), Spring 2011

## Homework 4

**Credit:** 20 points.

## 1 Reading

Be sure you have read the assigned readings for classes through 3/28.

## 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) In the early days of the Internet, a popular way of disguising text that some might find offensive was to encode it using a scheme called rot13 (short for "rotate 13"). In this scheme, all letters are rotated 13 positions (so 'a' becomes 'n', 'b' becomes 'o', 'n' becomes 'a', etc.). Spaces, digits, punctuation, etc., are not changed. (An advantage of this scheme is that if you apply it twice, you get the original text back. Think about why!)

   Write a C program that, given two command-line arguments *infile* and *outfile*, encodes the text from *infile* using rot13 and writes the result to *outfile*. It should print an appropriate error message if called with fewer than two arguments, or if either file cannot be opened.

   For example, if *infile* contains

   ```
   Now is the time for all good persons to come to the aid of their
   party.  Hello world!  1234  !@#$
   ```

   then *outfile* will contain the following

   ```
   Abj vf gur gvzr sbe nyy tbbq crefbaf gb pbzr gb gur nvq bs gurve
   cnegl.  Uryyb jbeyq!  1234  !@#$
   ```

   *Hints:*

   - The "sample programs" page contains an example of reading an input file one character at a time and copying it to an output file, which you could use as a starting point: copy-file.c[1]

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2011spring/SamplePrograms/Programs/copy-file.c`

- You could do the actual encoding in a function, perhaps along the lines of this one:

```
#include <string.h> /* for strchr */
/* encode one character */
int encode(int inchar) {

    char* alphabet_lc = "abcdefghijklmnopqrstuvwxyz";
    char* alphabet_uc = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    if (strchr(alphabet_lc, inchar) != NULL) {
        /* encode inchar using alphabet_lc and return */
    }
    else if (strchr(alphabet_uc, inchar) != NULL) {
        /* encode inchar using alphabet_uc and return*/
    }
    else
        return inchar;
}
```

2. (10 points) Write a C program that performs some of the same functions as the UNIX command `grep` — searches in a file for all lines that match a given search string, and prints the ones that do. For example, if you have a file containing the following lines:

```
Hello world!
There is a German word for worldview that I have forgotten.
World without end.
Goodbye!
```

and you search for `world`, the first and second lines should print. (It's easiest to do the match in a case-sensitive way, so the third line does not match.)

The program should accept two command-line arguments, the first containing the text to search for and the second containing the name of the file to search. (For extra credit, make the program work more like `grep`, which allows searching any number of files, or none, in which case it searches for the specified text in standard input. If you do this, and more than one file is specified, print matching lines in a way that makes it clear which file they came from. `grep` does this by preceding each matching line with the name of the file it came from, e.g.,

```
input.txt: There is a German word for worldview that I have forgotten.
```

but you may think of some format you like better.)

For this problem, read the file a line at a time, but be as careful as you can about reading text — it is very easy to read more characters than will fit into the character array you're putting them in. (I usually recommend using the library function `fgets` for reading text line by line.) If the input file contains lines that are too long, you can just print warning messages about them.

*Hints:*

- The "sample programs" page contains an example of using `fgets()` to get a whole line of text and echo it:

  echo-lines.c[2]

  `fgets()` puts a newline character in its output area if it was able to read the whole line; if the line was too long, it does not. If that happens, the sample program discards the rest of the line. You could do something similar.

- You may find the library function `strstr()` useful in searching for matching text.

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2011spring/SamplePrograms/Programs/echo-lines.c`