Lab 2: UNIX stdio I/O and Strings

Please read this entire assignment. Take some notes. Think about it some. Take some more notes. Look for answers to your questions in this document.

Due: As shown in Canvas. Submit a single tar.gz file.

Do not place ANY directories in your submitted tar file. I will not change into any sub-directories to hunt down your source files. When you create your `tar.gz` file to submit, do it within the directory where you created the source files, **NOT from a higher level directory. If I cannot find your source files in the same directory where I extract your submitted tar file, I will simply give you a zero on the assignment**. Submit a single `tar.gz` file to Canvas for this assignment.

In this assignment, you will be working with UNIX file I/O system calls and library functions. **I urge you to not delay beginning it**.

Part 1 – Makefile (50 points)

Write a Makefile to build the other parts.

1.1. You must have a single `Makefile` for this lab. Your `Makefile` must contain (at least) the following targets:

Target	Action
`all`	Builds all dependent C code modules for your applications in the directory. This should be the default target in your `Makefile`.
`caesar`	Builds all dependent C code modules for the `caesar` application in the directory. The dependency of `caesar` should be `caesar.o`.
`caesar.o`	Compiles the `caesar.c` module for the `caesar` application in the directory, based off of any changed dependent modules. The dependency for `caesar.o` should be `caesar.c`.
`xor`	Builds all dependent C code modules for the `xor` application in the directory. The dependency of `xor` should be `xor.o`.
`xor.o`	Compiles the `xor.c` module for the `xor` application in the directory, based off of any changed dependent modules. The dependency for `xor.o` should be `xor.c`.
`clean`	Deletes all executable programs, object files (files ending in `.o` produced by `gcc`), and any editor chaff (# files from vi and ~ files from **emacs**). Make sure you use this before you bundle all your files together for submission.

1.1.1. When I build your assignment, I should be able to just type

```
make clean
```

```
make
```

to have it completely clean the directory and build both `caesar` and `xor`.

1.1.2. I also recommend that you **use some form of revision control on your source files**. Not only does this reduce the possibility of catastrophic file loss, but it is a LOT better than making `.BAK1`, `.BAK17`, `.BAK4c` copies of your code.

1.1.2.1. Putting this into your `Makefile`, as described in the notes about `make` would make your life better.

1.1.3. You must compile your program using the following flags for `gcc`

1.1.3.1. Putting these flags into the `Makefile` with the `CFLAGS` variable will make your life better.

1.1.3.2. When compiled with the flags, the `gcc` compiler should emit no errors or warnings.

1.1.3.3. **Any warnings from the compiler is an automatic 20% deduction**.

```
-Wall -Wextra -Wshadow -Wunreachable-code -Wredundant-decls
-Wmissing-declarations -Wold-style-definition
-Wmissing-prototypes -Wdeclaration-after-statement
-Wno-return-local-addr -Wunsafe-loop-optimizations
-Wuninitialized -Werror
```

Part 2 – Caesar Cipher (100 points)

This part is about *reading text from stdin and writing text to stdout*.

You will be implementing the Caesar cipher. Unlike the original Caesar cipher (which only worked with an early Latin alphabet of 23 letters), your implementation will work on all the printable ASCII characters (space to ~). All non-printing characters are ignored.

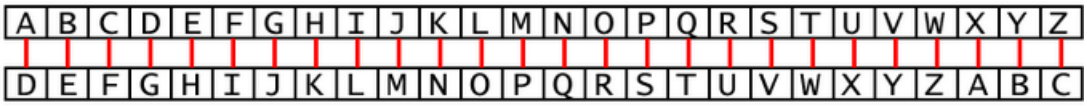
Your program will read input from **`stdin`**, shift the characters by the shift amount (a value >= 0 and <= 95) and then print the encrypted text back to **`stdout`**. All the text in this portion of the assignment is plain ASCII text.

Command line options:

Option	Action
`-e`	Encrypt `stdin` and print it to `stdout`. If neither `-d` nor `-e` are given on the command line, the default is to encrypt.
`-d`	Decrypt `stdin` and print it to `stdout`. If neither `-d` nor `-e` are given on the command line, the default is to encrypt.
`-s #`	The amount by which the characters are shifted. Must not be less than 0 and not greater then 95.

If neither the -e (encrypt) nor the -d (decrypt) are on the command line, the default is to encrypt.

The valid range for the shift amount is greater-than or equal to 0 and less-than or equal to 95. If a shift value is not given on the command line, the default value is 3 (just like Julius Caesar). A shift value of 0 or of 95 will result in the input and output being the same.



Alphabet shifted by 3 spaces.

A simplified representation of the Caesar Cipher, with a shift of 3.

If the shift value is out of range (less than 0 or greater than 96), issue an error message and exit.

If the shift value is not a valid base-10 number, issue an error message and exit. An example command line for this could look like:

```
./caesar -s jesse
```

Use `getopt()` to handle the command line. This is a valid command line:

```
./caesar -s 17 -e -s 13 -d -s 4 -eds9
```

This really is just a text in, transform, text out application. If you are typing input from the keyboard and need to end the input, type a Control-d (the end-of-file character in Unix).

Your source file must be named **`caesar.c`**. My solution has 134 lines.

Part 3 – `xor` Cipher (100 points)

Moving on from the simple to break Caesar cipher, you will now implement the `xor` cipher. In the Caesar cipher, the key was a single value, the key can have multiple characters.

Command line options (not much here):

Option	Action
`-k string`	The string used is encrypt/decrypt the input.

Since the **`xor`** cipher is a symmetric encryption algorithm, the encryption key is the same as the decryption key. Unlike the Caesar cipher we implement above, the output from the encryption using xor cipher may not be plain text. Similarly, the input for decryption may not be plain text. This means that you CANNOT use things like **`scanf()`** or **`fgets()`** for input and CANNOT use **`printf()`** or **`fputs()`** for output. **You must use `read()` for input and `write()` for output.**

If an encryption/decryption key is not specified on the command line (using `-k <string>`), the default value is "`Xerographic`". Since the key is a string, any printable characters in the key are acceptable.

Your source file must be named **`xor.c`**. My solution has 71 lines.

Testing Your Code

Under development. Check back soon.

Submitting your code

When you are ready to submit your code, create a single tar.gz file for the C source files and the `Makefile`. Upload the file into Canvas. Your submitted file tar.gz should contain exactly 3 files: **caesar.c, xor.c**, and **Makefile**. Additional files or any directories can result in a grade of zero. Creating a target in your Makefile to automate creation of the tar.gz file would be an excellent decision. If you don't remember how to do this, check the slides about make, or ask me or the TA.

Final note

The labs in this course are intended to give you basic skills. **In later labs, we *assume* that you have mastered the skills introduced in earlier labs.** If you don't understand, ask questions.