CS 215 - Fundamentals of Programming II
Spring 2017 - Homework 1
10 points

Out: January 13, 2017
Due: January 18, 2017 (Wednesday)

The purpose of this homework is to practicing using the development environment expected in this course. Therefore, please type in your program from scratch in the UNIX environment and used the clang++ compiler for this homework. **Please ask for assistance sooner rather than later if you need it.**

A *cipher* is a form of cryptography in which individual letters of a message are replaced by other letters or symbols. The original form of the message is called the *plaintext*. It is enciphered using the cipher into the *ciphertext*. The act of recovering the plaintext is called *deciphering*.

A very simple cipher was used by the Romans and is now called the Caesar shift cipher. In the Caesar shift cipher, an alphabetic character (i.e., a letter) is replaced by the letter located *n* places to the right in the alphabet circularly. Another way to say this is that the plaintext letter is shifted circularly to the right *n* places to obtain the ciphertext letter. For example, if the shift is 8, then 'A' becomes 'I', 'B' becomes 'J', etc. to 'Z' becomes 'H' This idea also can be represented as a substitution table, as shown below for a shift of 8.

Plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H

Using the corrected file copying program from Lecture 3 as a base, modify it to perform a Caesar shift encipherment of **only** the alphabetic characters in the input file before writing the (enciphered) character to the output file. The source code for this program must be in a file named **`cipher.cpp`**.

The encipherment must be **case sensitive**. That is, an uppercase letter must map to another uppercase letter (e.g., 'A' to 'I'), and a lowercase letter must map to another lowercase letter (e.g., 'a' to 'i'). The non-alphabetic characters (i.e., digits, punctuation, and whitespace) must be passed through as plaintext.

The program must accept exactly **three** command-line arguments that are the name of an input file, the name of an output file, and an integer. The input file is interpreted a plaintext message. The output file will contain the corresponding ciphertext. The integer is the shift distance. The program must do proper error checking of the number of command line arguments and the file opens. Here are some things to note:

- Since **`argv`** is an array of C-strings, the third command-line argument will need to be converted to an integer using the function **`atoi`** that is defined in **`<cstdlib>`**. It is used as follows:

```
int shift = atoi(argv[3]);
```

- To compute the right circular shift of a lowercase alphabetic character **`ch`**, we can use the following formula:

```
ch = (((ch - 'a') + shift) % 26) + 'a';
```

Computing the shift of an uppercase alphabetic character is similar using 'A' instead of 'a'.

- Deciphering a Caesar shift is the opposite of enciphering, but subtracting the shift value could result in a negative number. The equivalent positive shift value is 26 minus the shift value. This means that we can decipher any Caesar shift message in ciphertext with the same program by giving a shift value that is 26 minus the shift value used to encipher. E.g., if we encipher using shift value of 8, then to decipher the resulting ciphertext, we run the program using the ciphertext as the input file and a shift value of 18.

A reminder of how to compile this program using clang++:

```
$ clang++ cipher.cpp -o cipher -Wall
```

The `-Wall` option is used to turn on all of the warning messages. As usual, there shouldn't be any warning message unless you understand why it is being ignored. An example run of this program would look like:

```
$ ./cipher plaintext.txt ciphertext.txt 8
```

If file `plaintext.txt` contains the following "message":

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Then file `ciphertext.txt` should contain the following result:

```
ijklmnopqrstuvwxyzabcdefgh
IJKLMNOPQRSTUVWXYZABCDEFGH
```

Note: you create the `plaintext.txt` test file yourself using your text editor.

To decipher the above `ciphertext.txt`, you would run:

```
$ ./cipher ciphertext.txt newplaintext.txt 18
```

and `newplaintext.txt` would be exactly the same as the original `plaintext.txt`. You can check this by using the UNIX `diff` command:

```
$ diff plaintext.txt newplaintext.txt
```

When two files are exactly the same, the `diff` command produces no output.

At the beginning of class on Wednesday, January 18, the procedure for making electronic submissions will be demonstrated and you should try to submit your program at that time. If your program does not pass the submission tests at that time, you will have until 11:59pm to correct and resubmit for full credit. Please note that the automated submission system requires that all files be named exactly as specified in an assignment, and also requires that the output of the program be exactly as expected including whitespace.