Problem Set 2  
Out of 45 points  
Due Thursday, September 25th, 2014 at 11:55pm

Goals

• Get acquainted with functions and libraries
• Get acquainted with character values and strings

Recommended Reading

• Pages 11 – 14 and 39 of http://www.howstuffworks.com/c.htm
• Chapters 6, 7, 10, 17, 19, 21, 22, 30, and 32 of Absolute Beginner’s Guide to C
• Chapters 7, 8, and 10 of Programming in C

1. Print some lyrics (10 points)

Create a file in your ps2/ directory called oldman.c. In this file, write a program to print the song, “This Old Man.” The lyrics are as follows:

This old man, he played 1,  
He played knick-knack on my thumb;  
With a knick-knack paddy-whack, give a dog a bone,  
This old man came rolling home.

This old man, he played 2,  
He played knick-knack on my shoe;  
With a knick-knack paddy-whack, give a dog a bone,  
This old man came rolling home.

...

The remaining verses are

3 - on my tree  4 - on my door  
5 - on my hive  6 - on my sticks  
7 - up in heaven  8 - on my gate  
9 - on my spine  10 - on my hen

If you’re unfamiliar with the tune, there are plenty of versions to be found online (for a heavy-metal, anti-Barney version sung by dinosaurs, go to http://youtube.com and search for “this old man rockosaurus”).

Notice the repetition in the song’s verses. Clearly your program should make use of some sort of loop to generate repeated lyrics. Your program should probably also make use of one or more conditions in order to print the number referenced in each verse’s first line as well as where this old man played knick-knack in each verse’s second line. Using
switch is recommended but not required.

Before writing any code, think about how you might use hierarchical decomposition (i.e., more than one function) to solve this problem as efficiently and elegantly as possible. Make sure you comment your code adequately.

2. Caesar’s cipher (15 points)

In a file called caesar.c, write a program that encrypts messages using Caesar’s cipher. To review, Caesar’s cipher is a simple encryption technique in which each letter is replaced by another letter some fixed number of positions down the alphabet. For example, for a shift of 3, the letter 'A' would be replaced by a 'D', 'B' replaced by 'E', and so on.

Your program must accept a single command-line argument: a non-negative integer, k. If your program is executed without any command-line arguments, with more than one command-line argument, or with one command-line argument that contains any characters other than '0' through '9', your program should complain and exit immediately. Otherwise, your program must proceed to prompt the user for a string of plaintext and then output that text with each alphabetical character “rotated” by k positions; non-alphabetical characters should be outputted unchanged. After outputting this cipher text, your program should exit.

Although there exist only 26 letters in the English alphabet, you may not assume that k will be less than or equal to 26; your program should work for all non-negative integer values of k less than $2^{31}$. Even if k is greater than 26, alphabetical characters in your program’s input should remain alphabetical characters in your program’s output. For instance, if k is 27, 'A' should not become '[', even though ']' is 27 positions away from 'A' in ASCII; 'A' should become 'B', since 27 modulo (%) 26 is 1. In other words, values like k = 1 and k = 27 are effectively equivalent.

Your program must preserve case: capitalized letters, though rotated, must remain capitalized letters; lowercase letters, though rotated, must remain lowercase letters.

Your program must also use the inputlib functions for input, in particular the function getString(). You can download inputlib.h and inputlib.c from
this web directory:

http://marl.smusic.nyu.edu/CProgramming/Fall14/src/general/

To use them, put them both in your ps2/ directory. In order to use the inputlib functions, add the following include statement in your program:

#include "inputlib.h"

When you compile your program, you will want to add inputlib.c to the command line:

gcc -Wall caesar.c inputlib.c -o caesar

Functionally, your program might resemble the below. Underlined are some sample inputs.

prompt % caesar 20
Enter plaintext: The dog ate my 2618 homework!
Ciphertext: Nby xia uny gs 2618 bigyqile!

So that you don’t reinvent the wheel, check out
http://www.cppreference.com/stdstring/index.html for any functions that might be of assistance to you. As will often be the case, there is more than one way to solve the problem at hand. But be aware that isdigit and atoi might come in handy. You should probably dig up the ASCII chart handed out in class. And best not to forget about an operator like %.

3. Vigenère’s cipher (20 points)

Your next challenge is to write, in vigenere.c, a program that encrypts a message using Vigenère’s cipher. Vigenère’s cipher uses a Caesar cipher with a different shift at each position in the text; the value of the shift is determined by a repeating keyword.

For example, suppose the plaintext to be encrypted is:

WATERTHEPLANTS

The person sending the message chooses a keyword and repeats it until it matches the length of the plaintext. In the case of the keyword TURNIP it would be:

TURNIPTURNIPTU

Plaintext: WATERTHEPLANTS
Key: TURNIP
Ciphertext: PUKRZIAYGYYICMM

For example, here is a visualization of how one of the characters is encoded:

WATERTHEPLANTS
TURNIPTURNIPTU
PUKRZIAYGYYICMM

Letters used in the plaintext message are in the top row, and the keyword letters are in the leftmost column:

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

Your program must accept a single command-line argument: a keyword, \( k \), composed entirely of alphabetical characters. If your program is executed without any command-line arguments, with more than one command-line argument, or with one command-line argument that contains any non-alphabetical character, your program should complain and exit immediately. Otherwise, your program must proceed to prompt the user for a string of plaintext, \( p \), which it must then encrypt according to Vigenère’s cipher with \( k \), ultimately printing the result and exiting.

As for the characters in \( k \), you must treat ‘A’ and ‘a’ as 0, ‘B’ and ‘b’ as 1, . . . , and ‘Z’ and ‘z’ as 25, just as we did in lecture. In addition, your program must only apply Vigenère’s cipher to a character in \( p \) if that character is a letter. All other characters (numbers, symbols, spaces, punctuation marks, etc.) must be outputted unchanged.

Moreover, if your code is about to apply the \( i \)th character of \( k \) to the \( j \)th character of \( p \), but the latter proves to be a non-alphabetical character, you must wait to apply that \( i \)th character of \( k \) to the next alphabetical character in \( p \)—you must not yet advance to the next character in \( k \). Finally, your program must preserve the case of each letter in \( p \).

Functionally, your program might resemble the below. Underlined is a sample input.

```
prompt % vigenere Foobar
Enter plaintext: Hello, World.
```
Ciphertext: Mszmo, Ntfze.

4. Submitting Your Work

You need to submit all .c files (including inputlib.h and inputlib.c) on New Classes. As you did for ps1, rename and create a compressed archive of your ps2 files: first cd to the directory above ps2, then rename your directory by typing

```bash
mv ps2 ps2_Lastname_Firstname
```

Then compress the directory by typing

```bash
tar cvzf ps2_Lastname_Firstname.tgz ps2_Lastname_Firstname
```

Now submit `ps2_Lastname_Firstname.tgz` via the NYU Classes platform. Make sure to submit before the problem set’s deadline, or you won’t be graded. Be warned that the timestamps for submitted work are based on the server’s clock, not your computer’s clock.