Computer Programming and Data Structures: MTH 409
Department of Mathematics and Statistics
Indian Institute of Technology - Kanpur August 27, 2014

Practice Problem Set 2

A. The Caesars cipher is a type of encryption strategy where messages are scrambled rotating each letter by k positions, wrapping around from 'Z' to 'A' as needed. Here, k, a non-negative integer, is known as the *secret key*. To elaborate further on how this encryption methodology works, let m be some text message to be encrypted and m_i is the $i-$th character in m. Then the $i-$th letter c_i in the ciphertext (encrypted text), c, is computed as

$$c_i = (m_i + k) \ \% \ 26.$$

For example, suppose that the secret key, k, is 13 and that the message, m, also known as plaintext, is "Be sure to submit your homework on time!" When we encrypt this m with this k, we get the ciphertext, c, "Or fher gb fhozvg lbhe ubzrjbex ba gvzr!" by rotating each of the letters in m by 13 places. Notice how O (the first letter in the ciphertext) is 13 letters away from B (the first letter in the plaintext). Similarly r (the second letter in the ciphertext) is 13 letters away from e (the second letter in the plaintext). And so on.

Your goal is to implement, in caesar.c, a program that encrypts messages using Caesar's cipher. Your program must accept a single command-line argument: a non-negative integer, k. If your program is executed without any command-line arguments or with more than one command-line argument, your program should complain about it to the user by printing a message on the screen and return a value of 1 (which is used to indicate an error) immediately via the statement below:

return 1;

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 ciphertext, your program should exit, with main returning 0.

Note that even if k is greater than 26, an alphabetical character in your program's input should remain an alphabetical character in your program's output. Moreover, a capitalized alphabetical character should remain capitalized and a non-capitalized should remain a non-capilalized character.

B. Your next goal is to implement, in a file called vigenere.c, an improved encryption algorithm that uses a string composed entirely of alphabetical characters, called the *secret keyword*, instead of a non-negative integer as secret key used in Caesar's cipher. In Vigenere's cipher, we can view this keyword as a sequence of keys that are used for "rotating" alphabets according to the rule

$$c_i = (m_i + k_j) \ \% \ 26.$$

where the $i-$th character c_i of the ciphertext is obtained from the m_i, the $i-$th character in the message m and k_j, the $j-$th character of the secret keyword, which is obtained as follows : 'A' and 'a' is teated as 0, 'B' and 'b' as 1, ... , 'Z' and 'z' as 25. Obviously, a difficulty arises when length of m is greater that the length of k, that is, there are more characters c_i than k_j in above equation. This can be remedied by using the characters in k in a cyclic order for applying the encryption rule.

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, with main returning 1 (thereby signifying an error that our own tests can detect). Otherwise, your program must proceed to prompt the user for a string of plaintext, m, which it must then encrypt according to Vigenere's cipher with k, ultimately printing the result and exiting, with main returning 0. In addition, your program must only apply Vigenere's cipher to a character in m 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 $j-$th character of k to the $i-$th character of m, but the latter happens to be a non-alphabetical character, you must wait to apply that $j-$th character of k to the next alphabetical character in m; you must not yet advance to the next character in k. Finally, your program must preserve the case of each letter in m as you did in the first problem.