Tara Moses

S. Gauch

Programming Foundations 1

February 23, 2015

**Problem Statement:**

This program was designed to take in a four-letter message and encrypt or decrypt it using the Caesar cipher method. The user inputs a prime number key to shift the message, chooses whether to encrypt or decrypt the message by inputting either an ‘e’ or a ‘d’, and inputs the message itself. The program outputs the encrypted or decrypted message. If the user inputs a composite number as the key, the program prompts the user to try again. Similarly, if the user inputs a character other than an ‘e’ or ‘d’, the program asks again. The program also checks whether the user’s message is made of 4 letters. If the user enters a character that isn’t alphabetical, then the letter is asked for again and again until it is acceptable.

**Design:**

The program was designed to be very function-heavy; there are functions to obtain the encryption/decryption choice, to obtain and check the prime number key, and to obtain the message. There were no data structures used, but the message was read in one char at a time and concatenated into a string for the final result. When checking to make sure the input key was prime, the algorithm used was basically a function consisting of a for-loop that returned false if the key was divisible by anything between 2 and the square root of the key itself. If the for-loop was exited before the key was found to be divisible by anything, then the function returned positive.

The main advantage of creating a program laden with functions was to make the main function easier to read, seeing as this was the first program in the class where collaboration was allowed. The isPrime() function was particularly advantageous because it made it easy to handle user input errors in the function where the prime number key was collected.

**Implementation:**

After adding comments to outline the program, the functions were filled in. The shift() function, which took in a letter, the prime key, and the encoding/decoding character and returned the corresponding letter, was the first to be finished since it was the most crucial. The functions that asked for the prime key, the encoding/decoding character, and the message itself were all written in turn. After the functions were done, the main function was written. Then error-handling was added to each of the user-input functions.

We didn’t start with any sample code. The functions themselves took about fifteen minutes to complete, excepting the Shift() and isPrime() functions, which required more critical thinking and were therefore finished in about thirty minutes each.

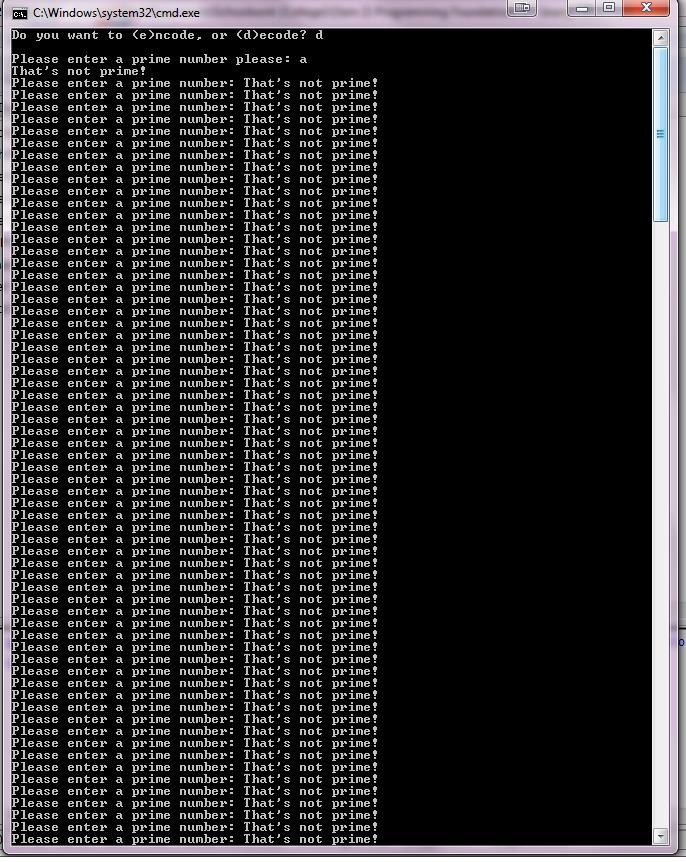
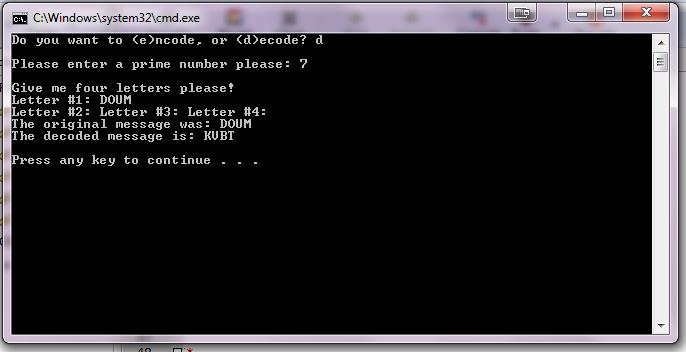
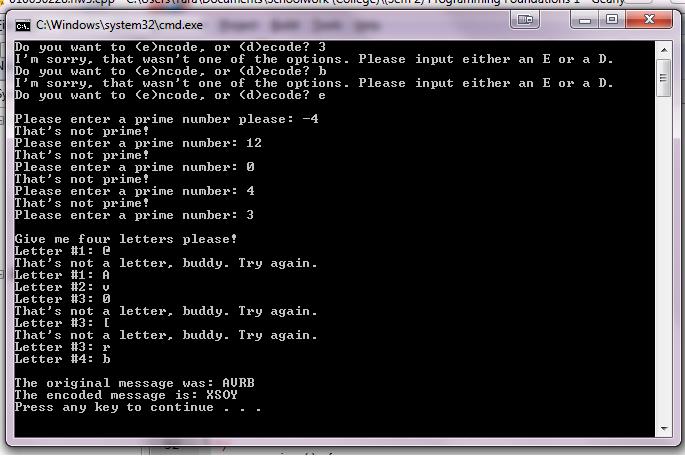
**Testing:**

After each function was written, it was tested with border cases as well as working cases. Error-handling wasn’t tested until everything else was written and worked. The function that gathered the message from the user was tested to make sure that the program didn’t accept any non-alphabetical values. The letters A and Z were tested, as were the characters ‘@’ and ‘[‘ because those characters had ascii values of 64 (one below A) and 91 (one above Z). Various integers were used to test the isPrime() function, including negative numbers.

**Conclusions:**

Overall, the project was a success. The program successfully takes in a four-letter message and encrypts or decrypts it. In the future, I might want to make a program that can encrypt more than one message before the program ends. It was pretty tedious to have to run the program multiple times for all the tests. Also, I would have liked to have made the isPrime() function more efficient by modding the prime key by 26 before sending it through the loop. If the user inputs, say, 26\*100 + 3, it would take a long time for the for-loop to finish when changing the prime key to 3 would yield the same encrypted message but with a much shorter run time.

**Tests:**

****