Documentation – CIS7 Course Project, Case 3

Team Solace – Jacob Arsenault

In this project, with the case I decided to take on, I was tasked with solving the problem of creating a program that utilized a Vigenère cipher that would both encrypt and decrypt messages from the user. With the structure I had in mind, the user would also be able to choose which they wanted to do at the time and continue with decryption or encryption if they still had something they wanted to decrypt or encrypt. In order to solve this problem, I needed to implement a few algorithms for key generation as well as the encryption and decryption itself.

After the user decided which they wanted to do, the algorithm for key generation would work by using the plaintext and cipher strings the user would feed into it shortly afterwards. The algorithm would then take the size of the plaintext string as an int and use it in a for loop; the loop would push back into the key the char value based on the current value of i, using the size it had taken to loop i back around, until the size of the key was the size of the plaintext string. The newly generated key would then be returned for further use.

Next, if the user had decided they wanted to encrypt something, the encryption algorithm would then take the original plaintext string and the key that had just been generated as an argument and create a string to hold the returned text once it was ready. It would then utilize a for loop, running as long as i was less than the plaintext string’s size, wherein the numeric value of the string would be run through the encryption formula ((Pi + Ki) % 26) before being converted back into an ASCII value by adding ‘A’ and being pushed back into the string holding the encrypted text. This would, of course, run until the plaintext had been fully run through the algorithm, at which point it would return the encrypted text to be displayed to the user.

The process for decrypting was similar in execution, such as creating a string to hold the decrypted text, utilizing a for loop, adding ‘A’ to the value once it had been run through the formula and pushing the result back into that string. The formula itself, however, had to be altered to properly decrypt; the formula for decryption was (Ei – Ki + 26) % 26. This is because, because the encryption was shifting the value over to the ‘right’, representing A-Z as 0-25, the way to represent that would be addition; meanwhile, decryption would go in the opposite direction, requiring subtraction, and 26 would need to be added afterwards in order to make sure that the values don’t go into the negatives, as those don’t translate into ASCII accurately.

After encryption or decryption, the program would ask if there was anything else the user wanted to encrypt or decrypt; if the user inputted ‘Y’, then the program would loop back to the beginning, and if the user inputted ‘N’, then the program would simply end. This makes it so the user can encrypt or decrypt as many things as they like without having to stop and restart the program over and over again.

There are limitations on the program, however—namely, the things that the program can and cannot encode. The program as it currently is requires both the plaintext/encrypted phrase and the keys to be in allcaps, as it does not necessarily deal with lowercase letters accurately. It has similar issues dealing with numbers, as well as spaces in the plaintext/encrypted phrase or the key phrase. To improve on these limitations, I would tweak the algorithm a little more to see if I can’t create one capable of properly encrypting and decrypting these characters.

N

Y

Print encrypted phrase to user

End

Ask if anything else is needed

Print decrypted phrase to user

Run key & phrase through encryption algorithm, return encrypted phrase

Run key & phrase through decryption algorithm

Create key

Create key

Decrypt – Ask for encrypted phrase and cipher

Encrypt – Ask for plaintext and Cipher

D

E

If statement – Encrypt or Decrypt?

Start