Whatcom COMMUNITY COLLEGE

NCYTE CENTER

VIGENÈRE CIPHER LESSON

Sean Gallop – Colorado Academy, Boulder, CO

This material is based upon work supported by
the **National Science Foundation** under Grant No.1548315.

Additional materials may be found at www.ncyte.net

VIGENÈRE CIPHER

MULTI-DAY PROGRAMMING EXERCISE

Students write a program which demonstrates the key generation, encryption, and decryption aspects of the Vigenère Cipher methodology.

PURPOSE

The three purposes of this exercise are

1) for students to develop a deeper and more connected understanding of Substitution Ciphers
2) for students to become more practiced programmers, and
3) to connect student learning between simple Caesar Ciphers and the substantively more complex Public Key Exchange methodologies.

ASSIGNMENT

Students model the three important aspects of the Vigenère Cipher encryption methodology: 1) generate a random, character-based key which has the same number of letters as the starting cleartext, 2) encrypt the cleartext (letter by letter) using key, and 3) decrypt the ciphertext by using the key. The starter code includes 1) detailed instructions, 2) Global Constants, 3) representative versions of cleartext, ciphertext, and the key, and 4) some starter code.

Students must do all of the following:

1) Create a random, character-based key
2) Create a ciphertext version of the cleartext based on the generated key
3) Create a cleartext version of the ciphertext based on the generated key

NOTES

Students should be able to complete the initial key generation part with little help. The encryption and the decryption parts of this assignment will require some teacher input. This input will come in three different ways.

Firstly, the teacher should direct student to review the code which they produced as part of the Caesar Cipher CCL. In reality, the Caesar Cipher is a subset of the Vigenère Cipher in that a Caesar Ciphers are a Vigenère Cipher in which the key letter does not change. Given this similarity, students already have reliable starter code with which they are very familiar.

Secondly, the teacher should direct students to review the line-by-line comments included with the Encrypt() and Decrypt() functions. These comments are sufficiently specific that students will be able complete this assignment if they encode those comments directly.

Thirdly, the teacher should review with students the specific steps that are required to complete the Encrypt() and Decrypt() functions. By breaking down a relatively complicated process into a series of simple, sequential steps students will get much needed practice at computational thinking. The GenerateKey(), Encrypt(), and Decrypt() line-by-line steps are detailed immediately below and as part of the introductory presentation. Given the complexity of these algorithms and the lack of programming experience held by the students, the teacher should encourage the students to avoid trying to program these two functions independently. There are, in fact, many ways to program these algorithms, it is unlikely however that students will be able to program at this level of complexity without much unnecessary frustration. The teacher should encourage students to start with the programming of the GenerateKey() algorithm. That algorithm is suitable for students' first programming attempt in that the algorithm is similar to but substantively simpler than the Encrypt() and Decrypt() algorithms.

1) GenerateKey(**cleartext**) The User-defined function, GenerateKey() returns a string key (**keyString**) which is of the same length as the cleartext and takes a single parameter: **cleartext**.
2) A zero-character string (**keyString**) should be defined at the top of GenerateKey(). **keyString** will eventually be the Vigenère Cipher key which will be returned out of GenerateKey().
3) A **for** loop whose index spans the length of **cleartext** is implemented. Students will use the **for** loop to add one (1) random character to **keyString** for each and every character in **cleartext**.

4) Create a variable called **randomAlphabetIndex**. **randomAlphabetIndex** is assigned a random integer value between 0 and 25 (number of letters in the English Alphabet minus one).
5) Next, append to **keyString** the letter of **ALPHABET** which is indexed by **randomAlphabetIndex**.
6) Once the **for** loop has visited and processed every corresponding letter in **cleartex**, **keyString** is returned out of GenerateKey().

The **Encrypt()** algorithm shares several similarities with the **GenerateKey**() algorithm (described immediately above). Firstly, both algorithms require access to a list of all of the capitalized letters of the English Alphabet (**ALPHABET**). Secondly, both algorithms start with an empty string variable (e.g., **key** immediately above). The return value for both algorithms is built up letter-by-letter from this initially empty string variable. Thirdly, both algorithms use a **for** loop as the iterator for the respective data processing steps.

1) Encrypt(**cleartext**,**keyString**) The User-defined function, Encrypt() returns the encrypted cleartext (**ciphertext**) and takes two parameters: a) the cleartext (**cleartext**) and b) the key string (**keyString**).
2) A zero-character string (**ciphertext**) should be defined at the top of the user-defined function. **ciphertext** will eventually be the encrypted cleartext which will be returned out of the user-defined function.
3) Vigenère ciphers require and the code depends on the length of **keyString** and **cleartext** being the same. A conditional statement should precede the **for** loop which verifies that these two parameterized variables are of the same length.
4) A **for** loop whose index spans the length of **keyString** and **cleartext** is created. Students will use the **for** loop's control variable (**i**) to manipulate each character of the **cleartext** based on the corresponding character in the **keyString**.
5) The first thing to do inside the for loop is to isolate the cleartext letter which we wish to encrypt and assign it to a variable (**clearTextLetter**)
6) Next, we isolate the corresponding **keyString** letter and assign it to a variable (**keyLetter**)
7) We then calculate the encryption shift (**shift**) by measuring the distance of **keyletter** distance from the start of the alphabet ('A'). This is accomplished by subtracting the **keyletter** from A'.
8) **alphabetIndex** in the next line is the integer distance of **cleartextLetter** from the start of the alphabet ('A').
9) Now that we know the index of the **cleartextLetter** (**alphabetIndex**) and the amount of the shift implied by the **keyLetter** (**shift**), we can determine

the index of the encrypted letter (**ciphertextIndex**) by adding **alphabetIndex** and **shift**.

10) An if statement is required here because it is possible that **ciphertextIndex** will be greater than the length of alphabet. For example, an **alphabetIndex** of 24 ('Y') and a shift of eight would result in **ciphertextIndex** being assigned the value 32. The program would fail on the last line before the return if we use a **ciphertextIndex** value of 32 as an index into **ALPHABET**. We address this by subtracting 26 (the number of letters in the alphabet) in the event that **shift** is greater than or equal to 26. This produces the desired wrap-around effect.
11) The last command in the **for** loop is to append the letter indexed by **ciphertextIndex** to the growing ciphertext variable (**ciphertext**). This produces the desired wrap-around effect.
12) Once the **for** loop has visited and processed every corresponding letter in **ciphertext** and **key**, the ciphertext variable (**ciphertext**) is returned out of the User-Defined function.

The Decrypt() algorithm is very similar to the Encrypt() algorithm. Both algorithms take two parameters (a String text value and a Vigenère key), and both return a String value. Both algorithms build up a returned String character-by-character. The main difference between the two algorithms is that the Encrypt() algorithm creates ciphertext by shifting cleartext characters to the right by the corresponding character value in the key. Decrypt() algorithm creates cleartext by reversing the process. Where Encrypt() created cipher text by adding a shift value, Decrypt creates cleartext by subtracting a shift value based on the corresponding letter in the key. For this reason, where Encrypt() must verify that the addition has not resulted in a value greater than 26, Decrypt() must verify that the subtraction operation has not resulted in a value less than zero.

Otherwise, the two algorithms are identical I have even tried to use similar variable names (where reasonable) to highlight these similarities.

1) Decrypt(ciphertext,keyString) The User-defined function, Encrypt() returns the decrypted cleartext (**cleartext**) and takes two parameters: a) the ciphertext (**ciphertext**) and b) the key string (**keyString**).
2) A zero-character string (**cleartext**) should be defined at the top of the user-defined function. **cleartext** will eventually be the decrypted ciphertext which will be returned out of the user-defined function.
3) Vigenère ciphers require and the code depends on the length of **keyString** and **ciphertext** being the same. A conditional statement should precede

the **for** loop which verifies that these two parameterized variables are of the same length.

4) This is followed by a **for** loop whose index (**i**) spans the length of **keyString** and **ciphertext**. Students will use the **for** loop's control variable (**i**) to manipulate each character of the **ciphertext** based on the corresponding character in the **keyString**.
5) The first thing to do inside the **for** loop is to isolate the **ciphertext** letter which we wish to decrypt and assign it to a variable (**ciphertextLetter**)
6) Next, we isolate the corresponding **keyString** letter and assign it to a variable (**keyLetter**)
7) We then calculate the encryption shift (**shift**) by measuring the distance of **keyletter** from the start of the alphabet ('A'). This is accomplished by subtracting the **keyletter** from 'A'.
8) **ciphertextIndex** in the next line is the integer distance of ciphertext from the start of the alphabet ('A').
9) Now that we know the index of the **ciphertextLetter** (**ciphertextIndex**) and the amount of the shift implied by the **keyLetter** (**shift**), we can determine the index of the decrypted letter (**cleartextIndex**) by subtracting **shift** from **ciphtertextIndex** (**ciphertextIndex-shift**).
10) An if statement is required here because it is possible that **cleartextIndex** will be less than zero. For example, an **ciphertextIndex** of 1 ('B') and a shift of eight would result in **cleartextIndex** being assigned the value -7. The program would fail on the last line before the return if we use a **cleartextIndex** value of -7 as an index into **ALPHABET**. We address this by add 26 (the number of letters in the alphabet) in the event that **shift** is negative. This produces the desired wrap-around effect.
11) The last command in the **for** loop is to append the letter indexed by **cleartextIndex** to the growing cleartext String variable (**cleartext**).
12) Once the **for** loop has visited and processed every corresponding letter in **ciphertext** and **key**, the cleartext variable (**cleartext**) is returned out of the User-Defined function.

IMAGES USED:

- Alessandro Cardinal Farnese by Titian. Public Domain.
- Blaise de Vigenère by Thomas de Leu. Public Domain.