Fay, Kamin
NCS 320
Final Project.

So for this final project I choose to do the coding portion of it, which includes implementing five ciphers of our choice. I decided to use Ceaser Cipher, Monoalphabetic Cipher, Vigenere Cipher, Rail Fence Cipher, and Row Transposition Cipher. I found these to be sort of the easiest and had no idea how I would have implemented Playfair cipher. From this project I have become every so curious as to how the opposite of this would work, how would decryption work it has me confused because in order to do most of these ciphers you need to remove spaces form the strings. So how would you know how to put them back in when decrypting. This is by far not my best work but also a good shot at creating the 5 ciphers.

House Keeping:

Github: https://github.com/AlwaysKamin/Encryption-Demo

So I discovered early on that there could be a few problems, like what if the user inputs a string that is to big, or what about if they choose something that is not on the given menu. Those are user entered problems, the other problems revolved around the ciphers themselves. The key problem would be dealing with spaces.

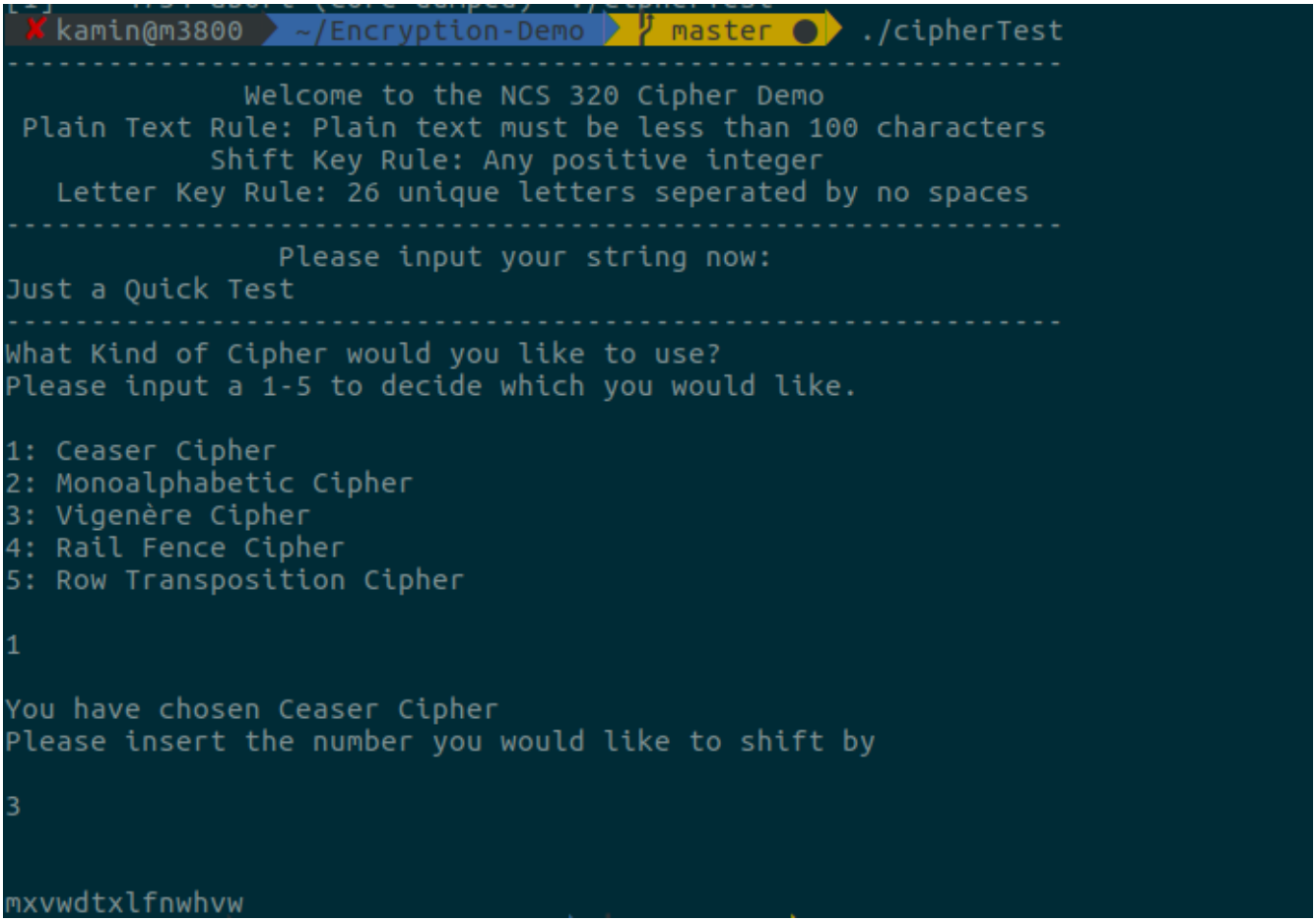
```
74      while(keyPress > 5 || keyPress == 0)
75      {
76          std::cin >> keyPress;
77          std::cout << std::endl;
78          if(keyPress <= 5 && keyPress > 0)
79          {
80              switch(keyPress)
81              {
```

The above screenshot is how I dealt with keyPresses. If the keypresses were not within 1-5 it would loop the program and ask the user to try again and input a different number.

```
180 /**
181 * parse Input takes in the inputed string and uses the .erase function as well as the
182 * standard remove function to find and remove all of the spaces in the function
183 * as well as sets it all to lowercase.
184 **/
185 void parseInput(std::string &inputText)
186 {
187   inputText.erase( std::remove( inputText.begin(), inputText.end(), ' ' ), inputText.end() );
188   std::transform(inputText.begin(), inputText.end(), inputText.begin(), ::tolower);
189 }
190
```

The above is the main things I want to look at there, it takes the inputed text, checks it, parses it and deletes all spaces and shifts the letters back together after doing that, it also transforms the text so that it is all lowercase. This makes it so the user can not see what letters were uppercase and which ones were lowercase.

Ceaser:

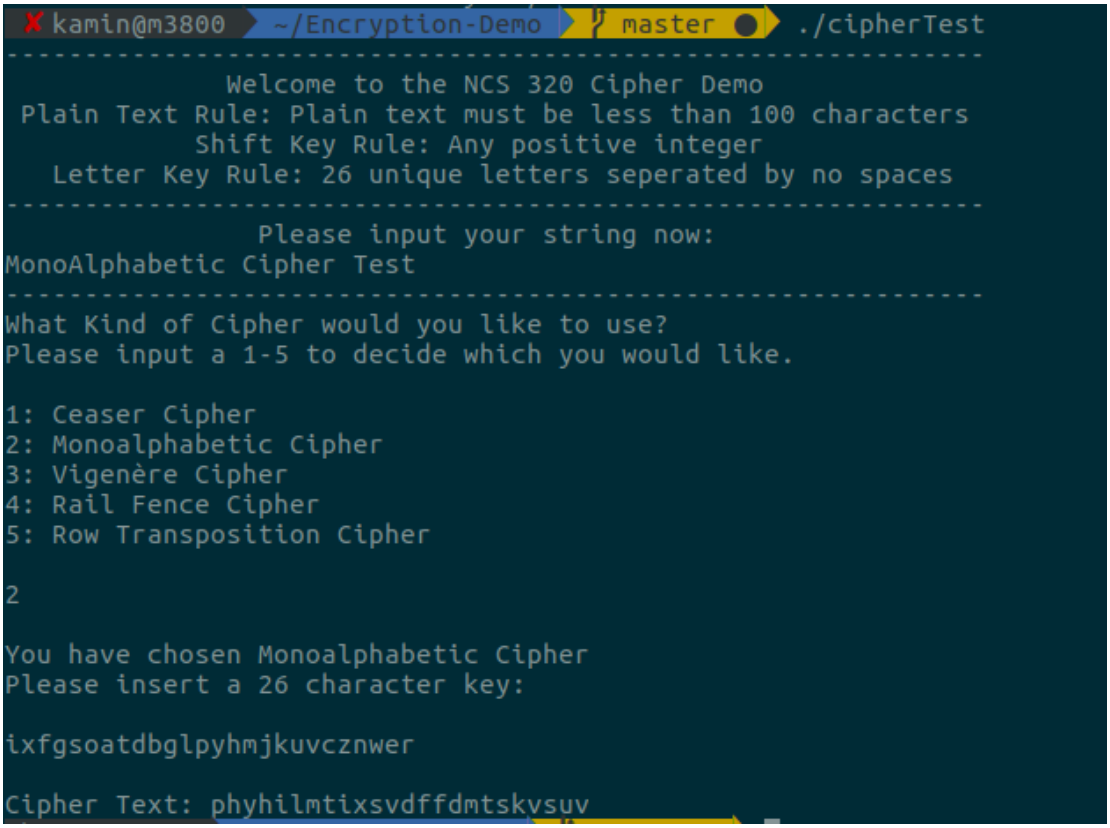


Above is a quick screen shot of my Ceaser Cipher, after inputing the string you would like to encrypt and selecting 1 for Ceaser Cipher the program will then prompt you for the key number that you would like to shift the letters. It then completes the shifting of the letters. This code can be seen below.

```
6  /*
7  * Deals with the encryption, gets sent the plaintext after it
8  * was parsed in the parseInput function in main.cpp. It was also
9  * passed the alphabet in a character array, and a key for shifting
10 */
11 void Ceaser::Encrypt(int key, std::string plainText, char* ALPHA)
12 {
13
14     for(int i = 0; i < plainText.length(); i++)
15     {
16         for(int j = 0; j < 26; j++)
17         {
18           if(ALPHA[j] == plainText[i])
19           {
20             //inserts the key# of leters over and places that in the cipher text spot
21             cipherText.insert(i, 1, ALPHA[(j + key) % 26]);
22           }
23         }
24     }
25 }
```

The code then calls the displayCipher Function within the Ceaser and the new cipherText is displayed to the screen.

MonoAlphabetic Cipher:



Above you can see the screenshot from the Monoalphabetic Cipher. After inputing the string to be encrypted and selecting the Monoalphabetic option you will be prompted to input a 26 character key. This is 26 letters from the alphabet completely randomized so that each is random but also a unique letter. So in other words my cipher does not take into account multiples of the same letter in the inputed text. And also doesn't check for that in the back end, so that could easily break the code. You can see the code for the Monoalphabetic encryption bellow.

```
3  /*
4  * Deals with the encryption, gets sent plaintext, a string key, and the alphabet
5  * Uses a unique 26! alphabet to generate the new string.
6  * for example a key of ixfqsoatdbglpyhmjkuvcznwer.
7  */
8  void MonoAlpha::Encrypt(std::string &plainText, std::string &stringKey, char* ALPHA)
9  {
10   for(int i = 0; i < plainText.length(); i++)
11   {
12     for(int j = 0; j < 26; j++)
13     {
14       if(ALPHA[j] == plainText[i])
15       {
16         cipherText.insert(i, 1, stringKey[j]);
17       }
18     }
19   }
20 }
21
```

The code then calls the displayCipher function and it displays it to the screen yet again.

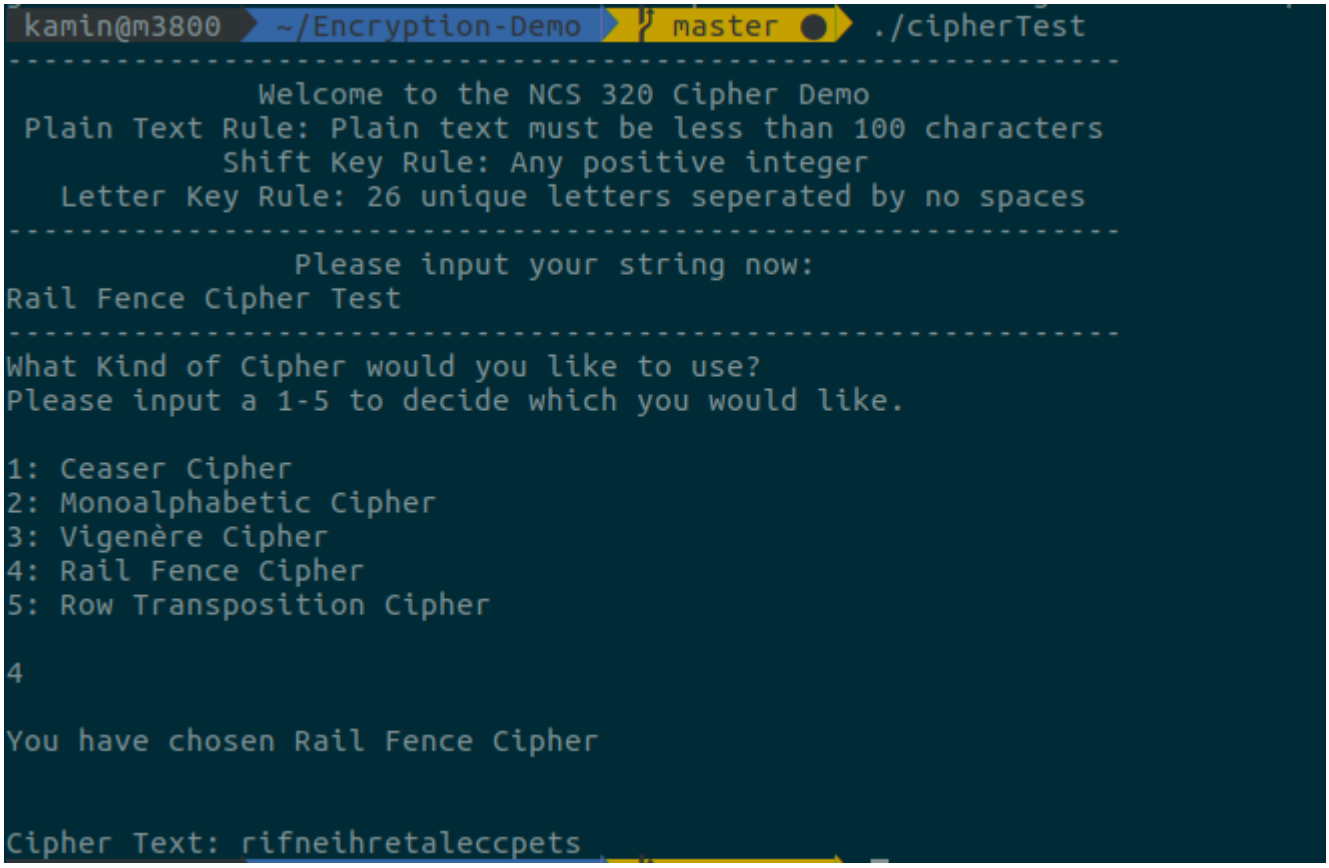
Vigenere Cipher:

```
kamin@m3800 ~/Encryption-Demo  master ● ./cipherTest
-------------------------------------------------------------
            Welcome to the NCS 320 Cipher Demo
 Plain Text Rule: Plain text must be less than 100 characters
           Shift Key Rule: Any positive integer
   Letter Key Rule: 26 unique letters seperated by no spaces
-------------------------------------------------------------
              Please input your string now:
we are discovered save yourself
-------------------------------------------------------------
What Kind of Cipher would you like to use?
Please input a 1-5 to decide which you would like.

1: Ceaser Cipher
2: Monoalphabetic Cipher
3: Vigenère Cipher
4: Rail Fence Cipher
5: Row Transposition Cipher

3

You have chosen Vigenère Cipher
Please input a key to use:

deceptive

PlainText: 22 4 0 17 4 3 8 18 2 14 21 4 17 4 3 18 0 21 4 24 14 20 17 18 4 11 5

Key: 3 4 2 4 15 19 8 21 4 3 4 2 4 15 19 8 21 4 3 4 2 4 15 19 8 21 4 3 4 2 4 15 19 8 21 4 3 4 2 4
15 19 8 21 4 3 4 2 4 15 19 8 21 4 3 4 2 4 15 19 8 21 4 3 4 2 4 15 19 8 21 4 3 4 2 4 15 19 8 21 4
3 4 4 15 19 21 4 4 4 21 4 4 4 21 4 4 4 21 4 4 4 21 4

CipherText: 25 8 2 21 19 22 16 13 6 17 25 6 21 19 22 0 21 25 7 2 16 24 6 11 12 6 9
```

The above screenshot shows the Vigenere Cipher in action, after first adding the string you would like to encrypt and selecting number 3 for Vigenere it will then prompt you for the key. This key will be repeated to be as long as the plainText so don't worry about that side of things. It will then display the plainText in number format, as well as the key, and cipherText in number format. I did not have time to

change it form number format the actual letters for the cipherText sorry about that. The code is also to long to show here in this document but you can see it in the github that was linked above.

Rail Fence Cipher:

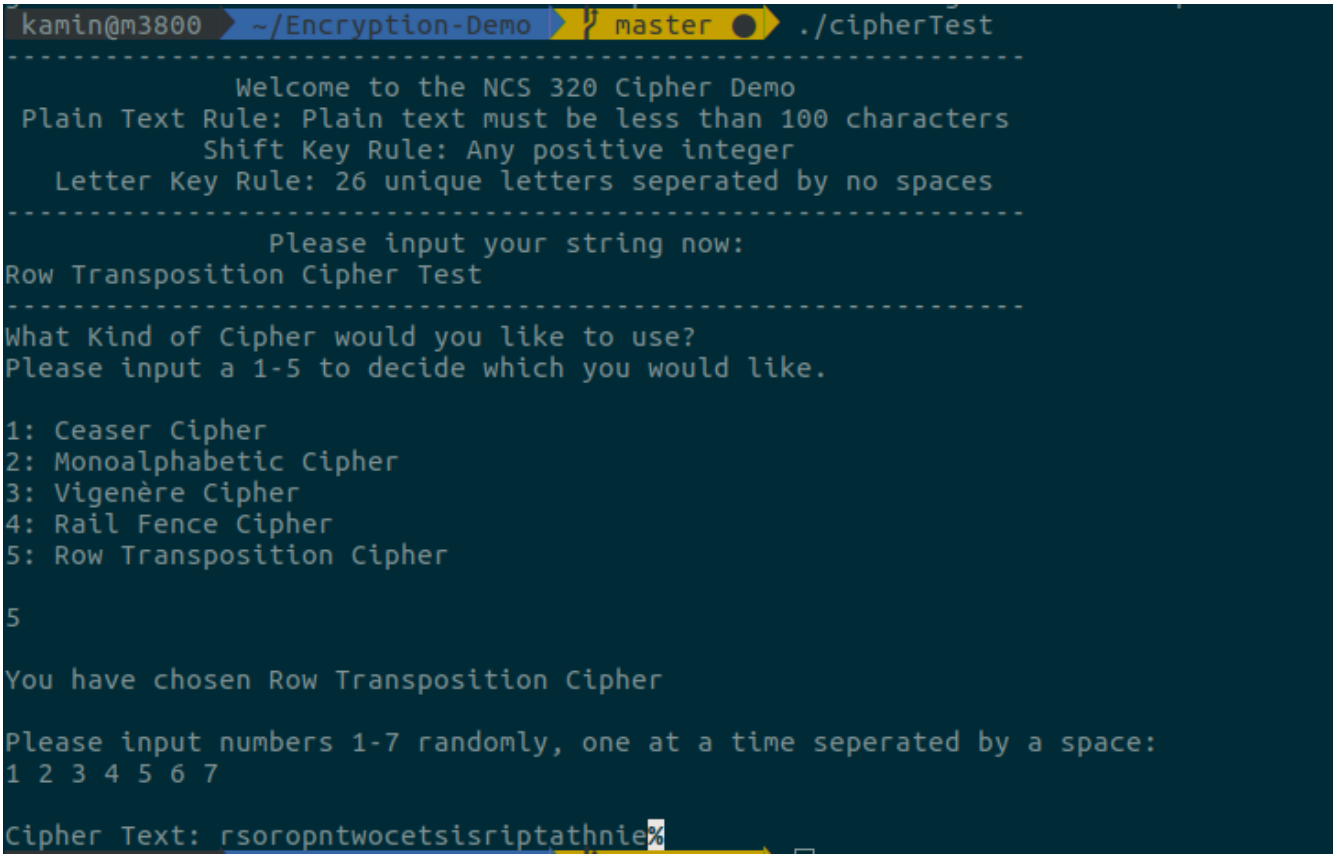


This screenshot shows the Rail Fence Cipher running, yet again it prompts the user at the beginning for the string that they would like to encrypt. After then being prompted and selecting 4 for Rail Fence Cipher all of the code runs in the background. It splits up the string into two ‘rails’ the top rail and the bottom rail. Every other letter goes to top and the other goes to bottom, from there those to ‘rails’ are place do together end to end to create the cipherText. You can see this in the screenshot below.

```
/*
* Deals with the encryption, gets sent the plaintext after it
* was parsed in the parseInput function in main.cpp
*/
void RailFence::Encrypt(std::string &plainText)
{
  b = 0;
  t = 0;

  for(int i = 0; i < plainText.length(); i ++)
  {
      if(i % 2 == false)
      {
        topRailText.insert(t, 1, plainText[i]);
        t++;
      }else
      {
        bottomRailText.insert(b, 1, plainText[i]);
        b++;
      }
  }

  cipherText = topRailText + bottomRailText;
}
```

Seen above is the code that deals with the Rail Fence Cipher. It takes in the plainText from the main and then performs the required operations to create the cipherText. The if(i % 2 == false) section deals with determining which rail to put each character on. If its odd it goes on the top rail, even numbers go on the bottom rail. They are then conjoined to create the cipherText. That is sent to the displayCipher function and it is displayed to the screen.

Row Transposition Cipher:



Above is the screen shot from the Row Transposition Cipher. Yet again we take the input and selection from the main section. It then has you input the numbers 1-7 as this determins the which columns are taken and when. The way this cipher works is the string is placed into a x by x matrix, in this case a 7 by 7 matrix. The user then defines the ‘key’ which is the numbers 1-7. These determine which columns are taken and the order in which they are taken. Those columns are read top to bottom from 1 to 7 and then added together to create the cipherText. Again the code for this one is just a tad bit to long so If you would like to see it just check the github at the top.