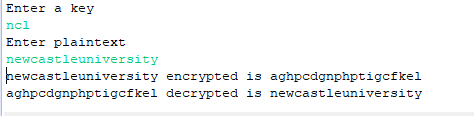
**CSC 3621 Coursework 1**

**Exercise 2 Report**

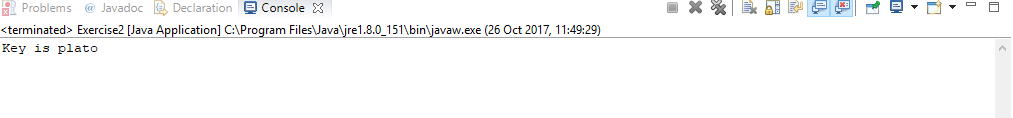
This exercise was to test understanding of the Vigenere cypher. Initially we were just required to implement an ‘encrypt’ and ‘decrypt’ method that would pass in a string and a key and return the string modified accordingly. Given the same key, performing encrypt and then decrypt would return the original text. To these methods, we had to encrypt the same large text file from exercise 1. The key I used was ‘key’. Upon encrypting, the ciphertext produced the following letter frequencies.

Comparing these results to the letter frequencies of the cyphertext from exercise 1, it can be seen that these results are a lot more even. Whilst this may be due to the much larger input file, the overall trend is uniform which means that there is less of a chance of two letters being confused and a more accurate result. In the swap cypher, the biggest value is nearly .14 and the smallest value is 0. This large discrepancy between the most and least frequent letters, and indeed the similarly large gap between the first and second most frequently occurring, mean that errors are very likely to happen. In exercise 1 errors did occur, resulting in the ten manual swaps that needed to be performed before the correct text was displayed. Meanwhile, each letter in the vigenere cypher has its own distinct frequency. Furthermore, all the frequencies are all within 0.005 of each other and there are no outliers or zero values- the trend is uniform. Therefore, using a vigenere cypher will produce a more accurate cypher than a swap cypher, as will be shown in later stages of this exercise.

Next, we were told to encrypt the string ‘newcastle university’ with the key ‘ncl’, to show the encryption and decryption methods were working correctly. The output to the console is shown below. Both the key and plaintext is user inputted, however this functionality has been removed from the final code since it’s more likely that the plaintext will be long and as such will be read from a file.



After this, we had to write a series of methods to decrypt encrypted text when a key was not provided. There are several methods of doing this, however the one we were told to use was calculating the IC (index of coincidence), the probability that two random elements of a string are identical. Once the IC is calculated, this can be used to find the length of the key and then eventually the key itself. To test this functionality, we were provided with a ciphertext file and told to recover both the key and the plaintext. Incidentally, the ciphertext (and the subsequent plaintext, due to its lack of spaces) are unable to be viewed directly in the console due to eclipse’s character buffer. This can easily be corrected by copying the text into another program such as notepad or word. In the case of the given cyphertext, running it through the Find Key method revealed that the key was ‘plato’.



Appropriately, the plaintext is an extract from Plato. Unlike exercise 1, there is no manual tuning required- aside from adding spaces and punctuation, all the characters are correctly placed. This is due to the uniqueness of each individual letter as discussed above.



If I were to repeat this exercise, I would make several changes. Firstly, I would correct the off by one error present in ‘keyLengthEstimate’ method. At present, the method simply adds one to the value that the equation returns. Whilst this makes sure that the other methods work as intended, if I had more time to work on this project I would correct the equation so the method was correct. Further, I would modify the letterFreq method to utilise the hashmap generated by the sortedCharCount method in exercise 1, to save on computational power. However they would need to be unsorted first to be provided in alphabetical order, so I deemed it simpler to write a new method. Also, I would make the encrypt and decrypt methods print to a file instead of the command line to correct the character limit issue mentioned earlier. Also, I would allow other sources to be encrypted and decrypted other than text files, such as web pages. If the program was to be made more complicated, spaces and punctuation could be considered so the cyphertext would be more readable.