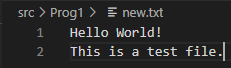
**C4 – Pointers, Files and Strings**

3.1 Reading Files

Test text file



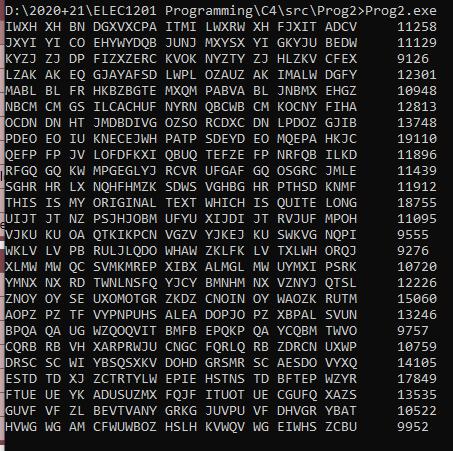
|  |  |
| --- | --- |
|  |  |
|  |  |

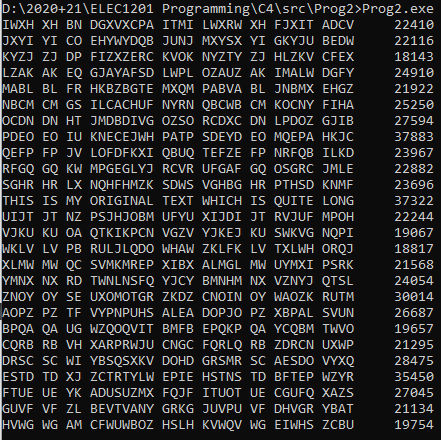
3.2 Manipulation Strings

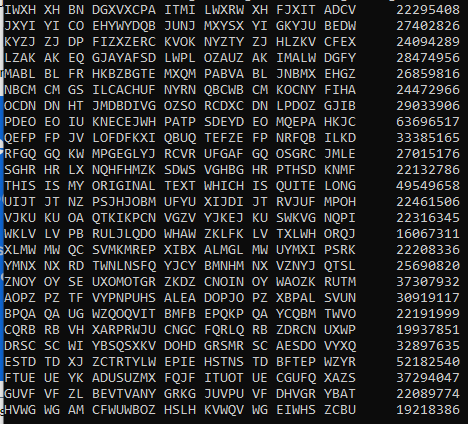
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

3.3 Code Breaking

|  |
| --- |
|  |
|  |
|  |

I have now used a text file which has a lot of text in (source <http://randomtextgenerator.com/>) which then generated its own histogram for each letter. Then I take the text to be deciphered and sum the product of each of its histogram against the source histogram (e.g. a \* a, …). Then it repeats, changing the increment each time. In theory the letters which are most common in both will yield a higher total “strength factor” and are therefore a possible greater match. In practice, the original correct text is usually one of the highest ones but not always the highest as seen below.

After I had increased the size of the source text (doubled in size using text from the same website as before) this created a greater differences of bad matches and good matches but my original sentence still didn’t come out as the highest but by a very small margin of 37322 compared to the highest which was 37883.

Next I created a new function which can square the histogram of the source text so that there is a greater difference between the popular and unpopular characters. Unfortunately this made a larger difference between the false positive result and the official result as show below.

Therefore in conclusion to make the results more accurate the text file should be as long as possible to create the greatest accuracy of what English text best consists of. The program should also output the best 3 or 4 matches since the “best match” is usually not the actual answer.