Project 2: Vigenere Decrypt

General Notes:

I have attached screenshots below of my program running with the tests provided. My program will automatically take in the values in the dictionary file and parse them by length. Additionally, commands will be taken in from the "commanddata.txt" file, so if you want to change the input, then that's where the change must occur.

Originally I designed my program as discussed in the instructions. I generated all possible keys, used them to decode the ciphertext, and checked the first word against the dictionary. Unfortunately, it quickly became apparent that this would not yield practical results, beyond the first few cases. The issue is that the time complexity of the program in this manner, is determined primarily by the length of the key. Each time we increase the size of the key, we multiply the runtime by a factor of 26. While my initial program probably could have searched the dictionary file more efficiently and made the comparisons quicker, at the end of the day, efficiency only gets us so far. We haven't escaped the issue, which is the time scaling with key length. I thought, "What if we change what primarily determines the complexity?" Then we would be able to keep runtimes reasonable even in the face of a large key length.

My current codes time complexity is based primarily on the size of my dictionary file. Since we are able to fairly easily divide the dictionary file based on word length, we are able to get a functional dictionary size of around 20,000 or so. Since this value is never changing even when the key length changes, we are able to keep the times down. I'm not saying that times won't increase at all, since as key lengths increase, the number of comparisons that need to be performed will increase regardless of implementation. However, the primary determiner of time will remain the length of the dictionary file.

My program works by finding offset values between the characters in the ciphertext and the characters in each word of the dictionary for the same word length. Since we know the length of the key, we know that the offsets determined for characters whose difference in index is equal to the key length, must be equal. If they are not, then we may simply discard the value, but if they are, then we've already found the key since we have already determined the offsets. Also, by approaching the problem in this manner, we have a max number of "comparisons" equal to the size of the dictionary for the length of the first word. If we brute force by checking all key combinations then we have a maximum number of "comparisons" equal to 26^n (where n is the length of the key). This method will work well as long as the dictionary file size doesn't become unreasonable. Even if we estimate the number of words in the English language to be a little over one million, we are still able to parse the dictionary into manageable pieces. Either way this implementation should still be significantly faster than a "test all key" approach.

Compile and Run:

I wrote my program in C++ and have included a makefile. The compiled program will be called "Program". No input is needed, you simply need to run the program to view the inputs and results.

```
C:\Users\Philip\Documents\EECS_565\Project2_PhilipWood>Program.exe
Don't blink or you'll miss it!!!



<---------------------------------- Test 1 ---------------------------------->
This is the original message: MSOKKJCOSXOEEKDTOSLGFWCMCHSUSGX
First word length: 6
Key length: 2

Here are the possible values:
****************************************

Cipher: MSOKKJ
Dict: CAESAR
Key: KS
Plaintext: CAESARSWIFEMUSTBEABOVESUSPICION

Decode runtime: 0.012011 seconds
****************************************
<---------------------------------- End Test 1 ---------------------------------->



<---------------------------------- Test 2 ---------------------------------->
This is the original message: OOPCULNWFRCFQAQJGPNARMEYUODYOUNRGWORQEPVARCEPBBSCEQYEARAJUYGWWYACYWBPRNEJBMDTEAEYCCFJNENSGWAQRTSJTGXNRQRMDGFEEPHSJRGFCFMACCB
First word length: 7
Key length: 3

Here are the possible values:
****************************************

Cipher: OOPCULN
Dict: FORTUNE
Key: JAY
Plaintext: FORTUNEWHICHHASAGREATDEALOFPOWERINOTHERMATTERSBUTESPECIALLYINWARCANBRINGABOUTGREATCHANGESINASITUATIONTHROUGHVERYSLIGHTFORCES

Decode runtime: 0.009009 seconds
****************************************
<---------------------------------- End Test 2 ---------------------------------->



<---------------------------------- Test 3 ---------------------------------->
This is the original message: MTZHZEOQKASVBDOWMWMKMNYIIHVWPEXJA
First word length: 10
Key length: 4

Here are the possible values:
****************************************

Cipher: MTZHZEOQKA
Dict: EXPERIENCE
Key: IWKD
Plaintext: EXPERIENCEISTHETEACHEROFALLTHINGS

Decode runtime: 0.006005 seconds
****************************************
<---------------------------------- End Test 3 ---------------------------------->
```

```
<---------------------------------- Test 4 ---------------------------------->
This is the original message: HUETNMIXVTMQWZTQMMZUNZXNSSBLNSJVSJQDLKR
First word length: 11
Key length: 5

Here are the possible values:
*****************************************

Cipher: HUETNMIXVTM
Dict: IMAGINATION
Key: ZIENF
Plaintext: IMAGINATIONISMOREIMPORTANTTHANKNOWLEDGE

Decode runtime: 0.017830 seconds
*****************************************
<---------------------------------- End Test 4 ---------------------------------->



<---------------------------------- Test 5 ---------------------------------->
This is the original message: LDWMEKPOPSWNOAVBIDHIPCEWAETYRVOAUPSINOVDIEDHCDSELHCCPVHRPOHZUSERSFS
First word length: 9
Key length: 6

Here are the possible values:
*****************************************

Cipher: LDWMEKPOP
Dict: EDUCATION
Key: HACKER
Plaintext: EDUCATIONISWHATREMAINSAFTERONEHASFORGOTTENWHATONEHASLEARNEDINSCHOOL

Decode runtime: 0.026084 seconds
*****************************************
<---------------------------------- End Test 5 ---------------------------------->



<---------------------------------- Test 6 ---------------------------------->
This is the original message: VVVLZWWPBWHZDKBTXLDCGOTGTGRWAQWZSDHEMXLBELUMO
First word length: 13
Key length: 7

Here are the possible values:
*****************************************

Cipher: VVVLZWWPBWHZD
Dict: INTELLECTUALS
Key: NICHOLS
Plaintext: INTELLECTUALSSOLVEPROBLEMSGENIUSESPREVENTTHEM

Decode runtime: 0.020797 seconds
*****************************************
<---------------------------------- End Test 6 ---------------------------------->

Author's Notes:  That was quick!  Simply take a different approach.
If the issue you have is with the time complexity due to your key length,
then maybe try changing the variable determining time complexity. I altered
my code in a way that made time complexity primarily based on the size of our
dictionary file which remains constant.  Furthermore, I am able to parse
the dictionary file by length which drastically reduces the applicable size
of the dictionary.  Worth!
```