CS 118 — Programming Fundamentals
Assignment #9

Due Date: Wednesday, December 11th at 11:55pm on Google Classroom.

Instructions: Assignments are to be done individually. **No late assignments will be accepted.** You must complete this assignment by yourself. You cannot work with anyone else in the class or with someone outside of the class. The code your write must be your own. You are encouraged to get help from the instructional staff. You may post general questions on Piazza. Do not post more than one line of code when using Piazza.

You must **submit a single zip file** containing your code and documentation on Google Classroom named ⟨*your_student_id*⟩*.zip* where ⟨*your_student_id*⟩ is something like *i19-XXXX*. This means that you must submit only **one file named *i19-XXXX.zip* containing only your source files**. Each file that you submit **must contain your name, student-id, and assignment#** on top of the file in comments. You submission must NOT contain multiple main() functions, otherwise it will not compile for grading. Test your program on a lab machine before submission.

Follow the instructions. Assignments not following the instructions will be awarded zero points.

Assignment Statement:

In the book *'The Dancing Men'* Sherlock Holmes cracks the code of a criminal mastermind writing letters using a sequence of stick figures to encrypt sentences. This encryption method is known as *substitution cipher* and the technique Holmes used to crack the code is known as *frequency analysis*.

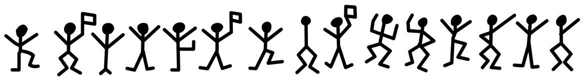

Figure 1: An example of encrypted message from The Dancing Men.

A more sophisticated version of substitution cipher was used by the Enigma machine during World War II. In this assignment you are expected to apply frequency analysis to a set of encrypted files and determine the cipher of the encrypted text. Your solution will be marked on its ability to determine the cipher correctly. Your program will take an input file as input and produce a cipher as output.

Background Information: A *substitution cipher* is a simple way of encrypting or encoding text to try and keep unwanted people from knowing the contents of a message. The key or cipher consists of a key as follows:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z decoded or plaintext letter
J I X O W H V Z M Q U D T B C Y P N A S E L G F K R encoded letter
```

To encrypt a message we simply match the letter we are trying to encrypt with the top row and write down the letter directly below it from the bottom row. For example, if we wanted to encrypt the sentence "`PF IS FUN`" we would see that `C` encrypts to `X`, `S` encrypts to `A` and so forth. We would get the message "`YH MA HER`". In order for the substitution cipher to work the sender and the receiver need to agree on the key beforehand. Since you know the key used to encrypt the message, what does "`WFJT MA VCMBV SC IW SCEVZ!`" decipher to?

At first the substitution cipher appears pretty hard to crack. Even if we only encrypt upper case letters there are $26! = (403, 291, 461, 126, 605, 635, 584, 000, 000)$ possible keys. It seems like an intractable task to try every possible key. However, code breakers have the English language (or whatever language the message is in) on their side. In English, as with other languages, certain characters are used much more frequently than others. A code breaker can use this fact and do frequency analysis on a message. Frequency analysis is performed by taking an encrypted message and counting up the occurrence of each letter or character. The longer the message the better, so this task is a good candidate for automation with a computer program.

Assignment Description: For this assignment you will write a program that does a frequency analysis on a file set of files. A key is created based on that frequency analysis. The file is then decrypted using that key.

For determining the fequency of printable ASCII letters, you can use as many documents as you like from the web (for e.g. from Wikipedia etc.). You do not need to display the frequency of all the printable ASCII characters. You only need to handle printable ASCII characters (with ASCII codes 10, 32 - 126). Where 10 represnts Line Feed (newline character). You can (Ignore ASCII chars 0 - 31 (except 10) and 127). See the Wikipedia article for more information on the printable ASCII characters. Thus your cipher will contain exactly 96 characters (as can be seen in cipher1.txt and cipher2.txt).

You are given the following files:

- `substitution`: this is an executable for encryption and decryption
- `plain1.txt`: A sample plain text file
- `encrypted1.txt`: Encrypted version of `plain1.txt`
- `cipher1.txt`: The cipher used to encrypt `plain1.txt` to `encrypted1.txt`
- `plain2.txt`: Another sample plain text file
- `encrypted2.txt`: Encrypted version of `plain2.txt`
- `cipher2.txt`: The cipher used to encrypt `plain2.txt` to `encrypted2.txt`
- `encrypted3.txt`: Encrypted version of an unknown file - try to figure out its cipher!

To run the program `substitution` you should download the given files and change the permissions of the executable using:

```
chmod 744 substitution
```

thereafter you can test the file to encrypt plaintext using:

```
./substitution -e cipher1.txt plain1.txt encrypted1.txt
```

The above command encrypts `plain1.txt` using the cipher given in `cipher1.txt` to generate the file `encrypted1.txt`. The ciper file must contain exactly 96 characters with ASCII index 10 and 32-126. Only the same cipher can be used to decrypt `encrypted1.txt` using the command:

```
./substitution -d cipher1.txt encrypted1.txt decrypted1.txt
```

Recall, the cipher is used to decrypt the message. The returned decrypted file relies on mapping given in the cipher. Thus the index of the character in the cipher indicates the ASCII code of the character in the encrypted text and the actual element is character than the encrypted character is changed into.

Here is an example. Assume we obtain the array of chars from the cipher file. Assume this is a portion of the array. (I don't show all of it.)

index	65	66	67	68	69	70	71	72	73	74
element	@	!	=	q	w	K	H	.	s	F

Again, that is only a portion of the array. Index 65 maps to ASCII character 65. ASCII code 65 is `'A'` mapping to `'!'`. Thus an `'@'` in the encrypted message will decrypt to a `'A'` based on the current key. ASCII code 66 is `'B'`. Thus a `'!'` in the encrypted message will decrypt to a `'B'` based on the current key. And so forth.
So if we had this encrypted message: `!w@F@=CB`
and used the key shown above to decrypt it we would get: `BEAJACK`

`'!'` is the first character in the encrypted message. `'!'` is located at the index 66 in the cipher. And 66 is the ASCII code for `'B'`. So `'!'` becomes `'B'`. The encrypted character in the message is `'F'`. `'F'` is located at the index has an 74 so `'F'` becomes `'J'` in the decrypted message. And so forth. Thus, the cipher array returned is used to transform the encrypted message to a decrypted message.

In your solution, you can use a lot of online text to calculate frequencies of printable characters and match these with the frequencies of letters appearing in the encrypted text. The trouble with this approach is that with short messages there will be differences between the expected frequency of letters and the actual frequencies. The decrypted text probably won't be perfect unless it is several thousand characters long. Even then there could be mistakes due to differences between the standard frequencies of characters and the actual frequency of characters in the original text. Therefore, your solution will be marked on its ability to determine the cipher as best as possible even if it is not 100% accurate. Your program will take an encrypted file as input (as a command line parameter) and produce a cipher on the standard output. For example, if your program is called `holmes` then the following is a sample run:

```
./holmes encrypted2.txt
] ~*;dfTy_<{8oXv'U(xW>VI[!L^l:Z|hDMH=j#`C\Ea6K,%N2Fus"b/74RzQO+$}5)Y&?@1Jpm
iS.tGw9Anc-r3e0kgqBP
```

Things to remember:
We are mapping the ASCII values of chars to indices in the array. You can convert from a char to an int without casting.

`int x = ch;` // if ch is a char, x now holds the ASCII code for that char

Converting from an int to a char requires casting

`char ch = (char) x;` // if x is an int, ch now holds the ASCII character associated with the code x

Break the problem up into methods. Test methods before going on. This will require writing some testing code that you delete before turning in your program. Simple printf statements are VERY useful when testing and debugging. As always you must use good program hygiene. (Constants, meaningful variable names, well structured code, removal of redundancy, correct spacing and tabbing and alignment of braces, and so forth as spelled out in the assignment page.)

Approach: Divide the program into parts. Complete and test each part before moving on. Use the methods from string and ctype (such as `isprint()`)libraries to help make your job easier.

Divide the program up into methods to provide a structured solution. Some of the methods will be void and others will return values. You will have to make use of parameters, for loops, while loops, and if statements. Do not write all your code in main().

Honor Policy
This assignment is a individual learning opportunity that will be evaluated based on your ability to think independently, work through a problem in a logical manner solve the problems on your own. You may however discuss verbally or via email the general nature of the conceptual problem to be solved with your classmates or the course instructor, but you are to complete the actual assignment without resorting to help from any other person or other resources that are not authorized as part of this course. If in doubt, ask the course instructor. You may not use the Internet to search for solutions to the problem.