Ave Caesar! Morituri te salutamus!!

(Assignment adapted from Harvard's CS50)

1. Prior Reading and Orientation

Suggested reading: Caesar Cipher
Take your time, read, learn and get it right primarily through your own effort!

This assignment requires you to use command line arguments. It is good that you learn about command line arguments. If you are unfamiliar with the command line, the resources below (and other similar ones) can you help learn the basics:

NB: you can choose to either use Windows or Ubuntu. If you choose Windows, follow option a) below. If you choose Ubuntu, follow option b). If you are mainly used to Windows, I would encourage you to experiment and learn a bit of the Ubuntu environment (but it is not compulsory).

a. Command Line For Windows

- How to use the Windows command line
- A video on running C++ in the command line
- A video on how to use command line arguments

b. Command Line For Ubuntu

This is for the more adventurous students. Go through Part 2, 3 and 4 of the document kap0.pdf. Mvumilivu hula mbivu :-)

c. Video for Orientation

Read the instructions below carefully and then watch this video by Zamyla Chan. Note that we are using C++, whereas the video gives function examples based on C. Therefore, only apply the concepts illustrated in the video *mutatis mutandis* i.e. making the necessary changes. E.g. do not use `get_string()` as suggested; rather than using `strlen()` use `myString.length()`, where `myString` is a C++ string.

2. Caesar's Cipher

Caesar's cipher encrypts (i.e., scrambles in a reversible way) messages by "rotating" each letter by k positions, wrapping around from Z to A as needed:

In other words, if `p` is some `plaintext` (i.e., an unencrypted message), p_i is the i-th character in `p`, and `k` is a secret key (i.e., a non-negative integer), then each letter, c_i, in the ciphertext, `c`, is computed as:

$$c_i = (p_i + k) \% 26$$

This formula perhaps makes the cipher seem more complicated than it is, but it's really just a nice way of expressing the algorithm precisely and concisely. And engineers and computer scientists love precision and concision.

For example, suppose that the secret key, k, is 13 and that the plaintext, p, is "`Be sure to drink your Ovaltine!`" Let's encrypt that p with that k in order to get the ciphertext, c, by rotating each of the letters in p by 13 places, whereby:

```
Be sure to drink your Ovaltine!
```

becomes:

```
Or fher gb qevax lbhe Binygvar!
```

Notice how O (the first letter in the ciphertext) is 13 letters away from B (the first letter in the plaintext). Similarly is r (the second letter in the ciphertext) 13 letters away from e (the second letter in the plaintext). Meanwhile, f (the third letter in the ciphertext) is 13 letters away from s (the third letter in the plaintext), though we had to wrap around from z to a to get there. And so on.

Not the most secure cipher, to be sure, but fun to implement! Incidentally, a Caesar cipher with a key of 13 is generally called ROT13.

a. Specification:

Your goal is to write, in caesar.cpp , a program that encrypts messages using Caesar's cipher.

Your program should accept a non-negative integer, `k`, supplied through a command line argument. If your program is executed without any command-line arguments or with more than one command-line argument, your program should print the following message and **do nothing else**.

```
One argument expected!
```

You can assume that, if a user does provide a command-line argument, it will be a non-negative integer (e.g., 1). No need to check that it's indeed numeric. So you don't have to worry about the user typing, say, `foo` , just to be difficult (even though you could try and have a solution that catches such).

Otherwise, your program must proceed to prompt the user for a string of plaintext and then output that text with each alphabetical character "rotated" by k positions; no alphabetical characters should be output unchanged. After outputting this ciphertext, your program should terminate .

Although there exist only 26 letters in the English alphabet, you may not assume that k will be less than or equal to 26; your program should work for all non-negative integral values of k.

Now, even if k is greater than 26, alphabetical characters in your program's input should remain alphabetical characters in your program's output. For instance, if k is 27, A should not become [even though [is 27 positions away from A in ASCII; A should become B , since 27 modulo 26 is 1, as an engineer or computer scientist might say. In other words, values like `k = 1` and `k = 27` are effectively equivalent.

Okay, so once you've got k stored as an int, you'll need to prompt the user for some plaintext. Odds are, `getline` can help you with that.

Once you have both k and some plaintext, it's time to encrypt the latter with the former. Your program must output the corresponding ciphertext, with each alphabetical character in the plaintext "rotated" by k positions; non-alphabetical characters should be output unchanged.

Your program must be case-sensitive: capitalized letters, though rotated, must remain capitalized letters; lowercase letters, though rotated, must remain lowercase letters.

After outputting ciphertext, you should print a newline. Your program should then exit by returning 0 from main.

b. Hints:

Note that that you can iterate over the characters in a string, p, printing each one at a time, with code like the below:

```
for (int i = 0, n = p.length(); i < n; i++)
{
   cout << p[i];
}
```

Note also that a string is an array of char s. And so you can use square brackets to access individual characters in string s. Of course, printing each of the characters in a string one at a time isn't exactly cryptography. Well, maybe technically if k is 0.

Other pointers: the function `isdigit` sounds interesting and you may or may not use it. And, with regard to wrapping around from Z to A (or z to a), don't forget about `%` , C++'s modulo operator. You might also want to check out an ascii table, which reveals the ASCII codes for more than just alphabetical characters, just in case you find yourself printing some characters accidentally.

3. Starter code

Here is some code to get you started. You can copy paste it into your favourite IDE and code away… NB: the five TODOs in the code below are to guide you on what you need to do.

```cpp
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;

// function prototype
string caesar(int k, string plaintext);

int main(int argc, char* argv[])
{
    // TODO 1: declare an int to hold k as well as a string to hold the plaintext

    // TODO 2: ensure that the right number of command line arguments are entered
    // If the wrong number of arguments are entered exit by returning 3.
    // If the right number are entered assign k the appropriate value

    // TODO 3: read in the plaintext using getline

    // TODO 4: run the function caesar with appropriate arguments and assign the value returned to a string called ciphertext

    // Print out the ciphertext
    cout << ciphertext;
}

// TODO 5: implement the caesar function
// Remember to take care of a case in which k is greater than 26
string caesar(int k, string plaintext){

}
```

4. Sample input and output:

The following is a sample output of what your program should do, assuming that you are running from an Ubuntu environment.

```
$ ./caesar
One argument expected!

$ ./caesar 1 2 3 4 5
One argument expected!

$ ./caesar 1
HELLO
IFMMP

$ ./caesar 13
hello, world
uryyb, jbeyq

$ ./caesar 13
be sure to drink your Ovaltine
or fher gb qevax lbhe Binygvar

$ ./caesar 26
be sure to drink your Ovaltine
be sure to drink your Ovaltine
```

If you are running from a Windows environment, the flow would be as follows (assuming the compiled file is called caesar.exe)

```
> caesar.exe
One argument expected!

> caesar.exe 1 2 3 4 5
One argument expected!

> caesar.exe 1
HELLO
IFMMP

> caesar.exe 13
hello, world
uryyb, jbeyq

> caesar.exe 13
be sure to drink your Ovaltine
or fher gb qevax lbhe Binygvar

> caesar.exe 26
be sure to drink your Ovaltine
be sure to drink your Ovaltine
```