Objective:

- Use of class level information (data/operations).

Task-1: discussed in class/lecture

```
class CMath
{
public:
    static float calcPower(int base, int exponent);
    static int calcGCD(int numerator, int denominator);
    static CString toCString(long long int num);
    static long long int toInteger(CString);
    //you may add other mathematical functions in the same way
};
```

Task-2

There is still a possibility of creating more than one objects of the following class (discussed in lecture as well). Hunt that flaw but if you get exhausted then do discuss with me.

```
class Singleton
{
private:
    Singleton()
    {};
    ~Singleton()
    {}
    static Singleton * ptr;
public:
    static Singleton * CreateObject()
    {
        if (!ptr)
            ptr = new Singleton;
        return ptr;
    }
    static void freeObject()
    {
        if (ptr)
        {
            delete ptr;
            ptr=0;
        }
    }
};
Singleton * Singleton::ptr=0;
```

Task-3:

Cryptography: the science of secret massages and their applications. This field studies ways of performing **encryption**, which takes a message, called the **plaintext**, and converts it into a scrambled message, called the **cipher text**. Likewise, cryptography also studies corresponding ways of performing d**ecryption**, which takes a cipher text and turns it back into its original plaintext.

Arguably the earliest encryption scheme is **Ceaser Cipher**, which is named after Julius Ceaser, who used this scheme to protect important military messages. (All of Ceaser's messages were written in Latin, of course, which already makes them unreadable for most of us!). The Ceaser Cipher is a simple way of obscure a message written in a language that forms words with an alphabet.

The Ceaser Cipher involves replacing each letter in a message with the letter that is three letters after it in the alphabet for the language. So, in English message, we would replace each A with D, each B with E, and so on. We continue this approach all the way up to W, which is replaced by Z. Then, we let the substitution pattern wrap around, so that we replace X with A, Y with B and Z with C

We would like to generalize this scheme by not fixing it to shift to 3 letter only rather making it n-shift cipher.

For example, a three shift looks like:

Plaintext:

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

Cipher Text

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

In this example, the message:

WHAT KIND OF CAKE SHOULD WE HAVE? ALICE.

From Alice to Bob would look like:

TEXQ HFKA LC ZXHB PELRIA TB EXSB? XIFZB.

Suppose that Bob responds to Alice's message with:

GLB OBXIIV IFHBP ZELZLIXQB ZXHB. YLY.

How do we go about decrypting this message? It's easy, we just reverse the process and swap each letter of encrypted message with the letter 3 positions to its left. Thus mapping roles of the alphabets above are reversed:

Plaintext:

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

Cipher Text

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

Now we can decrypt the message:

JOE REALLY LIKES CHOCOLATE CAKE. BOB.

We see that it is a bit tedious to encrypt and decrypt the message. One way to help ease this process is to think of each letter as a number, with A corresponding to 1, B to 2, and so on up to Z corresponding to 26. Then, we can represent a shift of ***n*** to the right as simply adding ***n*** to each number. For example, suppose Carol wants to send the following message to Dave, with a shift of 7:

PARTY STARTS AT NINE PM. CAROL.

First, she writes out the message in number form:

16-1-18-20-25 19-20-1-18-20-19 1-20 14-9-14-5 16-13. 3-1-18-15-12.

Now all she does is adds 7 to each of the numbers:

23-8-25-27-32 26-27-8-25-27-26 8-27 21-16-21-12 23-20. 10-8-25-22-19.

Does anyone see a problem with the encrypted message above? If you spotted that we don't have a letters corresponding to numbers bigger than 26, good for you. There is an easy fix to this problem, however. Remember originally that we 'wrapped around' once we got to Z. The number equivalent to wrapping around is subtracting 26 if the number is too big. Thus, Carol's encrypted message becomes:

23-8-25-1-5 26-1-8-25-1-26 8-1 21-16-21-12 23-20. 10-8-25-22-19.

To decrypt this, Dave simply subtracts 7 from each of these numbers and adds 26 to anything he gets that is negative. As you will try, this is much faster than our first method, which used only letters. Unfortunately, the messages we have looked at so far are particularly vulnerable to attack. In particular, including punctuation marks gives clues to an attacker about the structure of the sentence. Also due to spaces, attackers can know the length of words. Since there aren't very many one and two letter words in the English language, this could be a big help. We start by renumbering the alphabet with numbers and also assigning numbers to other common symbols and punctuation marks:

Space ↔ 0		
A ↔ 1	0 ↔ 27	. ↔ 37
B ↔ 2	1 ↔ 28	! ↔ 38
…	…	? ↔ 39
…	…	, ↔ 40
Y ↔ 25	8 ↔ 35	; ↔ 41
Z ↔ 26	9 ↔ 36	

Suppose Alice wants to send Bob the following message with a shift of 4:

I have 150 balloons! Alice.
Converting this into numbers we get:

09-00-08-01-22-05-00-28-32-27-00-02-01-12-12-15-15-14-19-38-00-01-12-09-03- 05-37

Next we add 4 to each number:

13-04-12-05-26-09-04-32-36-31-04-06-05-16-16-19-19-18-23-42-04-05-16-13-07- 09-41

In our first attempt using numbers for letters, numbers larger than 26 had no meaning so we had to adjust those. Under our new scheme, however, the numbers from 1 to 41 all correspond to some symbol. It is only the numbers larger than 42 that we need to worry about now. We handle them as before: we subtract 42. Looking back, we need only replace the 42 which corresponds to the ! shifted by 4. It is shown in italic/bold below.

13-04-12-05-26-09-04-32-36-31-04-06-05-16-16-19-19-18-23-***00***-04-05-16-13-07- 09-41

Our encrypted message is thus:

mdlezid594dfeppssrw depmgi;

Implementation Guidelines:

```
//ShiftCipher.h file
	class ShiftCipher
	{
	    static const char mapArray[42];
	public:
	    static CString cipher(CString plainText, int);
	    static CString decipher(CString cipherText, int);
	    //it returns cipher text in CString object
	};
//ShiftCipher.cpp file
	const char ShiftCipher::mapArray[42] = {' ','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','0','1','2',
	'3','4','5','6','7','8','9','.','!','?',',',';'};
	CString ShiftCipher::cipher(CString plainText, int shiftAmount)
	{
	    CString cipherText;

	    //need to do ABRACADABARA here

	    return cipherText;
	}
	CString ShiftCipher::decipher(CString cipherText, int shiftAmount)
	{
	    CString plainText;

	    //need to do ABRACADABARA here

	    return plainText;
	}
//driver.cpp
	int main()
	{
	    CString secret = ShiftCipher::cipher("Hello Please Cipher Me",2);
	    secret.display();
	    return 1;
	}
```