Week Seven

Max Olivier

March 7, 2009

Readings
For this week please read Chapter 12 in Overland on constructors, which you already used in a limited capacity last week. In addition, please read section 5.2.1 in Stinson on the Euclidean Algorithm for finding GCF's and the extended Euclidean algorithm for finding multiplicative inverses.

Main Ideas

- Different types of class constructors and how they are declared in C++.
- Reference variables and the copy constructor.
- How to use the extended Euclidean algorithm to find multiplicative inverses given a modulus.

Exercise 1 (Exercises from the Reading)
All the exercises from the reading in Overland.

Exercise 2 (Cryptology Program)
Now that you know how to find multiplicative inverses you can write the decode function in your Mult_Cipher class. So for this week write a function called inverse in the Mult_Cipher class that that takes a number as a parameter and returns that number's multiplicative inverse for the modulus made in the constructor. For example, if the modulus declared in the constructor is 26, then inverse(5) should return 21 since

$$5 \times 21 \equiv 1 \pmod{26}.$$

I would suggest you first write a separate program to make sure your function works. Also be aware that the number passed into inverse() may not already be reduced, and that there are come special cases (for example, what is the multiplicative inverse of 1? 0?).

Once you have written this function, use it to write the decode function in the Mult_Cipher class. Also, use your new inverse function as well as a GCF function to ensure that all encoding and decodings use a key that is coprime to the modulus. So if the user tries to enocde a message using a multiplicative cipher but enters a key that is not coprime to the moulus, including 0, it should politely tell them that it is not valid and ask for another value. Make sure that the encode_random() function only encodes with a proper key as well.

Hint 1: Tweak the GCF function to return 0 if one of the entries into it is 0. Given how the GCF function should factor into your encode(), encode_rand(), and decode() functions, this should fix any problems you may have with 0 keys.

Hint 2: Since the user can enter a key that is not reduced for the given modulus (i.e. 75 even if the modulus is 12) make sure that you do not use the GCF function on the initial key the user enters, but rather what the key is modulo the modulus.

Finally, use your now functions to guide you in writing an Affine_Cipher class similar its Add and Mult counterparts. As with the Mult_Cipher class make sure that all the multkey's are proper. There are of course solutions available on the course website.