Programming Assignment 1
Joker's Wild – WSS Encryption

Design due: Monday January 30, 2012 11pm
Studio lab: Tuesday January 31, 2012
Implementation due: Saturday February 4, 2012 11pm

Lab Objectives

- Creating effective abstractions to model a concrete problem.
- See that a solution to a larger problem can be developed in terms of smaller solutions to smaller problems.
- See that a solution to a complex problem can be made more manageable by using an appropriate collection.
- Understand and apply a selection algorithm.
- Continue to develop a personal process oriented around a daily build.
- Continue to adopt a test-first mindset.

Problem Description

Encryption and cryptography are branches of mathematics and computer science that lie at the heart of most economic and security systems. Since many cryptographic codes are very, very computationally expensive to break, both the good guys and the bad guys have a crucial interest in cryptographic systems. Governments recognize the power of strong cryptographic systems and many have restrictions on their use and/or distribution. In fact, the program that you will develop as part of this assignment would have been considered a munition by the United States government not that many years ago. Cool.

This assignment requires you to implement a little-known encryption algorithm developed by one of the most famous criminal minds of all time – The Joker. While much of The Joker's past is unknown, most people agree that he was a brilliant engineer before he turned to crime. It was his engineering expertise that allowed The Joker to develop the Why So Serious (WSS) Encryption Algorithm.

WSS encryption is a stream cipher much like RC4 and one of the modes of DES. Stream ciphers convert *plaintext* (the message to be encrypted) to *ciphertext* (the encrypted message) by using a *keystream* (a stream of pseudorandom characters that are combined with the plaintext characters to produce the ciphertext). The Joker's approach to generating the keystream was particularly brilliant: It can be done with a standard deck of cards. This allows The Joker to communicate privately with anyone he wishes, provided that both he and the other person have an identical deck of cards. The keystream generation algorithm has become known as Joker's Wild, not only because of its author, but also because of its use of the joker cards to cut and shuffle the deck.

Your task for this assignment is to implement WSS encryption with Joker's Wild keystream generation. You must write a Java program that can encrypt messages based on these algorithms.

WSS Encryption

The WSS algorithm is as follows:

1. Ensure that the number of characters in the plaintext is a multiple of five. (This is just tradition and doesn't really matter, but we'll do it anyway. It helps to obscure the length of the real message.) Use X's pad to the right of the plaintext as necessary.

2. Use the Joker's Wild algorithm (below) to generate one keystream value for each character in the plaintext.
3. Convert the plaintext characters into numbers (A=1, B=2, etc.)
4. Add the plaintext numbers to the corresponding keystream values, modulo 26. ("Modulo 26" just means that if the sum is more than 26, subtract 26 from the result. Note that this might not give exactly the same result as the % operator in Java.)
5. Convert the resulting numbers back into characters. This character stream is the ciphertext.

The decryption strategy is essentially the same process, just in reverse.

1. Split the ciphertext into five-character groups. (Since the ciphertext resulted from the above process, the number of characters will already be a multiple of five.)
2. Use the Joker's Wild algorithm (below) to generate one keystream value for each character in the ciphertext.
3. Convert the ciphertext characters into numbers.
4. Subtract the keystream values from the corresponding ciphertext numbers, modulo 26. (Here "modulo 26" means that if the first number is less than or equal to the second number, add 26 to the first number before subtracting. So 1-22=? Becomes 27-22=5.)
5. Convert the resulting numbers back into characters. This character stream is the original plaintext.

Joker's Wild Keystream Generation

The Joker's Wild algorithm refers to the following "joker" terms:

Joker A – This is one of the two jokers in the card deck. We will use the black joker as the "A joker."

Joker B – This is the other of the two jokers in the card deck. We will use the red joker as the "B joker."

Top joker – The joker (either A or B) that is closest to the top of the card deck.

Bottom joker – The joker (either A or B) that is closest to the bottom of the card deck.

Here is the Joker's Wild algorithm for producing a *single* keystream character.

1. Find the A joker and move it one card down. If the joker is the bottom card in the deck, move it just below the top card. (That is, treat the deck like it's circular or a loop.)
2. Find the B joker and move it two cards down. If the joker is the bottom card of the deck, move it just below the second card. If the joker is one up from the bottom card, move it just below the top card.
3. Perform a triple cut. That is, swap the cards above the top joker with the cards below the bottom joker.
4. Perform a count cut. Look at the bottom card and convert it into a number from 1 to 53. (Use the bridge order of suits: clubs, diamonds, hearts, and spades. So, the ace of clubs is 1, the jack of hearts is 37, and both the A joker and the B joker are 53.) Count down from the top of the deck that number of cards. (Start counting with the top card.) Cut after the card that you counted down to, leaving the bottom card on the bottom. Note that a deck with a joker on the bottom will be unchanged by this step.
5. Find the keystream value card. Look at the top card and convert it to a number from 1 to 53 using the same technique as in the previous step. Count down from the top of the deck that many cards. (Start counting with the top card.) The *next* card is keystream value card, unless it's a joker. If it's a joker, you have to start over at step 1 and try again for a non-joker keystream value card.
6. Convert the keystream value card to a number from 1 to 26 (because there are 26 letters in our alphabet) using the following technique: Use the bridge order of suits (clubs, diamonds, hearts, and spades from

lowest to highest). Then, assign clubs the values 1-13, diamonds the values 14-26, hearts the values 1-13, and spades the values 14-26 also. So, the ace of clubs is 1, the eight of diamonds is 21, the jack of hearts is 11, and the ace of spades is 14. This number is the keystream value.

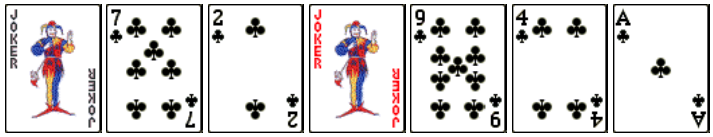
Since this only produces a single keystream value, you will need to repeat all six steps for each character in the plaintext.

Illustrating the Six Steps of Joker's Wild

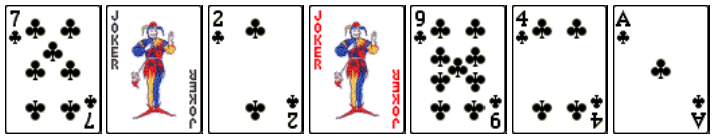
Here are the six WSS steps with examples. I'm only using a few cards so everything will fit on one line.

1. Find the A joker and move it one card down. If the joker is the bottom card in the deck, move it just below the top card. (That is, treat the deck like it's circular or a loop.)

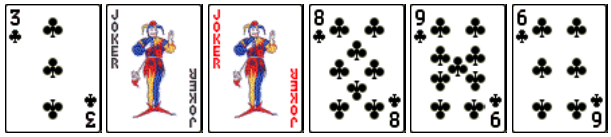
Example 1:
Before:



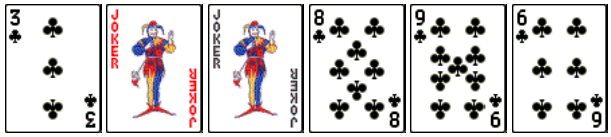
After:



Example 2:
Before:

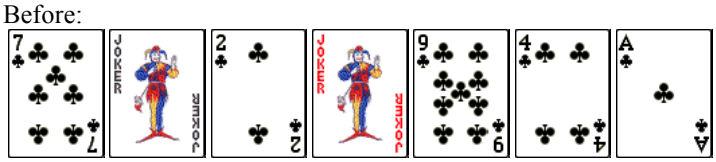


After:

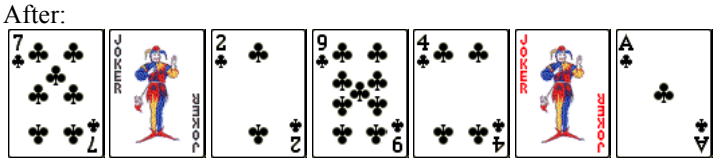


2. Find the B joker and move it two cards down. If the joker is the bottom card of the deck, move it just below the second card. If the joker is one up from the bottom card, move it just below the top card.

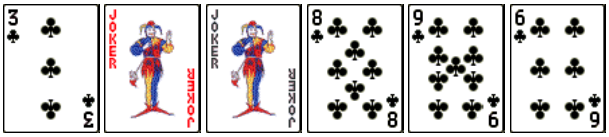
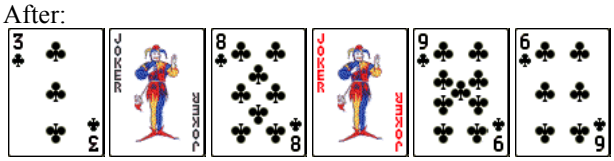
Example 1:
Before:



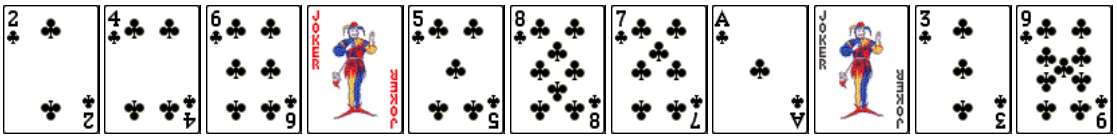
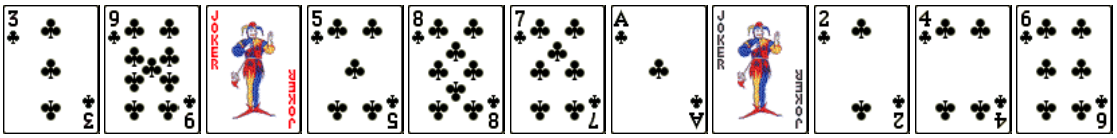
After:

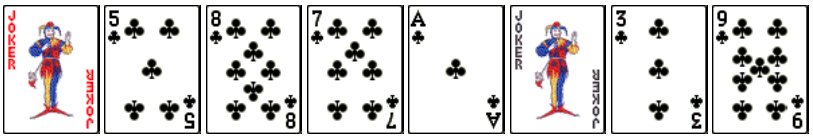
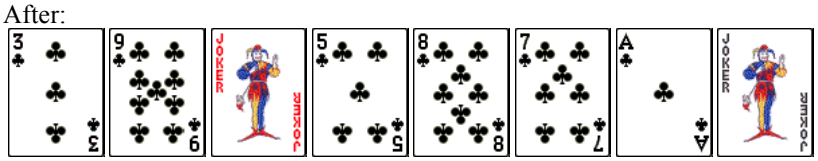


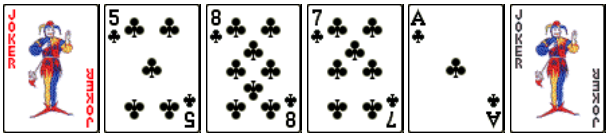
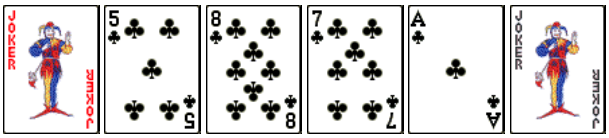
Example 2:
Before:


After:


3. Perform a triple cut. That is, swap the cards above the top joker with the cards below the bottom joker.

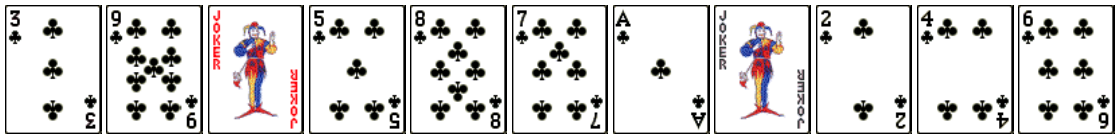
Example 1:
Before:

After:


Example 2:
Before:

After:


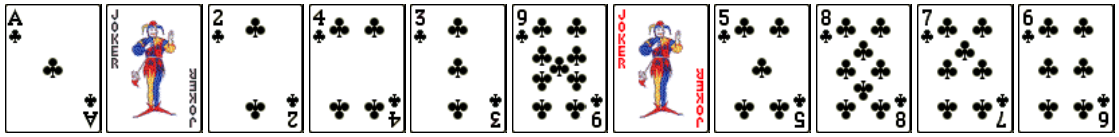
Example 3:
Before:

After:


4. Perform a count cut. Look at the bottom card and convert it into a number from 1 to 53. (Use the bridge order of suits: clubs, diamonds, hearts, and spades. So, the ace of clubs is 1, the jack of hearts is 37, and both the A joker and the B joker are 53.) Count down from the top of the deck that number of cards. (Start counting with the top card.) Cut after the card that you counted down to, leaving the bottom card on the bottom. Note that a deck with a joker on the bottom will be unchanged by this step.

Example 1:
Before:

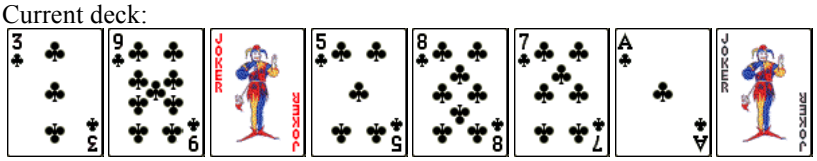


After:

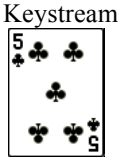


5. Find the keystream value card. Look at the top card and convert it to a number from 1 to 53 using the same technique as in the previous step. Count down from the top of the deck that many cards. (Start counting with the top card.) The *next* card is keystream value card, unless it's a joker. If it's a joker, you have to start over at step 1 and try again for a non-joker keystream value card.

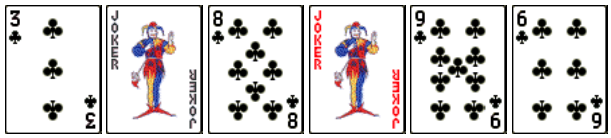
Example 1:
Current deck:



Keystream card:



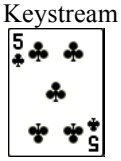
Example 2:
Current deck:



Keystream card:
None. Must begin again at step one.

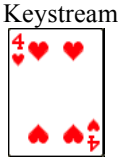
6. Convert the keystream value card to a number from 1 to 26 (because there are 26 letters in our alphabet) using the following technique: Use the bridge order of suits (clubs, diamonds, hearts, and spades from lowest to highest). Then, assign clubs the values 1-13, diamonds the values 14-26, hearts the values 1-13, and spades the values 14-26 also. So, the ace of clubs is 1, the eight of diamonds is 21, the jack of hearts is 11, and the ace of spades is 14. This number is the keystream value.

Example 1:
Keystream card:



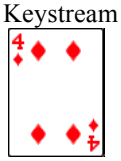
Keystream value: 5

Example 2:
Keystream card:



Keystream value: 4

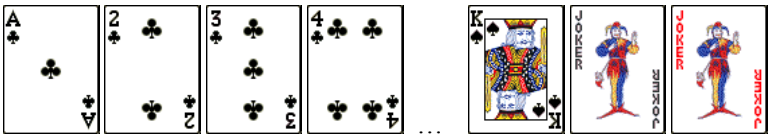
Example 3:
Keystream card:



Keystream value: 17

Generating Multiple Keystream Values

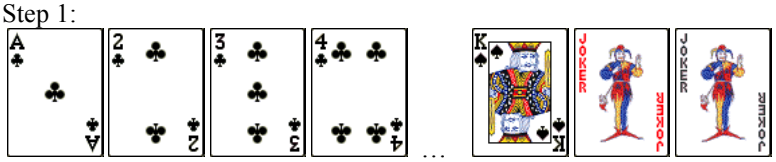
Let's say that we started with a standard deck of cards just out of the wrapper, and it looked like this:



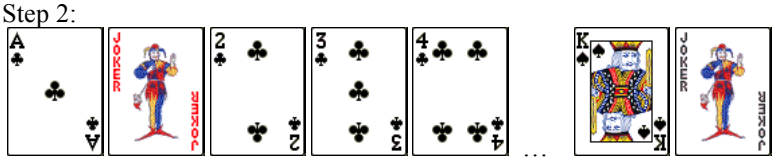
Applying Joker's Wild to generate multiple keystream values would work as follows.

Generating the first keystream value:

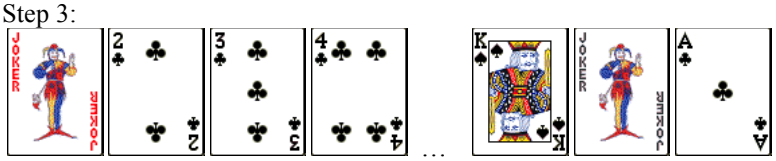
Step 1:



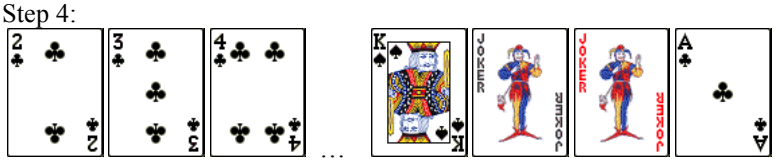
Step 2:



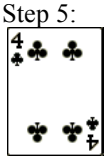
Step 3:



Step 4:



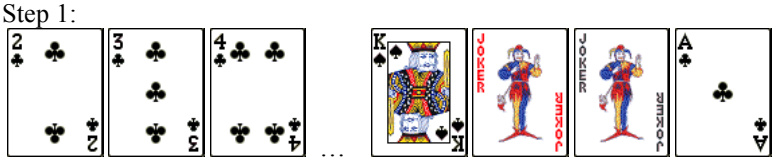
Step 5:

 (keystream card)

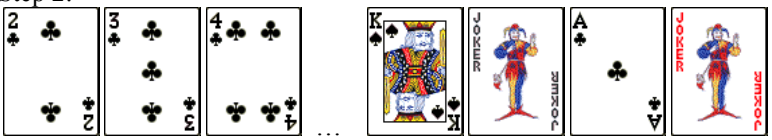
Step 6:
4 (keystream value)

Generating the second keystream value:

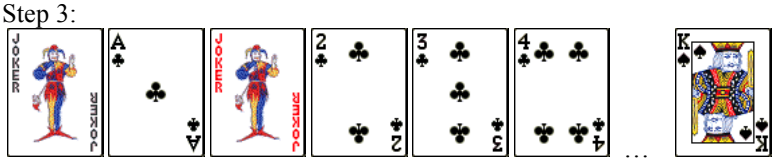
Step 1:



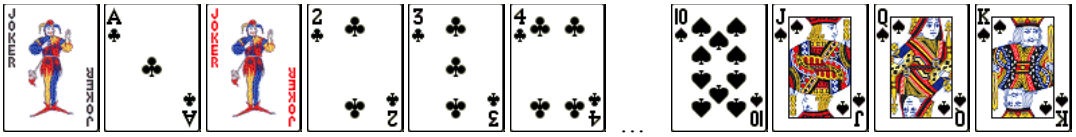
Step 2:



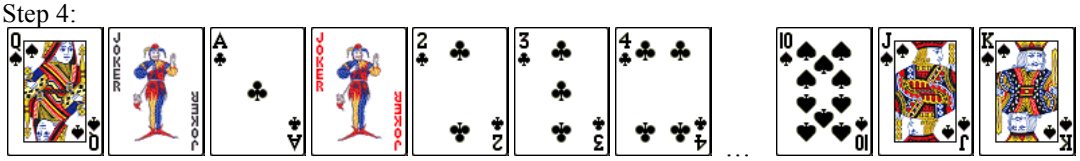
Step 3:



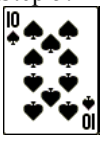
(For the next step, we need to see a few of the cards before the king of spades, so I'll expand the previous line to the following.)



Step 4:



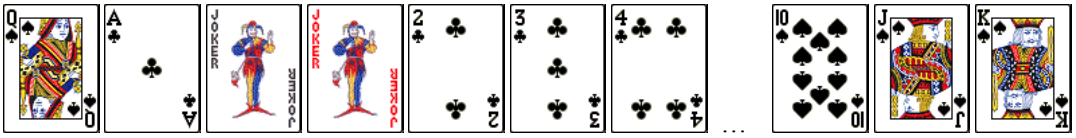
Step 5:

 (keystream card)

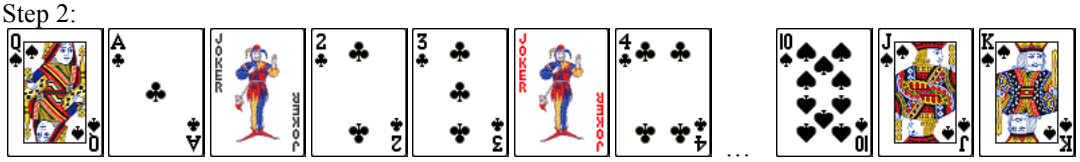
Step 6:
23 (keystream value)

Generating the third keystream value:

Step 1:



Step 2:



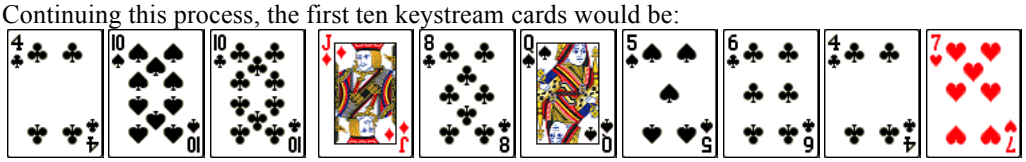
Step 3: …

Step 4: …

Step 5: …

Step 6: …

Continuing this process, the first ten keystream cards would be:



(Note that a joker is generated between the 10 of clubs and the jack of diamonds, but it can't be used as a keystream card.)

The corresponding keystream values would be:
4 23 10 24 8 25 18 6 4 7

An Encryption Example

Finally, to make sure that you understand WSS encryption you need to encrypt a short sample message. Here is a step-by-step example of WSS encryption for the plaintext "JOKER":

Phase 1: Split the plaintext into five-character groups.
JOKER (already done)

Phase 2: Use the Joker's Wild algorithm to generate one keystream value for each character in the plaintext.
(Let's use the keystream values that we just generated in the example above.)
4 23 10 24 8

Phase 3: Convert the plaintext characters into numbers (A=1, B=2, etc.)
10 15 11 5 18

Phase 4: Add the plaintext numbers to the corresponding keystream values, modulo 26.
14 12 21 3 26

Phase 5: Convert the resulting numbers back into characters. This character stream is the ciphertext.
NLUCZ

Your Assignment

You must write a program in Java that encrypts plaintext using WSS encryption with Joker's Wild keystream generation. Your solution must be in terms of a class named WSS, which has the following constructor and methods.

- A parameterless constructor WSS(). This constructor must initialize a standard deck of 54 cards in bridge order and prepare the class to begin encrypting plaintext.
- A public method named initDeck() with no parameters. This method initializes or resets the current deck being used by the WSS class to a standard deck of 54 cards in bridge order.
- A public method named encrypt() that takes a single parameter of type String and returns a String as its return value. This method takes the plaintext as its parameter and returns the ciphertext as its return value. The signature of this method must appear exactly as shown below.

```
public String encrypt(String plaintext)
```

The WSS class must be defined in a file named WSS.java. You may add any other public or private methods that you would like to the WSS class. However, your submission will be graded for correctness using only these three methods. Specifically, your solution will be graded in a manner similar to the following code.

```
public class WSSCLient
   {
      public static String[] plaintextArray = {"JOKER", "ATTACKATDAWN",
                                               "HELPME", "WAREAGLE"};

      public static void main(String[] args)
      {
         WSS wss = new WSS();

         for (int i = 0; i < plaintextArray.length; i++)
         {
            String plaintext = plaintextArray[i];
            String ciphertext = wss.encrypt(plaintext);
            System.out.println("Plaintext:  " + plaintext);
            System.out.println("Ciphertext: " + ciphertext);
            System.out.println();
         }
      }
   }
```

Note that the initial state of the deck must be bridge order, like the "JOKER" example worked out above. Thus, in the first call to encrypt(), the plaintext will be encrypted with a standard deck in bridge order. In each subsequent call to encrypt(), the plaintext must be encrypted using the current state of the deck. That is, the deck does not automatically reset to encrypt the next message. Only an explicit call to initDeck() will reset the deck to bridge order.

Sample Input

```
JOKER
ATTACKATDAWN
HELPME
WAREAGLE
```

Sample Output

For the sample input above, which corresponds to the example code, a correct implementation of the WSS class would generate the following output.

```
Plaintext:  JOKER
Ciphertext: NLUCZ
```

```
Plaintext:  ATTACKATDAWN
Ciphertext: ZLZEJENMLQRIPVH

Plaintext:  HELPME
Ciphertext: HDIOUHMTPJ

Plaintext:  WAREAGLE
Ciphertext: SSMKNFSZFC
```

Something to think about …

Although it's not required for the assignment, implementing the decryption phase isn't any harder than encryption and it gives you a built-in way of testing yourself. If you can retrieve the original plaintext by decrypting what your program encrypted, then your program should be right on target. An even better approach would be to test your solution with other students and make sure that you can encrypt/decrypt each other's test input.

Lab Turn-In

There are multiple submissions associated with this assignment. Refer to the first page of this document and Canvas for all deliverables and dates.

Acknowledgments

- Card images are from: http://www.jfitz.com/cards/. All copyrights observed.
- The Joker images copyright DC Comics and Warner Bros.
- INQSVOXTPRSXZPR