CSC 231 HOMEWORK 7
NUMBER THEORY AND CRYPTOGRAPHY PROJECT

PROFESOR GODFREY MUGANDA
DEPARTMENT OF COMPUTER SCIENCE

In this assignment, you will implement that RSA cryptosystem based on the material covered in lecture and in the class textbook. You will do this by following a provided outline of steps and implementing a collection of stubbed methods provided in a Netbeans project shell. A link to the shell is on the course website.

You will test your code on data provided in Example 12 on page 254, and Examples 8 and 9 on pages 300 and 301.

To Test your code, begin by commenting out all the code in the `main()` method in the `RSAUtilities` class. Develop the project by following the steps below. As you complete each step, uncomment the code that tests that step and run the project.

This homework is due Friday night of week 10.

1. Fast Modular Exponention

Begin by reading the section on *Modular Exponentiation* on page 253 of the text book, and studying Algorithm 5 on page 254.

Using the stub provided in the `ModularExponentiation` class in the Project shell provided, Complete the

```
long exp(long b, long n, long m)
```

to implement that algorithm to compute $b^n \mod m$.

Test your code by running the main method in this class. You can do this by right-clicking on the file node under the project and selecting `Run File` from the pop-up menu.

The static method `Long.toBinaryString(long i)` will be useful.

2. The RSA CryptoSystem

Read the section on RSA Cryptosystem, RSA encryption, and RSA decryption on page 299-302 in the class text. Make sure you understand Examples 8 and 9. Use your understanding of these examples to write a program for RSA encryption / decryption by following the steps below:

2.1. **Encoding and Decoding.** Implement the methods `hw6encode()`

```
String hw6encode(String str)
String hw6decode(String str)
```

`RSAUtilities` class in the Project shell provided.

Test your methods by running the main method in this class.

2.2. **Determining a block size.** Write an

```
int  blockSize(int maxCode, long n)
```

method that returns the size of the blocks into which to divide the "plain text" after it has been encoded as a stream of numbers. Each character takes up the same number as digits as the `maxCode`, the maximum number used to code a single character. Since all numbers must be members of Z_n, the number represented by a single block must be at most $n - 1$. This method returns a number σk where k is the maximum number of concatenations of maxCode that is less than n, and σ is the number of digits in a single character code.

Test your methods by running the main method in this class.

2.3. **Splitting a string of encoded input into blocks of fixed size.** Write a method

```
 List<String> splitStringByBlockSize(String str,
                               int blockSize, String pad)
```

that splits a string of digits into a list of blocks of a given fixed size. Each block will be a string. The last block may have to be padded to bring it up to the block size: use the string supplied as the given pad parameter. If we adopt the convention that we will always pad by the last character, say `'Z'`, then the pad string will be `"25"`.

2.4. **Merging a list of blocks into a string.** This method is sort of the inverse of the method that splits a string into a list of blocks.

```
 String mergeBlocksIntoString(List <String> blockList)
```

Encrypting a list of blocks: This is performed by a method

```
 List<String> encrypt(List<String> input, Long n, long key,
                                               int blockSize)
```

This method just converts each block (string) into a long integer, encrypts int a long integer using the modular exponentiation according to the RSA encryption method, and then transforms the long integer back to a string. If the integer resulting from the modular exponention is shorte that the block size, it is padded be prefixing it with 0s.

2.5. **Decrypting a list of blocks.** This is performed by a method

```
 List<String> decrypt(List<String> input, Long n, long key,
                                               int blockSize)
```

This method just converts each block (string) into a long integer, decrypts int a long integer using the modular exponentiation according to the RSA decryption method, and then transforms the long integer back to a string. If the integer resulting from the modular exponention is shorte that the block size, it is padded be prefixing it with 0s.

3. Sample Output

When you run the complete testing code included in `main` in the `RSAUtilities` class, your output should look like this:

```
Test Modular Exponentian: 36

Test hw6encode and hw6decode methods:
abc efg hij xyz a
0001022604050626070809262324252600
ABC EFG HIJ XYZ A

Another Test of encode and decode:
STOP
18191415
STOP

Test block size method
4

Test splitting into blocks:
[1819, 1415]

Test Encryption and Decryption:
[2081, 2182]
[0704, 1115]

Decoding of decrypted text:
HELP
```