RSA

After learning about Modular Arithmetic, Mr. Panda wants to apply it to something practical, thus he begins to read up on RSA. RSA (Rivest-Shamir-Adleman) is a widely use cryptographic algorithm that is used to transmit data securely. RSA is an example of an asymmetric algorithm because the encryption and decryption is done using different keys. Suppose Alice wants to allow other people to send encrypted messages to her that no one else can read. She will generate a public key and a private key using this algorithm:

1. Choose 2 distinct large primes **p** and **q**
2. Compute **N = pq**
3. Compute **M = (p – 1)(q – 1)**
4. Choose a value of **E** between 1 and **M**
5. Compute **D** which is the inverse of **E** under the modulus **M**

After these steps, **N** and **E** are released as public keys which other people will use to encrypt the messages they send to Alice, while **D** is kept by Alice as the private key which she will use to decrypt the messages sent to her.

Suppose Bob wants to send a message to Alice. The protocol is as follows:

1. Convert the message into an array of integers
2. For each integer **K**, calculate $\mathbf{K^E \pmod N}$ using Alice's public key **E,** which can be done by repeatedly performing multiplication under modular arithmetic with modulus **N**
3. Convert the array of new integers back to a string and send that as the encrypted message

When Alice receives the encrypted message, she decrypts the message as follows:

1. Convert the encrypted message back to an array of integers
2. For each integer **K**, calculate $\mathbf{K^D \pmod N}$ using Alice's private key **D**
3. Convert this array back to a string to obtain the original message

Mr. Panda wants to help his friends create a simplified RSA system so they can send messages securely[1] to each either. He helps each person pick 2 primes and generate the public and private keys for each person and create an *RSAUser* class to store all this information. After that, they can then send messages to each other securely using the protocols above.

Mr. Panda has coded a skeleton code for the *RSAUser* including functions for converting between a string and an array of integers, but he has been too busy to complete the rest of the class, so he has requested your help to complete the remainder of the code. **You should also make use of the completed *Modulo class* from the other Take-Home Lab to code some of the functionalities.** He also needs your help to code subroutines to read input and write output in the format described below.

[1] Note that this is a highly simplified version of RSA and is not secure at all in practice

Input
The first line of input will contain an integer **N** representing the number of friends Mr. Panda friends. Mr. Panda will have at most 20 friends.

The next **N** lines will each contain the name of the friend, followed by 3 integers representing the two primes **p, q** and public key **E** respectively. It is guaranteed that the name will **only contain lower and uppercase alphabets and will not contain spaces**. In addition, both **p** and **q** will be positive primes less than 150 and **E** will be a value between 1 and **(p – 1)(q – 1)**, and the inverse of **E** modulo **(p – 1)(q – 1)** will exist.

The next line of input will contain an integer **Q** representing the number of queries. There will be at most 40 queries.
Each query will be represented by two lines.
The first line will contain two strings, the first of which is the name of a friend and the second string is either "ENCRYPT" or "DECRYPT". An "ENCRYPT" means you need to encrypt a message using the friend's public key to be sent to the friend. A "DECRYPT" means you need to decrypt an encrypted message using the friend's private key so that it can be read.
The second line will contain the message to be encrypted/decrypted. It will contain **only capital letters and spaces, and will not start or end with a space**. It will also be at most 50 characters long. You should make use of the *nextLine()* function to input this line.

Output
For each query line, print the result of the encryption/decryption.

Sample Input (**rsa1.in**)	Sample Output (**rsa1.out**)
2 Alice 61 71 3977 Bob 101 109 11 4 Alice ENCRYPT HELLO THERE Alice DECRYPT NLNUELO LHHH Bob ENCRYPT GOOD BYE Bob DECRYPT CPFHHG EV	NLNUELO LHHH HELLO THERE CPFHHG EV GOOD BYE

Skeleton

You are given the skeleton file *RSA.java*. You should see a non-empty file containing the partially completed class *Modulo* and *RSAUser* when opening it.

Notes:

1. You should develop your program in the subdirectory `ex1` and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. If your algorithm is different from the given skeleton, you are free to write a solution according to your own algorithm.
3. You do not need to use OOP for this sit-in lab. However, you must use the *RSAUser* class provided.
4. You are free to define your own helper methods. **Remember to use <u>private methods</u> whenever possible.** You are also allowed to modify the main function.
5. Please be reminded that the marking scheme is:

   Input : 10%
   Output : 10%
   Correctness : 50%
   Programming Style : 30%, which consists of:
   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity (incremental programming, proper modifiers [public / private]): 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

   Compilation Error **:** Deduction of **50% of the total marks obtained**.