```
/*
   Computer Networks Laboratory (Lab) 15CSL77
   9. Write a program for simple RSA algorithm to encrypt and decrypt the data.
*/

/* RSA ( Ron Rivest, Adi Shamir, and Leonard Adleman) asymmetric Public Key
     cryptography algorithm

   Asymmetric - Two keys, one key is public other private

   Cryptography - Encryption algorithm

   Cipher Text

   Based on the practical difficulty of the factorization of the product of
     two large prime numbers, the "factoring problem"

   Integer factorization is the decomposition of a composite number into a
     product of smaller integers
   If only prime numbers are used, the process is called prime factorization

   Publish a public key based on two large prime numbers

   Prime numbers must be kept secret.
   Anyone can use the public key to encrypt a message. (Sender)

   Only someone with knowledge of the prime numbers can decode the message
     feasibly (Receiver)
*/

/* Generating Public Key
   1. Choose two distinct prime numbers, p, and q
   2. Compute n = p*q
   3. Compute the totient of the product as λ(n) = (p − 1) * (q − 1)
      λ(n) can also be, λ(n) = lcm(λ(p), λ(q)) = lcm(p − 1, q − 1)
   4. Choose any number 1 < e < λ(n) that is coprime to λ(n),        e

   Coprime: two integers a and b are said to be relatively, mutually or co prime
     if the only positive integer (factor) that divides both of them is 1
   a, and b themself need not be prime.
   Example 14, 15 are co prime, but they themself are not prime,
     only common divisor is 1

   ( n , e )  is the Public key

   Generating Private Key
   5. Compute d, the modular multiplicative inverse of e (mod λ(n))
      Inverse
      Multiplicative
      Modular
      d * e mod λ(n) = 1

   ( n , d )  is the Private key
*/

/*To encrypt, message m, into cipher c
  c = ( m power e ) mod n
  using public key (n, e) at sender
*/

/*To decrypt, cipher text c to message m
  m = ( c power d ) mod n
  using private key, (n, e) at receiver
*/

/*Program should:
  Read prime numbers p, q
  Read message, may be in an array
```

```
  calculate n = p*q

  calculate lambdaN = (p-1) * (q-1)

  find e, which is in between 1 and lambdaN and coprime to lambdaN

  ( n, e ) will be Public key

  encrypt message m, use number representation of character, by ( m power e )mod n
  save as cipher text c, c = ( m power e ) mod n

  To decrypt cipher text, use ( c power d ) mod n
  m = ( c power d ) mod n
  Then change number to its character representation
*/

/*Example:
  p = 3,    q = 11
  n = 33

  λ(n) = (p − 1) * (q − 1) = (3-1)*(11-1) = 2*10 = 20
  Choose any number 1 < e < λ(n) that is coprime to λ(n)
  e = 3

  ( n , e ), is the Public key =    ( 33, 3 )

  Compute d, the modular multiplicative inverse of e (mod λ(n))
  d such that, d * e mod λ(n) = 1
  d * e mod λ(n) = d * 3 mod 20 = 7 * 3 mod 20 = 21 mod 20 = 1

  d = 7

  ( n , d ), is the Private key =   ( 33, 7 )
*/

/*To encrypt, message m to Cipher text c
  c = ( m power e ) mod n

  Consider m = 2 3 4 , say its b c d, b is 2, c is 3 and d is 4
  c = ( m power 3 ) mod 33

  c = ( 2 power 3 ) mod 33 = 8
  c = ( 3 power 3 ) mod 33 = 27
  c = ( 4 power 3 ) mod 33 = 64 mod 33 = 31

  For message m = 2 3 4 , Cipher c = 8 27 31
*/

/*To decrypt, cipher text c back to message m
  m = ( c power d ) mod n

  m = ( c power 7 ) mod 33

  c = 8 27 31

  m = (  8 power 7 ) mod 33 = 2
  m = ( 27 power 7 ) mod 33 = 3
  m = ( 31 power 7 ) mod 33 = 4

  For Cipher c = 8 27 31 , message m = 2 3 4
*/

/*Example
  p = 13,   q = 17
  n = p * q = 13 * 17 = 221

  λ(n) = ( p - 1 ) * ( q - 1 ) = 12*16 = 192

  e, such that, 1 < e < λ(n) and coprime to λ(n),    e = 35
```

```
  ( n , e ) is Public key = ( 221 , 35 )

  Private key, d, such that,    d * e mod λ(n) = 1

   d * 35 mod 192 = 1
  11 * 35 mod 192 = 1

  ( n , d ) is Private key. ( 221 , 11 )
*/

/*To encrypt, message m to Cipher c
  c = ( m power e ) mod n

  Suppose, m = 1 2 3 , say its a b c, a is 1, b is 2 and c is 3
  c = ( m power 35 ) mod 221

  c = ( 1 power 35 ) mod 221 = 1
  c = ( 2 power 35 ) mod 221 = 59
  c = ( 3 power 35 ) mod 221 = 61

  For message m = 1  2 3, the cipher c = 1 59 61
*/

/*To decrypt, cipher c back to message m
  m = ( c power  d ) mod n
  m = ( c power 11 ) mod 221

  c = 1 59 61

  m = (  1 power 11 ) mod 221 = 1
  m = ( 59 power 11 ) mod 221 = 2
  m = ( 61 power 11 ) mod 221 = 3

  From cipher c = 1 59 61, message obtained, m = 1 2 3
*/

#include <stdio.h>
#include <math.h>

int gcd(int m , int n )// ALGORITHM Euclid(m, n), gcd: greatest common divisor
 {                      //   Computes gcd(m, n) by Euclid’s algorithm
   int r = 0;           //   Input: Two nonnegative, not-both-zero integers m and n
   char temp;           //   Output: Greatest common divisor of m and n
   while( n!=0 )        //   while n != 0
    {                   //     do
      r = m % n;        //       r ← m mod n
      m = n;            //       m ← n
      n = r;            //       n ← r
    }                   //     done
   return m;            //   return m
 }

/*
  Procedure to computes a to power of b mod n

  MODULAR-EXPONENTIATION( a, b, n )
    c ← 0
    d ← 1
    let ( b_k , b_k-1 , . . . b_1 , b_0 ) be the binary representation of b

    for i ← k down to 0
        c ← 2c
        d ← ( d * d ) mod n

        if b_i == 1
            c ← c + 1
            d ← ( d * a ) mod n
```

```
    return d

    d is a^b mod n
*/

int modularExponentiation( int a, int b, int n )
 {  // returns d = a to the power of b mod n
    int c = 0;  // c ← 0
    int d = 1;  // d ← 1

    int num = b;
    int binary[16]; // b_k, b_k-1,...b_1, b_0 be binary representation of b
    int k = 0; // length of binary representation of b

    int i;

    while ( num != 0 ) // convert b to binary
     {
       binary[ k++ ] = num % 2;
       num = num / 2;
     }

    for( i = k-1 ; i>=0; i-- ) // i ← k down to 0
     {
        c = 2*c ;              // c ← 2c
        d = ( d * d ) % n;     // d ← ( d * d ) mod n

        if ( binary[i] == 1)                  // if b_i == 1
          {
            c = c + 1;                        // c ← c + 1
            d = ( d * a ) % n;                // d ← ( d * a ) mod n
          }
     }

    return d;
 }


int main()
 {
   int p, q, n, lambdaN, d, e, length, i;
   int message[10], cipher[10];

   // read two distinct prime numbers, p, and q

   // Compute n = p*q
   // Compute totient of the product λ(n)=(p−1)*(q−1)

   // choose number e, such that 1 < e < λ(n) and is coprime to λ(n)
   // Find e, such that gcd( e, λ(n) ) = 1, Greatest common divisor of e and λ(n)
   // Two numbers e and λ(n) are co prime if gcd( e, λ(n) ) = 1,
   // That is no common divisor other than 1

   printf("\n Public key = ( %d, %d )", n, e);

   // Private key, d, such that,    d * e mod λ(n) = 1
   printf("\n Private key = ( %d, %d )", n, d);

   printf("\n Enter length of message: ");    scanf("%d", &length);

   // read message

   // At sender, encrypt message to cipher, cipher

   // At receiver, decrypt cipher to message, message

   return 0;
 }
```

```
/*Textbook:
  Behrouz Forouzon - Data Communications and Networking, McGraw Hill Edition
  Anany Levitin, Introduction to the design & analysis of algorithms
  Thomas H. Cormen , Charles E. Leiserson , Ronald L. Rivest, Clifford Stein
    Introduction to Algorithms
*/
```