```
# A Python3 program to demonstrate
# working of Chinise remainder Theorem

# k is size of num[] and rem[].
# Returns the smallest number x
# such that:
# x % num[0] = rem[0],
# x % num[1] = rem[1],
# ..................
# x % num[k-2] = rem[k-1]
# Assumption: Numbers in num[]
# are pairwise coprime (gcd for
# every pair is 1)
def findMinX(num, rem, k):
    x = 1; # Initialize result

    # As per the Chinise remainder
    # theorem, this loop will
    # always break.
    while(True):

        # Check if remainder of
        # x % num[j] is rem[j]
        # or not (for all j from
        # 0 to k-1)
        j = 0;
        while(j < k):
            if (x % num[j] != rem[j]):
                break;
            j += 1;

        # If all remainders
        # matched, we found x
        if (j == k):
            return x;

        # Else try next number
        x += 1;

# Driver Code
num = [3, 4, 5];
rem = [2, 3, 1];
k = len(num);
print("x is", findMinX(num, rem, k));

# This code is contributed by mits
```

```c
// C program to demonstrate working of extended
// Euclidean Algorithm
#include <stdio.h>

// C function for extended Euclidean Algorithm
int gcdExtended(int a, int b, int *x, int *y)
{
    // Base Case
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }

    int x1, y1; // To store results of recursive call
    int gcd = gcdExtended(b%a, a, &x1, &y1);

    // Update x and y using results of recursive
    // call
    *x = y1 - (b/a) * x1;
    *y = x1;

    return gcd;
}

// Driver Program
int main()
{
    int x, y;
    int a = 35, b = 15;
    int g = gcdExtended(a, b, &x, &y);
    printf("gcd(%d, %d) = %d", a, b, g);
    return 0;
}
```

```
# Python program to demonstrate working of extended
# Euclidean Algorithm

# function for extended Euclidean Algorithm


def gcdExtended(a, b):

    # Base Case
    if a == 0:
        return b, 0, 1

    gcd, x1, y1 = gcdExtended(b % a, a)

    # Update x and y using results of recursive
    # call
    x = y1 - (b//a) * x1
    y = x1

    return gcd, x, y


# Driver code
a, b = 35, 15
g, x, y = gcdExtended(a, b)
print("gcd(", a, ",", b, ") = ", g)
```

```
# Python for RSA asymmetric cryptographic algorithm.
# For demonstration, values are
# relatively small compared to practical application
import math


def gcd(a, h):
    temp = 0
    while(1):
        temp = a % h
        if (temp == 0):
            return h
        a = h
        h = temp
```

```
p = 3
q = 7
n = p*q
e = 2
phi = (p-1)*(q-1)

while (e < phi):

    # e must be co-prime to phi and
    # smaller than phi.
    if(gcd(e, phi) == 1):
        break
    else:
        e = e+1

# Private key (d stands for decrypt)
# choosing d such that it satisfies
# d*e = 1 + k * totient

k = 2
d = (1 + (k*phi))/e

# Message to be encrypted
msg = 12.0

print("Message data = ", msg)

# Encryption c = (msg ^ e) % n
c = pow(msg, e)
c = math.fmod(c, n)
print("Encrypted data = ", c)

# Decryption m = (c ^ d) % n
m = pow(c, d)
m = math.fmod(m, n)
print("Original Message Sent = ", m)


# This code is contributed by Pranay Arora.
```

```c
// C program for RSA asymmetric cryptographic
// algorithm. For demonstration values are
// relatively small compared to practical
// application
#include<stdio.h>
#include<math.h>

// Returns gcd of a and b
int gcd(int a, int h)
{
    int temp;
    while (1)
    {
        temp = a%h;
        if (temp == 0)
          return h;
        a = h;
        h = temp;
    }
}

// Code to demonstrate RSA algorithm
int main()
{
    // Two random prime numbers
    double p = 3;
    double q = 7;

    // First part of public key:
    double n = p*q;

    // Finding other part of public key.
    // e stands for encrypt
    double e = 2;
    double phi = (p-1)*(q-1);
    while (e < phi)
    {
        // e must be co-prime to phi and
        // smaller than phi.
        if (gcd(e, phi)==1)
            break;
        else
            e++;
    }

    // Private key (d stands for decrypt)
    // choosing d such that it satisfies
    // d*e = 1 + k * totient
```

```c
    int k = 2;  // A constant value
    double d = (1 + (k*phi))/e;

    // Message to be encrypted
    double msg = 20;

    printf("Message data = %lf", msg);

    // Encryption c = (msg ^ e) % n
    double c = pow(msg, e);
    c = fmod(c, n);
    printf("\nEncrypted data = %lf", c);

    // Decryption m = (c ^ d) % n
    double m = pow(c, d);
    m = fmod(m, n);
    printf("\nOriginal Message Sent = %lf", m);

    return 0;
}
// This code is contributed by Akash Sharan.
```