**Garvin Chanderia**

**2203132**

**LY-ISA1**

**Experiment No:1**

**Title:**  Chinese Remainder Theorem

**Problem Statement:** Implement a number theory such as the Chinese remainder Theorem

**Aim:**  To study & implement the Chinese Remainder Theorem

**Theory:** Chinese Remainder Theorem is used to solve set of congruent equations with one variable but different modulus, which are relatively prime

x ≡ a1 mod m1

x≡ a2 mod m2 x ≡ a3 mod m3

……

     x ≡ ak mod mk

The Chinese Remainder Theorem states that the above equations have a unique solution if the moduli are relatively prime. Below are the steps needed to follow to solve set of congruent equations using Chinese Remainder Theorem

**Step I:** Find M = m1 x m2 x m3…mk where M is common modulus

**Step II:** Find M1 = M/m1, M2 = M/m2 and so on

**Step III:** Find multiplicative inverses for M1, M2 and so on

**Step IV:** Put the values in the below equation to solve for X

X = (a1 x M1 x M1-1 +a2 x M2 x M2-1+  a3 x M3 x M3-1) mod M

Example

X = 4 mod 5

X = 6 mod 8

X = 8 mod 9

**Step I:** M = 5 \* 8\* 9 = 360

**Step II:** M1 = M/m1 = 360 / 5 = 72

M2 = M/m2 = 360 / 8 = 45

M3 = M/m3 = 360 / 9 = 40

Step III:

To find the M1 inverse, Solve for GCD (m1, M1)  using Extended Euclidean Algorithm. GCD (5, 72)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *q* | *r1* | *r2* | *r* | *t1* | *t2* | *t* |
| 0 | 5 | 72 | 5 | 0 | 1 | 0 |
| 14 | 72 | 5 | 2 | 1 | 0 | 1 |
| 2 | 5 | 2 | 1 | 0 | 1 | -2 |

The inverse value cannot be negative, so add modulus into it to make it positive.

M1 inverse = -2 + 5 = 3

To find M2 inverse, Solve for GCD (m2, M2) using Extended Euclidean Algorithm. GCD (8, 45)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *q* | *r1* | *r2* | *r* | *t1* | *t2* | *t* |
| 0 | 8 | 45 | 8 | 0 | 1 | 0 |
| 5 | 45 | 8 | 5 | 1 | 0 | 1 |
| 1 | 8 | 5 | 3 | 0 | 1 | -1 |
| 1 | 5 | 3 | 2 | 1 | -1 | 2 |
| 1 | 3 | 2 | 1 | -1 | 2 | -3 |

M2 inverse = -3 + 8 = 5

To find the M3 inverse, Solve for GCD (m3, M3) using Extended Euclidean Algorithm. GCD (9, 40)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *q* | *r1* | *r2* | *r* | *t1* | *t2* | *t* |
| 0 | 9 | 40 | 9 | 0 | 1 | 0 |
| 4 | 40 | 9 | 4 | 1 | 0 | 1 |
| 2 | 9 | 4 | 1 | 0 | 1 | -2 |

M3 inverse = -2 + 9 = 7

**Step IV:** Put the values in the below equation to solve for X

X = (a1 x M1 x M1-1 + a2 x M2 x M2-1 + a3 x M3 x M3-1) mod M X = (4\*72\*3 + 6\*45\*5 + 8\*40\*7) mod 360

X = (864 + 1350 + 2240) mod 360

X = 4454 mod 360

X = 134

**Application:**

The Chinese Remainder Theorem has several applications in cryptography. One is to solve the quadratic congruence and the other is to represent very large number in terms of list of small integers

**Algorithm/Pseudocode:**

 # A Python 3 program to demonstrate

# working of Chinese remainder Theorem

 # Returns modulo inverse of a with

# respect to m using extended

# Euclid Algorithm. Refer below

# post for details:

# multiplicative-inverse-under-modulo-m/

def inv(a, m) :

    m0 = m

    x0 = 0

    x1 = 1

    if (m == 1) :

        return 0

    while (a > 1) :

        # q is quotient

        q = a // m

        t = m

        # m is remainder now, process

        # same as euclid's algo

        m = a % m

        a = t

        t = x0

        x0 = x1 - q \* x0

        x1 = t

    # Make x1 positive

    if (x1 < 0) :

        x1 = x1 + m0

    return x1

# k is the 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) :

    # Compute product of all numbers

    prod = 1

    for i in range(0, k) :

        prod = prod \* num[i]

    # Initialize result

    result = 0

    # Apply above formula

    for i in range(0,k):

        pp = prod // num[i]

        result = result + rem[i] \* inv(pp, num[i]) \* pp

          return result % prod

  # Driver method

num = [3, 4, 5]

rem = [2, 3, 1]

k = len(num)

print( "x is " , findMinX(num, rem, k))

**Code:**

#include<bits/stdc++.h>

using namespace std;

int findMinX(int num[], int rem[], int k)

{ int x = 1;

while (true)

{ int j;

for (j=0; j<k; j++ )

if (x%num[j] != rem[j])

break;

if (j == k)

return x;

x++;

} return x;

}

int main(void)

{

int num[] = {3, 4, 5};

int rem[] = {2, 3, 1};

int k = sizeof(num)/sizeof(num[0]);

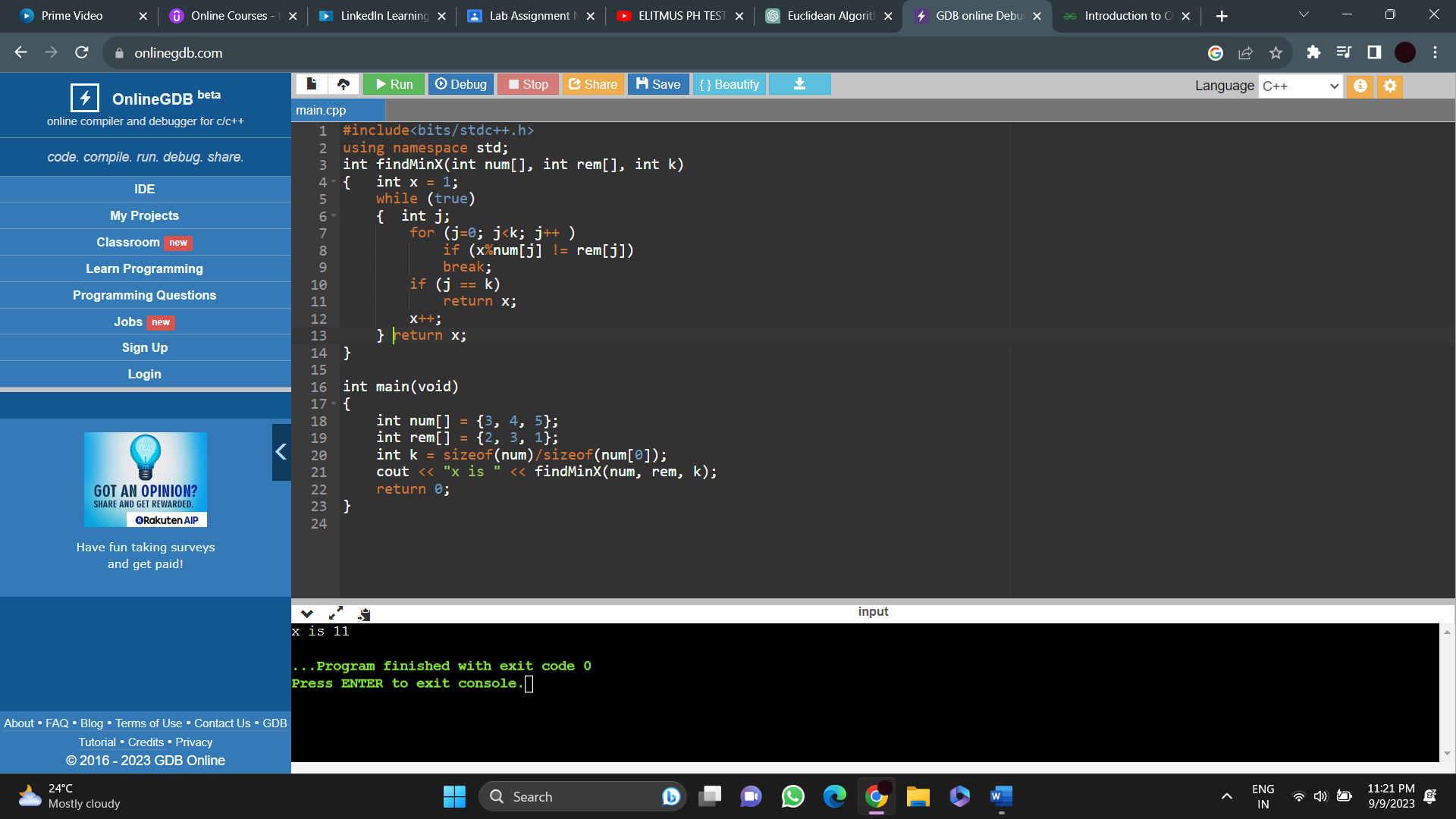
cout << "x is " << findMinX(num, rem, k);

return 0;

}

**Output:**

x is 11



**Conclusion:** We have studied & implemented the Chinese Remainder Theorem.

