**4CS401: Cryptography and Network Security**

**B.Tech. (CSE) – I [ 2022-23 ]**

**Assignment No - 11**

**Chinese Reminder Theorem**

**Title: Chinese Reminder Theorem**

**Aim: To Demonstrate Chinese Reminder Theorem**

**Theory:**

**Chinese Remainder Theorem states that there always exists an x that satisfies given congruences**

**Code :**

**#include <bits/stdc++.h>**

**#define ll long long**

**#define ul unsigned long long**

**#define pb emplace\_back**

**#define po pop\_back**

**#define vi vector<ll>**

**#define vii vector<vector<ll>>**

**using namespace std;**

**const int MODVALUE = 1e9;**

**long long gcdExtended(long long a, long long b, long long \*x, long long \*y)**

**{**

**cout << a << " " << b << " "**

**<< " " << \*x << " " << \*y << "\n";**

**// Base Case**

**if (b == 0)**

**{**

**return \*x;**

**}**

**long long q = a / b;**

**long long x1 = \*y;**

**long long y1 = \*x - q \* (\*y);**

**long long t1 = gcdExtended(b, a % b, &x1, &y1);**

**cout << a << " " << \*x << "\n";**

**if (\*x == 0 && t1 < 0)**

**return a + t1;**

**else**

**return t1;**

**// return gcd;**

**}**

**int main() {**

**char patternChar = '-';**

**char resetChar = ' ';**

**int lineWidth = 90;**

**int initialWidth = 50;**

**cout << setfill(patternChar) << setw(lineWidth) << patternChar << endl;**

**cout << setfill(resetChar);**

**cout << setw(initialWidth) << "Chinese Remainder Theorm" << endl;**

**cout << setfill(patternChar) << setw(lineWidth) << patternChar << endl;**

**cout << setfill(resetChar);**

**cout << "Enter the total number of equations involved: ";**

**int n;**

**cin >> n;**

**vector<int> divisor(n, 0);**

**vector<int> remainder(n, 0);**

**// M = m1 \* m2 \* m3 \* .....**

**long long int M = 1;**

**cout << "Enter the divisors of " << n << " the equations: \n" << endl;**

**for(int i = 0; i < n; i++){**

**cin >> divisor[i];**

**M \*= divisor[i];**

**M %= MODVALUE;**

**}**

**cout << "Enter the remainders of " << n << " equations: \n" << endl;**

**for(int i = 0; i < n; i++){**

**cin >> remainder[i];**

**}**

**// finding m1, m2, m3, ...**

**vector<int> mValues(n);**

**vector<int> invMValues(n);**

**for(int i = 0; i < n; i++){**

**mValues[i] = M/divisor[i];**

**long long x=0, y=1;**

**x = gcdExtended(divisor[i],mValues[i],&x, &y);**

**cout<<"The inverse for M"<<(i+1)<<" = "<<mValues[i]<<" is "<<x<<"\n";**

**invMValues[i] = x;**

**}**

**long long ans = 0;**

**for(int i = 0; i < n; i++){**

**ans += (((1LL\* remainder[i] \* mValues[i])%M)\*invMValues[i])%M;**

**ans %= M;**

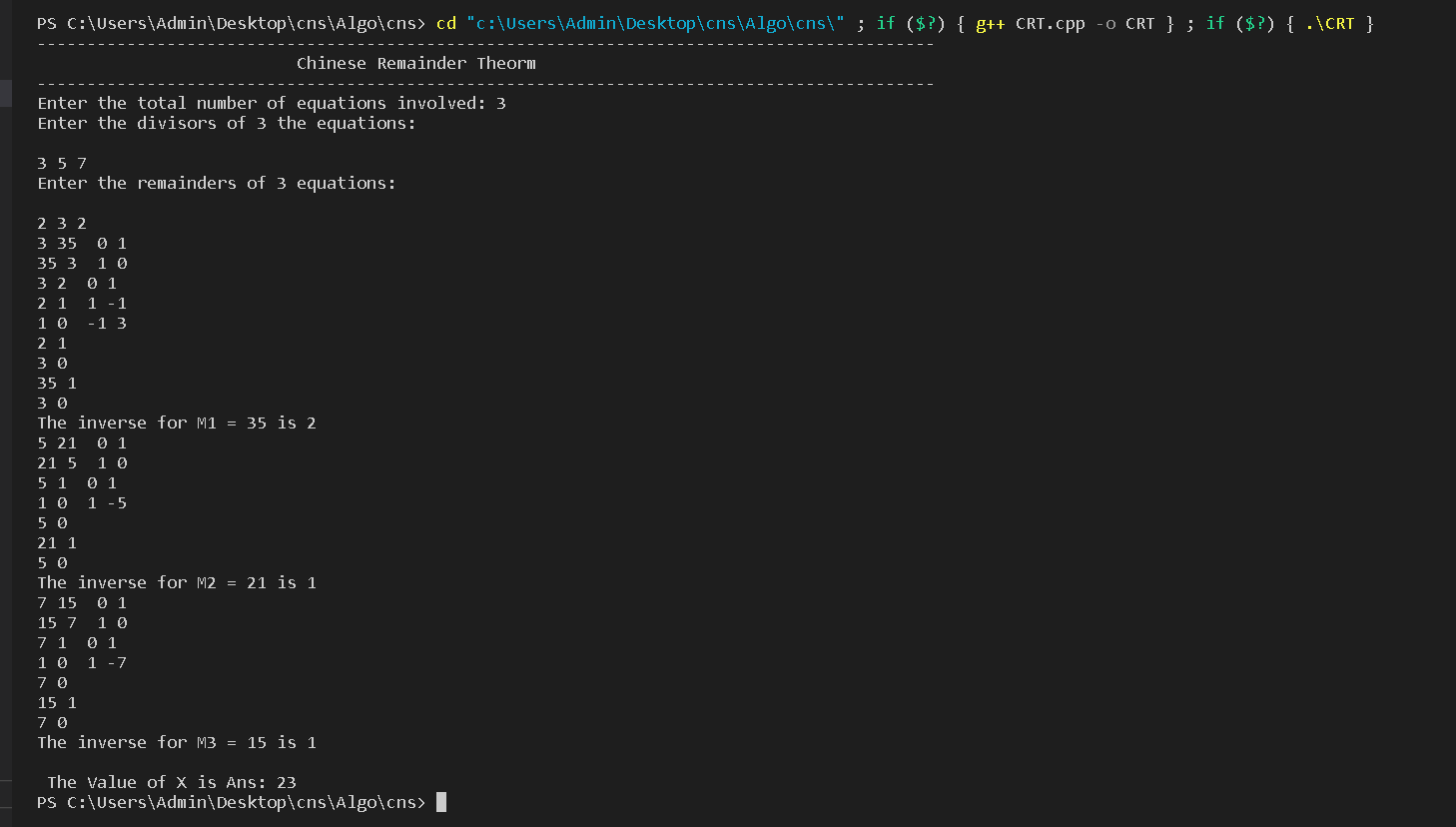
**}**

**cout << "\n The Value of X is Ans: " << ans << endl;**

**return 0;**

**}**

**Output:**

****