Name: Rutikesh Sawant

Batch: B2

Subject: CNS Lab

PRN: 2019BTECS00034

Aim: Chinese Remainder Theorem implementation

Theory:

x = a1 (mod n1)

...

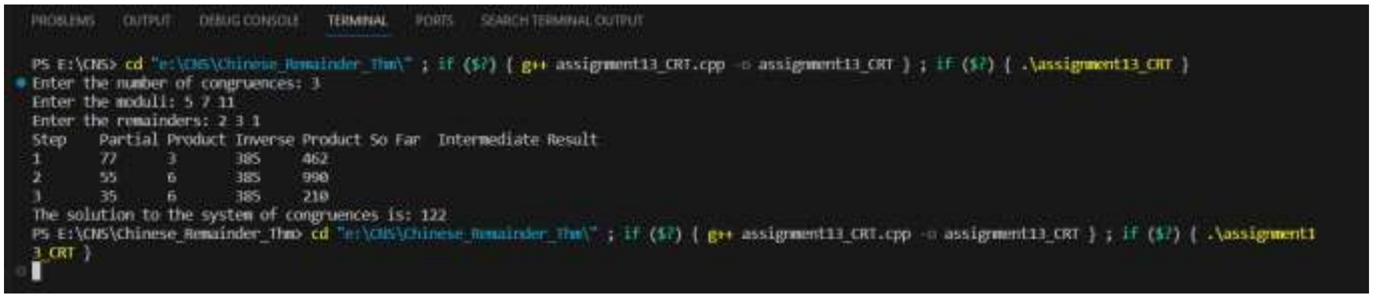
x = ak (mod nk)

This is equivalent to saying that x mod ni = ai (for i=1…k). The notation above is common in group theory, where you can define the group of integers modulo some number n and then you state equivalences (or congruence) within that group. So x is the unknown; instead of knowing x, we know the remainder of the division of x by a group of numbers. If the numbers ni are pairwise coprimes (i.e. each one is coprime with all the others) then the equations have exactly one solution. Such solution will be modulo N, with N equal to the product of all the n_i.

Code:

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the modular inverse using extended Euclidean algorithm
int modInverse(int a, int m) {
    a = a % m;
    for (int x = 1; x < m; x++) {
        if ((a * x) % m == 1) {
            return x;
        }
    }
    return -1; // Modular inverse doesn't exist
}

// Function to find the solution to the system of congruences using CRT
int chineseRemainderTheorem(vector<int>& num, vector<int>& rem) {
    int product = 1;
    int n = num.size();

    for (int i = 0; i < n; i++) {
        product *= num[i];
    }

    vector<int> partialProducts(n);
    vector<int> inverse(n);

    int x = 0;

    cout << "Step\tPartial Product\tInverse\tProduct So Far\tIntermediate
Result" << endl;

    for (int i = 0; i < n; i++) {
        partialProducts[i] = product / num[i];
        inverse[i] = modInverse(partialProducts[i], num[i]);
        int intermediateResult = partialProducts[i] * inverse[i] * rem[i];
        x += intermediateResult;

        cout << i + 1 << "\t" << partialProducts[i] << "\t" << inverse[i] <<
"\t" << product << "\t" << intermediateResult << endl;
    }

    x = x % product;

    return x < 0 ? x + product : x;
}

int main() {
    int n;
    cout << "Enter the number of congruences: ";
    cin >> n;

    vector<int> num(n);
    vector<int> rem(n);

    cout << "Enter the moduli: ";
    for (int i = 0; i < n; i++) {
        cin >> num[i];
    }

    cout << "Enter the remainders: ";
    for (int i = 0; i < n; i++) {
        cin >> rem[i];
    }

    int result = chineseRemainderTheorem(num, rem);

    cout << "The solution to the system of congruences is: " << result << endl;

    return 0;
}
```

Output:



```
PS E:\CNS> cd "e:\CNS\Chinese_Remainder_Thm\" ; if ($?) { g++ assignment13_CRT.cpp -o assignment13_CRT } ; if ($?) { .\assignment13_CRT }
Enter the number of congruences: 3
Enter the moduli: 5 7 11
Enter the remainders: 2 3 1
Step    Partial Product Inverse Product So Far  Intermediate Result
1       77      3       385     462
2       55      6       385     990
3       35      6       385     210
The solution to the system of congruences is: 122
PS E:\CNS\Chinese_Remainder_Thm> cd "e:\CNS\Chinese_Remainder_Thm\" ; if ($?) { g++ assignment13_CRT.cpp -o assignment13_CRT } ; if ($?) { .\assignment1
3_CRT }
```