# NTUST OOP Midterm Problem Design

**Subject: Chinese Remainder Theorem**

**Author: 謝鈞曜 (CHUN-YAO HSIEH)**

**Main testing concept:** Array、GCD、LCM、Recursion

| Basics | Functions |
|---|---|
| ■ C++ BASICS | □ SEPARATE COMPILATION AND NAMESPACES |
| ■ FLOW OF CONTROL | □ STREAMS AND FILE I/O |
| ■ FUNCTION BASICS | ■ RECURSION |
| □ PARAMETERS AND OVERLOADING | □ INHERITANCE |
| ■ ARRAYS | □ POLYMORPHISM AND VIRTUAL FUNCTIONS |
| □ STRUCTURES AND CLASSES | □ TEMPLATES |
| □ CONSTRUCTORS AND OTHER TOOLS | □ LINKED DATA STRUCTURES |
| □ OPERATOR OVERLOADING, FRIENDS, AND REFERENCES | □ EXCEPTION HANDLING |
| | □ STANDARD TEMPLATE LIBRARY |
| □ STRINGS | □ PATTERNS AND UML |
| □ POINTERS AND DYNAMIC ARRAYS | |

## Description:

The Chinese Remainder Theorem is a theorem for solving systems of congruences. It states that if there are $n$ equations, each of the form $x \equiv a_i (\, mod\, m_i)$, *where* $m_1$, $m_2$ ... $m_i$ are pairwise coprime positive integers, then these equations can be simultaneously satisfied, and $x$ can be uniquely solved. In other words, if there are multiple congruence equations and the modulo in the equations are pairwise coprime, then the solutions to these equations can be found using the Chinese Remainder Theorem. Assuming we have the following system of congruences:

$$\mathrm{x} \equiv 2\ (\, mod\ 3), \mathrm{x} \equiv 3\ (\, mod\ 5), \mathrm{x} \equiv 2\ (\, mod\ 7)$$

First, we need to check if the modulo in each equation are relatively prime, i.e., $gcd(3,5) = gcd(5,7) = gcd(3,7) = 1$, if the modulo in the equations are not relatively prime, **print "No solution"**. Since each pair of modulo is relatively prime, we can use the Chinese Remainder Theorem to solve the system. According to the Chinese Remainder Theorem, we need to first calculate the values of $M_i$, where $M$ is the product of all the modulo, i.e., $M = 3 \times 5 \times 7 = 105$. $M_i$ is the quotient when $M$ is divided by the modulus $m_i$, i.e.:

$$M_1 = \frac{M}{3} = 35,\ M_2 = \frac{M}{5} = 21,\ M_3 = \frac{M}{7} = 15$$

Next, we need to calculate the modular multiplicative inverse $b_i$ of each $M_i$. For each $M_i$, we need to find an integer $b_i$ such that $b_i \times M_i \equiv 1\ (mod\ m_i)$. Afterwards, calculate the solution *x*, and according to the Chinese Remainder Theorem, $x = \Sigma(a_i \times M_i \times b_i)\ (mod\ M)$. In the end, the smallest $x$ is the solution to this system of congruences.

- $2 \leq n < 10$
- $2 \leq a_i < m_i < 2147483648$
- $0 \leq M < 9{,}223{,}372{,}036{,}854{,}775{,}808$

## Input:

n
$a_1\ m_1$
.
.
$a_n\ m_n$

## Output

**x**

## Sample Input / Output：

| Sample Input | Sample Output |
|---|---|
| 2<br>21477 214719<br>2147483 21474836 | 150927294891 |
| 5<br>625 797<br>477 5261<br>153 631<br>2718 19949<br>3545 3989 | 138999336506034318 |
| 3<br>2315 15625<br>123 625<br>9 19 | No solution |

□ **Eazy,Only basic programming syntax and structure are required.**

□ **Medium,Multiple programming grammars and structures are required.**

■ **Hard,Need to use multiple program structures or more complex data types.**

## Expected solving time:

60 minutes

## Other notes:

The notation $x \equiv a \ (mod\ m)$ represents a congruence relation between $x$, $a$, and $m$, where $x$ is an integer that leaves a remainder of a when divided by $m$. For example, if we divide $x$ by 7, the remainder should be 3. So, $x$ could be 3, 10, 17, etc. They all leave a remainder of 3 when divided by 7.

Since $\boldsymbol{M}$ may **overflow**, you should use a multiplication algorithm to reduce $R$, and $R = (A * B)\ mod\ M$. Calculate $R$ by following steps:

1. *Define* $A, B, M, R$
2. $R \leftarrow 0$
3. $A \leftarrow (A\ mod\ M)$
4. *While* $B$ *is greater than 0, do the following:*
   a. *If the* $B$ *is odd,* $R \leftarrow ((A + R)\ mod\ M)$
   b. $A \leftarrow ((2 * A)\ mod\ M)$
   c. $B \leftarrow (B \div 2)$
5. *Return R.*

***The Extended Euclidean Algorithm for finding the inverse of a number mod n.***

Start from $Step\ 0$. The quotient obtained at step i will be denoted by $q_i$. As we carry out each step of the Euclidean algorithm, we will also calculate an auxiliary number, $p_i$. For the first two steps, the value of this number is given: $p_0 = 0$ and $p_1 = 1$. For the remainder of the steps, we recursively calculate $p_i = p_{i-2} - p_{i-1}\ q_{i-2}$ (mod n). Continue this calculation for one step beyond the last step of the Euclidean algorithm. The algorithm starts by "dividing" n by x. If the last non-zero remainder occurs at step k, and if this remainder is 1, x has an inverse and it is $p_{k+2}$. (If the remainder is not 1, then x does not have an inverse.) Here is an example: **Find the inverse of 15 mod 26.**

$Step\ 0$: $26 = 1(15) + 11$ $p_0 = 0$

$Step\ 1$: $15 = 1(11) + 4$ $p_1 = 1$

$Step\ 2$: $11 = 2(4) + 3$ $p_2 = 0 - 1(1)\ mod\ 26 = 25$

$Step\ 3$: $4 = 1(3) + 1$ $p_3 = 1 - 25(1)\ mod\ 26 = -24\ mod\ 26 = 2$

$Step\ 4$: $3 = 3(1) + 0$ $p_4 = 25 - 2(2)\ mod\ 26 = 21$

$Step\ 5$: $p_5 = 2 - 21(1) mod\ 26 = -19\ mod\ 26 = 7$

**Notice that 15(7) = 105 = 1 + 4(26) ≡ 1 (mod 26), so 7 is the inverse of 15 mod 26.**