**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  ■ FLOW OF CONTROL  ■ FUNCTION BASICS  □ PARAMETERS AND OVERLOADING  ■ ARRAYS  □ STRUCTURES AND CLASSES  □ CONSTRUCTORS AND OTHER TOOLS  □ OPERATOR OVERLOADING, FRIENDS, AND REFERENCES  □ STRINGS  □ POINTERS AND DYNAMIC ARRAYS | □ SEPARATE COMPILATION AND NAMESPACES  □ STREAMS AND FILE I/O  ■ RECURSION  □ INHERITANCE  □ POLYMORPHISM AND VIRTUAL FUNCTIONS  □ TEMPLATES  □ LINKED DATA STRUCTURES  □ EXCEPTION HANDLING  □ STANDARD TEMPLATE LIBRARY  □ PATTERNS AND UML | |
| **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 *, where*  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:  , ,  First, we need to check if the modulo in each equation are relatively prime, i.e., , 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 where is the product of all the modulo, i.e., . is the quotient when is divided by the modulus , i.e.:  *, ,*  Next, we need to calculate the modular multiplicative inverse of each . For each , we need to find an integer such that × ≡ . Afterwards, calculate the solution *x*, and according to the Chinese Remainder Theorem, . In the end, the smallest is the solution to this system of congruences.  **Input:**  n  .  .  **Output**  **x**  **Sample Input / Output：**   |  |  | | --- | --- | | Sample Input | Sample Output | | 2  21477 214719  2147483 21474836 | 150927294891 | | 5  625 797  477 5261  153 631  2718 19949  3545 3989 | 138999336506034318 | | 3  2315 15625  123 625  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 represents a congruence relation between , , and , where is an integer that leaves a remainder of a when divided by . For example, if we divide by 7, the remainder should be 3. So, could be 3, 10, 17, etc. They all leave a remainder of 3 when divided by 7.  Since ***M*** may **overflow**, you should use a multiplication algorithm to reduce , and = . Calculate by following steps:   1. *Define* 2. *While is greater than 0, do the following:*    1. *If the is odd,* 3. *Return R.*   ***The Extended Euclidean Algorithm for finding the inverse of a number mod n.***  Start from . The quotient obtained at step i will be denoted by qi. As we carry out each step of the Euclidean algorithm, we will also calculate an auxiliary number, pi. For the first two steps, the value of this number is given: p0 = 0 and p1 = 1. For the remainder of the steps, we recursively calculate pi = pi-2 - pi-1 qi-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 pk+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.**  **Notice that 15(7) = 105 = 1 + 4(26)**  **1 (mod 26), so 7 is the inverse of 15 mod 26.** |