**CPP Problem Design Example**

|  |
| --- |
| **Contributor︰Cheryl Huang, Jun-Yu Chen** |
| **Subject：The Translation Machine** |
| **Main testing concept：**   |  |  | | --- | --- | | **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：**  Here is a translation machine. You are given the possible translations of letters which are a list of pairs of original and deciphered characters. Your task is to verify whether a word can be translated into another given translatable letter pairs. Two words match if the they have the same length and each character of the first word can be turned into a corresponding character of the second word after the translation process. For example, a word “sky”, given a translation pair making ‘k’ the same as ‘p’, yields the word ‘spy’. As a result, we deem ‘sky’ the same as ‘spy’, with the position of letters having to correspond to the other by using the available translations zero or more times.  **Input：**  Users input through standard input. The input contains several test cases, each of them as described below.   1. The first line of input contains two integers **m (1 ≤ m ≤ 500)** and **n (1 ≤ n ≤ 50)**, where **m** is the number of translations of letters and **n** is the number of word pairs. All given test inputs will fall in this range. 2. Each of the next **m** lines contains two distinct space-separated letters **a** and **b**, indicating that the letter **a** can be translated to the letter **b**. Each ordered pair of letters **(a, b)** appears at most once. Following this are **n** lines, each containing a word pair to check. Remember, there are chances where one letter could be translated into many others. For example, for the pairs (i, b), (l, i), (y, l) the word ‘y’ could be translated into either ‘l’, ‘i’ or ‘b’. 3. Translations and words use only lowercase letters ‘a’…‘z’ and each word contains at least 1 and at most 50 letters. 4. Exit the program while **m** and **n** are both **0**.   **Output：**  For each pair of words, display ‘**yes**’ if the two words match, and ‘**no**’ otherwise, on a line by itself.  **Error handling：**  Output “ERROR” when uppercase letters (i.e., ‘L’) or other invalid inputs (i.e., numbers) are read. However, the input still counts towards the input tally while the information is ignored. (The input counter adds one but the inputted information will not be used). Please refer to Sample Input / Output for a more thorough understanding regarding error handling.  **Sample Input / Output :**   |  |  | | --- | --- | | **Sample Input** | **Sample Output** | | 10 5 |  | | c t  i r  k p  o c  r o  t e  t f  u h  w p  E c | ERROR | | we we  can the  work people  it of  out the | yes  no  no  yes  yes | | 3 4 |  | | a c  b a  a b |  | | aaa abc  abc aaa  acm bcm  a3 bb | yes  no  yes  ERROR | | 0 0 |  | |
| **□ Easy, only basic programming syntax and structure are required.**  **■ Medium, multiple programming grammars and structures are required.**  **□ Hard, need to use multiple program structures or complex data types.** |
| **Expected solving time:**  40 minutes |
| **Other notes：** |