ICPC NEERC 2018 J - JS Minification

Time limit: 1.5s **Memory limit:** 512M

International Coding Procedures Company (ICPC) writes all its code in Jedi Script (JS) programming language. JS does not get compiled, but is delivered for execution in its source form. Sources contain comments, extra whitespace (including trailing and leading spaces), and other non-essential features that make them quite large but do not contribute to the semantics of the code, so the process of minification is performed on source files before their delivery to execution to compress sources while preserving their semantics.

You are hired by ICPC to write JS minifier for ICPC. Fortunately, ICPC adheres to very strict programming practices and their JS sources are quite restricted in grammar. They work only on integer algorithms and do not use floating point numbers and strings.

Every JS source contains a sequence of lines. Each line contains zero or more tokens that can be separated by spaces. On each line, a part of the line that starts with a hash character (`#` code 35), including the hash character itself, is treated as a comment and is ignored up to the end of the line.

Each line is parsed into a sequence of tokens from left to right by repeatedly skipping spaces and finding the longest possible token starting at the current parsing position, thus transforming the source code into a sequence of tokens. All the possible tokens are listed below:

- A reserved token is any kind of operator, separator, literal, reserved word, or a name of a library function that should be preserved during the minification process. Reserved tokens are fixed strings of non-space ASCII characters that do not contain the hash character (`#` code 35). All reserved tokens are given as an input to the minification process.
- A number token consists of a sequence of digits, where a digit is a character from zero (`0`) to nine (`9`) inclusive.
- A word token consists of a sequence of characters from the following set: lowercase letters, uppercase letters, digits, underscore (`_` code 95), and dollar sign (`$` code 36). A word does not start with a digit. Note, that during parsing the longest sequence of characters that satisfies either a number or a word definition, but that appears in the list of reserved tokens, is considered to be a reserved token instead.

During the minification process words are renamed in a systematic fashion using the following algorithm:

1. Take a list of words that consist only of lowercase letters ordered first by their length, then lexicographically: `a`, `b`, ..., `z`, `aa`, `ab`, ..., excluding reserved tokens, since they are not considered to be words. This is the target word list.
2. Rename the first word encountered in the input token sequence to the first word in the target word list and all further occurrences of the same word in the input token sequence, too. Rename the second new word encountered in the input token sequence to the second word in the target word list, and so on.

The goal of the minification process is to convert the given source to the shortest possible line (counting spaces) that still parses to the same sequence of tokens with the correspondingly renamed words using these JS parsing rules.

Input

The first line of the input contains a single integer n $(0 \le n \le 40)$ — the number of reserved tokens.

The second line of the input contains the list of reserved tokens separated by spaces without repetitions in the list. Each reserved token is at least one and at most 20 characters long and contains only characters with ASCII codes from 33 (`!`) to 126 (`~`) inclusive, with exception of a hash character (`#` code 35).

The third line of the input contains a single integer m $(1 \leq m \leq 40)$ — the number of lines in the input source code.

Next m lines contain the input source, each source line is at most 80 characters long (counting leading and trailing spaces). Each line contains only characters with ASCII codes from 32 (space) to 126 (`~`) inclusive. The source code is valid and fully parses into a sequence of tokens.

Output

Write to the output a single line that is the result of the minification process on the input source code. The output source line shall parse to the same sequence of tokens as the input source with the correspondingly renamed words and shall contain the minimum possible number of spaces needed for that. If there are multiple ways to insert the minimum possible number of spaces into the output, use any way.

Sample Input 1

```
16
fun while return var { } ( ) , ; > = + ++ - --
9
fun fib(num) { # compute fibs
  var return_value = 1, prev = 0, temp;
  while (num > 0) {
    temp = return_value; return_value = return_value + prev;
    prev = temp;
    num--;
  }
  return return_value;
}
```

Sample Output 1

```
fun a(b){var c=1,d=0,e;while(b>0){e=c;c=c+d;d=e;b--;}return c;}
```

Sample Input 2

```
10
( ) + ++ : -> >> >>: b c)
2
($val1++ + +4 kb) >> :out
b-> + 10 >>: t # using >>:
```

Sample Output 2

```
(a+++ +4c )>> :d b->+10>>:e
```