A=B

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

Marisa has learned an interesting language called A=B. She finds that this language has the advantages of simple syntax, easy to learn and convenient to code.

Here is the user manual of A=B:

(Note that it may differ from the original game "A=B". So please read the statement carefully.)

- Instruction set

  A=B's instruction set includes:

  1. `string1=string2`
     Find the leftmost occurrence of `string1` in the string and replace it with `string2`.
  2. `string1=(return)string2`
     If `string1` is found, replace the entire string with `string2` and end the program immediately.

- Program structure

  An A=B program consists of several lines of instructions. Each line must include exactly one equal sign ('`=`').

  Following characters are reserved: '`=`', '`(`', '`)`'.

- Execution order

  1. Read the input string.
  2. Starting from the topmost line, find the first line that can be executed.
  3. If found, execute that line and go to step 2.
  4. If none is found, return the current string as output.

Pixiv ID: 68375845

Marisa once introduced A=B to Alice. However, "You called this a programming language? You can't even write a program that can check if string t is a substring of string s!" said Alice.

Now Marisa comes to you for help. She wants you to design an A=B program for this problem and show A=B's efficiency.

Your program needs to meet the following requirements given by Marisa:

- Read the input string (the input format is s`S`t. '`S`' is the separator. s and t are two **non-empty** strings consists of characters '`a`', '`b`', '`c`'.)
- If t is a substring of s, the program should return 1 as output, else return 0 as output.
- The character set that your program can use is {'`a`'∼'`z`', '`A`'∼'`Z`', '`0`'∼'`9`', '`=`', '`(`', '`)`'}. Remember, '`=`', '`(`', '`)`' are reserved character in A=B and you can't use it in `string1` or `string2`.
- In the previous instruction's format, the length of `string1` and `string2` should be at most 3.
- Suppose the length of the input string is L, the number of instruction's execution can't exceed $\max(2L^2, 50)$ and the length of string during execution can't exceed $2L + 10$.
- The number of instructions in your A=B program can't exceed 100.

Input

Input an integer Tid ($0 \le Tid \le 2 \times 10^9$). It is used for generating test sets and may be no use to you.

Output

Output your A=B program containing several lines of instructions.

The number of tests will not exceed 20. In each test, the checker will use Tid in the input file to generate several lines of input strings and their corresponding answers. Your A=B program is considered correct if and only if for each input string in all tests, your A=B program gives the correct output.

It's guaranteed that for each input string in all tests, the length L satisfies $3 \le L \le 1\,000$.

Examples

standard input	standard output
114514	514=(return)1 =514
1919810	S=Sakuya =(return)0

/*
Sample input: caba
Sample output: aabc
 	
Sample input: cbacab
Sample output: aabbcc
*/
	ba=ab
	ca=ac
	cb=bc

/*
Sample input: bababb
Sample output: b
 	
Sample input: aababbaa
Sample output: a
*/
	ba=ab
	ab=
	bb=b
	aa=a

/*
Sample input: abc
Sample output: true
 	
Sample input: cabc
Sample output: false
 	
Sample input: ca
Sample output: false
*/
	b=a
	c=a
	aaaa=(return)false
	aaa=(return)true
	=(return)false

/*
Sample input: 10111+111
Sample output: 11110
 	
Sample input: 101+10110
Sample output: 11011
*/
	A0=0A
	A1=1A
	B0=0B
	B1=1B
	0A=a
	0B=b
	1A=b
	1B=ca
	A=a
	B=b
	ac=b
	bc=ca
	0+=+A
	1+=+B
	+=
	0c=1
	1c=c0
	c=1
	a=0
	b=1

Note

The first and second samples show how you should submit your answer. You should read an integer and then output your A=B program. This integer will be used to generate a fixed data set. The third to sixth samples give problems and their corresponding programs to help you get familiar with the A=B language. All these programs **may not** satisfy the requirements given by Marisa.

- Sample 1

  If the input is "`abcaSab`", the process will be:

  `abcSab` $\xrightarrow{\textit{line 2}}$ `514abcSab` $\xrightarrow{\textit{line 1}}$ `1`

  (Note that an empty string can be found at the beginning of any string.)

  If the input is "`abcaSccc`", the process will be:

  `abcSccc` $\xrightarrow{\textit{line 2}}$ `514abcSccc` $\xrightarrow{\textit{line 1}}$ `1`

  So this program will get a WRONG ANSWER verdict.

- Sample 2

  In the first instruction, `string2` is equal to "`Sakuya`", which has a length more than 3.

  So this program will get a WRONG ANSWER verdict.

- Sample 3

  Input: A string consists of '`a`', '`b`', '`c`'.

  Output: Sort the input in alphabetical order.

  Test1: `caba` $\xrightarrow{\textit{line 1}}$ `caab` $\xrightarrow{\textit{line 2}}$ `acab` $\xrightarrow{\textit{line 2}}$ `aacb` $\xrightarrow{\textit{line 3}}$ `aabc`

  Test2: `cbacab` $\xrightarrow{\textit{line 1}}$ `cabcab` $\xrightarrow{\textit{line 2}}$ `acbcab` $\xrightarrow{\textit{line 2}}$ `acbacb` $\xrightarrow{\textit{line 1}}$ `acabcb` $\xrightarrow{\textit{line 2}}$

  `aacbcb` $\xrightarrow{\textit{line 3}}$ `aabccb` $\xrightarrow{\textit{line 3}}$ `aabcbc` $\xrightarrow{\textit{line 3}}$ `aabbcc`

- Sample 4

  Input: A string consists of '`a`', '`b`'. The number of '`a`' and '`b`' are different.

  Output: The most common letter.

  Test1: `bababb` $\xrightarrow{\textit{line 1}}$ `abbabb` $\xrightarrow{\textit{line 1}}$ `ababbb` $\xrightarrow{\textit{line 1}}$ `aabbbb` $\xrightarrow{\textit{line 2}}$ `abbb` $\xrightarrow{\textit{line 2}}$ `bb` $\xrightarrow{\textit{line 3}}$ `b`

  Test2: `aababbaa` $\xrightarrow{\text{several times of line 1}}$ `aaaaabbb` $\xrightarrow{\text{several times of line 2}}$ `aa` $\xrightarrow{\textit{line 4}}$ `a`

- Sample 5

  Input: A string consists of '`a`', '`b`', '`c`'.

  Output: Return true if the input contains exactly three letters. Otherwise, return false.

- Sample 6

  Input: Two binary numbers, separated by a '`+`'.

  Output: The sum of these two numbers. It is also in binary form.

  This program performs addition from low bit to high bit. For each step, the program first checks if the first number is empty (Line 15). Then consumes the lowest bit of the first number (Line 13 or 14) and then writes it at the back of the second number (Line 1 - 4). And then performs 1-bit addition (Line 5 - 8). Here, '`a`' and '`b`' represent '`0`' and '`1`', '`c`' represents the carry character. It may happen that in some steps the first number is non-empty but the second one is empty, so Line 9 - 10 are used to solve this case. Line 11 - 12 are to solve the carry in the previous step. At last, we transform '`a`', '`b`', '`c`' to their real number (Line 18 - 20) and solve the case that the second number is non-empty (Line 16 - 17).