Czech ACM Student Chapter
CTU in Prague, Dept. of Computer Science and Engineering
Charles University in Prague, MFF
VŠB TU Ostrava, FEI, Dept. of Computer Science
SUT Bratislava, FEI, Dept. of Computer Science and Engineering

CTU Open Contest 2003

Base Numbers

`numbers.p`, `numbers.c`, `numbers.C`

Any integer number can be written as a sequence of digits. The most popular system is decimal, which uses ten digits (its *base* is 10). Besides, other systems with different bases can be used. For instance, the binary system (with the base of 2) is often used in relation to computers.

Generally, if we have a non-negative integer number n expressed as a sequence of digits

$$d_k d_{k-1} \dots d_2 d_1 d_0$$

in the system with the base r ($r > 1$), then the value of the number is

$$n = d_k \cdot r^k + d_{k-1} \cdot r^{k-1} + ... + d_2 \cdot r^2 + d_1 \cdot r + d_0.$$

Under any circumstances, each digit must be smaller than the base, $0 \leq d_i < r$ for every i, $0 \leq i \leq k$.

Some problems arise when we use bases greater than 10, because then we need more than 10 popular digits. A common solution is to use letters for "digits" greater than 9. Although this solution pushes limits a little further, it does not avoid the problem itself — the set of letters is still very limited. Thus, higher bases could not be used.

Another solution is to write every digit separately as a number in the decimal system. For example, the hexadecimal number 1A8D (in usual notation) can be written as "$(1{-}10{-}8{-}13)_{16}$". Please note that the number is always followed by the base value (even in the decimal system) to avoid misinterpretations. This number format is called a *decimal-encoded notation* in this text.

A number in the decimal-encoded notation is considered valid only if it contains no unnecessary zeros, i.e., additional leading zeros cannot appear in the base value or in any of the digits. Moreover, all digits must always be smaller than the base value. Thus, "$(1-0-0-0)_7$", "$(4{-}7689)_{7690}$", and "$(0)_{16}$" are valid decimal-encoded numbers, on the other hand, "$(0{-}3{-}6)_8$", "$(1-02-3)_6$", "$(3-2-1)_{07}$", and "$(9)_6$" are invalid.

Due to some implementation reasons, ACM stores some data in valid decimal-encoded formats with various bases. Due to a software bug, a file containing a set of such numbers was damaged. All decimal digits has been preserved in the right order, but all of the dashes and parentheses were lost. Thus, each representation of a number was transformed to a *code* consisting of decimal digits only. Unfortunately, these codes are very ambiguous, most of them could represent many different numbers. For instance, "1234" could stand for "$(1-2-3)_4$", "$(12)_{34}$", "$(1-2)_{34}$", or "$(1)_{234}$". Your task is to determine the number of different representations that a code could stand for.

Input Specification

The input file consists of several lines. Each line contains one code, i.e., the string consisting of decimal digits. The maximum length of any code is 35, the minimum length is 1. The last line of the input file contains a single hash character ("`#`").

Output Specification

For each code, output a single line of text. If there is no valid representation of a number resulting in the given code, print a single line containing the text "`The code` CCC `is invalid.`". Otherwise, print the text "`The code` CCC `can represent` X `numbers.`". Replace CCC with the given code. Replace X with the total number of different representations in the decimal-encoded notation that would result in the given code when dashes and parentheses are removed. Note that the representations of numbers are considered different even if they result in the same value.

Sample Input

```
1234
102
201
123456
#
```

Output for Sample Input

```
The code 1234 can represent 4 numbers.
The code 102 can represent 1 numbers.
The code 201 is invalid.
The code 123456 can represent 13 numbers.
```