Problem A
Convert a Boolean Formula
Time limit: 2 seconds

In this problem, we consider two algebraic structures: Boolean algebra and finite field GF(2).

In mathematical logic, Boolean algebra is the algebra in which the values of the variables are the truth values **false** or **true**, usually represented by 0 and 1, respectively. There are 3 operations: $\vee$, $\wedge$, and $\neg$.

$$0 \vee 0 = 0, \quad 0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1.$$

$$0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0, \quad 1 \wedge 1 = 1.$$

$$\neg 0 = 1, \quad \neg 1 = 0.$$

Boolean formulas are formulas containing Boolean variables with operators $\vee$, $\wedge$, and $\neg$. For simplicity, in this problem we consider only *simple* Boolean formulas of the form

$$x_1 \vee x_2 \vee \cdots \vee x_n,$$

where each x_i is a variable or the negation of a variable. Each variable is different, and no variable and its negation will both appear in the formula.

Since computer keyboards may not have $\vee$, $\wedge$, and $\neg$ keys, we will replace $\vee$ by "`+`", $\wedge$ by "`x`" and $\neg$ by appending a `'` after the variable.

In finite field GF(2), there are only two elements 0 and 1. The two operations in GF(2) are $+$ and $\times$.

$$0 + 0 = 1 + 1 = 0, \quad 0 + 1 = 1 + 0 = 1.$$

$$0 \times 0 = 0 \times 1 = 1 \times 0 = 0, \quad 1 \times 1 = 1.$$

If you are not familiar with GF(2), you can think of GF(2) as integers. After each addition (or multiplication) of 2 integers, the final result is the remainder of the sum (or the product) divided by 2.

A polynomial P in GF(2) is a polynomial with coefficient 0 and 1. The addition and multiplication of 2 polynomials are the same as ordinary polynomials, except that each final coefficient is the remainder of the coefficient divided by 2. For example,

$$(a + b)^2 = a^2 + 2ab + b^2 = a^2 + b^2 = a + b.$$

Note that $x^2 = x$ in GF(2).

Let B be a Boolean formula and P be a GF(2) polynomial with the same set of variables. B and P are equivalent, if for every assignment of the valuables, B and P always have the same value. The following table gives a useful set of equivalent B's and P's.

B	P
x	x
$x \vee y$	$x + y + xy$
$x \wedge y$	xy
x'	$1 + x$

Based on the above table, write a program to convert a simple Boolean formula into its equivalent polynomial in GF(2).

Input File Format

There are more than one test cases in the input file. Each test case contains a simple Boolean formula of n, $1 \leq n \leq 8$, valuables in $\{a, b, c, d, e, f, g, h\}$. For simplicity, if a Boolean formula contains $n < 8$ variables, the first n variables in the list a, b, c, d, e, f, g, h will be used. For example, if $n = 3$, then the variables are a, b, c. Furthermore, the variables will appear in increasing order. For example, $a + b' + c$, not $b' + a + c$ or $c + a + b'$. There will be no spaces in each input line. The last line of the input file contains a single 0. After scanning this line, your program should stop.

Output Format

For each Boolean formula, print its equivalent polynomial in GF(2). The polynomial should be simplified and expressed as the sum of products. For example, express a polynomial as $a + ab + ac + bc$, not $(a + b)(a + c)$. If the polynomial is a constant, print the constant. Otherwise, follow the following rules in printing out the polynomial.

1. If the coefficient of a term is 0, do not print this term.
2. For each term with coefficient 1, print only the product of the valuables. For example, print `ab`, not `1ab`.
3. The product of the valuables must be printed in alphabetical order. For example, print `abc`, not `cab`.
4. If the polynomial contains more than 1 term:
   (a) Print the terms with fewer number of variables first. For example, print $1 + a + ab$, not $a + ab + 1$.
   (b) For terms with the same number of variables, print them in dictionary order. For example, print $ab + bc$, not $bc + ab$.
   (c) Insert the string " `+` " (a space followed by "`+`" followed by another space) between two adjacent terms.

Sample Input

```
a+b'
a+b+c
a'+b+c
0
```

Output for the Sample Input

```
1 + b + ab
a + b + c + ab + ac + bc + abc
1 + a + ab + ac + abc
```