Given an expression and some rules, a corresponding parse-tree can be drawn. If the leaves of the trees are traversed from left to right, then the expression from which the tree was generated can be found. The leaves contain the terminal symbols, and the internal nodes contain the non-terminal symbols. For this problem, terminals consist of {`i`, `+`, `-`, `*`, `/`, `(`, `)`} and the non-terminals are {`E`, `T`, `F`}. The rules for this specific problem are:

- E -¿ E + T
- E -¿ E - T
- E -¿ T
- T -¿ T * F
- T -¿ T / F
- T -¿ F
- F -¿ i
- F -¿ (E)

This is NOT a parse tree

Input

Input consists of several test cases. Each case is in a line by itself. Each line contains a non-empty expression which doesnt have any blank space in it. There will be no invalid and ambiguous input, i.e., there will always be a unique parse-tree for the input. Input is terminated by EOF.

Output

For each test case, print the parse-tree. The leftmost leaf should be at column 1, the leaf next to that should be at column 4, the next leaf should be at column 7, and so on. Each non-terminal should have a '|' on the same column in the following line and a non-terminal in the next line on the same column. A string of '=' should spread on both sides of '|' just enough to cover its immediate children. There should be no blank line within a parse-tree.

Print a single blank line between test cases. No line should have blank spaces at the end.

You can safely assume that a single parse tree wont need more than 200 lines to be drawn and no line will need more than 200 characters.

Look at the sample outputs for clearer idea.

Sample Input

```
i+i*i
i+i*(i+i)
```

Sample Output

```
   E
===|======
E  +     T
|      ===|===
T      T  *  F
|      |     |
F      F     i
|      |
i      i


   E
===|======
E  +     T
|      ===|=========
T      T  *        F
|      |      ======|======
F      F      (     E     )
|      |         ===|===
i      i         E  +  T
                 |     |
                 T     F
                 |     |
                 F     i
                 |
                 i
```