J. (Jiandan) Mua(I) - Lexical Analyzer

In this problem-series, you're to implement a subset of the Lua language (version 5.1), called mini-lua (mua). This is one of Rujia Liu's experimental languages, mainly for implementing algorithms, not real-world programs.

This is the first problem in the series, which requires you to write a lexical analyzer (lexer), i.e. split the input program into a stream of **tokens** (defined below).

There are six kinds of tokens in mini-lua:

RESERVED: The reserved words, listed below. Note that mini-lua is case-sensitive, so AND is not a reserved word. Note that not all of them are actually used in mini-lua, but we want any valid mini-lua program to be valid in Lua.

```
and       break     do        else      elseif
end       false     for       function  if
in        local     nil       not       or
repeat    return    then      true      until     while
```

NUMBER: There are two kinds of numbers:

- *Integers:* Decimal integers consist of one or more digits (0-9), and hexadecimal integers consists of a prefix (0x or 0X), then followed by one or more hexadecimal digits(0-9, a-f, case-insensitive). Note that leading zeros are simply ignored (e.g. 0123 is the same as 123).
- *Floating-point numbers:* Always in decimal notation, e.g. 1.23. Scientific notation may be used by adding e or E followed by a decimal exponent (e.g. 1.23e2, which has the value 123.0). Either a decimal point or an exponent is required, but the integer part can be omitted (like .2e3). If the integer part is omitted, the decimal point and at least one digit after the decimal point are required (so .e2 is illegal). Hexadecimal floating-point numbers are not supported. Note that the exponent can be prefixed by "+" or "-", e.g. 1e+10 and 4e-3 are legal.

Note that "negative numbers" consists of two tokens: the unary "minus" operator, and the absolute number. For example, -34 has two tokens. Similarly, +7e8 also has two tokens.

STRING: A string enclosed by `""` or `''`. Only four escaped characters are supported: `\" \' \\ \n`. A string cannot contain a real newline character.

SYMBOL: Symbols that have special meanings, listed below:

```
+       -       *       /       %       ^       #
==      ~=      <=      >=      <       >       =
(       )       {       }       [       ]
;       :       ,       .       ..      ...
```

NAME: an identifier starting with a letter and followed by letters, digits and underscores. Note that reserved words cannot be used as names.

EOL: End of a physical line.

COMMENT: Start with `--`, and may not cross a line (the newline character following the comment is part of it. It is a separate **EOL** token). Note that white spaces output any string constants are always ignored, so `1+1` and `1 +1` have no differences for the lexer.

The lexer should be greedy. i.e. if there are more than one way to split the input into tokens, always maximize the length of the first token, then the second one, etc. For example, the only way to tokenize "`abc123<=x`" is "`abc123`", "`<=`" and "`x`".

In order to simplify the problem, a mini-lua program can only contain ASCII characters, regardless of your current locale (if you couldn't understand this, just ignore it).

Input

A valid mini-lua program.

Output

Non-comment tokens, one for each line. formatted as "[TYPE] token", where "token" is printed as in the input. If type is EOL, print an empty text.

Sample Input

```
print("Hello".." ".."--\"World\"--")
x=-3+4
this_1s_a_variable = 0xabcF-.012e-56+7.e8+9e+10--8
if 1 then
  y=x
end
print (y)
```

Output for Sample Input

```
[NAME] print
[SYMBOL] (
[STRING] "Hello"
[SYMBOL] ..
[STRING] " "
[SYMBOL] ..
[STRING] "--\"World\"--"
[SYMBOL] )
[EOL]
[NAME] x
[SYMBOL] =
[SYMBOL] -
[NUMBER] 3
[SYMBOL] +
[NUMBER] 4
[EOL]
[NAME] this_1s_a_variable
[SYMBOL] =
[NUMBER] 0xabcF
[SYMBOL] -
[NUMBER] .012e-56
[SYMBOL] +
[NUMBER] 7.e8
[SYMBOL] +
[NUMBER] 9e+10
[EOL]
[RESERVED] if
[NUMBER] 1
[RESERVED] then
[EOL]
[NAME] y
[SYMBOL] =
[NAME] x
[EOL]
[RESERVED] end
[EOL]
[NAME] print
[SYMBOL] (
[NAME] y
[SYMBOL] )
[EOL]
```