Fun With Fortran IV (prob12)

The Problem

As everyone knows (or should know), Fortran is the ancestor of all high-level programming languages. The most significant version was Fortran IV, which is the version we shall consider here. Fortran had a number of interesting “quirks” when compared to modern languages. For example, spaces were completely insignificant. Thus, the statements “`GOTO 23`”, “`GO TO 23`” and “`G O T O 2 3`” were all equivalent (and valid). Also, there were no reserved keywords, so “`IF`,” “`DO`,” etc. could be used as variable names (or as a prefix to a variable name), and variables did not have to be declared. These three aspects of the language combined to cause a very famous bug: a `DO` statement that should have been

```
      DO 97 I = 1,5
```

was actually coded as

```
      DO 97 I = 1+5
```

which was recognized as a valid assignment statement (assigning the value 6 to the variable `DO97I`) instead of looping 5 times, and an experimental rocket blew up.

Your job is to write a program that parses Fortran IV statements. The Fortran subset that you are to work with is described by the following grammar:

```
⟨statement⟩                ::=⟨dimensionstatement⟩|⟨assignmentstatement⟩
                              |⟨ifstatement⟩|⟨dostatement⟩|⟨gotostatement⟩
                              |⟨endstatement⟩
⟨dimensionstatement⟩       ::= ‘DIMENSION’ ⟨dimensionlist⟩
⟨dimensionlist⟩            ::=⟨dimensionitem⟩
                              |⟨dimensionitem⟩‘,’⟨dimensionlist⟩
⟨dimensionitem⟩            ::=⟨identifier⟩‘(’⟨integerconstant⟩‘)’
⟨assignmentstatement⟩      ::=⟨variable⟩‘=’⟨expression⟩
⟨variable⟩                 ::=⟨identifier⟩|⟨identifier⟩‘(’⟨expression⟩‘)’
⟨expression⟩               ::=⟨integerconstant⟩|⟨variable⟩| ‘(’⟨expression⟩‘)’
                              |⟨expression⟩ ⟨operator⟩ ⟨expression⟩
⟨operator⟩                 ::= ‘+’ | ‘-’ | ‘*’ | ‘/’ | ‘**’
⟨ifstatement⟩              ::= ‘IF’ ‘(’⟨expression⟩‘)’⟨integerconstant⟩‘,’
                                  ⟨integerconstant⟩‘,’⟨integerconstant⟩
⟨dostatement⟩              ::= ‘DO’⟨integerconstant⟩ ⟨identifier⟩‘=’⟨dolist⟩
⟨dolist⟩                   ::=⟨doitem⟩‘,’⟨doitem⟩
                              |⟨doitem⟩‘,’⟨doitem⟩‘,’⟨doitem⟩
⟨doitem⟩                   ::=⟨integerconstant⟩|⟨identifier⟩
⟨gotostatement⟩            ::= ‘GOTO’⟨integerconstant⟩
⟨endstatement⟩             ::= ‘END’
⟨identifier⟩               ::=⟨letter⟩|⟨identifier⟩ ⟨letter⟩|⟨identifier⟩ ⟨digit⟩
⟨integerconstant⟩          ::=⟨digit⟩|⟨integerconstant⟩ ⟨digit⟩
⟨letter⟩                   ::= ‘A’|‘B’|‘C’|‘D’|‘E’|‘F’|‘G’|‘H’|‘I’|‘J’|‘K’|‘L’|‘M’
                              |‘N’|‘O’|‘P’|‘Q’|‘R’|‘S’|‘T’|‘U’|‘V’|‘W’|‘X’|‘Y’|‘Z’
⟨digit⟩                    ::= ‘0’|‘1’|‘2’|‘3’|‘4’|‘5’|‘6’|‘7’|‘8’|‘9’
```

Input

Each input line is a non-empty string representing a potential Fortran statement. No line will be longer than 72 characters, and will consist only of characters in the standard Fortran characters set: uppercase letters, digits, spaces, and the “special” characters

```
=+-*/(),.$
```

Your program should exit after processing the `END` statement.

Output

For each line of input (including the `END` statement), there will be one line of output, stating which kind of Fortran statement was recognized: a ‘`#`’ sign followed by the input line count; a space; one of the words “`ASSIGNMENT`”, “`DIMENSION`”, “`DO`”, “`END`”, “`GOTO`”, “`IF`” or “`INVALID`”; a space; and the word “`STATEMENT`”.

Sample Input

```
A = 9
X(3) = (5 + 7) ** 9
DIMENSION A(3), B(5), C(100)
DO 97 I = 1,5
DO 97 I = 1+5
GO TO 23
IF(X – 3)21,32,33
IF(X - 3) = 5 000 000
GOTO SOMEWHERE
DIMENSION X(5 + 3)
END
```

Sample Output

```
#1: ASSIGNMENT STATEMENT
#2: ASSIGNMENT STATEMENT
#3: DIMENSION STATEMENT
#4: DO STATEMENT
#5: ASSIGNMENT STATEMENT
#6: GOTO STATEMENT
#7: IF STATEMENT
#8: ASSIGNMENT STATEMENT
#9: INVALID STATEMENT
#10: INVALID STATEMENT
#11: END STATEMENT
```