Symbolic Logic Mechanization

Marvin, the robot with a brain the size of a planet, followed some . . . markedly less successful robots as the product line developed. One such was Monroe, the robot — except, to help him recognize his name, he was referred to as Moe. He is sufficiently mentally challenged that he needs external assistance to handle symbolic logic.

Polish notation is the prefix symbolic logic notation developed by Jan Łukasiewicz (1929). [Hence postfix expressions are referred to as being in Reverse Polish Notation — RPN.] The notation developed by Łukasiewicz (referred to as PN below) uses upper-case letters for the logic operators and lower-case letters for logic variables (which can only be **true** or **false**). Since prefix notation is self-grouping, there is no need for precedence, associativity, or parentheses, unlike infix notation. In the following table the PN operator is shown, followed by its operation. Operators not having exactly equivalent C/C++/Java operators are shown in the truth table (using 1 for **true** and 0 for **false**). [The operator J is not found in Łukasiewicz' original work but is included from A.N.Prior's treatment.]

PN Operator	Operation
Cpq	conditional
Np	not
Kpq	and
Apq	(inclusive) or
Dpq	nand
Epq	equivalence
Jpq	exclusive or

Truth Tables				
p	q	Cpq	Dpq	Epq
0	0	1	1	1
0	1	1	1	0
1	0	0	1	0
1	1	1	0	1

For every combination of PN operators and variables, an expression is a "well-formed formula" (WFF) if and only if it is a variable or it is a PN operator followed by the requisite number of operands (WFF instances). A combination of symbols will fail to be a "well-formed formula" if it is composed of a WFF followed by extraneous text, it uses an unrecognized character [upper-case character not in the above table or a non-alphabetic character], or it has insufficient operands for its operators. For invalid expressions, report the ***first*** error discovered in a left-to-right scan of the expression. For instance, immediately report an error on an invalid character. If a valid WFF is followed by extraneous text, report that as the error, even if the extraneous text has an invalid character.

In addition, every WFF can be categorized as a tautology (true for all possible variable values), a contradiction (false for all possible variable values), or a contingent expression (true for some variable values, false for other variable values).

The simplest contingent expression is simply "p", true when p is true, false when p is false. One very simple contradiction is "KpNp", both p and not-p are true. Similarly, one very simple

tautology is "ApNp", either p is true or not-p is true. For a more complex tautology, one expression of De Morgan's Law is "EDpqANpNq".

Input

Your program is to accept lines until it receives an empty character string. Each line will contain only alphanumeric characters (no spaces or punctuation) that are to be parsed as potential "WFFs". Each line will contain fewer than 256 characters and will use at most 10 variables. There will be at most 32 non-blank lines before the terminating blank line.

Output

For each line read in, echo it back, followed by its correctness as a WFF, followed (if a WFF) with its category (tautology, contradiction, or contingent). In processing an input line, immediately terminate and report the line as not a WFF if you encounter an unrecognized operator (even though it may fail to be well-formed in another way as well). If it has extraneous text following the WFF, report it as incorrect. If it has insufficient operands, report that. Use ***exactly*** the format shown in the **Sample Output** below.

Sample Input

```
q
Cp
Cpq
A01
Cpqr
ANpp
KNpp
Qad
CKNppq
JDpqANpNq
CDpwANpNq
EDpqANpNq
KCDpqANpNqCANpNqDpq
[this is an empty line]
```

Sample Output

```
q is valid: contingent
Cp is invalid: insufficient operands
Cpq is valid: contingent
A01 is invalid: invalid character
Cpqr is invalid: extraneous text
ANpp is valid: tautology
KNpp is valid: contradiction
Qad is invalid: invalid character
CKNppq is valid: tautology
JDpqANpNq is valid: contradiction
CDpwANpNq is valid: contingent
EDpqANpNq is valid: tautology
KCDpqANpNqCANpNqDpq is valid: tautology
```