Instrutor: **Ramtin Khosravi** Deadline: Sat. 6 Bahman 97, 23:55 PM

Advanced Coffee Makers

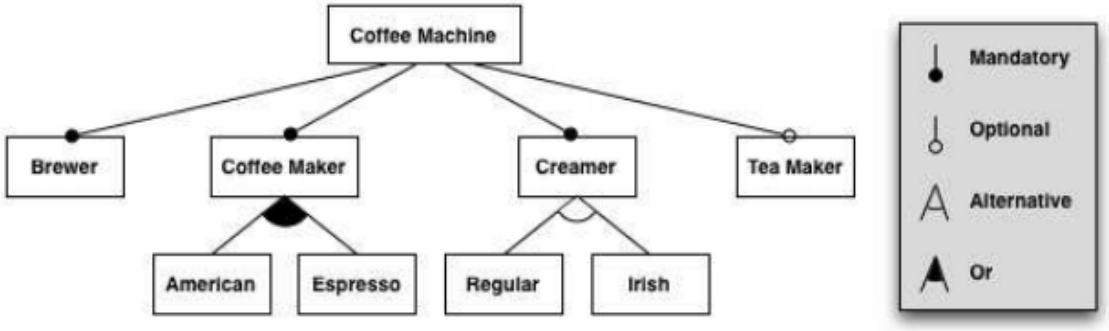
Ref: *ACM ICPC Regional 2011, Tehran, Asia Region*

Advanced Coffee Makers (ACM) Company holds a large portion of the coffee vending machine market. Facing with numerous variations of coffee machines that the market demands for, ACM tries to develop a product line, so that each time a customer requests for a specific configuration of coffee machines, it can easily produce and deliver the product in a shorter time and with lower cost. This way, if a customer asks for example, for a coffee machine that serves both coffee and tea, but does not offer options for adding cream to coffee, ACM can produce the corresponding machine including coffee and tea features, but not having creamer features.

To this end, ACM has prepared a feature model of the coffee machine products. The feature model contains a hierarchy of the features appearing in a coffee machine, organized as a rooted tree. The root always corresponds to the whole coffee machine product, while other nodes represent the features of a coffee machine. Relationships between a parent feature and its child features (or sub-features) are categorized as:

- **Mandatory**: child feature is required.
- **Optional**: child feature is optional.
- **Or**: at least one of the sub-features must be selected.
- **Alternative (xor)**: exactly one of the sub-features must be selected.

For example, the following figure represents a sample feature model for coffee machines:



In the above figure, every coffee machine must have Brewer, Coffee Maker, and Creamer component. However, it may or may not have a Tea Maker component. It can provide American or Espresso types of coffee, or both. But, the Creamer can be either Regular or Irish.

Having a feature model, a customer may ask for a specific configuration, i.e., a set of features. For example, Coffee Machine, Brewer, Coffee Maker, American, Creamer, Regular form a valid configuration. Note that mandatory features must be present in a valid configuration if their parents are present. In contrast, the configuration Coffee Machine, Brewer, Coffee Maker, Creamer, Regular, Tea Maker is not valid, since no sub-feature of Coffee Maker is selected. Note that the whole Coffee Machine (the root) must be included in all valid configurations.

A valid configuration must include the root feature. A mandatory feature must be included in the configuration if its parent is included. From a set of features having 'or' (resp. 'xor') relationship with an included parent feature, at least (resp. exactly) one must be present. Also, if a child is included in a valid configuration, its parent must also be included. You may assume that each feature appears only once in the feature model tree.

Your program must input a feature model, together with a set of configurations, and for each configuration determines whether the configuration is valid.

Input

The input consists of a number of test cases. Each test case has two parts. The first part represents the feature model and the second part lists the configurations to be validated. The feature model is described as a set of lines of the following forms, each describing a (non-leaf) feature F that is composed from a set of $n > 1$ subfeatures:

- F = F_1 + F_2 + $\cdots$ + F_n, where F_i is either a feature name, or a question mark followed by a feature name.
- F = F_1 | F_2 | $\cdots$ | F_n, where F_i is a feature name
- F = F_1 ^ F_2 ^ $\cdots$ ^ F_n, where F_i is a feature name

Each line defines the sub-features of a feature F. In the first case, optional features are preceded by a question mark (see the sample input). It is assumed that

the first line defines the whole coffee machine (that must be present in all valid configurations). The feature names are sequences of upper-case and lower-case letters (with no blanks in between). However, there may be arbitrary number of blank characters in the beginning or at the end of the lines, or around the symbols `=`, `+`(Mandatory), `?`(Optional), |(Or), and `^`(Alternative). You may safely assume that each feature is defined once, and the feature model forms a tree. As said before, each feature appears only once in the feature tree (e.g. we cannot have `A = B + C` and `B = C + D`). Also assume that the total number of features is at most 1000 (and of course at least one).

After the last line of a feature model description there is a line containing a single `#` character, after which come the lines describing the configurations that you must validate. Each configuration comes in a separate line of the form `{`F_1`,`F_2`,` $\cdots$ `,`F_n`}` where F_i is a feature name and $n > 0$. There may be blank characters anywhere in the line except in the feature names. You may assume that each F_i has appeared in the feature model description and there are no duplicate feature names in a configuration. The last line of the test case is a single line containing `##`. After the last test case, there is a line containing `###` which should not be processed.

Output

For each configuration in each test case, print one line in the output containing either Valid or Invalid words, indicating whether the corresponding configuration is valid. Print a line containing `+++` (three plus sign) after each test case.

Sample Input and Output

Sample Input	Sample Output
CoffeeMachine = Brewer + CoffeeMaker + Creamer + ?TeaMaker CoffeeMaker = American | Espresso Creamer = Regular ^ Irish # {CoffeeMachine, Brewer, CoffeeMaker, American, Creamer, Regular} {CoffeeMachine, Brewer, CoffeeMaker, Creamer, Regular, TeaMaker} ## A=?B+?C B=D|E # {A,B,C,D,E} {A,D} ## ###	Valid Invalid +++ Valid Invalid +++