10788 Parenthesizing Palindromes

Validating an expression for parenthesis balance, like many other routine tasks, is a trivial matter for a computer program. But for human eyes, this simple task can actually be quite painful. Just to get a feel, see how straining it can be to check if the following expression is correctly parenthesized:

(( ) (( )) (( ) ( )) (( ) (( )) ( )) (( )) ( )) ( ))

To save our precious eyes from the trouble, we find many of the editors and spreadsheet applications using colors or emphasized fonts to show the nesting of the parenthesis structure. For example, the parenthesis structure shown here uses parenthesis of different sized fonts to help us understand the nesting levels:

$$\Big(\Big(( ) ( ) \big(( ) \big) \Big) \big(\big(( ) \big) \big) \Big)$$

But the solution that we have for those applications are of little use for our age old console applications. Neither do we have colors, nor do we have different fonts. Here is our naïve suggestion: We may not have colors but we have different letters. And if we run out of letters, we can always use palindromes! Strange as it may seem at the first sight, this is nothing new to us. We are frequently using `/*` and `*/` to bracket (not to parenthesize) our comments in the C/C++ or Java code. For this problem, we are only taking it one step higher. According to our suggestion, we could say "abbaddcc" is a correctly nested parenthesis structure since it can be interpreted as "(ab ba) (d d) (c c)".

However, it takes little thought for one to realize that parenthesis structures represented in this way can be ambiguous. Expressions like aaabba can leave us wondering whether to think of it as "(a (a a) (b b) a)" or "(a a) (ab ba)"; whereas the previous expression "abbaddcc" had only one interpretation.

For this problem, you are required to write a computer program that checks if a parenthesis structure defined in terms of palindromes is valid and has a unique interpretation.

Input

Input consists of several test cases. The first line of the input file contains a single integer, T ($1 \leq T \leq 20$) indicating the number of test cases. Then T test cases follow. For each of the test cases, you will be given a string S containing lower case letters only. You can assume that none of the strings would be more than 100 characters long.

Output

The output for each test case starts with the test case number in the format '`Case` x`:`' where $1 \leq x \leq T$. Then any of the following three strings would follow:

- '`Invalid`' – the parenthesizing structure is unbalanced
- '`Valid, Unique`' – the parenthesizing structure is balanced and has unique interpretation
- '`Valid, Multiple`' – the parenthesizing structure is balanced but has multiple interpretation

Sample Input

```
3
aaabba
aabb
bbababba
```

Sample Output

```
Case 1: Valid, Multiple
Case 2: Valid, Unique
Case 3: Invalid
```