This problem requires you to write a program to syntactically validate some simple text written using HTML, the HyperText Markup Language in which all documents available on the World Wide Web are written. We'll not consider the semantics, or meaning, of these documents, and will only consider simplified syntactical rules. In these documents you'll find ordinary text (with arbitrary line lengths) interspersed with markup tags. The markup tags we consider will always occur in pairs. An example will illustrate this point:

```
This is some ordinary text. <ATAG>  Here's some additional
text. <BOLD>  This will be boldface. </BOLD>  Still more
</ATAG>
```

There are two pair of markup tags in the example. The meaning of `ATAG` and `BOLD` is unimportant to us in this problem, but typically a markup tag requests particular treatment of the text to which it applies. The markup tags are easily identified because they always appear in angle brackets (that is, a less than symbol and a greater than symbol). The tags we'll consider will always be written as a sequence of no more than 10 upper-case alphabetic characters. The end of the document region affected by the tag is indicated by a tag with the same name preceded by a forward slash, '`/`' As illustrated, the tagged regions may encompass more than one line of text. Also as shown in the example, the HTML tags must nest properly, just like `BEGIN...END` pairs in Pascal, or '`{`' and '`}`' in C/C++.

Input

The input will consist of a number of test cases. Each will begin with a separate line containing an integer specifying the number of lines of text in the test case, NL. NL will never exceed 32767. The end of input is marked by a value of zero (`0`) for NL.

Following the line specifying NL in each test case there will be NL lines of text which are to be checked for syntactic conformance. Remember that there is no maximum line length limit.

Output

For each test case, output the test case number (they are numbered sequentially starting with 1). If the text is in conformance with the rules specified above, then output the word '`OK`'. If the text is erroneous, then output one of the following messages for the first error only:

```
line #: bad character in tag name
line #: too many/few characters in tag name
line #: expected </xxxxxxxxxx>
line #: no matching begin tag.
```

The '`#`' in these messages is to be replaced by the line number of the test case when the offending tag was detected. Examples of when these messages are displayed are found in the example input and expected output immediately following.

If an error is detected, after producing the appropriate error message your program must skip any remaining lines in the erroneous test case to reach the beginning of the next test case.

Sample Input

```
6
This is some ordinary text.
<BEGIN> This is included in the BEGIN tag </BEGIN>
        <START>    Here's some stuff
and so is this
    more stuff. </START>
2
   This has a null tag <>
   And an extra line after the error
5
   This has some good stuff <OKAY> and some bad stuff later on.
   <GOOD> All is still okay, but later on we'll have an error.
   </GOOD> We're still in the pink! <THISISTOOLONG>
   This line will be skipped.
   As will this one.
1
   This is an interesting error: <ERROR
2
   This one is okay                                <IN> </IN>

1
   Mismatch <START> </STOP>
1
    Missing start symbol:   <OK></OK></NOTOK>   more garbage...
1
<ELEVENChars>Both too long and invalid letter</ELEVENChars>
1
<ELEVENCHARs>Both too long and invalid letter</ELEVENCHARs>
1
<ELEVENCHARS!>Both too long and invalid letter</ELEVENCHARS!>
0
```

Sample Output

```
Test Case 1
OK
Test Case 2
line 1: too many/few characters in tag name.
Test Case 3
line 3: too many/few characters in tag name.
Test Case 4
line 1: bad character in tag name.
Test Case 5
OK
Test Case 6
line 1: expected </START>
Test Case 7
line 1: no matching begin tag.
Test Case 8
line 1: bad character in tag name.
Test Case 9
line 1: bad character in tag name.
Test Case 10
line 1: too many/few characters in tag name.
```