Problem 3—Presentation Error

One of the main burdens of the judges (and contestants) of the ACM Programming Contest is not to decide whether a submitted is incorrect—that's generally pretty obvious—but how to classify the error. In the past, some of the classifications were "Failed Test Case," "Wrong Answer," "Wrong Output Format," and "Too much/Too Little Output." The interpretation of these messages depended on which judge was doing the interpreting. What's the difference between "Wrong Answer," and "Failed Test Case" anyway? Does it have fail every test case to be "wrong?"

While Dr. Poe wishes the ACM would follow his lead and simply judge a program "right" or "wrong," the ACM has improved its policy from the early days: "Accepted," "Wrong Answer," and "Presentation Error" are the relevant responses for this problem. (There are other messages, involving the program crashing and infinite loops, but they are not our present concern.)

To eliminate any judge's bias in deciding between a "Wrong Answer" and a "Presentation Error" imagine that there is a exact procedure to decide which of the three above responses should be given.

In the description of the rules, we distinguish between CorrectOut and SubmitOut, as the correct output of the program, and the output submitted by the contestant. CorrectOut contains parts which are considered essential in the output of a correct program. These essentials are placed between square brackets. These brackets are not part of the output, they should not appear in SubmitOut. The algorithm to decide between "Accepted," "Wrong Answer," and "Presentation Error" is as follows:

1. From each line in both outputs, all trailing spaces (spaces at the end of a line) should be removed. After that, all trailing empty lines (empty lines at the end of an output) should be removed.

2. If after step 1, CorrectOut and SubmitOut are identical, "Accepted" should be printed. If not, continue.

3. All letters in both outputs are changed to uppercase (including those in square brackets).

4. We name the essentials E(1) through E(n) in sequential order.

5. If each of the strings E(1) through E(n) occurs as a string in SubmitOut, and E(i) comes after (without overlapping) E(i-1) for all $2 \leq i \leq n$, then the algorithm prints "Presentation Error". Otherwise,

6. "Wrong Answer" is printed.

You are to write a program that implements the above algorithm.

INPUT SPECIFICATION

The first line contains the number of test cases, followed by **<EOLN>**. Each test case begins with a line containing two integers representing the number of lines of CorrectOut and the number of lines of SubmitOut. The integers will be separated by one space and will be terminated by **<EOLN>**. Neither output will be longer than 10 lines; no line will be longer than 80 characters discounting the **<EOLN>**. Essentials are never empty, never cross line boundaries, never begin or end with a space, and are never nested. Square brackets will never appear in CorrectOut in any connotation other than the indications of essentials. Square brackets will never appear in SubmitOut at all. There will be no more than 100 test cases.

OUTPUT SPECIFICATION

The output for each test case should appear in the order in which they appear in the input. The output for each test case must be `Accepted`, `Presentation Error`, or `Wrong Answer`. Each output case should be terminated by **<EOLN>**. No extra spaces or **<EOLN>**'s should appear in the file.

SAMPLE INPUT

```
4<EOLN>
1 2<EOLN>
Just one line?<EOLN>
Just one line?<EOLN>
<EOLN>
2 2<EOLN>
The first characters of the alphabet are:<EOLN>
[abcde]<EOLN>
Here they come:<EOLN>
a b c d e<EOLN>
1 1<EOLN>
That's it: [abcde]<EOLN>
That's it: AbCdE<EOLN>
1 1<EOLN>
[2] and [3] make [5]<EOLN>
I guess 2 and 3 are less than 50.<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
Accepted<EOLN>
Wrong Answer<EOLN>
Presentation Error<EOLN>
Presentation Error<EOLN>
<EOF>
```