2003/2004 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Go Scoring

The Bog City Go Club has a scoreboard installed in the main competition room of the Club. The scoreboard is used to display the board position, current move, and score of one of the games taking place in the room. Although it is controlled by a microprocessor, the scoreboard is a simple device that merely responds to operator commands by displaying them directly. This is really all that is needed, with one exception. It is very difficult for the operator to continually calculate and enter a running score. Your task will be to automate the calculation, so the operator can enter it easily.

The Bog City Go Club uses the proposed international Go rules devised by the people on the "go-rules" mailing list. The rules relevant to scoring follow:

(General)

2. One player uses black stones, the other white.
4. The board is a grid of 19 horizontal and 19 vertical lines forming 361 intersections.
5. Two intersections are adjacent if they have a line but no intersection between them.
6. Two intersections with either black, white, or no stones on them are connected if they are adjacent or if there is a chain of adjacent intersections of their type between them.
7. A region consists of an intersection and any intersections connected to it.

(Move)

2. A move is either a play of an own stone on an empty intersection, or a pass.

(Scoring)

1. The points of each player are the numbers of intersections
   a) with his stones, and
   b) of the empty regions only adjacent to intersections with his stones.

When the operator wants your program to calculate a score, he will tell the scoreboard to download a textual representation of the board, which your program can read from the standard input stream. Your program will calculate the score and output it on the standard output stream.

Input for each board position consists of 20 lines, each terminated by a newline character. The first line is a spacer with just the newline character on it. The 19 lines that follow it represent the board position. On each of these lines, there are 19 pairs of characters before the newline that represent the intersections on that line of the go board. The first character of each pair is a space. " +" and " *" represent empty intersections. " @" represents an intersection with a black stone, and " O" represents an intersection with a white stone.

When your program has read 20 lines (1 spacer and 19 lines defining a board), it should calculate and output the score for that board. The scores for any number of Go boards may be requested. When the operator is done, he will turn off the scoreboard, which your program will see as an end-of-file on the input stream. When it sees the end-of-file, it should exit.

Output for each board should be a single line terminated by a newline character. The line should contain two decimal integers separated by a single blank character, with no leading or trailing whitespace. There should be no leading zeros in either integer, unless the value is zero, in which case the integer should be a single "`0`". The first integer should be the score calculated for black, and the second should be the score calculated for white. Your program should output one line of output for every 20 lines of input.

Problem 2
Go Scoring (continued)

Sample Input

(the first line is blank)

```
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + * + + + + + * + + + + + * + + +
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + * + + + + + * + + + + + * + + +
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + @ O + + + + + + + + + + + + + +
+ + @ * @ O + + + * + + + + + * + + +
+ + + @ O + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + +

+ + + + O O O @ + + + + @ @ O + + + +
+ + O + O @ O @ @ @ @ @ @ O O + + + +
+ + + O @ @ O @ + @ + + @ O + + + O +
+ + O @ @ @ @ @ @ O @ + @ O + * + + +
+ O O O @ + @ O O O @ @ O O + + O + +
+ + O @ @ @ + @ @ O O @ + + O O + O +
O O O @ O O @ + O + O @ O @ @ @ @ O +
@ + @ @ O + O O O O + O O @ + + @ O +
+ @ @ O O @ @ @ @ O + O @ @ @ @ O + +
O O O * + O O @ @ @ O O @ O @ O O O O
@ O O O O + O @ O @ @ @ @ O @ O @ @ @
@ @ O @ O + O O O @ + + @ O O O O O @
+ @ @ @ @ O O + + O @ + @ O + O O @ @
+ + @ @ + @ O O O O @ @ @ + O O @ @ @
+ @ + + + @ O @ @ O O O @ @ @ O O @ +
+ + @ * @ @ O @ O O O @ @ O @ O + @ +
+ @ + @ + + @ @ @ @ @ O O O O O @ + +
+ + + + + + + + + @ O O O + + O @ @ @ +
+ + + + + + + + + @ @ O + + + O O @ + +
```

Output for the Sample Input

```
5 3
177 175
```