2015 Mid-Atlantic Regional Programming Contest
Practice Round

Welcome to the practice round for the 2015 ICPC Mid-Atlantic Regional. Before you start the contest, please take the time to review the following:

The Contest

1. There is one (1) practice problem. Please submit solutions or request clarifications **for this problem only.** Unless you have a real question about the problem, please submit at most one clarification request, and at most two runs. It is important that everyone have a chance to see how the system works. Even if you do not solve the practice problem, you should submit once just to practice with the system.

2. Before completing the practice problem, please read all of the notes listed here. They are designed to help you solve the problems during the contest.

3. The winning team is the one that successfully completes the most problems in the time allowed.

   If teams are tied with the same number of problems solved, the tie is broken in favor of the team with the fewest penalty points. For each problem *solved correctly*, penalty points are charged as the sum of

   - the number of minutes elapsed since the start of the contest to when the successful submission was made, and
   - 20 points for each incorrect submission prior to the successful one.

   No penalty points are added for problems that are never solved.

4. In the event that you feel a problem statement is ambiguous or incorrect, you may request a clarification. Read the problem carefully before requesting a clarification.

   If a clarification is issued during the contest, it will be broadcast to *all* teams.

   If the judges believe that the problem statement is sufficiently clear, you will receive the response, "No response, read problem statement." If you receive this response, you should read the problem description more carefully. If you still feel there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you may have found.

Submitting

5. Solutions for problems submitted for judging are called runs. Each run will be judged.

6. The judges will respond to your submission with one of the following responses.

Response	Explanation
Yes	Your submission has been judged correct.
Wrong Answer	Your submission generated output that is not correct.
Output Format Error	Your submission's output is not in the correct format or is misspelled.
Incomplete Output	Your submission did not produce all of the required output.
Excessive Output	Your submission generated output in addition to or instead of what is required.
Compilation Error	Your submission failed to compile.
Run-Time Error	Your submission experienced a run-time error.
Time-Limit Exceeded	Your submission did not solve the judges' test data within 30 seconds.
Other-Contact Staff	Contact your local site judge for clarification.

7. In the event that more than one response is applicable, the judges may respond with any of the applicable responses. For example, a program that runs too long but produces incorrect output before it is killed might receive either a "Wrong Answer" or a "Time-Limit Exceeded" response. A program that crashes before completing the test data set might receive either an "Incomplete Output" or a "Run-Time Error" response.

8. **Do not** request clarifications on when a response will be returned. If you have not received a response for a run within 30 minutes of submitting it, **you may have a runner ask the local site judge to determine the cause of the delay.**

   If, due to unforeseen circumstances, judging for one or more problems begins to lag more than 30 minutes behind submissions, a clarification announcement will be issued to all teams. This announcement will include a change to the 30 minute time period that teams are expected to wait before consulting the site judge.

9. The submission of code deliberately designed to delay, crash, or otherwise negatively affect the contest itself will be considered grounds for immediate disqualification.

Your Programs

10. Your program must be contained within a single source-code file. Java programs should be written in the default (unnamed) package, meaning that it should not contain a `package` statement at all.

    Use the filename extension "`.cpp`" for C++ program files. Use the extension "`.c`" for C program files. Use the extension "`.java`" for Java program files.

    Note that all filename extensions are lower case.

11. Your code will be compiled for judging as follows:

    C: `gcc -O2 -std=gnu99 -static` *yourFileName* `-lm`

    C++: `g++ -O2 -std=c++11 -static` *yourFileName*

Java: `javac -encoding UTF-8 -sourcepath .` *yourFileName*

For Java, the compiled code will be executed using the command:

`java -Xss8m -Xmx1024m` *yourClassName*

12. All solutions must read from standard input and write to standard output.

    In C this is `scanf`/`printf`, in C++ this is `cin`/`cout`, and in Java this is `System.in`/`System.out`.

13. Unless otherwise specified, all lines of program output

    - must be left justified, with no leading blank spaces prior to the first non-blank character on that line,
    - must end with the appropriate line terminator (`\n`, `endl`, or `println()`), and
    - must not contain any blank characters at the end of the line, between the final specified output and the line terminator.

    You must not print extra lines of output, even if empty, that are not specifically required by the problem statement.

14. Unless otherwise specified, all numbers in your output should begin with the minus sign (−) if negative, followed immediately by 1 or more decimal digits. If the number being printed is a floating point number, then the decimal point should appear, followed by the appropriate number of decimal digits.

    For output of real numbers, the number of digits after the decimal point will be specified in the problem description (as the "*decimal digits of precision*").

    All floating point numbers printed to a given precision should be rounded to the nearest value. For example, if 2 decimal digits of precision is requested, then 0.0152 would be printed as "0.02" but 0.0149 would be printed as "0.01".

    In other words, neither scientific notation nor commas will be used for numbers, and you should ensure that you use a printing technique that rounds to the appropriate precision.

15. All input sets used by the judges will follow the input format specification found in the problem description. You do not need to test for input that violates the input format specified in the problem.

16. All lines of program input will end with the appropriate line terminator (e.g., a linefeed on Unix/Linux systems, a carriage return-linefeed pair on Windows systems).

17. If a problem specifies that an input is a floating point number, the input will be presented according to the rules stipulated above for output of real numbers, except that decimal points and the following digits may be omitted for numbers with no non-zero decimal portion. Scientific notation will not be used in input sets unless a problem explicitly allows it.

18. Every effort has been made to ensure that the compilers and run-time environments used by the judges are as similar as possible to those that you will use in developing your code. With that said, some differences may exist. It is, in general, your responsibility to write your code in a portable manner compliant with the rules and standards of the programming language. You should not rely upon undocumented and non-standard behaviors.

Good luck, and HAVE FUN!!!

Practice Problem: Lunacy

After several months struggling with a diet, Jack has become obsessed with the idea of weighing less. In an odd way, he finds it very comforting to think that, if he had simply had the luck to be born on a different planet, his weight could be considerably less.

Of course, the planets are far out of reach, but even the Earth's moon would yield a dramatic weight loss. Objects on the moon weight only 0.167 of their weight on Earth.

Input

Input consists of one or more lines, each containing a single floating point number denoting a weight (in pounds) on the Earth. The end of input is denoted by a negative floating point number.

Output

For each line of input data, your program should print a single line of the form

```
X Y
```

where `X` is the weight from the input and `Y` is the corresponding weight on the moon. Both output numbers should be printed to a precision of 2 digits after the decimal point. The numbers should be separated on the line by a single blank.

Example

Input:

```
100.0
12.0
0.12
120000.0
-1.0
```

Output:

```
100.00 16.70
12.00 2.00
0.12 0.02
120000.00 20040.00
```