2016 Mid-Atlantic Regional Programming Contest
Practice Round

Welcome to the practice round for the 2016 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 execute your program on multiple test cases. To be successful, each test execution must return the correct output, formatted as specified in the problem statement, and must complete execution within the appropriate time limit.

   If a problem does not specify a time limit as part of its output format description, then it must complete each test case within 2 seconds.

   - Although Python is accepted as a programming language in this contest, no guarantee is made that a Python solution is possible that runs within the time limits allowed for any given problem.

7. 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 one or more of the judges' test data within the allotted time period.
Other-Contact Staff	Contact your local site judge for clarification.

8. 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.

9. **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.

10. 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

11. 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. Use the extension "`.py`" for Python program files.

    Note that all filename extensions are lower case.

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

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

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

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

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

    `java XX:+UseSerialGC -Xss64m -Xms1024m -Xmx1024m` *yourClassName*

    Python 3: `python3` *yourFileName*

13. All solutions must read from standard input and write to standard output. Do not print to the standard error stream, as this may be mistaken for a run-time error signal.

14. 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 lines, 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.

15. 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, using unbiased (a.k.a. half-to-even) rounding. For example, if 2 decimal digits of precision is requested, then the table below shows how certain exact values would be printed:

exact	prints as
0.0149	0.01
0.0251	0.03
0.015	0.02
0.025	0.02

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.

16. 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.

17. 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).

18. 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.

19. 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: St. Ives

Robert the chapman (a medieval traveling merchant) made regular trips between his home village and St. Ives to peddle his cloth, ribbons, and needles. On one such trip he encountered a curious procession:

As I was traveling to St. Ives
I met a man with seven wives.
Every wife had seven sacks.
Every sack had seven cats.
Every cat had seven kits.
Kits, cats, sacks, wives -
How many were traveling to St. Ives?

The answer to this classic ancient riddle is 'one'. Robert was traveling to St. Ives. The others were all traveling away from St. Ives. However, if we prefer to ask the question of how many were traveling from St. Ives, we can add up:

- 1 man
- 7 wives
- 7*7 sacks
- 7*7*7 cats
- 7*7*7*7 kittens

for a total of 2801.

On his next trip to St. Ives, Robert met the same man, this time accompanied by 3 wives, each with 3 sacks, and so on. Becoming curious about what seemed to be a bizarre village ritual of some kind, Robert kept track of how many traveled with the man each time he encountered him during the subsequent year.

Help Robert compute the number of entities he has encountered during his travels to St. Ives.

Input

Input consists of a single line, containing one integer in the range $0 \ldots 100$. That integer indicates the number to be applied as the multiplier in the St Ives riddle.

Output

Print a line containing the integer number traveling from St. Ives (man, wives, sacks, cats, kits).

Example

Input:

Given the input

```
3
```

The output would be

```
121
```