icpc.foundation
icpc International Collegiate Programming Contest
TWO SIGMA
north america sponsor
programming tools sponsor
2018 ICPC Mid-Atlantic USA Regional Contest
event sponsor

2018 ICPC Mid-Atlantic North America Programming Contest

Contest Guide and Rules

Nov. 10, 2018

Welcome to the 2018 ICPC Mid-Atlantic Regional.

This document outlines the rules and procedures that will be in effect for both the practice round and the contest itself.

The Contest

1. There will be one problem for the practice round. For the actual contest, there will be eight problems.
2. 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.

Submitting

3. The contest web interface is found at `https://mausa18.kattis.com`.
4. You will submit problems via the web interface.

   Log in to the web interface and choose "submit". You will be presented with a web form where you can upload files.

Your Programs

5. 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.
6. Your source code file names should begin with an alphabetic character and thereafter be composed of alphabetic characters, numeric digits, or the punctuation marks period ('.'), hyphen ('-'), or underscore ('_').

   No blank spaces within file names!

   - Use the filename extension "`.cpp`" or "`.cc`" for C++ program files. (This is the only circumstance in which a '+' should appear within a file name.)
   - Use the extension "`.c`" for C program files.
   - Use the extension "`.java`" for Java program files.
   - Use the extension "`.py`" for Python program files.

   All filename extensions are lower case.

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

   Output sent to the standard error stream will be ignored and will not affect the judging of whether your output is correct. You are not obligated, therefore, to remove debugging output printed to standard error. (Such output does, however, take time and, if excessive, could cause your program to be rejected due to a Time Limit Exceeded error.)

8. 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.

9. 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).

10. 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.

11. If a problem calls for floating-point output, it will specify "N decimals precision" in the output description. That means that you should print, in non-scientific notation, a number with N digits after the decimal point. The final position should be rounded to most closely approximate the remaining digits. (Although there are slight differences in rounding schemes favored by various programming languages, the judge's data has been chosen so that these differences will be insignificant.)

    On some problems, the judges may relax these requirements if they are considered inconsequential to the judging process. However you should not rely upon such relaxation.

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

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

    C++: `g++ -O2 -std=gnu++14` -static *yourFileName*

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

13. Programs in interpreted languages will be executed as follows:

    Java: `java -XX:+UseSerialGC -Xss64m -Xms1024m -Xmx1024m -Dfile.encoding=UTF-8` *yourClassName*

    Python 2: `pypy` *yourFileName*

    Python 3: `python3` *yourFileName*

14. 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.

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

Judging

16. Solutions for problems submitted for judging are called runs. Each run will be judged. This judging may be entirely automated or may be managed by a human.

17. 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 1 second (CPU time) apiece.

    Of course, you cannot know for sure whether your own processor is faster or slower than the judging hardware, nor by how much this might be true. Coping with this uncertainty is considered to be part of the challenge in the contest, so you should generally attempt to design solutions that beat the time limit by a healthy margin.

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

Response	Explanation
Accepted	Your program has been judged correct.
Compile Error	Your program failed to compile.
Wrong Answer	Your program finished within the time limit, but the output produced was incorrect.
Run Time Error	Your program crashed during testing.
Time Limit Exceeded	Your program ran for too long on one of the test cases.
Memory Limit Exceeded	Your program tried to allocate more memory than is available.
Ouput Limit Exceeded	Your program generated an unacceptable amount of output (far exceeding any correct output).
Judge Error	Your program revealed a bug or mis-configuration in the judging software. (This should not happen during the contest.)

    Additional notes on these responses:

    - Test cases are processed one by one until all have been completed or a response other than "Accepted" is indicated. It is possible, therefore, that a program would have generated "Wrong Answer", "Run Time Error", and "Time Limit Exceeded" on different test cases. Which one of these gets reported back to you is entirely a matter of luck, depending upon the order in which the test cases happened to be run.
    - A "Compilation Error" can be issued if, during submission, you give the wrong information about the main Java class, or about the programming language (including using an incorrect file extension).
    - A "Wrong Answer", "...Limit Exceeded", or "Runtime Error" response can be generated if, during submission, you mis-identify which problem your source code tries to solve.

- Make sure you return 0 from your C++/Java programs or you may get a "Run Time Error". In Java uncaught exceptions or termination with a non-zero exit status will be judged as Run Time Errors.

19. You can track the progress of your submissions by logging into the web interface and choosing your name from the top right menu. On this page you will see a list of all submissions you have made, in reverse chronological order. As the submission proceeds through the judgement process your submissions page will reflect this. The states a submission will pass through while it is being judged are:

    - New
    - Waiting for Compile
    - Compiling
    - Waiting for Run
    - Running
    - Final Judgement

Clarifications

20. 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.

21. To issue a clarification request, click on Clarifications in the contest web interface.

    The clarifications page has three sections:

    - submitted clarification requests from your team that have not yet been answered,
    - a form for submitting a clarification request, and
    - clarification requests with answers from the judges.

    When you submit a clarification request, please select a subject (either one of the problems or general) and write your request *clearly*.

    > By the day of the the contest, the problem descriptions have been reviewed carefully by many different people. So if you submit a clarification that simply says "Problem A is ambiguous." or "The sample output for problem B is incorrect.", the judges will have no particular reason to believe you and will quickly respond "*No response, read problem statement.*"
    >
    > You will need to explain carefully what part of the problem statement you think is ambiguous or incorrect and why you think so if you want to convince the judges that a more substantive response is justified.

22. Do not use clarifications to ask "What would the correct output be for this input data?" You are expected to figure that out yourself from the problem description.

    You may, however, wish to include sample test inputs and your anticipated output as part of a careful argument that the problem description is ambiguous or incorrect.

23. Do not use clarifications to ask questions about your local environment (physical or electronic), such as "Where will lunch be served?" or "Can someone replace this bad keyboard?". This is a *distributed* contest being conducted over many sites, and the judge who recieves your clarification request will not even know if he/she is in the same state as you, much less the details of what is located where in your building.

24. **Do not** use clarifications to request 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.

25. Notifications that there are new clarification replies are displayed on all Kattis web pages, but not on other pages (e.g., API documentation and scoreboards).

    You may need to manually refresh those pages to see if new notifications have arrived.