The 3rd Universal Cup Finals

Judging Notes (v1.0)

The Universal Cup Scientific Committee & Technical Committee

April 19th, 2026

This document contains important information related to the way submissions are judged at the Universal Cup Finals. It complements the *Contest Rules* and the *Technical Notes*; teams are required to read all three documents carefully before the contest.

1 Clarifications

1. Historically, the Judges have issued very few formal clarifications, and those were generally limited to genuine ambiguities. Please read the problem statement and examine the sample test cases carefully before submitting any clarification request.
2. All clarification requests must be written in **English only** and submitted through DOMjudge. Replies given privately to a single team carry no precedence; only public clarifications issued by the Scientific Committee modify the problem set.
3. Should the Scientific Committee issue a public clarification (an explanation, supplementary detail, additional example, or amendment), an announcement will also be made publicly at the competition venue.

2 Input and Output

The following guidelines apply to handling input and output in your programs:

- All input comes from *standard input*, and all output goes to *standard output*. See the *Technical Notes* for details on standard error.
- Output formatting should follow the sample output in the problem statement, although extra whitespace within reason is acceptable. For example, an extra blank at the end of a line, or an extra blank line at the end of the output, is acceptable; however, printing an excessive amount of whitespace (or any other extraneous data) may exceed the output limit and will be judged as **Output Limit Exceeded (OLE)**, or as **Wrong Answer (WA)** when it changes the meaning of the output.

- For problems with floating-point output, the Judges will accept a range of answers as correct provided they satisfy the constraints described in the problem statement. These constraints will be specified as an absolute and/or relative tolerance, which will be given.
- There is no such thing as "Presentation Error" or "Format Error". If, for example, you misspell the word "infinity" and the problem requires that word as output, then your submission will be judged as **Wrong Answer (WA)**.
- Unless a problem specifically indicates that uppercase or lowercase letters are important, then either will be accepted. For example, "Impossible" or "impossible" would be treated the same, but "Impassable" would be judged as **Wrong Answer (WA)**.

3 Verdicts

1. Each submission will be evaluated and strictly categorized as "Accepted" or "Rejected". Only an "Accepted" verdict will result in points being awarded for the corresponding problem; no partial credit is given.
2. Rejected submissions will be marked with one of the following verdicts: Compilation Error (CE), Runtime Error (RTE), Time Limit Exceeded (TLE), Wrong Answer (WA), No Output (NO), Output Limit Exceeded (OLE).
3. Your program may be run on multiple input files. If your program has more than one type of error (for example, Time Limit Exceeded on one test and Wrong Answer on another), the verdict reported is the first error discovered, and you may receive either of them. See the *DOMjudge Team Guide* for details on how the judging works.
4. Compilation Errors do **not** count toward penalty time. The same applies to Python submissions which fail the syntax check (see the *Technical Notes*).
5. In the event of a malfunction within the judging system, the judges reserve the right to initiate a *rejudge* of some or all submissions for a given problem. Rejudges will normally only be applied to previously "Rejected" submissions; "Accepted" submissions will be rejudged only in the cases described in the *Contest Rules*.

4 Limits

1. The **Time Limit** for each problem is specified in the problem statement and on the judging system. Output-only problems do not have a time limit.
2. The default **Memory Limit** is 2 Gigabytes (GB) and the default **Output Limit** is 64 Megabytes (MB), unless otherwise stated in the problem statement. The

same memory allocation limit applies to every language; for interpreted languages this includes the runtime environment (see the *Technical Notes*).

3. Input size constraints on test cases are given as part of the problem statement. If the input contains multiple test cases, the problem will also state an upper bound on the number of test cases.

5 Problem Types

1. The contest may include the following problem types:
   - **Standard I/O Problems:** programs read from standard input and write to standard output.
   - **Interactive Problems:** programs communicate dynamically with an *interactor* via standard input/output. See the next section for additional notes.
   - **Multiple-Run Problems:** the program is executed multiple times, each time on a different input set.
   - **Output-Only Problems:** teams do not submit source code; they submit the final answer file directly.

6 Additional Judging Notes on Interactive and Multi-Pass Problems

- In most programming environments, program output is buffered to speed up I/O operations. With interactive problems, it is crucial to make sure the output is actually sent from your program and not simply stored in internal buffers. This typically means flushing the output buffers after each line of output (that is, after each newline character) as follows:
  - In C (or C++ using `cstdio`), you can use `fflush(stdout)`.
  - A C++ output stream is flushed automatically each time you write the `endl` manipulator. When using other means or if you want to be sure, call `cout.flush()`.
  - In Java and Kotlin, the `System.out` stream has so-called "auto-flush" functionality and its buffer is therefore flushed automatically with each newline character. When using other streams or if you want to be sure, invoke the `flush()` method of the stream.
  - In Python, you can use `sys.stdout.flush()`.
- Interactive problems are judged in a way similar to other problems, but there are some differences:

  - When your program attempts to read data, it will wait until more data are available or until the judge program terminates the input (unless you read in a non-blocking way, which is beyond the scope of these notes). Thus, if your program attempts to read more input than can currently be provided, then the program will stall indefinitely and your submission will get **Time Limit Exceeded (TLE)**.
  - As usual, the verdict given to an incorrect submission is the first error discovered, but this does not always mean exactly the same thing as for traditional problems.
  - The time limit for an interactive problem is how much time *your submission* may spend; the time spent by the judge program is not counted towards this.
  - The judge program may behave in an *adversarial* way and adapt the input provided to your program based on your previous output, with the intent to discover errors in your algorithm.
  - Because of timing between the interactive judge program and your submission, verdicts for incorrect submissions to interactive problems are not necessarily deterministic.

- Multi-pass problems are judged in a way similar to other problems, but there are some differences:
  - A multi-pass problem is evaluated by running your submission multiple times ("passes") on related inputs, as described in the problem statement. The input provided in later passes may depend on the output produced in earlier passes.
  - Each pass is a fresh execution of your program; no in-memory state is preserved between passes.
  - The time limit and memory limit are applied independently to each pass and are identical for all passes.
  - For a given test case, passes are executed sequentially. If an error is detected in an early pass, the remaining passes for that test case will not be executed.
  - A pass itself may be interactive. In that case, the submission must follow the same conventions and requirements as for interactive problems.

- For some problems, the judges may provide a simple testing tool simulating the judging process. If this is the case, you will find such a tool where you normally see the sample data. Executing the tool with a "`-h`" option should describe how to use the tool. Of course, the testing tool will only implement some test scenarios and only some functionality of the real judge program.