Problem format

From ICPC-Contest Control Standard
Jump to:navigation, search

This is a draft.

Contents

- 1 Overview
  - 1.1 General Requirements
  - 1.2 Programs
- 2 Problem Metadata
  - 2.1 license
  - 2.2 limits
- 3 Problem Statements
- 4 Test data
  - 4.1 Annotations
  - 4.2 Test Data Groups
- 5 Example Submissions
- 6 Input Validators
  - 6.1 Invocation
  - 6.2 Output
  - 6.3 Exit codes
    - 6.3.1 Dependencies
- 7 Output Validators
  - 7.1 Default Output Validator Specification
- 8 See also

Overview

This document describes the problem format used at the ICPC World Finals. It is intended to be a proper subset of the Kattis problem package format, i.e. problems following this spec should be valid according to the Kattis format as well, with no changes, but not necessarily vice versa.

General Requirements

The package consists of a single directory containing files as described below, or alternatively, a ZIP compressed archive of the same files using the file extension `.kpp` . The name of the directory or the base name of the archive must consisting solely of lower case letters a-z and digits 0-9.

All file names for files included in the package must match the following regexp

```
[a-zA-Z0-9][a-zA-Z0-9_.-]*[a-zA-Z0-9]
```

I.e., it must be of length at least 2, consist solely of lower or upper case letters a-z, A-Z, digits 0-9, period, dash or underscore, but must not begin or end with period, dash or underscore.

All text files for a problem must be UTF-8 encoded and not have a byte order mark.

All floating point numbers must be given as the external character sequences defined by IEEE 754-2008 and may use up to double precision.

Programs

There are a number of different kinds of programs that may be provided in the problem package; submissions, input validators and output validators. All programs are always represented by a single file or directory. In other words, if a program consists of several files, these must be provided in a single directory. The name of the program, for the purpose of referring to it within the package is the base name of the file or the name of the directory. There can't be two programs of the same kind with the same name.

Validators and graders, but not submissions, in the form of a directory may include two POSIX-compliant scripts "build" and "run". Either both or none of these scripts must be included. If the scripts are present, then:

- the program will be compiled by executing the build script.
- the program will be run by executing the run script.

Programs without build and run scripts are built and run according to what language is used. Language is determined by looking at the file endings. If a single language from the table below can't be determined, building fails. In the case of Python 2 and 3 which share the same file ending, language will be determined by looking at the shebang line which must match the regular expressions in the table below.

For languages where there could be several entry points, the default entry point in the table below will be used.

Code	Language	Default entry point	File endings	Shebang
c	C		.c	
cpp	C++		.cc, .cpp, .cxx, .c++, .C	
csharp	C#		.cs	
go	Go		.go	
haskell	Haskell		.hs	
java	Java	Main	.java	
javascript	JavaScript	main.js	.js	
kotlin	Kotlin	MainKt	.kt	
objectivec	Objective-C		.m	
pascal	Pascal		.pas	
php	PHP	main.php	.php	
prolog	Prolog		.pl	
python2	Python 2	main.py	.py	Matches the regex "`^#!.*python2 `", and default if shebang does not match any other language

python3	Python 3	main.py	.py	Matches the regex "^#!.*python3 "
ruby	Ruby		.rb	
scala	Scala		.scala	

Problem Metadata

Metadata about the problem (e.g., source, license, limits) are provided in a UTF-8 encoded YAML file named `problem.yaml` placed in the root directory of the package.

The keys are defined as below. Keys are optional unless explicitly stated. Any unknown keys should be treated as an error.

Key	Type	Default	Comments
uuid	String		UUID identifying the problem.
author	String		Who should get author credits. This would typically be the people that came up with the idea, wrote the problem specification and created the test data. This is sometimes omitted when authors choose to instead only give source credit, but both may be specified.
source	String		Who should get source credit. This would typically be the name (and year) of the event where the problem was first used or created for.
source_url	String		Link to page for source event. Must not be given if source is not.
license	String	unknown	License under which the problem may be used. Value have to be one of the ones defined below.
rights_owner	String	Value of author, if present, otherwise value of source.	Owner of the copyright of the problem. If not present, author is owner. If author is not present either, source is owner. Required if license is something other than "unknown" or "public domain". Forbidden if license is "public domain".
keywords	String or sequence of strings		Set of keywords.
limits	Map with keys as defined below	see definition below	
validation	String	default	One of "default" or "custom".
validator_flags	String		Will be passed as command-line arguments to each of the output validators.

license

Allowed values for license.

Values other than *unknown* or *public domain* requires rights_owner to have a value.

Value	Comments	Link

unknown	The default value. In practice means that the problem can not be used.	
public domain	There are no known copyrights on the problem, anywhere in the world.	http://creativecommons.org/about/pdm
cc0	CC0, "no rights reserved"	http://creativecommons.org/about/cc0
cc by	CC attribution	http://creativecommons.org/licenses/by/3.0/
cc by-sa	CC attribution, share alike	http://creativecommons.org/licenses/by-sa/3.0/
educational	May be freely used for educational purposes	
permission	Used with permission. The author must be contacted for every additional use.	

limits

A map with the following keys:

Key	Comments	Default	Typical system default
time_multiplier	optional	5	
time_safety_margin	optional	2	
memory	optional, in MiB	system default	2048
output	optional, in MiB	system default	8
code	optional, in kiB	system default	128
compilation_time	optional, in seconds	system default	60
compilation_memory	optional, in MiB	system default	2048
validation_time	optional, in seconds	system default	60
validation_memory	optional, in MiB	system default	2048
validation_output	optional, in MiB	system default	8

For most keys the system default will be used if nothing is specified. This can vary, but you SHOULD assume that it's reasonable. Only specify limits when the problem needs a specific limit, but do specify limits even if the "typical system default" is what is needed.

Problem Statements

The problem statement of the problem is provided in the directory `problem_statement/`.

This directory must contain one file per language, for at least one language, named `problem.<language>.<filetype>`, that contains the problem text itself, including input and output specifications, but not sample input and output. Language must be given as the shortest ISO 639 code. If needed a hyphen and a ISO 3166-1 alpha-2 code may be appended to ISO 639 code. Optionally, the language code can be left out, the default is then English (`en`). Filetype must be `tex` for LaTeX files.

Auxiliary files needed by the problem statement files must all be in `<short_name>/problem_statement/` , `problem.<language>.<filetype>` should reference auxiliary files as if the working directory is `<short_name>/problem_statement/`. Image file formats supported are `.png`, `.jpg`, `.jpeg`, and `.pdf`.

The LaTeX file may include the Problem name using the LaTeX command `\problemname` in case LaTeX formatting of the title is wanted.

The problem statements must only contain the actual problem statement, no sample data.

Test data

The test data are provided in subdirectories of `data/`. The sample data in `data/sample/` and the secret data in `data/secret/`.

All input and answer files have the filename extension `.in` and `.ans` respectively.

Annotations

Optionally a hint, a description and an illustration file may be provided.

The hint file is a text file with filename extension `.hint` giving a hint for solving an input file. The hint file is meant to be given as feedback, i.e. to somebody that fails to solve the problem.

The description file is a text file with filename extension `.desc` describing the purpose of an input file. The description file is meant to be privileged information that explains the purpose of the related test file, e.g. what cases it's supposed to test.

The Illustration is an image file with filename extension `.png`, `.jpg`, `.jpeg`, or `.svg`. The illustration is meant to be privileged information illustrating the related test file.

Input, answer, description, hint and image files are matched by the base name.

Test Data Groups

The test data for the problem can be organized into a tree-like structure. Each node of this tree is represented by a directory and referred to as a test data group. Each test data group may consist of zero or more test cases (i.e., input-answer files) and zero or more subgroups of test data (i.e., subdirectories).

At the top level, the test data is divided into exactly two groups: `sample` and `secret`, but these two groups may be further split into subgroups as desired.

The *result* of a test data group is computed by applying a *grader* to all of the sub-results (test cases and subgroups) in the group. See Graders for more details.

Test files and groups will be used in lexicographical order on file base name. If a specific order is desired a numbered prefix such as 00, 01, 02, 03, and so on, can be used.

In each test data group, a file `testdata.yaml` may be placed to specify how the result of the test data group should be computed. If such a file is not provided for a test data group then the settings for the parent group will be used. The format of `testdata.yaml` is as follows:

Key	Type	Default	Comments
input_validator	String or map with the keys "name" and "flags"	empty string	If a string this is the name of the input validator that will be used for this test data group. If a map then this is the name as well as the flags that will be passed to the input validator.

output_validator	String or map with the keys "name" and "flags"	empty string	If a string this is the name of the output validator that will be used for this test data group. If a map then this is the name as well as the flags that will be passed to the output validator.

Example Submissions

Correct and incorrect solutions to the problem are provided in subdirectories of `submissions/`. The possible subdirectories are:

Value	Requirement	Comment
accepted	Accepted as a correct solution for all test files	At least one is required.
wrong_answer	Wrong answer for some test file, but is not too slow and does not crash for any test file	
time_limit_exceeded	Too slow for some test file. May also give wrong answer but not crash for any test file.	
run_time_error	Crashes for some test file	

Every file or directory in these directories represents a separate solution. Same requirements as for submissions with regards to filenames. It is mandatory to provide at least one accepted solution.

Submissions must read input data from standard input, and write output to standard output.

Input Validators

Input Validators, for verifying the correctness of the input files, are provided in `input_validators/`. Input validators can be specified as VIVA-files (with file ending `.viva`), Checktestdata-file (with file ending `.ctd`), or as a program.

All input validators provided will be run on every input file. Validation fails if any validator fails.

Invocation

An input validator program must be an application (executable or interpreted) capable of being invoked with a command line call.

All input validators provided will be run on every test data file using the arguments specified for the test data group they are part of. Validation fails if any validator fails.

When invoked the input validator will get the input file on stdin.

The validator should be possible to use as follows on the command line:

```
./validator [arguments] < inputfile
```

Output

The input validator may output debug information on stdout and stderr. This information may be displayed to the user upon invocation of the validator.

Exit codes

The input validator must exit with code 42 on successful validation. Any other exit code means that the input file could not be confirmed as valid.

Dependencies

The validator MUST NOT read any files outside those defined in the Invocation section. Its result MUST depend only on these files and the arguments.

Output Validators

Output Validators are used if the problem requires more complicated output validation than what is provided by the default diff variant described below. They are provided in `output_validators/`, and must adhere to the Output validator specification.

All output validators provided will be run on the output for every test data file using the arguments specified for the test data group they are part of. Validation fails if any validator fails.

Default Output Validator Specification

The default output validator is essentially a beefed-up diff. In its default mode, it tokenizes the files to compare and compares them token by token. It supports the following command-line arguments to control how tokens are compared.

Arguments	Description
`case_sensitive`	indicates that comparisons should be case-sensitive.
`space_change_sensitive`	indicates that changes in the amount of whitespace should be rejected (the default is that any sequence of 1 or more whitespace characters are equivalent).
`float_relative_tolerance` ε	indicates that floating-point tokens should be accepted if they are within relative error ≤ ε (see below for details).
`float_absolute_tolerance` ε	indicates that floating-point tokens should be accepted if they are within absolute error ≤ ε (see below for details).
`float_tolerance` ε	short-hand for applying ε as both relative and absolute tolerance.

When supplying both a relative and an absolute tolerance, the semantics are that a token is accepted if it is within either of the two tolerances. When a floating-point tolerance has been set, any valid formatting of floating point numbers is accepted for floating point tokens. So for instance if a token in the answer file says `0.0314`, a token of `3.14000000e-2` in the output file would be accepted. If no floating point tolerance has been set, floating point tokens are treated just like any other token and has to match exactly.

See also

- Output validator
- Sample problem.yaml
- Problem format directory structure
- Problem Format Verification

Navigation menu

Views

- Page
- Discussion
- View source
- History
- PDF Export

Personal tools

- Log in

Navigation

- Main page
- Recent changes
- Random page
- Help

Search

Go Search

Tools

- What links here
- Related changes
- Special pages
- Permanent link
- Page information