# DOMjudge team manual


## Summary

Here follows a short summary of the system interface. This is meant as a quick introduction, to be able to start using the system. It is, however, strongly advised that at least one of your team's members reads all of this manual. There are specific details of this jury system that might become of importance when you run into problems. **BE WARNED!**

DOMjudge works through a web interface that can be found at
`http://example.com/domjudge/team`.

### Reading and writing

Solutions have to read all input from 'standard in' and write all output to 'standard out' (also known as console). You will never have to open (other) files. See appendix A for some examples.

### Submitting solutions

You can submit solutions with the command-line program `submit` or by the web interface:

**Command-line**
: Use `submit <problem>.<extension>`, where `<problem>` is the label of the problem and `<extension>` is a standard extension for your language. For a complete reference of all options and examples, see `submit --help`.

**Web interface**
: From your team page, `http://example.com/domjudge/team`, browse to **submit** and select the file you want to submit. By default, the problem is selected from the base of the filename and the language from the extension.

### Viewing scores, submissions, etc.

Viewing scores, submissions and sending and reading clarification requests is done through the web interface. The menu buttons at `http://example.com/domjudge/team` speak for themselves.

*End of summary*

# 1 Submitting solutions

Submitting solutions can be done in two ways: with the command-line program `submit` or using the web interface. One of the interfaces might not be available, depending on the system configuration by the jury. A description of both methods follows.

## 1.1 Command-line: `submit`

**Syntax:** `submit [options] filename.ext`

The submit program takes the name (label) of the problem from `filename` and the programming language from the extension `ext`. This can be overruled with the options `-p problemname` and `-l languageextension`. See `submit --help` for a complete list of all options, extensions and some examples. Use `submit --help | more` when the help text does not fit on one screen.

`submit` will check your file and warns you for some problems: for example when the file has not been modified for a long time or when it's larger than the maximum source code size.

Then `submit` displays a summary with all details of your submission and asks for confirmation. Check whether you are submitting the right file for the right problem and language and press 'y' to confirm. `submit` will report a successful submission or give an error message otherwise.

The submit program uses a directory `.domjudge` in the home directory of your account where it stores temporary files for submission and also a log file `submit.log`. Do not remove or change this directory, otherwise the `submit` program might fail to function correctly. Furthermore a "public ssh-key" of the jury might have been added to the SSH configuration; this is also necessary for the functioning of `submit`.

## 1.2 Web interface

Solutions can be submitted from the web interface at http://example.com/domjudge/team. Navigate to **submit** to select the file for submission, and the problem and language. These can also be left on the default 'automatic' setting; then the problem and language will be determined from the base and extension of the filename respectively.

After you hit the submit button and confirm the submission, a page is shown with a confirmation of successful upload. Next, you should be able to see the submission in your submissions list.

# 2 Viewing the results of submissions

There is an overview of your submissions on your team web page. It contains all relevant information: submission time, programming language, problem and status. The address of your team page is http://example.com/domjudge/team. From here you can also view the public scoreboard page with the scores of all teams.

### 2.1 Possible results

A submission can have the following results:

**CORRECT** The submission passed all tests: you solved this problem!

**COMPILER-ERROR** There was an error when compiling your program. On the submission details page you can inspect the exact error (this option might be disabled).

**TIMELIMIT** Your program took longer than the maximum allowed time for this problem. Therefore it has been aborted. This might indicate that your program hangs in a loop or that your solution is not efficient enough.

**RUN-ERROR** There was an error during the execution of your program. This can have a lot of different causes like division by zero, incorrectly addressing memory (e.g. by indexing arrays out of bounds), trying to use more memory than the limit, etc. Also check that your program exits with exit code 0!

**NO-OUTPUT** Your program did not generate any output. Check that you write to standard out.

**WRONG-ANSWER** The output of your program was incorrect. This can happen simply because your solution is not correct, but remember that your output must comply exactly with the specifications of the jury.

**PRESENTATION-ERROR** The output of your program has differences in presentation with the correct results (for example in the amount of whitespace). This will, like WRONG-ANSWER, count as an incorrect submission. This result is optional and might be disabled.

**TOO-LATE** Bummer, you submitted after the contest ended! Your submission is stored but will not be processed anymore.

## 3 Clarifications

All communication with the jury is to be done with clarifications. These can be found on your team page. Both clarification replies from the jury and requests sent by you are displayed there.

There is also a link to submit a new clarification request to the jury. This request is only readable for the jury and they will respond as soon as possible. Answers that are relevant for everyone will be sent to everyone.

When you receive a new clarification from the jury, this will automatically be displayed as "(1 new)" in the clarification button in the menu bar. This will be updated automatically even without reloading the page.

# 4 How are submissions being judged?

The DOMjudge jury system is fully automated. In principle no human interaction is necessary. The judging is done in the following way:

## 4.1 Submitting solutions

With the `submit` program or the web interface (see section 1) you can submit a solution to a problem to the jury. Note that you have to submit the source code of your program (and not a compiled program or the output of your program).

There your program enters a queue, awaiting compilation, execution and testing on one of the jury computers.

## 4.2 Compilation

Your program will be compiled on a jury computer running Linux. Using a different compiler or operating system than the jury should not be a problem. Be careful however, not to use any special compiler and/or system specific things (you may be able to check compiler errors on the team page).

## 4.3 Testing

After your program has compiled successfully it will be executed and its output compared to the output of the jury. Before comparing the output, the exit status of your program is checked: if your program gives the correct answer, but exits with a non-zero exit code, the result will be a RUN-ERROR! There are some restrictions during execution. If your program violates these it will also be aborted with a RUN-ERROR, see section 4.4.

When comparing program output, it has to exactly match to output of the jury. So take care that you follow the output specifications. In case of problem statements which do not have unique output (e.g. with floating point answers), the jury may use a modified comparison function.

## 4.4 Restrictions

To prevent abuse, keep the jury system stable and give everyone clear and equal environments, there are some restrictions to which all submissions are subjected:

**compile time** Compilation of your program may take no longer than 30 seconds. After that compilation will be aborted and the result will be a compile error. In practice this should never give rise to problems. Should this happen to a normal program, please inform the jury right away.

**source size** The source code of your program may not exceed 256 kilobytes, otherwise your submission will be rejected.

**memory** During execution of your program, there are 524288 kilobytes of memory available. This is the total amount of memory (including program code, statically and dynamically defined variables, stack, Java VM, ...)! If your program tries to use more memory, it will abort, resulting in a run error.

**program output** You are not allowed to output more than 4096 kilobytes on either standard out or standard error, and will get a run error if you exceed this limit.

**number of processes** You are not supposed to create multiple processes (threads). This is to no avail anyway, because your program has exactly 1 processor fully at its disposal. To increase stability of the jury system, there is a maximum of 15 processes that can be run simultaneously (including processes that started your program).

People who have never programmed with multiple processes (or have never heard of "threads") do not have to worry: a normal program runs in one process.

## 4.5 Java class naming

Compilation of Java sources is somewhat complicated by the class naming conventions used: there is no fixed entry point; any class can contain a method `main`. Furthermore, a class declared `public` must be located in an indentically named file.

In the default configuration of DOMjudge this is worked around by autodetecting the main class. When this feature is not used, then the main class should be "`Main`", with method "`public static void main(String args[])`", see also the Java code example in appendix A.

# A Code examples

Below are a few examples on how to read input and write output for a problem.

The examples are solutions for the following problem: the first line of the input contains the number of testcases. Then each testcase consists of a line containing a name (a single word) of at most 99 characters. For each testcase output the string "Hello <name>!" on a separate line.

Sample input and output for this problem:

| **Input** | **Output** |
|---|---|
| 3<br>world<br>Jan<br>SantaClaus | Hello world!<br>Hello Jan!<br>Hello SantaClaus! |

Note that the number 3 on the first line indicates that 3 testcases follow.

A solution for this problem in C:

```
1   #include <stdio.h>
2
3   int main()
4   {
5           int i, ntests;
6           char name[100];
7
8           scanf("%d\n", &ntests);
9
10          for(i=0; i<ntests; i++) {
11                  scanf("%s\n", name);
12                  printf("Hello %s!\n", name);
13          }
14
15          return 0;
16  }
```

Notice the last `return 0;` to prevent a RUN-ERROR!

A solution in C++:

```
1   using namespace std;
2
3   #include <iostream>
4   #include <string>
5
6   int main()
7   {
8           int ntests;
9           string name;
10
11          cin >> ntests;
12          for(int i = 0; i < ntests; i++) {
13                  cin >> name;
14                  cout << "Hello " << name << "!" << endl;
15          }
16
17          return 0;
18  }
```

A solution in Java:

```
1   import java.io.*;
2
3   class Main
4   {
5           public static BufferedReader in;
6
7           public static void main(String[] args) throws IOException
8           {
9                   in = new BufferedReader(new InputStreamReader(System.in));
10
11                  int nTests = Integer.parseInt(in.readLine());
12
13                  for (int i = 0; i < nTests; i++) {
14                          String name = in.readLine();
15                          System.out.println("Hello "+name+"!");
16                  }
17          }
18  }
```

A solution in Pascal:

```
1   program example(input, output);
2
3   var
4           ntests, test : integer;
5           name         : string[100];
6
7   begin
8           readln(ntests);
9
10          for test := 1 to ntests do
11          begin
12                  readln(name);
13                  writeln('Hello ', name, '!');
14          end;
15  end.
```

And finally a solution in Haskell:

```
1   import Prelude
2
3   main :: IO ()
4   main = do input <- getContents
5             putStr.unlines.map (\x -> "Hello " ++ x ++ "!").tail.lines $ input
```