

Judging Details

Judge System

Your programs will be judged on the system once they're submitted.

- Your program must read input data from the standard input, and write its output to the standard output.
- Other outputs, e.g. writing to the standard error, will not be used for judging.
- You will never have to write to (open) a file, and are not allowed to do so.

Your programs will be run inside a *sandboxed environment*, i.e. with protections to prevent the system from being damaged. Specifically:

- Memory usage is limited to 2 GB in the environment. Note it is the total amount, not the amount you can use exclusively in your programs.
- The stack size is set unlimited (in C/C++), only capped by the total memory limit.
- Multi-processing or multi-threading is discouraged and unlikely beneficial, though not prohibited. Remember your programs will run on a single processor core. The total number of processes is limited to 64, including ones the system may create outside your programs.
- It is *never* recommended to run external commands. It is technically possible but probably does not work as you expect.

If you have no idea about what these mean — no worries. Just remember your programs should use the standard input and output, not files.

There are a couple more restrictions that apply:

- The total amount of source code must not exceed 256 KB in each submission.
- Your program must compile within 30 seconds.

See the DOMjudge team manual for more details about these restrictions.

Note about Platform

The judge system is running on Google Compute Engine, C2 machine type (`c2-standard-4`). For more information about Google Compute Engine, please visit the official website^{*1}.

1. <https://cloud.google.com/compute/docs/cpu-platforms>

Compilers & Options

The judge system uses the following compilers and execution environments (e.g., interpreters) with the following options. `"$@"` is substituted with your source file(s); `"$DEST"` is the name of the binary (which is `./a.out` by default) and is chosen arbitrarily by the system.

The **Run** commands indicated in the following table are for non-interactive problems. For interactive problems, standard input and output are connected to a judge program. See the "Note on Interactive Problems" section below for the details.

C
Version gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Compile <code>gcc -x c -g -O2 -std=gnu11 -static -o "\$DEST" "\$@" -lm</code>
Run <code>"\$DEST" < <i>infile</i> > <i>outfile</i></code>
C++
Version g++ (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Compile <code>g++ -x c++ -g -O2 -std=gnu++20 -static -o "\$DEST" "\$@"</code>
Run <code>"\$DEST" < <i>infile</i> > <i>outfile</i></code>
Java
Version OpenJDK 17.0.13 (build 17.0.13+11-Ubuntu-2ubuntu122.04)
Compile <code>javac -encoding UTF-8 -sourcepath . -d . "\$@"</code>
Run <code>java -Dfile.encoding=UTF-8 -XX:+UseSerialGC -Xss64m -Xms1920m -Xmx1920m <i>MainClass</i> < <i>infile</i> > <i>outfile</i></code>
Python 3 (PyPy)
Version Python 3.10.14 [PyPy 7.3.17 with GCC 10.2.1 20210130 (Red Hat 10.2.1-11)]
Compile <code>pypy3 -m py_compile "\$@"</code>
Run <code>pypy3 "\$@" < <i>infile</i> > <i>outfile</i></code>
Kotlin
Version kotlinc-jvm 1.7.21 (JRE 17.0.13+11-Ubuntu-2ubuntu122.04)
Compile <code>kotlinc -d . "\$@"</code>
Run <code>kotlinc -Dfile.encoding=UTF-8 -J-XX:+UseSerialGC -J-Xss64m -J-Xms1920m -J-Xmx1920m <i>MainClass</i> < <i>infile</i> > <i>outfile</i></code>

In Java and Kotlin, DOMjudge will detect the main class automatically; you do not have to name it `Main`. See the DOMjudge team manual for details.

In Python, **Compile** commands only verify the syntax. `*.pyc` files will not be used in the real run.

The compilers and the execution environments are also available on your workstation as the following commands:

- **C** — `compilegcc / runc`
- **C++** — `compileg++ / runcpp`
- **Java** — `compilejava / runjava`
- **Python 3** — `compilepython3 / runpython3`
- **Kotlin** — `compilekotlin / runkotlin`

Submission Results

The judges may have prepared multiple test cases for each problem. On each submission, DOMjudge decides one result for each test case. DOMjudge does *not* report results for each test case, but it reports one result for a submission, based on the following rules.

Results for test cases

For each test case, DOMjudge decides one of the following results:

- **CORRECT** - Your program ran successfully and passed the test case.
- **TIMELIMIT** — Your program did not finish within the time limit.
- **RUN-ERROR** — Your program crashed or exited with a non-zero exit status (e.g. because of missing `return 0;` in C/C++).
- **OUTPUT-LIMIT** — Your program produced excessive output (> 8 MB).
- **WRONG-ANSWER** — Your program neither crashed nor exceeded the time limit, but produced incorrect output.
- **NO-OUTPUT** — Your program did not produce any output.

See the DOMjudge team manual for more details about these results.

Results for submissions

For each submission, DOMjudge reports one of the following results:

Accepted

- **CORRECT** — Your program resulted in **CORRECT** for all test cases.

Rejected with 20-minutes penalty

- **TIMELIMIT, RUN-ERROR, OUTPUT-LIMIT, WRONG-ANSWER, NO-OUTPUT** — If your program receives any of these results for at least one test case, one of those results will be the overall result for the submission. If multiple of these results occur, the reported result is not guaranteed to be any specific one.

Rejected with no penalty

The following results imply your program did not even start. You do not receive any penalty for these results.

- **COMPILE-ERROR** — Your program did not compile in the judging environment. You can consult the error message(s) on the submission details page.
- **TOO-LATE** — Your program was submitted after the contest was over.^{*2}

Note on Interactive Problems

You may meet “interactive problems” in the contest. They are the same as other problems in a way that your program will read from standard input and print results to standard output. The difference is, the standard input and output are connected to a special program (judge program), with which you have to communicate back and forth. Unlike other problems where the input text is fixed for each test case, the input varies based on your previous outputs.

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 write.

- In C/C++ with `stdio.h` (or `cstdio`), you can use `fflush(stdout)`. Writing `\n` does not mean it will get flushed.
- In C++ with `iostream`, an output stream is flushed automatically each time you write the `std::endl` manipulator. When using other means or if you want to be sure, call `std::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()`.

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. Note that if your program attempts to read more input than can be provided currently (e.g., because you forgot to flush your previous output, or because of some other reason), then the program will stall indefinitely and your submission will get **TIMELIMIT**.

Note on Languages

The judges have solved all problems in languages from at least two of the three distinct language groups (Java/Kotlin, C/C++, and Python).

Note to Python Users

Only syntax errors will be reported as **COMPILE-ERROR**. Other types of errors, such as `NameError` or `ModuleNotFoundError`, will result in **RUN-ERROR** and incur a 20-minute penalty.

It is fine, though not needed, to start your scripts with an interpreter directive (line starting with `#!`, also known as shebang). *³

The full list of modules available in the judge system can be found in the following section.

2. Note that this does not mean your programs need to be judged before the end of the contest. Your programs will be judged as long as submitted (“queued”) within the contest time.

3. Some past versions of DOMjudge refused scripts that contain a shebang.

Available Python Modules

__decimal	_random	distutils	pypy_tools
__exceptions__	_rawffi	doctest	pypyjit
__future__	_resource_build	email	pyrepl
__pypy__	_resource_cffi	encodings	queue
_abc	_scproxy	ensurepip	quopri
_aix_support	_sha1	enum	random
_ast	_sha256	errno	re
_audiooop_build	_sha3	faulthandler	readline
_audiooop_cffi	_sha512	fcntl	reprlib
_blake2	_signal	filecmp	resource
_bootsubprocess	_sitebuiltins	fileinput	rlcompleter
_bz2	_socket	fnmatch	rumpy
_cffi_backend	_sqlite3	fractions	sched
_cffi_ssl	_sqlite3_build	ftplib	secrets
_codecs	_sqlite3_cffi	functools	select
_codecs_cn	_sre	future_builtins	selectors
_codecs_hk	_ssl	gc	shelve
_codecs_iso2022	_ssl_build	genericpath	shlex
_codecs_jp	_string	getopt	shutil
_codecs_kr	_strptime	getpass	signal
_codecs_tw	_struct	gettext	site
_collections	_structseq	glob	smtpd
_collections_abc	_sysconfigdata	graphlib	smtplib
_colorize	_sysconfigdata_linux_x86_64-linux-gnu	greenlet	sndhdr
_compat_pickle	_syslog_build	grp	socket
_compression	_syslog_cffi	gzip	socketserver
_contextvars	_testcapi	hashlib	sqlite3
_continuation	_testing	heapq	sre_compile
_cppyy	_testmultiphase	hmac	sre_constants
_crypt	_testmultiphase_build	html	sre_parse
_csv	_thread	http	ssl
_ctypes	_threading_local	identity_dict	stackless
_ctypes_test	_tkinter	idlelib	stat
_ctypes_test_build	_vmp prof	imaplib	statistics
_curses	_warnings	imghdr	string
_curses_build	_weakref	imp	stringprep
_curses_cffi	_weakrefset	importlib	struct
_curses_panel	_winapi	inspect	subprocess
_dbm	abc	io	sunau
_decimal_build	aifc	ipaddress	symtable
_ffi	antigravity	itertools	sys
_gdbm	argparse	json	sysconfig
_gdbm_build	array	keyword	syslog
_gdbm_cffi	ast	lib2to3	tabnanny
_hashlib	asynchat	linecache	tarfile
_hpy_universal	asyncio	locale	telnetlib
_immutables_map	asyncore	logging	tempfile
_imp	atexit	lzma	termios
_io	audiooop	mailbox	test
_jitlog	base64	mailcap	textwrap
_locale	bdb	marshal	this
_lsprof	binascii	math	threading
_lzma	binhex	mimetypes	time
_lzma_build	bisect	mmap	timeit
_lzma_cffi	builtins	modulefinder	tkinter
_markupbase	bz2	msilib	token
_marshal	cProfile	msvcrt	tokenize
_md5	calendar	multiprocessing	tputil
_minimal_curses	cffi	netrc	trace
_multibytecodec	cgi	nntplib	traceback

_multiprocessing	cgitb	ntpath	tracemalloc
_opcode	chunk	nturl2path	tty
_operator	cmath	numbers	turtle
_osx_support	cmd	opcode	turtledemo
_overlapped	code	operator	types
_pickle_support	codecs	optparse	typing
_posixshmem	codeop	os	unicodedata
_posixshmem_build	collections	pathlib	unittest
_posixshmem_cffi	colorsys	pdb	urllib
_posixsubprocess	compileall	pickle	uu
_pwdgrp_build	concurrent	pickletools	uuid
_pwdgrp_cffi	configparser	pipes	venv
_py_abc	contextlib	pkgutil	warnings
_pydecimal	contextvars	platform	wave
_pyio	copy	plistlib	weakref
_pypy_generic_alias	copyreg	poplib	webbrowser
_pypy_interact	cpyext	posix	wsgiref
_pypy irc topic	crypt	posixpath	xdrlib
_pypy_openssl	csv	pprint	xml
_pypy_testcapi	ctypes	profile	xmlrpc
_pypy_util_build	ctypes_support	pstats	zipapp
_pypy_util_cffi	curses	pty	zipfile
_pypy_util_cffi_inner	dataclasses	pwd	zipimport
_pypy_wait	datetime	py_compile	zlib
_pypy_winbase_build	dbm	pyclbr	zoneinfo
_pypy_winbase_cffi	decimal	pydoc	
_pypy_winbase_cffi64	difflib	pydoc_data	
_pypyjson	dis	pyexpat	