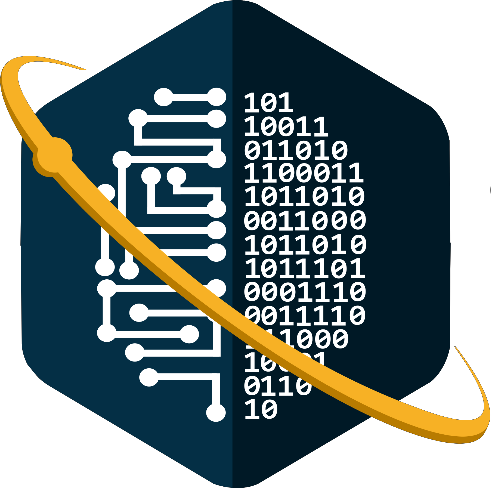
Dokumentasi *Setup* DOMJudge untuk Kontes Pemrograman Kompetitif



**Disusun Oleh**:

Muhammad Farhan Akbar

Muhammad Tsaqif Ammar

Vincentius Arnold Fridolin

Rayhan Zanzabila

Laboratorium *Computing*

Fakultas Informatika, Universitas Telkom

Agustus 2022

Daftar Isi

[1 Instalasi DOMJudge Menggunakan *Docker* 2](#_Toc112077333)

[1.1 Pendahuluan 2](#_Toc112077334)

[*1.2* Instalasi *Docker* 2](#_Toc112077335)

[1.3 *Setup* DOMJudge 2](#_Toc112077336)

[1.3.1 *Setup Database* Peserta 2](#_Toc112077337)

[1.3.2 *Setup* DOMserver *container* 3](#_Toc112077338)

[1.3.3 *Setup* Judgehost *container* 4](#_Toc112077339)

[2 *Setup Java Virtual Memory* (JVM) dan *Memstack* pada DOMJudge 6](#_Toc112077340)

[2.1 Java pada Pemrograman Kompetitif 6](#_Toc112077341)

[2.2 Solusi Masalah JVM dan *Memstack* pada DOMjudge 6](#_Toc112077342)

[2.3 Catatan Hasil Eksplorasi untuk JVM dan *Memstack* pada DOMjudge 8](#_Toc112077343)

[3 Cara Mengunggah Soal dan *Test-cases* ke DOMjudge 11](#_Toc112077344)

[3.1 Format *File* 11](#_Toc112077345)

[*3.2* Pengunggahan *Problem* 12](#_Toc112077346)

[3.3 Ubah Spesifikasi Soal yang Telah Diunggah 13](#_Toc112077347)

[4 Pembuatan Soal dengan *Custom Checker* di DOMjudge 17](#_Toc112077348)

[4.1 Pendahuluan 17](#_Toc112077349)

[4.2 Contoh Soal Kongkret 17](#_Toc112077350)

[*4.3* Pembuatan *Custom Checker* 17](#_Toc112077351)

[4.4 Pembuatan *Test-cases* 18](#_Toc112077352)

[4.5 Mengunggah Soal ke DOMJudge 18](#_Toc112077353)

[5 Pembuatan Soal Interaktif di DOMJudge 20](#_Toc112077354)

[5.1 Pendahuluan 20](#_Toc112077355)

[5.2 Contoh Soal Kongkret 20](#_Toc112077356)

[5.3 Pembuatan Interaktor 20](#_Toc112077357)

[5.4 Pembuatan *Test-cases* 21](#_Toc112077358)

[5.5 Mengunggah Soal ke DOMJudge 21](#_Toc112077359)

[5.6 Contoh Solusi Peserta 22](#_Toc112077360)

[6 Beberapa Isu Teknis Lanjut 24](#_Toc112077361)

[6.1 Pengaturan Jam pada DOMserver 24](#_Toc112077362)

[6.2 Penambahan Judgehost pada DOMjudge 24](#_Toc112077363)

[6.3 Isu *Test Case* Berlebih Spasi 24](#_Toc112077364)

[6.4 Isu *Test Case* Berakhiran Newline CRLF 25](#_Toc112077365)

# Instalasi DOMJudge Menggunakan *Docker*

## Pendahuluan

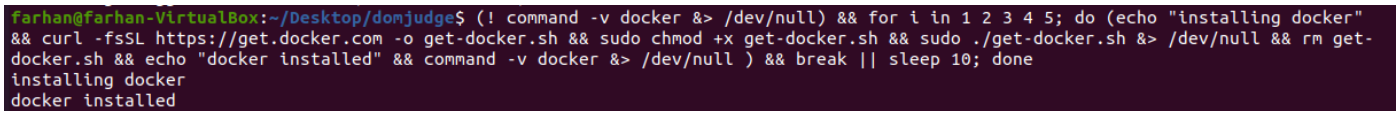
Sebelum masuk ke tahap instalasi, ada beberapa hal yang perlu diperhatikan. Dokumen ini dibuat pada tanggal 14 Juli 2022 dengan menggunakan spesifikasi sistem operasi **Ubuntu versi 20.04**.

Perlu diketahui bahwa instalasi DOMjudge melalui *Docker* perlu menggunakan sistem operasi Linux atau varian Unix lainnya. Namun, hal ini tidak menutup kemungkinan untuk membuatnya menggunakan WSL melalui sistem operasi Windows (referensi: [WSL in Win10](https://docs.microsoft.com/en-us/windows/wsl/install)).

## Instalasi *Docker*

Sebelum melakukan instalasi DOMjudge, kita perlu *setup* terlebih dahulu *Docker* di mesin yang akan kita gunakan. Untuk memudahkan, berikut *script* yang bisa digunakan di terminal Linux yang akan otomatis memasang *Docker* ke sistem. Sebelum menjalankan *script*, perlu menambahkan package curl jika terjadi *error* ketika menjalankan *script* untuk menambahkan Docker ke sistem. (*Credit to*: Abdurrohman Al Fatih)

sudo apt install curl



(! command -v docker &> /dev/null) && for i in 1 2 3 4 5; do (echo "installing docker" && curl -fsSLhttps://get.docker.com -o get-docker.sh && sudo chmod +x get-docker.sh && sudo ./get-docker.sh &>/dev/null && rm get-docker.sh && echo "docker installed" && command -v docker &> /dev/null ) &&break || sleep 10; done

## *Setup* DOMJudge

Instalasi DOMjudge *via Docker* pada dokumentasi ini bersumber pada [hub.docker.com.](https://hub.docker.com/r/domjudge/domserver/) Jika ada yang kurang jelas di dalam dokumentasi ini, bisa langsung membaca dokumentasi yang ada di *link* tersebut.

### *Setup Database* Peserta

Hal yang pertama yang perlu dipersiapkan adalah sebuah *database* MySQL/MariaDB, cara termudah untuk membuatnya adalah dengan menjalankan MariaDB *Docker* *container* dengan menjalankan *script* berikut.

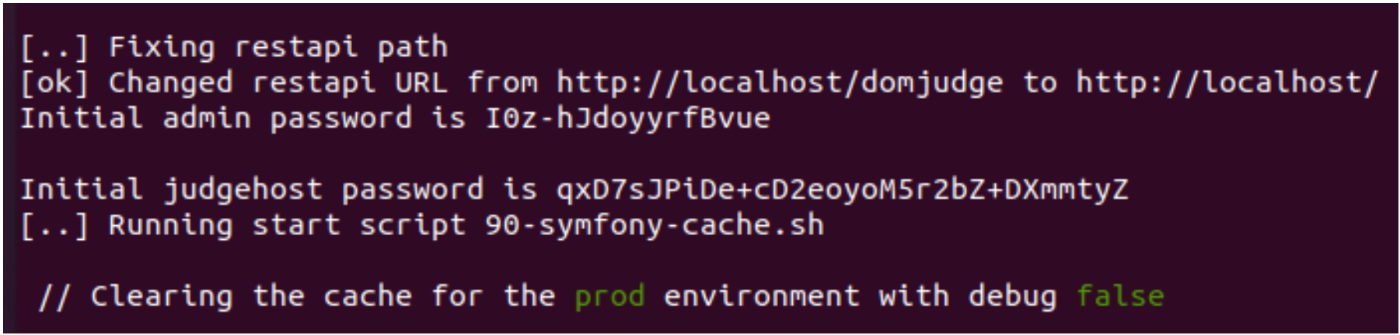
docker run -it --name dj-mariadb -e MYSQL\_ROOT\_PASSWORD=rootpw -eMYSQL\_USER=domjudge -e MYSQL\_PASSWORD=djpw -e MYSQL\_DATABASE=domjudge-p 13306:3306 mariadb --max-connections=1000



### *Setup* DOMserver *container*

Setelah sukses menyiapkan *database*, hal yang perlu dilakukan selanjutnya ialah membuat DOMserver *container*. Jalankan *script* berikut pada terminal Linux. Perlu diketahui bahwa terdapat beberapa *environment variables* yang dapat ditambahkan di perintah yang akan kita jalankan. Sebagai contoh, kita akan menambahkan opsi zona waktu ke *command* penambahan DOMserver dengan cara menambahkan opsi “-e CONTAINER\_TIMEZONE=Asia/Jakarta”. Opsi lengkap yang bisa digunakan dapat dilihat di [hub.docker.com.](https://hub.docker.com/r/domjudge/domserver/)  


docker run --link dj-mariadb:mariadb -it -e MYSQL\_HOST=mariadb -eMYSQL\_USER=domjudge -e MYSQL\_DATABASE=domjudge -e MYSQL\_PASSWORD=djpw -e MYSQL\_ROOT\_PASSWORD=rootpw -e CONTAINER\_TIMEZONE=Asia/Jakarta -p 12345:80 --name domserverdomjudge/domserver:latest



Perlu diperhatikan bahwa *port* yang digunakan sebagai DOMserver adalah 12345. Setelah menjalankan *script* tersebut, *initial password* untuk Admin dan pengguna Judgehost seharusnya ditampilkan di terminal. Jika tidak muncul, jalankan *script* berikut di terminal. Setelah itu, kita dapat mengakses *user interface* dari DOMjudge pada <http://localhost:12345/>.

docker exec -it domserver cat /opt/domjudge/domserver/etc/initial\_admin\_password.secret

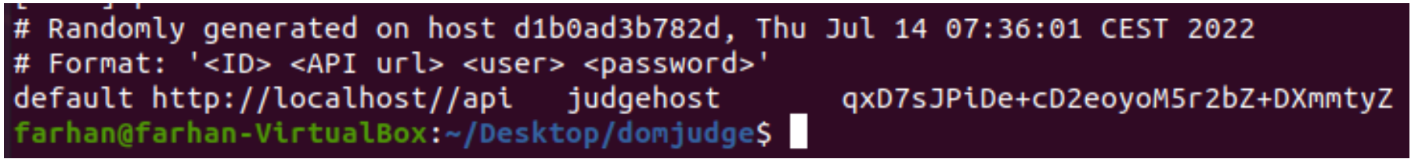
### *Setup* Judgehost *container*

Selanjutnya, kita akan menjalankan *script* yang digunakan untuk menjalankan Judgehost pada *Docker*.

docker run -it --privileged -v /sys/fs/cgroup:/sys/fs/cgroup:ro --name judgehost-0 --linkdomserver:domserver --hostname judgedaemon-0 -e DAEMON\_ID=0 -e JUDGEDAEMON\_USERNAME=<user> -e JUDGEDAEMON\_PASSWORD=<password>domjudge/judgehost:latest

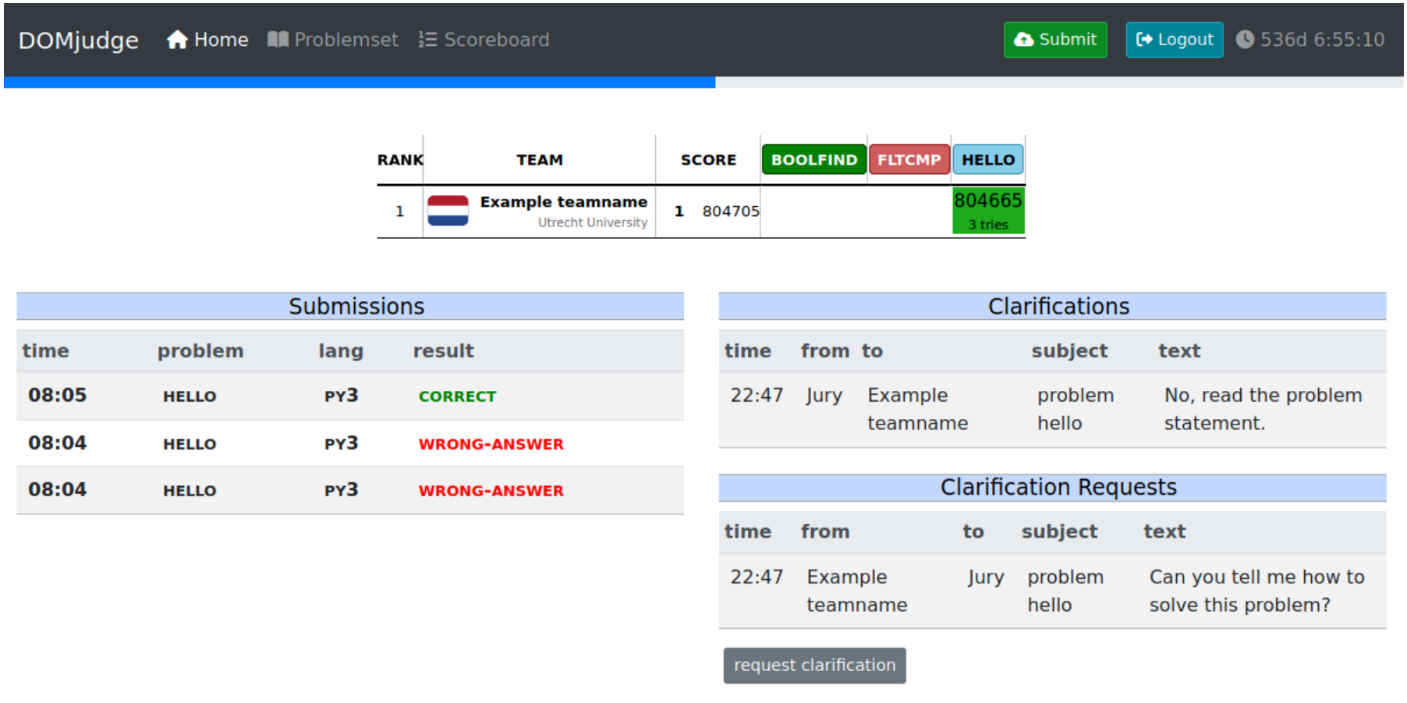
Perlu diketahui *<user>* dan *<password>* pada *script* tersebut dapat dilihat dengan menjalankan *script* berikut.

docker exec -it domserver cat /opt/domjudge/domserver/etc/restapi.secret





Setelah sukses melakukan segala proses di atas, DOMjudge telah dapat digunakan beserta Judgehost yang sudah bisa menerima *submission* dari sebuah *problem* yang dibuat. Sebagai contoh, berikut percobaan *submit* dan *upload* jawaban ke DOMjudge.



Untuk menambahkan Judgehost, kita hanya perlu menulis ulang *command* yang digunakan untuk menambahkan Judgehost

Tata cara dalam bentuk video dapat dilihat pada tautan <https://youtu.be/MqdDEY_2HBw>

# *Setup Java Virtual Memory* (JVM) dan *Memstack* pada DOMJudge

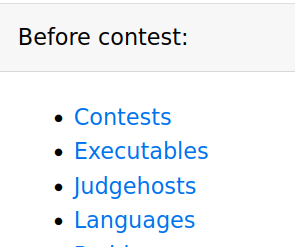
## Java pada Pemrograman Kompetitif

Di bahasa pemrograman Java, eksekusi program membutuhkan sesuatu yang bernama *Java Virtual Machine* (JVM). JVM berfungsi sebagai *runtime*/tempat bagi *code* Java untuk berjalan. Pada lomba pemrograman, tiap soal memiliki *memory limit* untuk setiap eksekusi solusi peserta. Untuk eksekusi solusi dengan Java, JVM akan memakan sebagian memori sehingga *memory limit* ini akan habis digunakan untuk JVM saja. Akibatnya, hanya tersisa sedikit memori untuk program yang dibuat peserta, sehingga akan memberikan kerugian bagi pengguna Java dibandingkan pengguna bahasa lain.

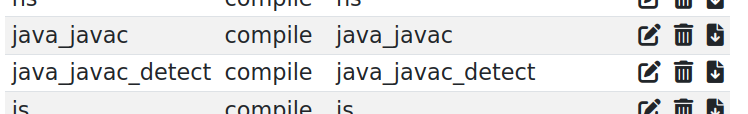
## Solusi Masalah JVM dan *Memstack* pada DOMjudge

Untuk mengatasi masalah tersebut, dapat dilakukan modifikasi batas memori yang digunakan untuk JVM dan *stack* pada *script* kompilasi Java di *website* DOMjudge sendiri. Tahapannya adalah sebagai berikut.

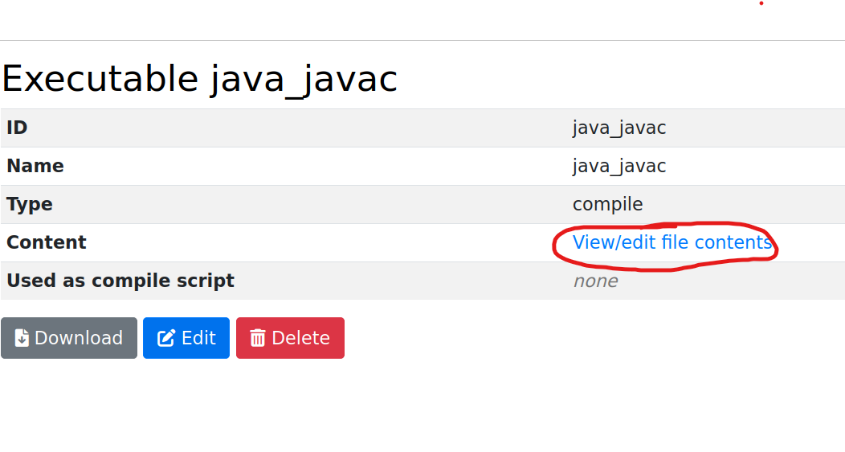
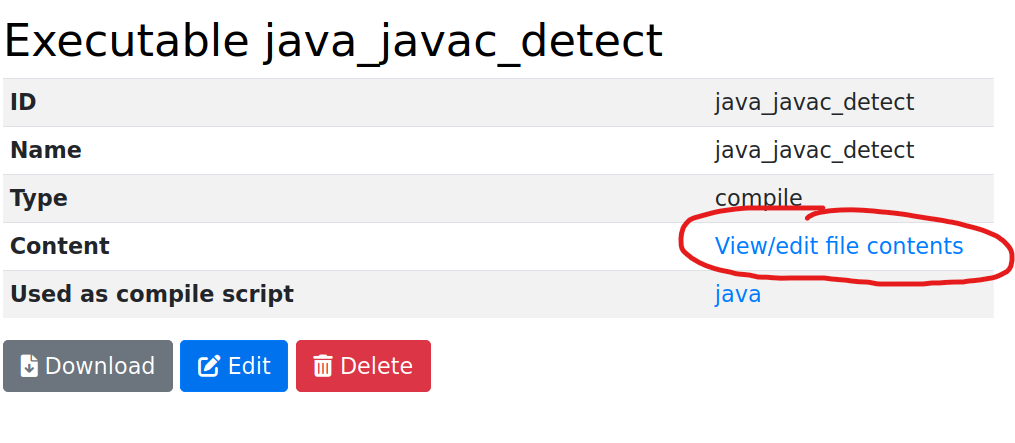
1. Pergi ke halaman “*Executables*” untuk mengakses *script compile* tersebut.



1. Di dalamnya, terdapat *script-script* khusus yang digunakan oleh DOMjudge, salah satunya adalah “javac”. Batas memori untuk JVM dan *stack* dapat diatur pada dua *file*, yaitu “java\_javac” dan “java\_javac\_detect”.



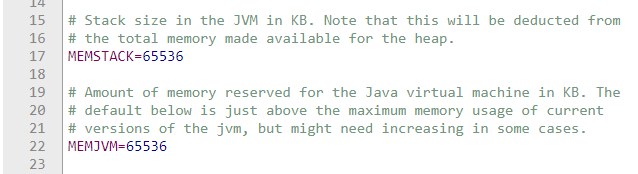
1. Klik “View/edit file contents” pada keduanya.

1. Bagian yang menyimpan informasi memori JVM dan *stack* tersebut dapat ditemukan di file run. Inilah *file* yang akan diedit.

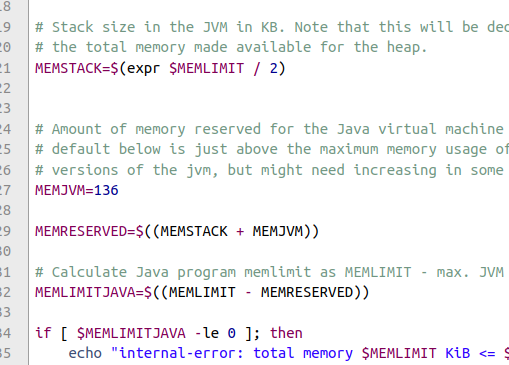

1. Di dalam file run tersebut, terdapat variabel MEMJVM dan MEMSTACK. Nilai *default* yang diberikan DOMjudge adalah 65536 untuk keduanya.



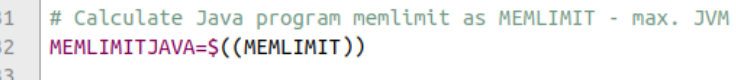
1. Ubah nilai tersebut, kemudian tekan “*Save files*” atau “*Submit*” di bagian paling bawah setelahnya. Lakukan ini untuk kedua *file* run yang terdapat di *script* “java\_javac” dan “java\_javac\_detect”. Nilai penggantinya akan dibahas di subbab selanjutnya.

## Catatan Hasil Eksplorasi untuk JVM dan *Memstack* pada DOMjudge

Solusinya hanya dengan mengubah baris:



menjadi



pada kedua script run di executable java\_javac dan java\_javac\_detect.

Dengan setup tersebut, berikut hasil eksekusi program:

* Solusi rekursif (faktorial) dengan kedalaman 10^3 memakan memori 11 MB.
* Solusi rekursif (faktorial) dengan kedalaman 10^4 memakan memori 11 MB.
* Solusi rekursif (faktorial) dengan kedalaman 10^5 memakan memori 16 MB.
* Solusi rekursif (faktorial) dengan kedalaman 10^6 memakan memori 37 MB.
* Solusi dengan inisialisasi array int sebesar 10^3 memakan memori sebesar 54 MB.
* Solusi dengan inisialisasi array int sebesar 10^4 memakan memori sebesar 54 MB.
* Solusi dengan inisialisasi array int sebesar 10^5 memakan memori sebesar 55 MB.
* Solusi dengan inisialisasi array int sebesar 10^6 memakan memori sebesar 60 MB.
* Solusi dengan inisialisasi array int sebesar 10^7 memakan memori sebesar 94 MB.

Pada DOMjudge yang baru, berikut hasil eksekusi program:

Solusi dengan arrayList int sebesar N diisi dengan nilai 10^9:

* N=10^3 memakan memori sebesar 14.10 MB.
* N=10^4 memakan memori sebesar 14.35 MB.
* N=10^5 memakan memori sebesar 17 MB.
* N=10^6 memakan memori sebesar 43 MB.
* N=10^7 memakan memori sebesar 337.53 MB.

Solusi dengan array int[] sebesar N diisi dengan nilai 109:

* N=10^3 memakan memori sebesar 10.59 MB.
* N=10^4 memakan memori sebesar 10.60 MB.
* N=10^5 memakan memori sebesar 10.97 MB.
* N=10^6 memakan memori sebesar 14.17 MB.
* N=10^7 memakan memori sebesar 49.73 MB.

Perbandingan dengan C++:

* Solusi dengan vector<int> sebesar N=10^7 diisi dengan nilai 10^9 memakan memori sebesar 64.40 MB.
* Solusi dengan array int sebesar N=10^7 diisi dengan nilai 10^9 memakan memori sebesar 38.37 MB.

Catatan: dokumen ini dibuat pada tanggal 16 Juli 2022 dan dites pada lomba.komputasi.id yaitu DOMjudge yang diperkirakan versi 8.1.0. Untuk tutorial dalam bentuk video dapat diakses pada [https://youtu.be/O1BCR4IYhM8.](https://youtu.be/O1BCR4IYhM8)

# Cara Mengunggah Soal dan *Test-cases* ke DOMjudge

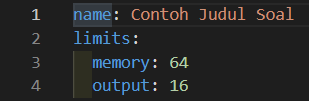
## Format *File*

Untuk mengunggah soal dan *test-cases* ke DOMjudge secara otomatis, perlu disiapkan format *file* sebagai berikut.

|  |
| --- |
| ├── data  │ └── secret  │ ├── 1.in  │ ├── 1.ans  │ ├── 1.desc  │ ├── 2.in  │ ├── 2.ans  │ ├── 3.in  │ └── 3.ans  ├── problem\_statement  │ └── problem.pdf  └── problem.yaml |

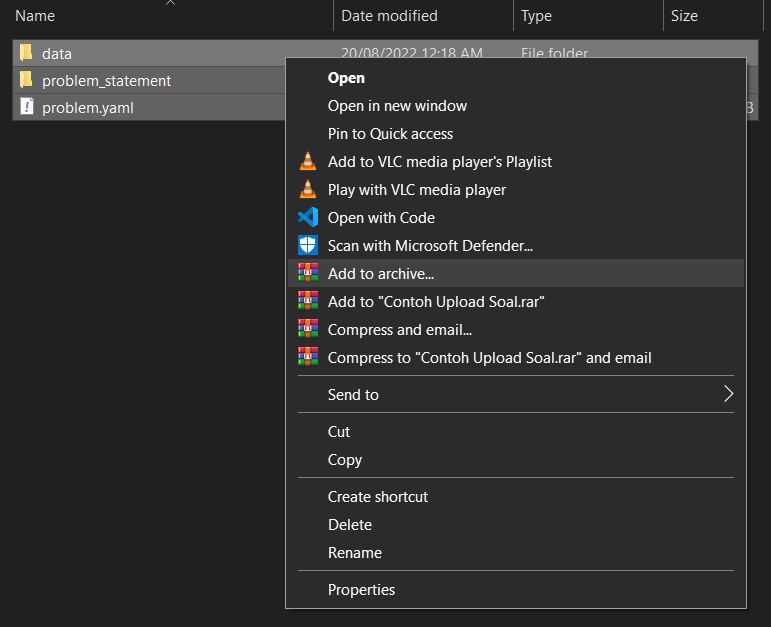
Tiap pasangan *file* in dan ans adalah masukan dan keluaran dari sebuah *test-case*. Tiap *test-case* juga dapat dilengkapi dengan deskripsi pada *file* desc, tetapi hal ini bersifat *optional*. Deskripsi ini nanti akan terlihat pada halaman detail *test-case* pada DOMjudge. *File* in, ans, dan desc ini berbentuk *plain text file*.

*File* problem.pdf merupakan *problem text* yang akan diberikan ke peserta. Sedangkan problem.yaml sendiri adalah spesifikasi dari tiap *problem*, yang dapat berisi (tanpa menutup kemungkinan spesifikasi lainnya) nama *problem* dan *memory limit* dalam MB. Spesifikasi lengkap dapat dilihat pada [dokumentasi terkait](https://icpc.io/problem-package-format/spec/problem_package_format). Spesifikasi *problem* yang tidak didefinisikan pada problem.yaml akan bernilai sama dengan nilai *default*. Contoh isi dari problem.yaml dapat dilihat sebagai berikut.



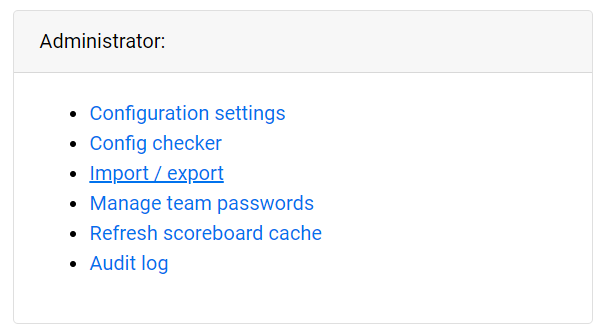
*Limit* yang dapat dispesifikasikan pada problem.yaml adalah *memory limit* dan *output limit*. Spesifikasi *time limit* hanya dapat dilakukan pada *interface* DOMjudge secara langsung, sehingga *time limit* awal akan selalu di-*set* dengan nilai *default*, yaitu 10 detik.

Setelah menyiapkan *file* yang dibutuhkan dengan ketentuan di atas, simpan *file-file* tersebut dalam bentuk zip. Nama *file* zip tidak mempunyai ketentuan tertentu. Namun, nama *file* zip ini tersimpan di DOMjudge sebagai externalid dari *problem*, sehingga nama *file* zip diunggah tidak boleh sama dengan nama *file* zip yang sudah diunggah sebelumnya. Keterbatasan penamaan ini tidak berlaku pada nama dari *problem* itu sendiri. Suatu *problem* dapat saja memiliki nama yang sama dengan *problem* lainnya, asalkan penamaan *file* zipnya berbeda. Jika nama *problem* tidak didefinisikan pada problem.yaml, maka namanya akan diambil dari nama *file* zip tanpa spasi.

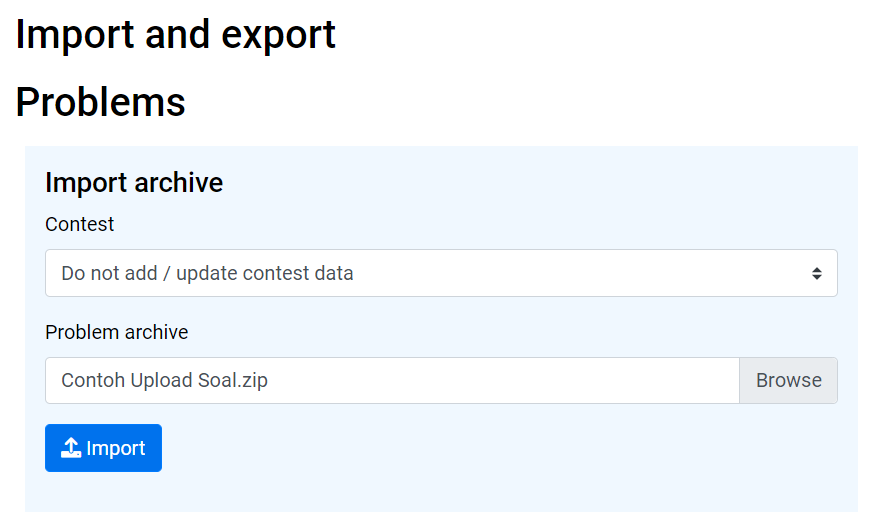


## Pengunggahan *Problem*

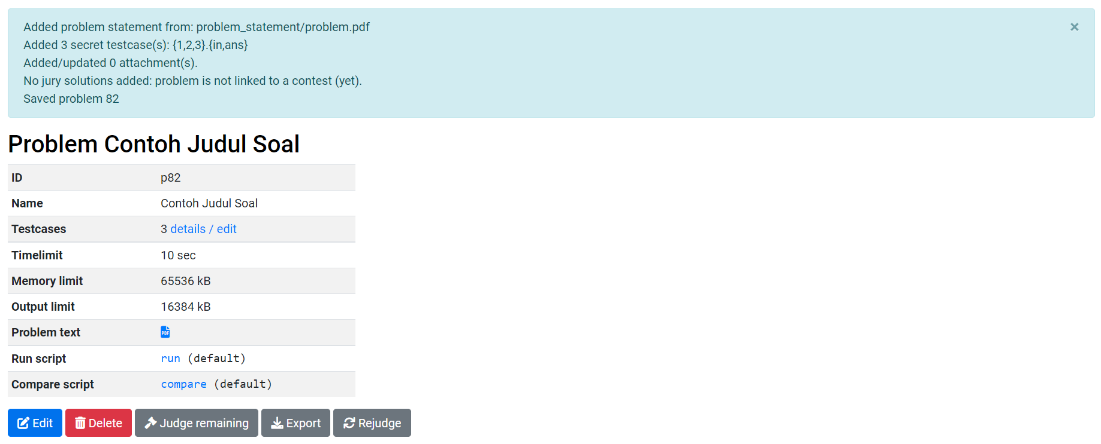
Dari halaman awal DOMjudge, klik “*Import / export*” pada bagian *Administrator*.



Di bagian “*Import archive*”, pilih “*Do not add / update contest data*” dan unggah *file* zip yang sudah dibuat sebelumnya. Kemudian klik “*Import*”.

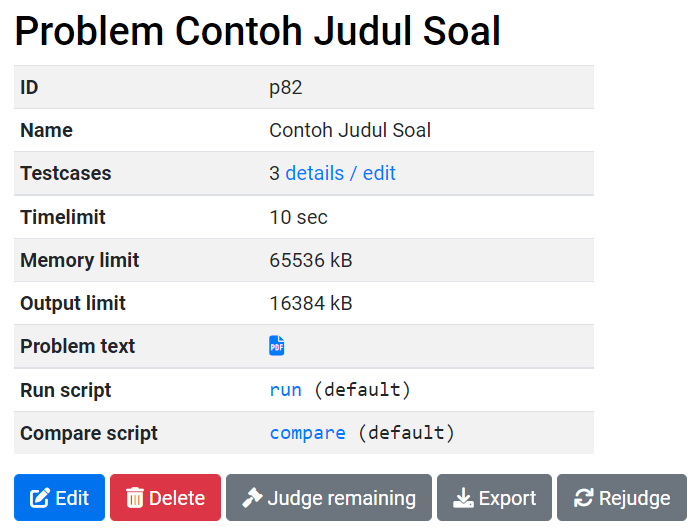


DOMjudge akan menunjukkan halaman berikut, yang berarti pengunggahan soal sudah berhasil.

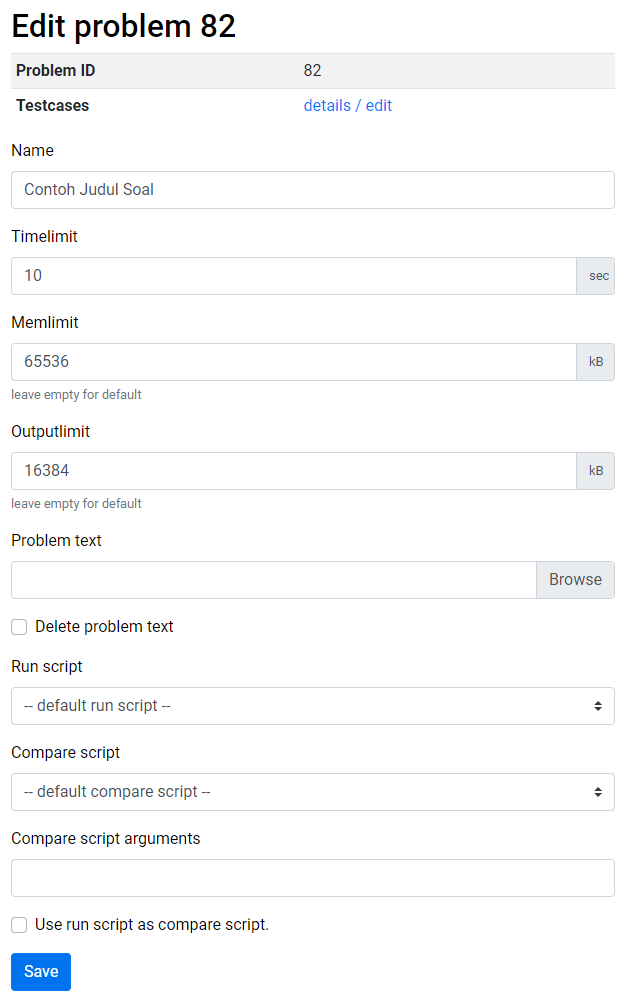


## Ubah Spesifikasi Soal yang Telah Diunggah

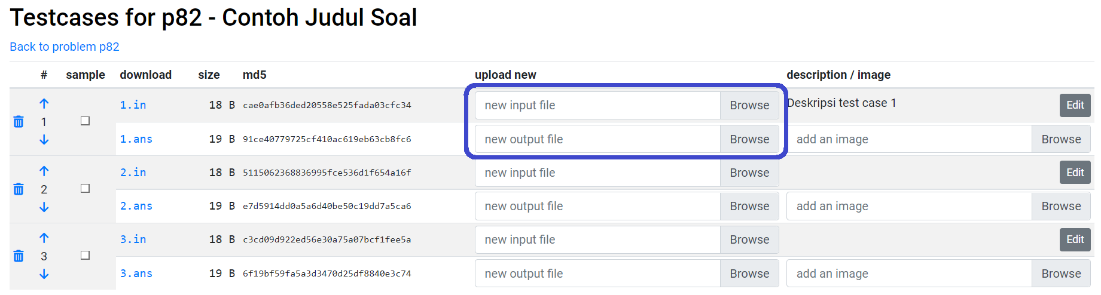
Jika spesifikasi soal perlu diubah setelah pengunggahan, klik tombol “*Edit*” pada halaman detail *problem*.



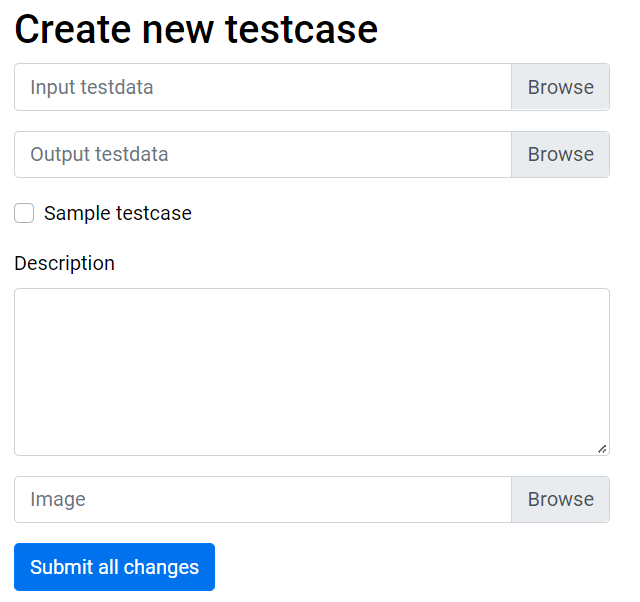
Ubah spesifikasi *problem* sesuai dengan yang diinginkan. Kemudian klik tombol “*Save*” untuk menyimpannya.



*Test-case* juga dapat diubah dengan mengklik “*details / edit*” yang akan membuka halaman detail *test-case*.



Pengubahan dapat dilakukan dengan mengganti *file test-case* yang sudah ada, dengan mengunggah *file* baru ke bagian yang ditandai pada gambar di atas. Selain itu, dapat juga dilakukan penambahan *test-case* baru pada bagian bawah halaman yang sama, yaitu pada bagian yang ditunjukkan gambar berikut.



Setelah melakukan pengubahan, klik “*Submit all changes*”.

Catatan: Tutorial pengunggahan soal ke DOMjudge ini juga tersedia dalam bentuk video yang dapat diakses dari <https://youtu.be/AeR7JXCMh2E>.

# Pembuatan Soal dengan *Custom Checker* di DOMjudge

## Pendahuluan

Pada lomba pemrograman, kadang terdapat soal yang *output valid*-nya terdapat lebih dari satu. Contohnya pada [soal berikut](https://codeforces.com/problemset/problem/1473/A), *output*-nya boleh “YES”, “yes”, atau “yEs”. Untuk menilai solusi peserta, tidak bisa hanya sekedar membandingkan kesamaannya dengan satu *file output* saja. Akan dimanfaatkan sebuah program khusus, *custom checker*, yang akan menilai *output* dari solusi peserta benar atau tidak.

Sebagai bagian dari juri, untuk persiapan soal dengan *custom checker* di DOMjudge, akan dimanfaatkan *library*, yaitu [testlib.h](https://codeforces.com/blog/entry/18455). Namun, di kasus ini akan digunakan “ekstensinya” ke DOMjudge, yaitu [testlib-for-domjudge](https://github.com/cn-xcpc-tools/testlib-for-domjudge), karena terdapat masalah saat pengintegrasian custom checker menggunakan testlib yang asli dengan DOMjudge.

Catatan: dokumen ini dibuat pada tanggal 16 Juli 2022 dan dites pada lomba.komputasi.id yaitu DOMjudge yang diperkirakan versi 8.1.0. Untuk tutorial dalam bentuk video dapat diakses pada <https://youtu.be/fEo3S_4LYV8>.

## Contoh Soal Kongkret

Diberikan dua bilangan bulat dan , keluarkan dua bilangan bulat dan yang memiliki jumlah sama atau lebih besar daripada . Terdapat juga batasan tambahan, yaitu . Jika terdapat lebih dari satu solusi yang *valid*, keluarkan salah satunya.

Solusinya jelas bahwa peserta bisa saja mengeluarkan kembali dan yang diberikan. Namun, terdapat banyak solusi lain yang mungkin (seperti mengeluarkan “10 10” setiap saat).

## Pembuatan *Custom Checker*

Seperti yang disebutkan sebelumnya, akan digunakan testlib.h untuk pembuatan interaktor oleh juri. Untuk panduan/*advice* dalam penulisan *custom checker*, dapat dilihat pada [blog ini](https://codeforces.com/blog/entry/18431). Berikut tahap-tahapnya:

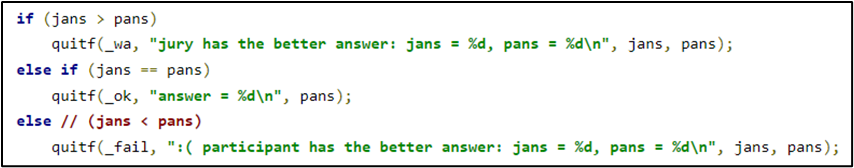
* Download file testlib.h yang ada pada folder pdf dokumentasi ini (atau langsung dari [repositori](https://github.com/cn-xcpc-tools/testlib-for-domjudge) githubnya).
* Pada direktori yang sama, buat file baru, yaitu checker yang akan kita buat, dan namakan checker.cpp.
* Dalam checker.cpp, *include* *library* testlib.h nya dan tulislah algoritma pengecekannya. Dengan testlib, terdapat 3 variabel global yang bisa kita dapatkan informasinya yaitu inf (input tc), ouf (output/jawaban peserta), dan ans (jawaban juri). Untuk contoh soal di atas, berikut *checker*-nya:

|  |
| --- |
| #include "testlib.h" #include <iostream> using namespace std;   int main(int argc, char\* argv[]) {  registerInteraction(argc, argv);    int a = inf.readInt();  int b = inf.readInt();    int c = ouf.readInt(-10, 10); // jika di luar [-10, 10], maka gagal  int d = ouf.readInt(-10, 10);    if ((c + d) >= (a + b))  quitf(\_ok, "Correct: c + d is greater than or equal to a + b.");  else  quitf(\_wa, "Wrong: c + d is smaller than a + b.");  } |

## Pembuatan *Test-cases*

Ada beberapa poin penting dari cara kerja kasus tes pada soal interaktif, yaitu:

* Seperti biasa, satu kasus tes terdiri dari dua *file* yaitu *input* dan *output* (jawaban) yang memiliki nama yang sama. *File input* memiliki ekstensi .in dan *file output* memiliki ekstensi .ans.
* Karena *verdict* solusi peserta langsung ditentukan di checker jurinya (seperti quitf(\_wa) atau quitf(\_ok) pada program di atas), maka *file output* tidak terlalu penting pada kasus tes. Di contoh soal ini, *file output* juri akan diabaikan dalam penilaian solusi, tetapi tetap harus disertakan saat pembuatan kasus tes. *File output* dapat dibiarkan kosong atau diisi dengan string acak.
* *File output* juri boleh saja diisi dengan benar dan dimanfaatkan pada *checker* (dengan memakai variabel global ans) apabila ingin membandingkan isinya dengan jawaban peserta seperti di blog panduan sebelumnya. Contohnya, dalam mencari solusi optimal:



Untuk contoh soal di atas, *input* yang diberikan ke interaktor juri adalah bilangan bulat dan . Berikut contoh kasus-kasus tes yang dapat dibuat:

1. *Test-case* 1

|  |  |
| --- | --- |
| 1.in | 1.ans |
| 0 0 | 1 2 |

1. *Test-case* 2

|  |  |
| --- | --- |
| 2.in | 2.ans |
| 10 10 | 10 10 |

1. *Test-case* 3

|  |  |
| --- | --- |
| 3.in | 3.ans |
| -10 -10 | -9 10 |

## Mengunggah Soal ke DOMJudge

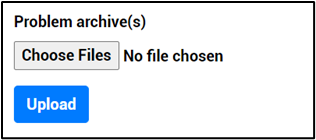
Setelah menyiapkan PDF soal, *program checker*, dan kasus-kasus tes, berikutnya akan dilakukan pengunggahan ke DOMjudge. Namun, sebelumnya, file-file soal ini akan dikemas ke dalam sebuah zip mengikuti [format package soal DOMjudge](https://www.domjudge.org/docs/manual/8.0/problem-format.html). Untuk contoh soal di atas, berikut isi zipnya (sederhana saja, informasi tambahan boleh disediakan jika ingin):

|  |
| --- |
| ├── data  │ └── secret  │ ├── 1.in  │ ├── 1.ans  │ ├── 2.in  │ ├── 2.ans  │ ├── 3.in  │ └── 3.ans  ├── output\_validators  │ └── validate  │ ├── checker.cpp  │ └── testlib.h  ├── problem\_statement  │ └── problem.pdf  └── problem.yaml |

Perlu diingat bahwa *file* problem.yaml juga wajib berisikan nama dan pengaturan validator seperti berikut:

|  |
| --- |
| name: Sum ABCD  validation: custom |

Setelah zip selesai disiapkan, unggah *file* zip tersebut di DOMjudge di halaman “*Problems*” pada bagian berikut:



# Pembuatan Soal Interaktif di DOMJudge

## Pendahuluan

Pada lomba pemrograman, kadang terdapat soal yang bertipe “interaktif”. Dalam soal bertipe interaktif, juri menulis sebuah program khusus—interaktor, yang memungkinkan program peserta dan interaktor juri untuk saling bertukar data nantinya. Peserta/juri menentukan apa yang dicetak berdasarkan "sejarah pertukaran data" tersebut. Sebagai contoh, dapat dilihat pada [blog ini](https://codeforces.com/blog/entry/45307).

Sebagai bagian dari juri, untuk persiapan soal interaktif di DOMjudge, akan dimanfaatkan suatu *library*, yaitu [testlib.h](https://codeforces.com/blog/entry/18455) agar pembuatan interaktornya lebih mudah. Namun, di kasus ini kita akan menggunakan “ekstensi” ke DOMjudgenya, yaitu [testlib-for-domjudge](https://github.com/cn-xcpc-tools/testlib-for-domjudge), karena terus terjadi masalah saat pengintegrasian interaktor menggunakan testlib yang asli dengan DOMjudge.

Catatan: dokumen ini dibuat pada tanggal 16 Juli 2022 dan dites pada lomba.komputasi.id yaitu DOMjudge yang diperkirakan versi 8.1.0. Untuk tutorial dalam bentuk video dapat diakses pada <https://youtu.be/FO9mhdx-tqI>.

## Contoh Soal Kongkret

Juri memilih sebuah bilangan bulat dalam interval dan peserta harus menulis sebuah program untuk menebaknya. Peserta dapat bertanya pada juri maksimal 50 pertanyaan. Dalam setiap pertanyaan, peserta mengatakan sebuah nomor dalam interval , dan akan dibalaskan oleh juri sebuah angka berdasarkan nomor tersebut:

* 1, jika nomor tersebut (oleh peserta) sama dengan bilangan bulat yang dipilih juri.
* 0, jika nomor tersebut lebih kecil dari bilangan juri.
* 2, jika nomor tersebut lebih besar dari bilangan juri.

## Pembuatan Interaktor

Seperti yang disebutkan sebelumnya, akan digunakan testlib.h untuk pembuatan interaktor oleh jurinya. Berikut tahap-tahapnya:

* Download file testlib.h yang ada pada folder PDF dokumentasi ini (atau langsung dari repositori githubnya).
* Pada direktori yang sama, buat file baru, yaitu interaktor yang akan kita buat, dan namakan interactor.cpp.
* Dalam interactor.cpp, *include library* testlib.h dan tulislah algoritma interaktornya. Dengan testlib, terdapat 3 variabel global yang bisa kita dapatkan informasinya yaitu inf (input tc), ouf (output/jawaban peserta), dan ans (jawaban juri). Untuk contoh soal di atas akan terlihat seperti berikut:

|  |
| --- |
| #include "testlib.h"  #include <iostream>  using namespace std;   int main(int argc, char\* argv[]) {  registerInteraction(argc, argv);  int n = inf.readInt(); // bilangan juri yang ingin ditebak  cout.flush(); // agar output tidak stuck di suatu buffer    int left = 50;  bool found = false;  while(left > 0 && !found) {  left --;  int a = ouf.readInt(1, 1000000000); // membaca nomor dari pertanyaan peserta  if(a < n)  cout << 0 << endl;  else if(a > n)  cout << 2 << endl;  else  cout << 1 << endl, found = true;  cout.flush();  }  if(!found)  quitf(\_wa, "couldn't guess the number with 50 questions");  quitf(\_ok, "guessed the number with %d questions!", 50 - left);  } |

## Pembuatan *Test-cases*

Ada beberapa poin penting dari cara kerja kasus tes pada soal interaktif, yaitu:

* Berbeda dengan soal pemrograman biasa, *input* kasus tes di sini akan dimasukkan ke dalam program interaktor juri (bukan solusi peserta).
* Seperti biasa, dalam satu kasus tes terdiri dari dua *file* yaitu *input* dan *output* (jawaban) yang memiliki nama yang sama. *File input* memiliki ekstensi .in dan *file output* memiliki ekstensi .ans.
* Karena *verdict* solusi peserta langsung ditentukan di interaktor jurinya (seperti quitf(\_wa) atau quitf(\_ok) pada program di atas), maka *file output* tidak terlalu penting pada kasus tes. Dapat dilihat pada interaktor juga, kita tidak menggunakan variabel global ans oleh testlib. *File output* juri akan diabaikan dalam penilaian solusi, tetapi tetap harus disertakan saat pembuatan kasus tes. File output dapat dibiarkan kosong atau diisi dengan string acak.

Untuk contoh soal di atas, input yang diberikan ke interaktor juri yaitu hanya 1 bilangan bulat (bilangan yang akan dicoba tebak oleh peserta). Berikut contoh kasus-kasus tes yang dapat dibuat:

1. *Test-case* 1

|  |  |
| --- | --- |
| 1.in | 1.ans |
| 238913 | wkwkwk, hello world! |

1. *Test-case* 2

|  |  |
| --- | --- |
| 2.in | 2.ans (kosong, tidak apa-apa) |
| 57891 |  |

## Mengunggah Soal ke DOMJudge

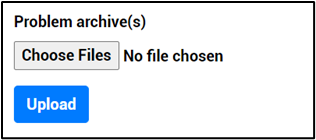
Setelah menyiapkan PDF soal, program interaktor, dan kasus-kasus tes, kemudian saatnya mengunggahnya ke DOMjudge. Namun, sebelumnya, *file*-*file* soal ini akan dikemas dalam sebuah zip mengikuti format *package* soal DOMjudge. Untuk contoh soal di atas, berikut isi zipnya (sederhana saja, informasi tambahan boleh disediakan jika ingin):

|  |
| --- |
| ├── data  │ └── secret  │ ├── 1.in  │ ├── 1.ans  │ ├── 2.in  │ └── 2.ans  ├── output\_validators  │ └── validate  │ ├── interactor.cpp  │ └── testlib.h  ├── problem\_statement  │ └── problem.pdf  └── problem.yaml |

Perlu diingat bahwa *file* problem.yaml juga wajib berisikan nama dan pengaturan validator seperti berikut:

|  |
| --- |
| name: Pencarian Biner Interaktif  validation: custom interactive |

Setelah zip selesai disiapkan, unggah *file* zip tersebut di DOMjudge di halaman “*Problems*” pada bagian berikut:



## Contoh Solusi Peserta

Berikut solusi peserta yang bisa dicoba submit pada soalnya untuk tes:

1. C++

|  |
| --- |
| #include <iostream> using namespace std;   int main() {  int l = 1, r = 1e9;  while (l <= r) {  int mid = (l + r)/2;  cout << mid << endl;    int ans;  cin >> ans;    if (ans == 0) {  l = mid + 1;  } else if (ans == 2) {  r = mid - 1;  } else {  break;  }  } } |

1. Python

|  |
| --- |
| import sys   l, r = 1, 10\*\*9 while l <= r:  mid = (l + r)//2  print(mid)  sys.stdout.flush()    ans = int(input())  if ans == 0:  l = mid + 1  elif ans == 2:  r = mid - 1  else:  break |

# Beberapa Isu Teknis Lanjut

## Pengaturan Jam pada DOMserver

Untuk mengatur zona waktu yang digunakan pada DOMserver, hanya perlu menambahkan *environment variable* CONTAINER\_TIMEZONE dengan nilai zona waktu yang kita inginkan. Contohnya seperti “-e CONTAINER\_TIMEZONE=Asia/Jakarta”. Contoh dengan *command* lengkap untuk menambahkan DOMserver *container*.

|  |
| --- |
| docker run --link dj-mariadb:mariadb -it -e MYSQL\_HOST=mariadb -eMYSQL\_USER=domjudge -e MYSQL\_DATABASE=domjudge -e MYSQL\_PASSWORD=djpw -e MYSQL\_ROOT\_PASSWORD=rootpw -e CONTAINER\_TIMEZONE=Asia/Jakarta -p 12345:80 --name domserverdomjudge/domserver:latest |

## Penambahan Judgehost pada DOMjudge

Penambahan Judgehost bisa saja dibutuhkan jika terdapat kasus di mana kontes dilaksanakan dalam rentang waktu tertentu dan ada kemungkinan peserta lomba mengakses atau mengunggah jawaban secara bersamaan di satu waktu. Penambahan Judgehost dapat membantu masalah ini agar jawaban tidak hanya dicek menggunakan satu Judgehost saja.

Untuk menambah Judgehost, kita hanya perlu menulis ulang *command* yang sama untuk menambah Judgehost dengan memberikan nilai atau nama yang berbeda pada opsi --name. Contoh jika kita ingin menambahkan 3 Judgehost *container*.

|  |
| --- |
| docker run -it --privileged -v /sys/fs/cgroup:/sys/fs/cgroup:ro --name judgehost-8.1.1-0 --link domserver-8.1.1-0:domserver-8.1.1-0 --hostname judgehost-8.1.1-0 -e DAEMON\_ID=0 -e JUDGEDAEMON\_USERNAME=<*username yang sama*> -e JUDGEDAEMON\_PASSWORD=<*password yang sama*> -e DOMSERVER\_BASEURL=http://domserver-8.1.1-0/ -e CONTAINER\_TIMEZONE=Asia/Jakarta domjudge/judgehost:8.1.1  docker run -it -d --privileged -v /sys/fs/cgroup:/sys/fs/cgroup:ro --name judgehost-8.1.1-1 --link domserver-8.1.1-0:domserver-8.1.1-0 --hostname judgehost-8.1.1-1 -e DAEMON\_ID=1 -e JUDGEDAEMON\_USERNAME=<*username yang sama*> -e JUDGEDAEMON\_PASSWORD=<*password yang sama*> -e DOMSERVER\_BASEURL=http://domserver-8.1.1-0/ -e CONTAINER\_TIMEZONE=Asia/Jakarta domjudge/judgehost:8.1.1  docker run -it -d --privileged -v /sys/fs/cgroup:/sys/fs/cgroup:ro --name judgehost-8.1.1-2 --link domserver-8.1.1-0:domserver-8.1.1-0 --hostname judgehost-8.1.1-2 -e DAEMON\_ID=2 -e JUDGEDAEMON\_USERNAME=<*username yang sama*> -e JUDGEDAEMON\_PASSWORD=<*password yang sama*> -e DOMSERVER\_BASEURL=http://domserver-8.1.1-0/ -e CONTAINER\_TIMEZONE=Asia/Jakarta domjudge/judgehost:8.1.1 |

## Isu *Test Case* Berlebih Spasi

***Test case* yang berlebih spasi dapat menimbulkan isu pada program Python**. Misalkan diberikan barisan bilangan bulat dan kita diminta menghitung jumlah bilangan-bilangan nya. Berkut contoh masukan yang berlebih spasi di akhir:

|  |
| --- |
| Masukan |
| 1 2 3 4 |

Program Python yang cukup umum ditulis oleh peserta yaitu dengan membaca input sebaris dan melakukan *split* berdasarkan karakter spasi seperti berikut:

|  |
| --- |
| nums = list(map(int, input().split(' ')))  print(sum(nums)) |

Namun, karena pada masukan *test case* terdapat ekstra spasi di belakang nya, maka input().split(' ') akan mengembalikan ['1', '2', '3', '4', '']. *String* kosong pada indeks terakhir tidak bisa dikonversi menjadi *integer* sehingga akan menimbulkan *runtime error*.

Untuk mencegah masalah tersebut, maka perlu hati-hati untuk tidak mencetak spasi berlebih saat pembuatan *test case* atau dapat menggunakan [*script* spesial](https://github.com/tsaqifammar/cp-scripts/tree/main/strip-whitespaces-tc) jika sudah terlanjur terbuat *test case* yang berlebih spasi.

## Isu *Test Case* Berakhiran Newline CRLF

Pada suatu berkas teks, terdapat dua jenis karakter *newline* berbeda tergantung sistem operasi nya. Pada sistem operasi Linux, karakter newline dapat direpresentasikan sebagai ‘\n’ atau disebut sebagai LF. Sedangkan pada sistem operasi Windows, karakter newline dapat direpresentasikan sebagai ‘\r\n` atau disebut sebagai CRLF. ***Test case* dari setiap soal yang ditambahkan pada DOMjudge disarankan memiliki akhiran *newline* bertipe LF**.

Contoh kasus yang dapat ditimbulkan oleh *test case* berakhiran CRLF yaitu misalkan diberikan input beberapa *string* dan kita ingin menghitung berapa banyak di antaranya yang bersifat palindrom (terbaca sama dari depan atau dari belakang). Berikut contoh *file* input dan bagaimana sketsa aktual nya secara *low-level* (dalam format bytes):

|  |  |
| --- | --- |
| Input | Bagaimana sebenarnya dalam format bytes |
| kakak  abccba  madam  racecar | kakak\r  abccba\r  madam\r  racecar\r |

Jika pengecekan palindrom dilakukan dengan melakukan *reverse* pada *string* awal dan mengecek apakah keduanya sama atau tidak, maka berikut yang dihasilkan jika menggunakan input() pada Python:

|  |  |  |
| --- | --- | --- |
| String | Reversed String | Sama? |
| kakak\r | \rkakak | Tidak |
| abccba\r | \rabccba | Tidak |
| madam\r | \rmadam | Tidak |
| racecar\r | \rracecar | Tidak |

Akibatnya, yang seharusnya banyak palindrom pada input adalah 4, kita malah mendapatkan 0.

Beberapa solusi untuk menyelesaikan masalah ini yaitu:

* Membuat *test case* dengan *script* [domjudge-simple-gen-tc](https://github.com/tsaqifammar/cp-scripts/tree/main/domjudge-simple-gen-tc) agar langsung berakhiran LF; atau
* Menjalankan [*script* spesial](https://github.com/tsaqifammar/cp-scripts/tree/main/replace-clrf-with-lf-on-tcs) pada kumpulan *test case* yang sudah terlanjur berakhiran CRLF. *Script* tersebut berfungsi untuk mengkonversi *newline* dari CRLF menjadi LF.