LEARNING MANAGEMENT SYSTEM TRAINING

VIRTUAL PROGRAMMING LAB

Hrvoje Leventić
hrvoje.leventic@ferit.hr

Osijek, June 13, 2019

AUTOMATIC GRADING OF PROGRAMMING ASSIGNMENTS

Often STDIN/STDOUT based workflow

Test cases

Online environment/compiler/IDE

Conform the code to the input and output

BENEFITS

Cross platform

Multilingual

Remotely accessible

Code tester

WHY USE AN AUTOGRADER

Student engagement

Accelerates marking

More time with students

Easy to help students

WHAT AUTOGRADERS EXIST?

There are a number of autograder options out there

Both commercial and free/open-source:

Notable mention:

repl.it

WHY VPL 1/2

Allows students to run code in browser

Validates student code against teacher-designed test cases

Provides instant feedback to students

Autogrades student submissions

Already linked to student credentials

WHY VPL 2/2

Validates student code against other submissions (Plagiarism Checker)

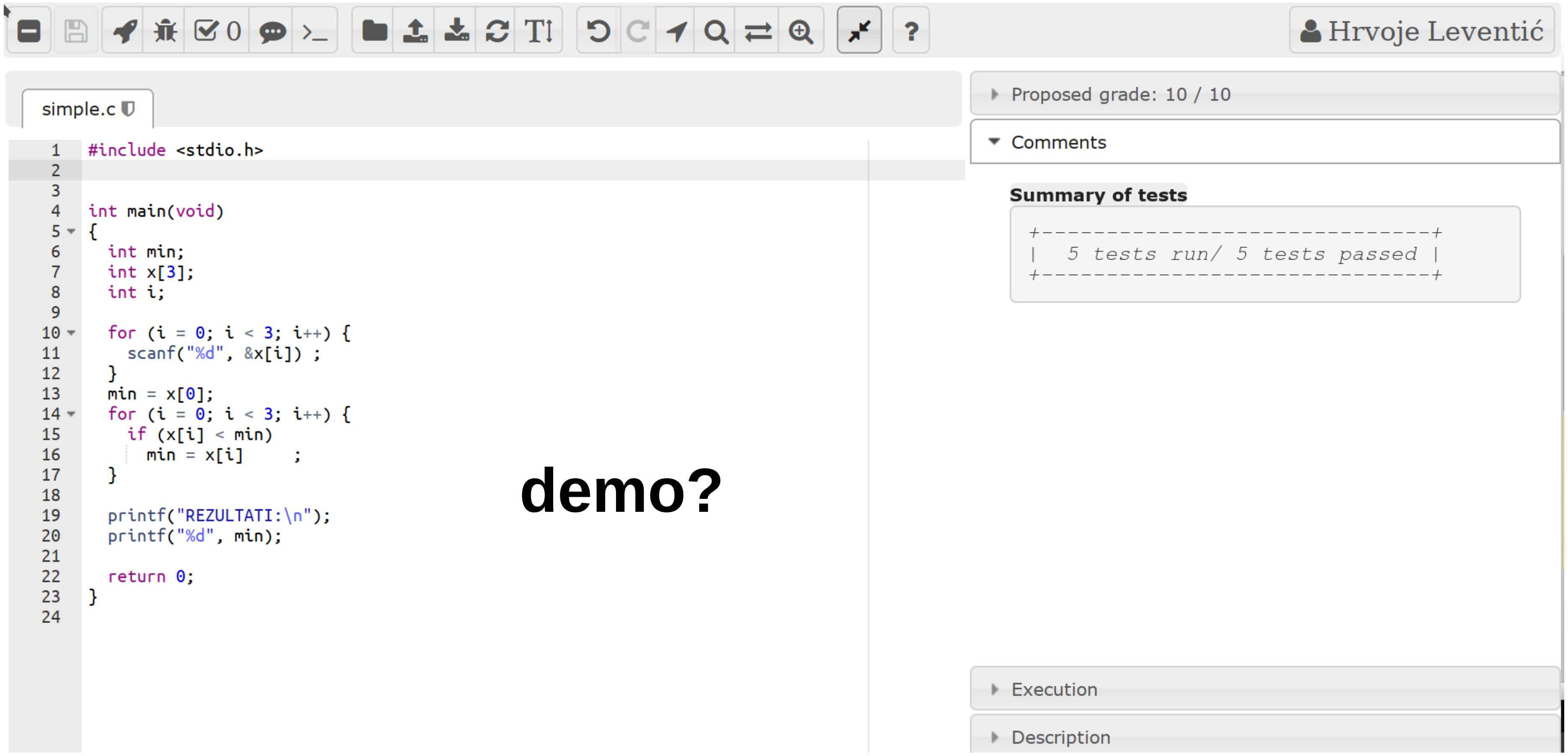
Grades go into Moodle Gradebook instantly

Transparency in Assessment

Support for multiple languages

Open source and in development since 2010

SO, HOW DOES IT WORK?



```
#include <stdio.h>


int main(void)
{
  int min;
  int x[3];
  int i;

  for (i = 0; i < 3; i++) {
    scanf("%d", &x[i]) ;
  }
  min = x[0];
  for (i = 0; i < 3; i++) {
    if (x[i] < min)
      min = x[i]     ;
  }

  printf("REZULTATI:\n");
  printf("%d", min);

  return 0;
}
```

Summary of tests

```
+------------------------------+
|  5 tests run/ 5 tests passed |
+------------------------------+
```

demo?

ANATOMY OF A TEST CASE

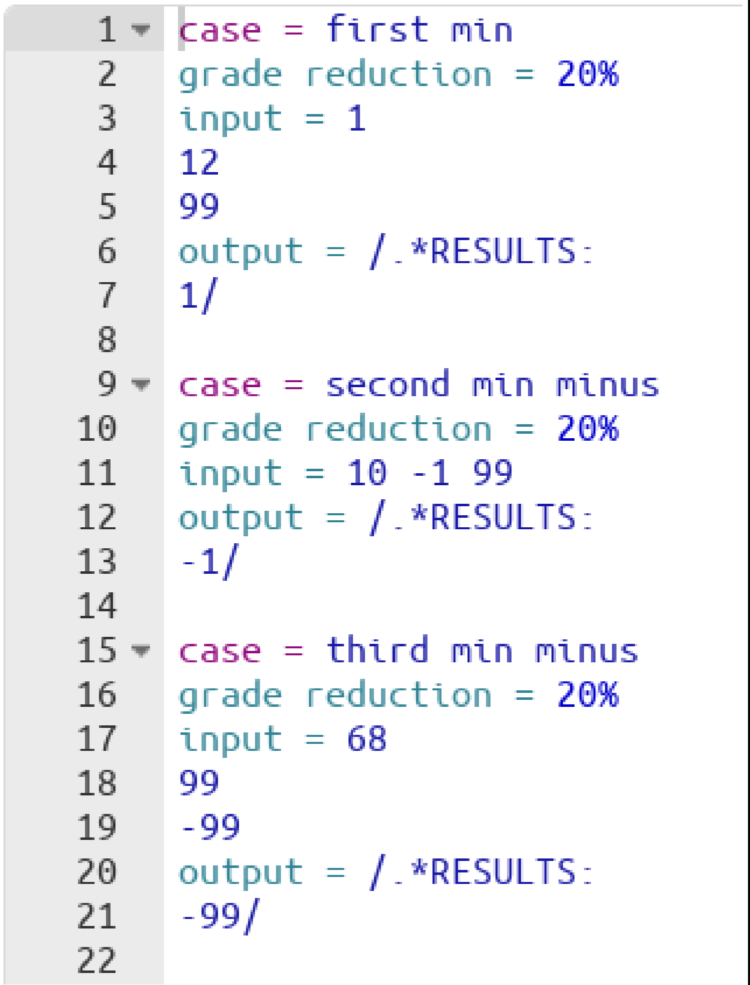
Simple synthax

Four basic commands

Line matching

Exact matching or regex

vpl_evaluate.cases

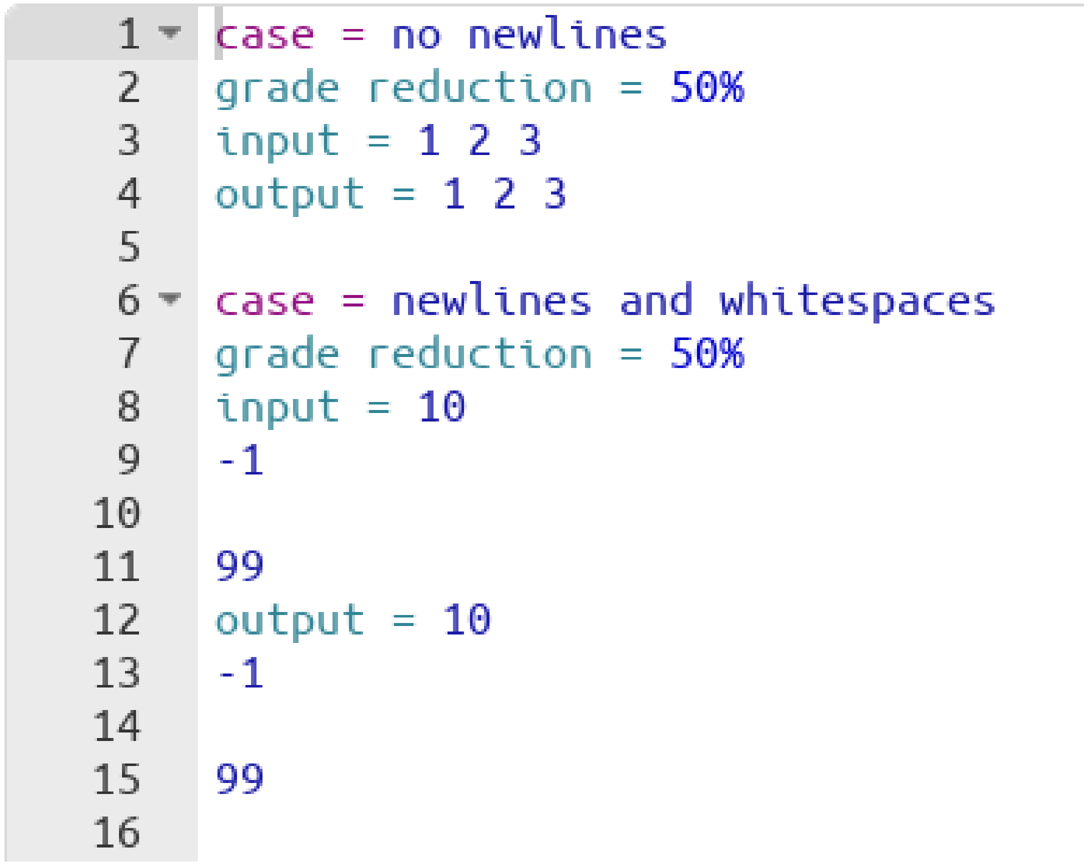

```
case = first min
grade reduction = 20%
input = 1
12
99
output = /.*RESULTS:
1/

case = second min minus
grade reduction = 20%
input = 10 -1 99
output = /.*RESULTS:
-1/

case = third min minus
grade reduction = 20%
input = 68
99
-99
output = /.*RESULTS:
-99/

```

DEMO 2

Simple line matching

Two test cases

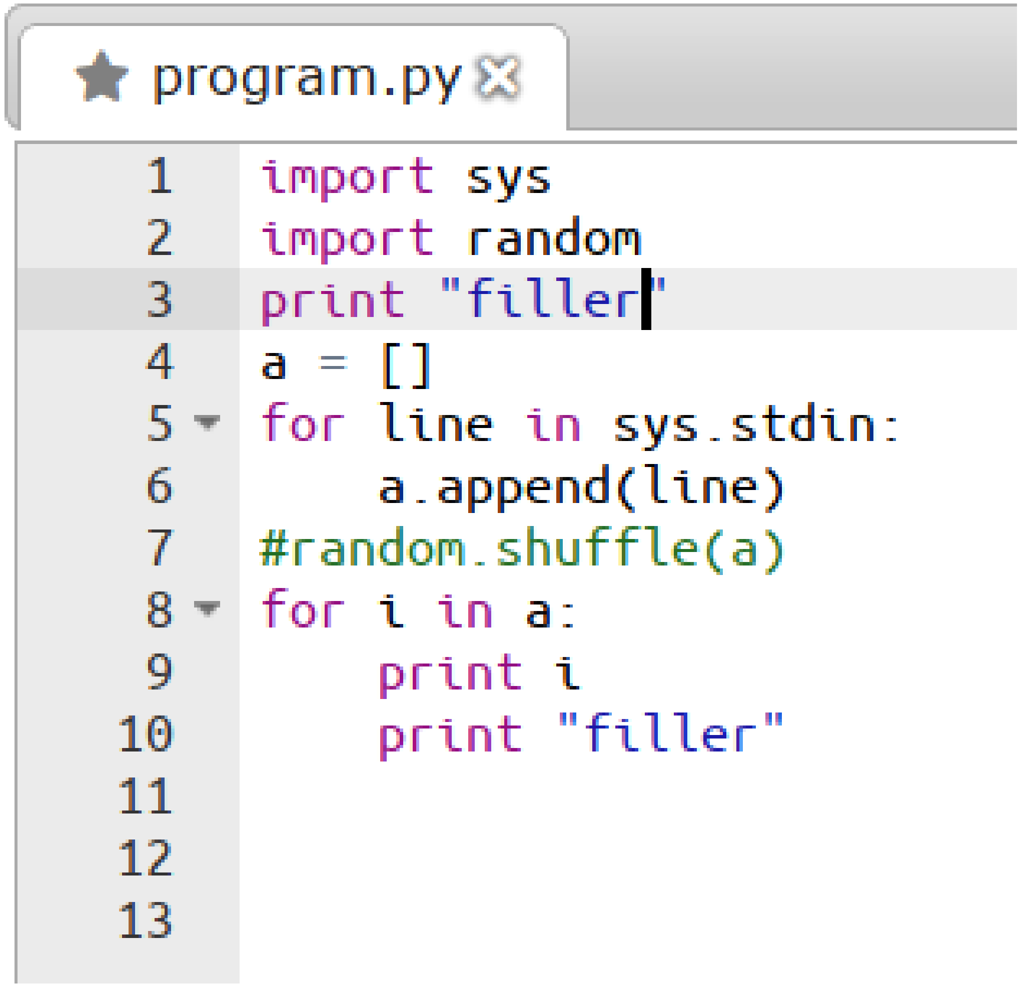
vpl_evaluate.cases



```
case = no newlines
grade reduction = 50%
input = 1 2 3
output = 1 2 3

case = newlines and whitespaces
grade reduction = 50%
input = 10
-1

99
output = 10
-1

99

```

DEMO 2

Add a simple python program



program.py

```
import sys
import random
print "filler"
a = []
for line in sys.stdin:
    a.append(line)
#random.shuffle(a)
for i in a:
    print i
    print "filler"
```

TEST CASES EASY TO FOOL

Test cases visible – easy to cheat:

Printing test cases

Hard to debug – tries to guess output

if number -> ignores text before

ignores whitespaces for numbers

numbers have to be in a correct order

DEMO 3

See if you can fool the grader

DEMO 4
USE REGEX IN OUTPUT

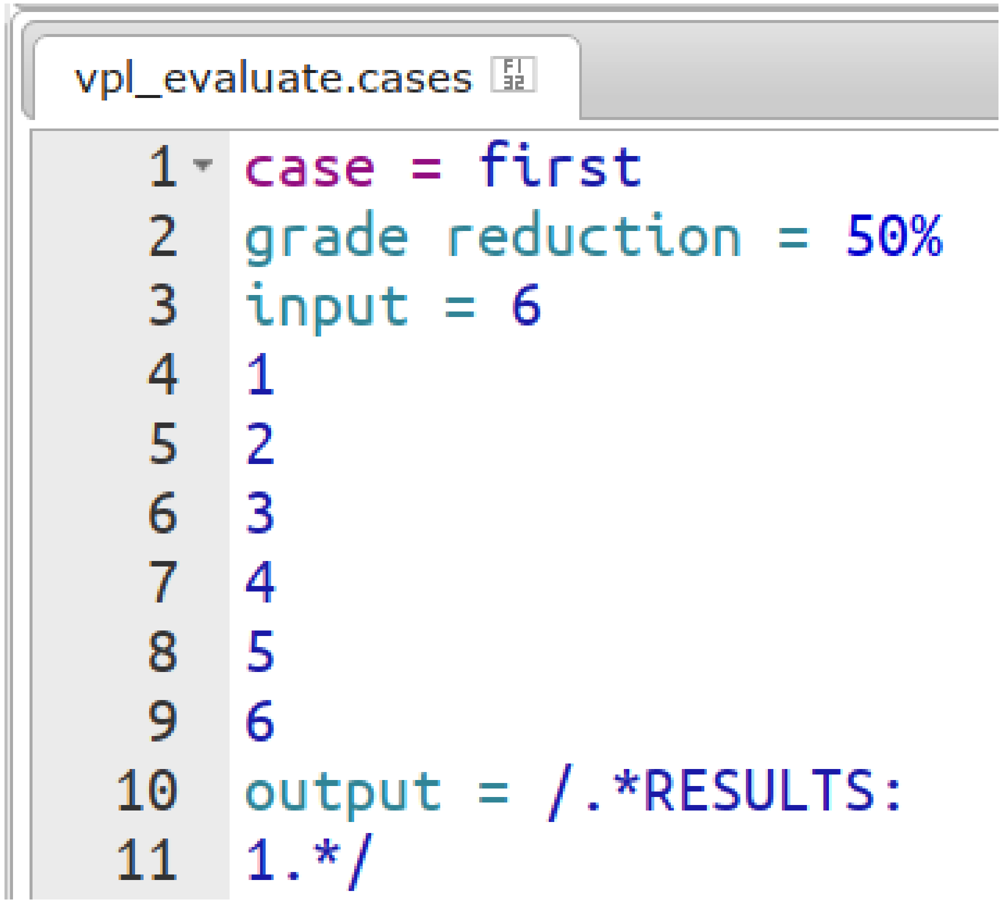
+Reduce cheating

+Reduce accidental correct answers

-Students have to be careful about whitespace

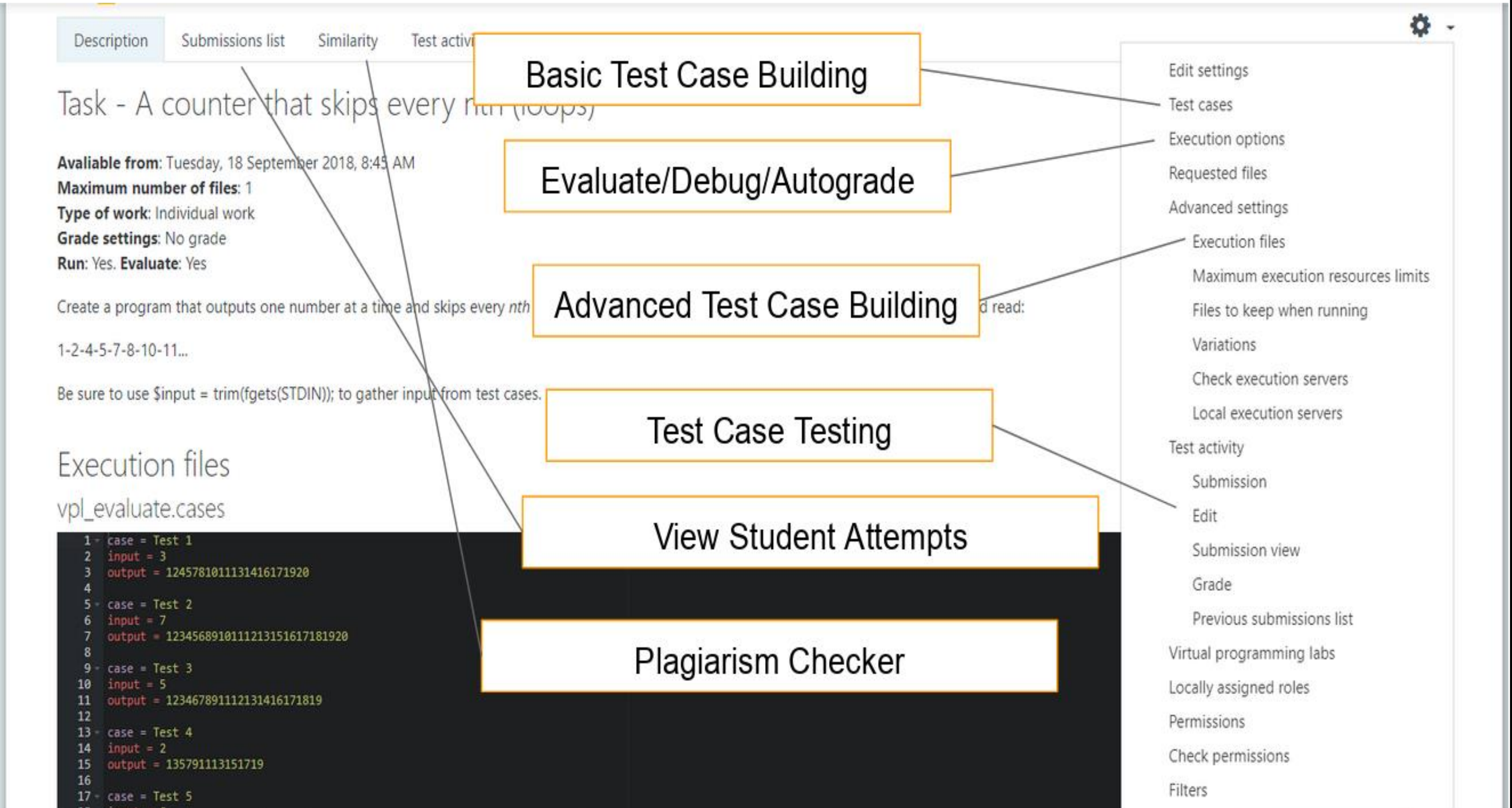
Constrain inputs

Allow debug printing



```
case = first
grade reduction = 50%
input = 6
1
2
3
4
5
6
output = /.*RESULTS:
1.*/
```

MAIN VPL FUNCTIONS



Task - A counter that skips every nth (loops)

Avaliable from: Tuesday, 18 September 2018, 8:45 AM
Maximum number of files: 1
Type of work: Individual work
Grade settings: No grade
Run: Yes. **Evaluate**: Yes

Create a program that outputs one number at a time and skips every *nth* … d read:

1-2-4-5-7-8-10-11...

Be sure to use $input = trim(fgets(STDIN)); to gather input from test cases.

Execution files

vpl_evaluate.cases

```
case = Test 1
input = 3
output = 124578101113141617192O

case = Test 2
input = 7
output = 1234568910111213151617181920

case = Test 3
input = 5
output = 123467891112131416171819

case = Test 4
input = 2
output = 135791113151719

case = Test 5
```

DEMO 5

Solve the Demo 5 exercise as we will use it to showcase:

- similarity checking
- automatic grading
- commenting on grades
- previous versions
- direct access to students code

REQUESTED FILES

Great for providing boilerplate code to students

Force a certain language:

1 Create requested files with the desired extensions

2 Limit number of files to the number of your files

3 Students won’t be able to delete them upon evaluation

EXECUTION FILES

Used to compile and evaluate student code

Not visible to students

vpl_evaluate.cases one of them

Available at compile time:

- Great for libraries students should not be able to change
- Overwrites student’s file if same name

DESIGN YOUR OWN VPL ACTIVITY

Create new VPL activity

Configure Execution options

Write Test cases

Add Requested files if you want the provide a boilerplate code for your students

Test the activity

TIPS AND TRICKS

Scanf vs. gets vs. fgets

Turn off stdout buffering

```
setvbuf(stdout, NULL, _IONBF, 0);
```

Safe exam browser

Duplicating activity does not duplicate grades – past exams free for practice

Creative cheaters – cheatsheet, grep

EVEN THIS MUCH CAN TAKE YOU QUITE FAR

SOME CREATIVITY, LARGE TEST CASES AND MANUALLY CHECKING STUDENTS' CODE SOLVES MOST OF THE PROBLEMS

OFTEN – IT'S THE EASIER AND LESS TIME CONSUMING OPTION

Lets Go Further

PREVENTING CHEATING WITH TWO SETS OF TEST CASES

Often used to prevent cheating by simple printing of the test cases

One set of test cases visible to students

Other set uploaded after deadline and regraded

Program has to work for both test cases

+Easy +Fast +NoSkillRequired^{TM}

- Hard to make test cases comparable and prevent edge cases

DYNAMIC TEST CASES

„These are too many students, I cannot possibly check every submission manually, let me automate this”

The Holy Grail of student cheating prevention

+Prevents cheating by printing the testcases as they are regenerated on each evaluation and different on every run

+It makes you look cool

+You don’t have to check the code of each submission manually

-It’s hard to do properly   -It’s time consuming

-You have to know a bit of linux and bash

VPL CODE EXECUTION LIFECYCLE

Three stages:

1. Compilation – Moves student code to server; `vpl_run.sh`/`vpl_debug.sh` creates `vpl_execution`
2. Running – Runs `vpl_execution` in execution jail with input from `vpl_evaluate.cases`
3. Evaluation – `vpl_evaluate.sh` parses output, calculates score

VPL EXECUTION FILES

`vpl_run.sh/vpl_debug.sh` – prepares for run, generates executable

`vpl_execution` – the executable, runs in jail, does not have access to other files

`vpl_evaluate.sh` - runs the executable, provides input, collects output, generates the grades according to testcases

`vpl_evaluate.cases` – the testcases

HIJACKING ONE OF THE EXECUTION STAGES

Two approaches:

1. Hijack `vpl_evaluate.sh` – write your own; provide input to executable, parse output, generate result text (weird synthax, hard, edge cases)
2. Hijack `vpl_run.sh` – insert your own code to run before creating the executable file, directly generate `vpl_evaluate.cases` file (easier to generate testcases, access to student code before compilation)

WRITING YOUR OWN EVALUATION LOGIC

Makes sense when the program output is not STDOUT

We used it to check writing to binary files

-The most buggy, most complained about lab exercise

+Lots of possibilities, e.g. Sending the file with curl somewhere else

```
Testing 1/2 : first

Testing 2/2 : newlines and whitespaces


<|--

-Failed tests

Test 1: first

Test 2: newlines and whitespaces

--|>


<|-- -Test 1: first (-50.000)

Incorrect program output

--- Input ---

> 6

>1

>2

>3

>4

>5

>6
```

```
--- Program output ---

> 6

>

>1

>

>2

>

>3

>

>4

>

>5

>


--- Expected output (regular
expression)---

>.*RESULTS:

>1.*
```

DYNAMICALLY GENERATING TESTCASES

Easier to accomplish

Benefit from a very well implemented evaluator logic

Testcase file synthax easy to generate dinamically

Pre-compilation checks

Free to use any language

OUR BEST APPROACH

vpl_run.sh:

```
#load common script and check programs
. common_script.sh
check_program gcc
get_source_files c

python generator.py

#compile
eval gcc -o vpl_execution -std=c99 $SOURCE_FILES
-lm -lutil
```

OUR EXPERIENCE SHOWCASE AND IDEAS

FERIT

Programming 2 exercises

Students only use C

Generators mostly in python

It’s hard to create fun exercises in C

DEMO

Build your own dynamic testcase generator