GitHub Classroom

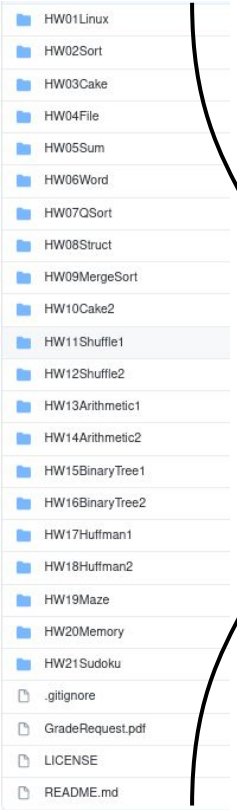
Grading System for ECE 264

GitHub Classroom Features

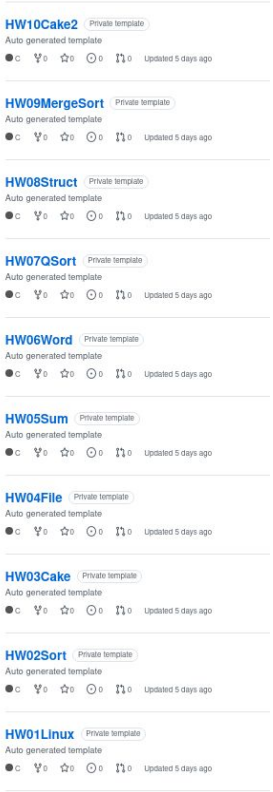
Automatic Template Repository Creation:

Automatically create assignment templates from master repository!

Master Repo:



Individual Assignment Repos:

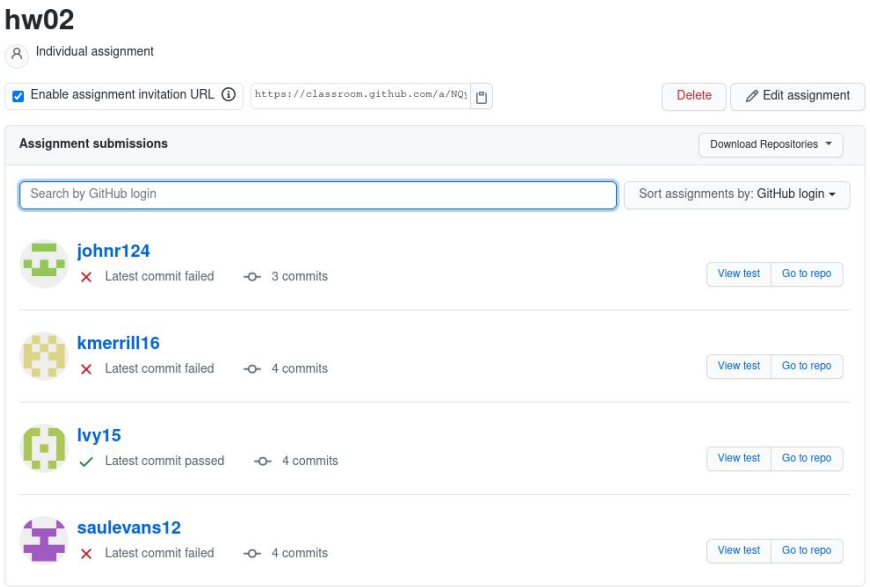


GitHub Classroom Features

Accept Assignments through Shared Link:

Professor can easily share assignments, all the student has to do is click the link!

Assignment Info:



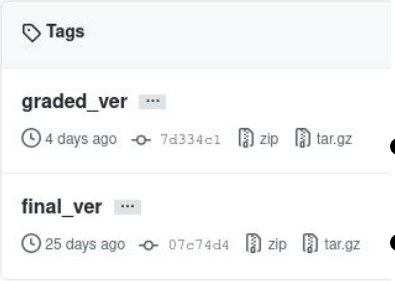
See autograding results and copy link from one page.
View commit history by clicking on specific student.

GitHub Classroom Features

Tags:

Students create tag when homework is ready to be graded. Graded tag is assigned after grading.

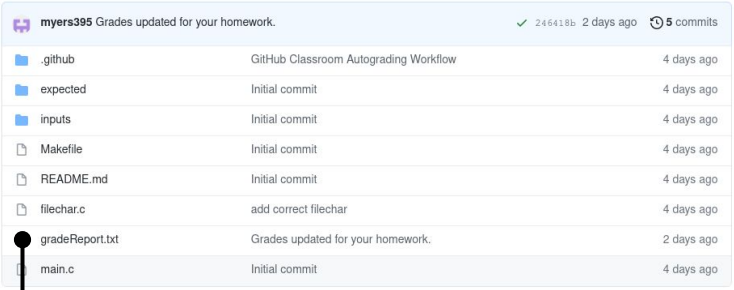
Tag List:



Automatically added

Added by student

graded_ver: Flags student code so that the homework is not regraded



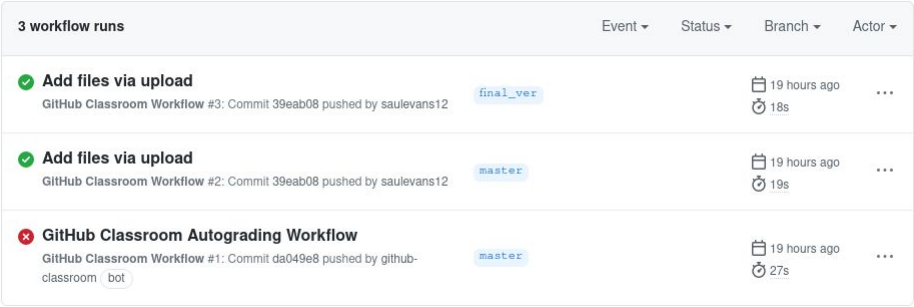
gradeReport: File that contains final grade and feedback for student pushed to student's repository after grading.

GitHub Classroom Features

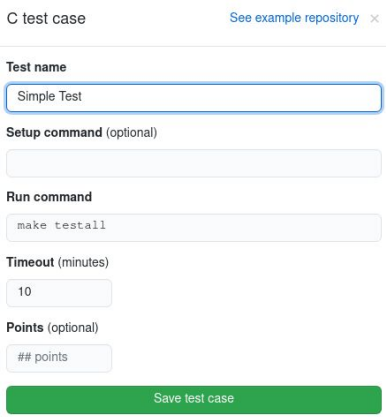
Autograding:

Give students feedback directly after submission through github actions.

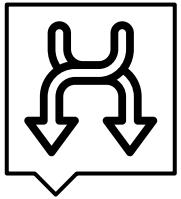
Passed Test Cases:



Simple Configuration:



System Overview

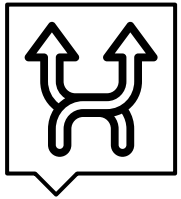


Cloning

A student repository with the appropriate tags is cloned to the local machine.

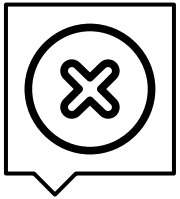
Grading

The homework is graded and the number grade and feedback is collected and added to a text file.



Pushing

The new grade file is added to the student repository and the change is pushed to remote.



Deleting

The cloned repository is deleted and the system moves on to the next student.

Demonstration

Configuration

- Certain variables should be specified by the professor in the config.JSON file

Organization Name

Name of the classroom organization

Authentication Username

The name of the GitHub account with access to the classroom.

Authentication Key

A token tied to this account that allows for automation of important GitHub API features.

- Professor files setup
- Other important variables

tagName

final_ver: Name of the submission tag

gradeFileName

gradeReport.txt: Location of student grade report file

profDir

/profFiles: Directory of professor files

gradesDir

/grades: Directory of grade reports

clonesDir

/clones: Directory of cloned repositories

Test Case Weights

`weights.json` contains information of:

- Weights of each **test case**

```
"due": "2022-08-01 15:59:59",
"weights": [
    {
        "test1": [
            0.3333333333333333
        ]
    },
    {
        "test2": [
            0.3333333333333333
        ]
    },
    {
        "test3": [
            0.3333333333333333
        ]
    },
```

The weights file can be easily generated with **`jsonfile_generator.py`**

```
jsonfile_generator.py --HW_name HW3 --case_num
3 --mem_coef 1.0 --late-coef 10
--grade_late_work false
```

```
    {
        "mem_coef": [
            1.0
        ]
    },
    {
        "late_coef": [
            10.0
        ]
    },
    {
        "grade_late_work": [
            false
        ]
    }
]
}
```

- Coefficient for deduction for **memory leaks**
- Coefficient for deduction for **late submissions**
- Whether to **grade late work** or not

Makefile Template

```
WARNING = -Wall -Wshadow --pedantic
ERROR = -Wvla -Werror
GCC = gcc -std=c99 -g $(WARNING) $(ERROR)

TESTFALGS = <need input!!!>                                          # Need modification

SRCS = <need input!!!>                                               # Need modification
OBJS = $(SRCS:%.c=%.o)
VAL = valgrind --tool=memcheck --log-file=memcheck.txt --leak-check=full --verbose

<NNN>: $(OBJS)                                                       # Need modification
    $(GCC) $(TESTFALGS) $(OBJS) -o <NNN>

.c.o:
    $(GCC) $(TESTFALGS) -c $*.c

test1: <NNN>                                                         # Need modification
    ./ <NNN> <Input arguments> <Name of Outputfiles>
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> > grade.txt
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> >> grade.txt
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> >> grade.txt

test2: <NNN>                                                         # Need modification
    ./ <NNN> <Input arguments> <Name of Outputfiles>
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> > grade.txt
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> >> grade.txt
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> >> grade.txt

test3: <NNN>                                                         # Need modification
    ./ <NNN> <Input arguments> <Name of Outputfiles>
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> > grade.txt
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> >> grade.txt
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> >> grade.txt

clean: # remove all machine generated files                          # Need modification
    rm -f <NNN> *.o output?
```

Makefile Example

```makefile
WARNING = -Wall -Wshadow --pedantic
ERROR = -Wvla -Werror
GCC = gcc -std=c99 -g $(WARNING) $(ERROR)

TESTFALGS = -DTEST_CONVERT

SRCS = main.c list.c convert.c
OBJS = $(SRCS:%.c=%.o)

hw14: $(OBJS)
    $(GCC) $(TESTFALGS) $(OBJS) -o hw14

.c.o:
    $(GCC) $(TESTFALGS) -c $*.c

test1: hw14
    ./hw14 inputs/test1 > output1
    diff output1 expected/expected1 > grade.txt

test2: hw14
    ./hw14 inputs/test2 > output2
    diff output2 expected/expected2 > grade.txt

test3: hw14
    ./hw14 inputs/test3 > output3
    diff output3 expected/expected3 > grade.txt

test4: hw14
    ./hw14 inputs/test4 > output4
    diff output4 expected/expected4 > grade.txt

test5: hw14
    ./hw14 inputs/test5 > output5
    diff output5 expected/expected5 > grade.txt

clean: # remove all machine generated files
    rm -f hw14 *.o output?
```

Set files that will be used for running this homework

Set <NNN> to hw14

Set <Input arguments> to inputs/test1

Set <Name of Output files> to output1

Set <Name of one of the expected outputfiles> to expected/expected1.

- All test cases must start with "test", or else they will not be found.
- Test cases must also start at 1 and increase by 1 each time, otherwise the program may not act as expected

Adding Test Cases

- Add files to inputs and expected folders
- Add test case to makefile as done in previous slide
  - Make sure to follow the template, writing the diff to grade.txt
- Add weight in weights.json
  - If the weight is not added, the program will assign the weight to be the average of all the previous weights

```
test1: <NNN>                                                    # Need modification
    ./ <NNN> <Input arguments> <Name of Outputfiles>
    diff <Name of one of the outputfiles> <Name of one of the expected outputfiles> > grade.txt
```

```
test1: sort
    ./sort inputs/test1 > output1
    diff output1 expected/expected1 > grade.txt
```

Running the System

As a user, everything is ran from *runSystem.py*.

There are a few options that can be specified when running. At least one of the three grade options *must* be specified for the program to run.

--hw_name

Grade a single homework. Specify this tag and the name or number of the homework.

```
runSystem.py --hw_name hw02sort
runSystem.py --hw_name 2
runSystem.py --hw_name hw02
```

--hw_range

Grade a range of homeworks. Specify this tag and two homework names/numbers to as the start and end of the range. The start and end indexes are inclusive.

```
runSystem.py --hw_range hw02sort hw10cake2
runSystem.py --hw_range 2 10
runSystem.py --hw_range 2 hw10cake2
```

--grade_all

Grade all homeworks. This will grade all homeworks for which a professor-created example folder exists in the professor directory.

```
runSystem.py --grade_all
```

Step 0: Setup

- A text file is created to collect output as the process runs (filteredOutput.txt)
- A list of all repositories that exist within the organization is collected
- A list of students and homeworks is also collected and added to the master CSV of grades
- Functions used:
  - **`fetchRepos(organization name, authentication username, authentication key)`**
    - `Returns JSON file`
  - **`fetchLists(JSON file of repository names)`**
    - `Returns lists of students, homeworks, and repository names`

Step 1: Cloning

- A student repository is passed to the cloning function
- A list of tags that exist for the repository is collected
- If the repository has the correct tags, it is cloned
- The number of hours late that the homework was submitted is collected
  - This is calculated from the submission date, which is the date that the tag was created, and the due date, which is specified in the weights JSON file
- Function:
  - `cloneFromRepos(org, repo, hwNum, tagName, authName, authKey, profPath, clonePath, outputFile)`
    - Returns true if repository needs to be graded and the number of hours late of the submission

Step 2: Grading

- The Grading Interface is called for the repository
- Test cases are evaluated and checked for memory leak
- The number grade and written feedback is added to a text file for the student
- Function:
  - `startGradingProcess(repo, hoursLate, hwName, outputFile, gradeDir, cloneDir, profDir)`
    - No returns, just text file creation

```
For repo: hw02sort-kmerrill16
--Calling grade_submission.py
--Grade is 100.0
--gradeReport.txt created
```

Example feedback for professor in filteredOutput.txt

```
Graded on 07-20 07:31:41
Grade: 100.0%
Submission was 0 hours late.
Feedback: hw compiled correctly! going to next step....
Test case 1 is correct!. Test case 2 is correct!. Test
case 3 is correct!. 3/3 test cases passed!. No memory
leak in test case 1. No memory leak in test case 2. No
memory leak in test case 3.
```

Example feedback for student in gradeReport.txt

Step 3: Pushing

- The grade report text file is pushed to the student repository
- A new tag that signifies that the repository has been graded is added to the student repository
- The number grade is added to a master CSV file for professor reference
- Functions:
  - `putGradesInRepos(gradesDir, clonesDir, fileName, repo`
  - `putGradesInCSV(profDir, gradesDir, fileName, repo)`
  - `pushChangeToRepos(clonesDir, fileName, repo)`

	GitHub Username	hw02sort	hw03cake
0	kmerrill16	100	0
1	lvy15	0	0
2	johnr124	0	0

Example CSV file

Step 4: Deletion

- Local folders of cloned repository and directory containing grade report are deleted
- Feedback file is closed
- System moves on to next repository to repeat steps 1-4

```
[[Currently grading : hw02sort]]
 * Cloned hw02sort-kmerrill16

For repo: hw02sort-kmerrill16
--Calling grade_submission.py
--Grade is 100.0
--gradeReport.txt created

Successfully ran startGradingProcess

Successfully ran putGradesInRepos

Successfully ran putGradesInCSV

Successfully ran pushChangeToRepos
```

filteredOutput.txt feedback for successful grading of one repository

Results

- Students who have submitted their homework (‘*final_ver*’ tag exists) have repositories cloned and graded
- Grade report pushed to each student repository
- Graded tag created (‘*graded_ver*’)
- Feedback collected in text file for professor

```
Graded on 07-29 12:13:19
Grade: 100.0%
Submission was 0 hours late.
Feedback: hw compiled correctly! going to next step.... Test case 1 is correct!.
Test case 2 is correct!. Test case 3 is correct!. 3/3 test cases passed!. No
memory leak in test case 1. No memory leak in test case 2. No memory leak in test
case 3.
```

Example feedback for student in gradeReport.txt

```
Ran on 07-29 12:16:54

Grading hw02sort

--[[Currently grading : hw02sort]]--

[Evaluating repo: hw02-lvy15]
  --Tags for this repo: final_ver
  * Cloned hw02-lvy15
  --Calling grade_submission.py
    --Grade is 100.0
    --gradeReport.txt created
  --Successfully ran startGradingProcess
  --Successfully ran putGradesInRepos
  --Successfully ran putGradesInCSV
  --Successfully ran pushChangeToRepos
[Finished grading hw02-lvy15]

Removed clones
Removed grades
***Finished grading process***

Requests Used this Runtime: 2
Hourly Requests Left: 4972
Total Runtime is: 1 Minutes and 37 Seconds
```

Example feedback for professor in filteredOutput.txt

Transition & Deployment

Support you can expect:

01 Setup

Comprehensive assistance guiding you through setting up for your class

02 Education

Thorough education for TAs on the ins and outs of the system, user's guide for quick reference

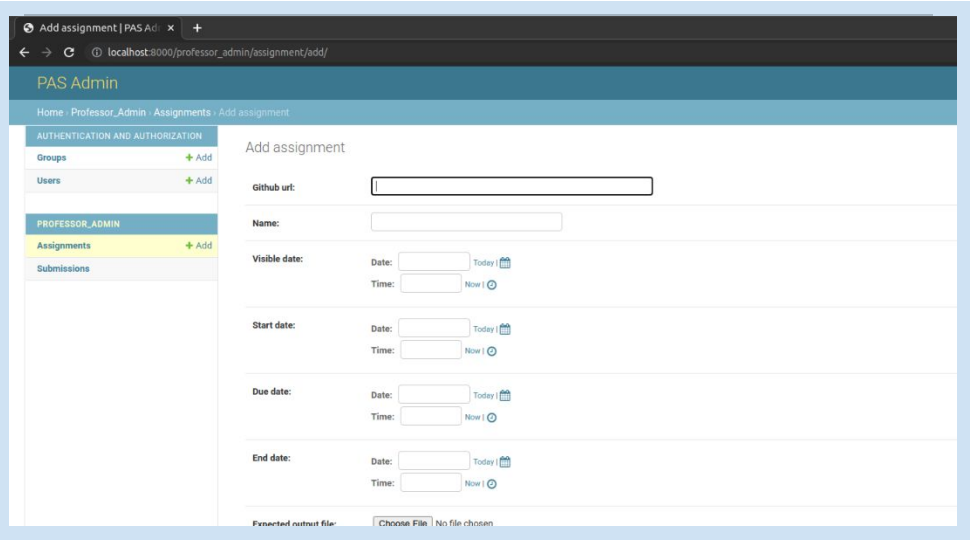
03 Troubleshooting

Rapid & effective solutions to any issues encountered

04 Personalization

Development & implementation of features specifically tailored to your desires

Future Plans



Professor webpage for streamlined data entry & download

CRON job set-up for repeated automatic grading at a specified time

Program analysis

Automatically generated test cases

In-depth feedback for students

Thanks!

Any questions?

Presenters:
Shan Huang
Yinhan Chen
Jack Meyers
Alex Gieson
Leila Yanni