

HOW TO USE

이용방법

리눅스 `bash` 셸에서 동작합니다. 리눅스 환경을 우선 준비해둬야 합니다.

1. 필수 요건 설치

`git`, `zip`, `unzip`, `dos2unix`, `perl`을 설치합니다.

```
sudo apt install git zip unzip dos2unix perl
```

2. 프로그램 설치

각 assignment 버전에 맞는 최신 release zip 파일을 다운받아 압축해제 합니다.

예를 들어, assignment 1 채점 프로그램을 얻는다고 하면, assignment 1이 붙은 release 중 가장 최신 버전을 다운받으면 됩니다.

The screenshot shows the GitHub repository page for 'auto_grading_release' by user 'saychuwho'. The repository is private and has 2 branches and 6 tags. The file list on the left shows various files like 'grading_cases', 'outputs', 'reports', 'student_submission', '.gitignore', 'README.md', and several shell scripts. The right sidebar shows the 'Releases' section with 'Assignment 1 Release 1.2' marked as the latest release, posted 1 hour ago. Below this, the 'Assets' section shows two downloadable files: 'Source code (zip)' and 'Source code (tar.gz)', both uploaded 2 hours ago. The main content area shows the release details for 'Assignment 1 Release 1.2', including a list of changes and a list of assets.

또는, 이 repository를 clone 한 후, assignment 버전에 해당하는 branch로 checkout합니다.

예를 들어, assignment 1 채점 프로그램을 얻는다고 하면, 이 repository를 clone한 후, assignment 1에 해당하는 branch인 **2024-CSE201-Assignment-1**로 checkout하면 됩니다.

```
git clone https://github.com/saychuwho/auto_grading_release.git
cd auto_grading_release/
git checkout 2024-CSE201-Assignment-1
```

```
(base) $ git clone https://github.com/saychuwho/auto_grading_release.git
Cloning into 'auto_grading_release'...
remote: Enumerating objects: 283, done.
remote: Counting objects: 100% (283/283), done.
remote: Compressing objects: 100% (77/77), done.
remote: Total 283 (delta 142), reused 256 (delta 118), pack-reused 0 (from 0)
Receiving objects: 100% (283/283), 61.90 KiB | 3.87 MiB/s, done.
Resolving deltas: 100% (142/142), done.
(base) $ cd auto_grading_release/
(auto_grading_release) $ git checkout 2024-CSE201-Assignment-1
Branch '2024-CSE201-Assignment-1' set up to track remote branch '2024-CSE201-Assignment-1' from 'origin'.
Switched to a new branch '2024-CSE201-Assignment-1'
(auto_grading_release) $
```

3. 스크립트에 실행 권한 주기

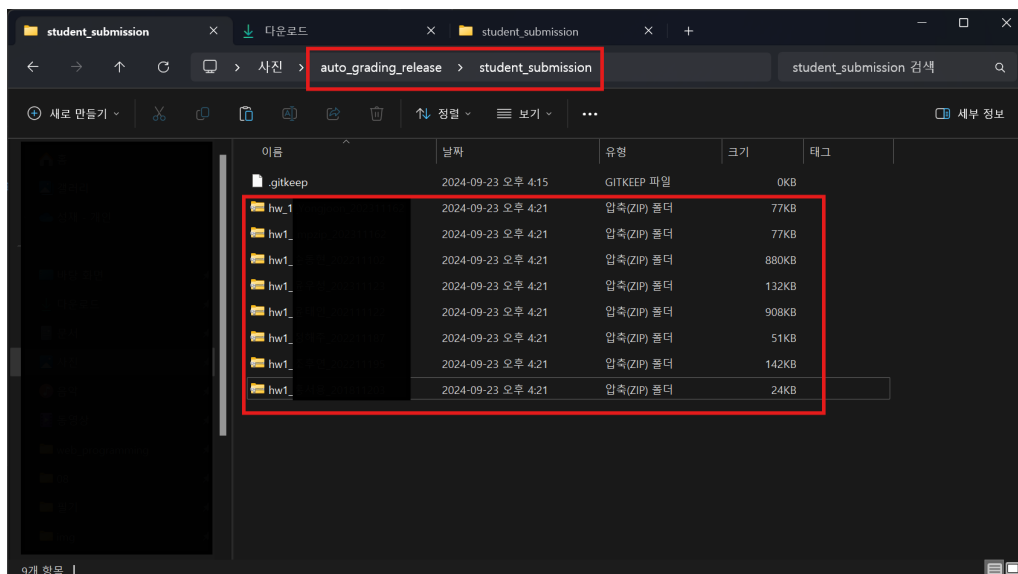
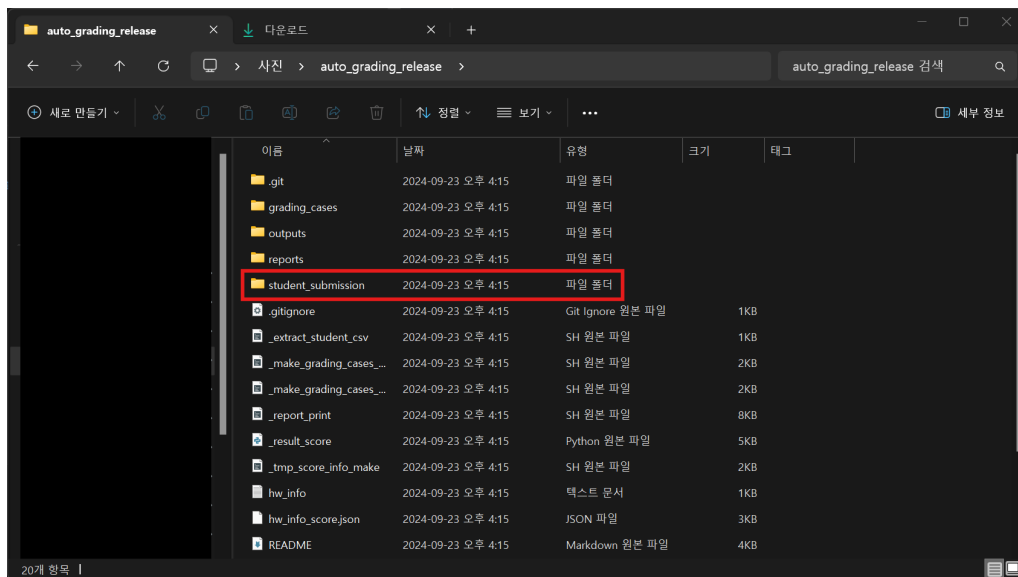
다음 명령어들을 실행합니다.

```
chmod 755 ./setup.sh
./setup.sh
```

```
(base) /auto_grading_release$ chmod 755 ./setup.sh
(base) /auto_grading_release$ ./setup.sh
dos2unix: converting file ./run.sh to Unix format...
dos2unix: converting file ./reset.sh to Unix format...
dos2unix: converting file ./student_list.txt to Unix format...
dos2unix: converting file ./_report_print.sh to Unix format...
dos2unix: converting file ./_result_score.py to Unix format...
dos2unix: converting file ./run extra.sh to Unix format...
(base) /auto_grading_release$
```

4. 채점 파일 준비하기

채점하고자 하는 학생들의 .zip 파일을 ./student_submission 폴더에 넣습니다.



5. 채점 시작

`./run.sh`를 실행합니다.

```
./run.sh
```

`run.sh`는 다음과 같이 실행될 때 현재 채점하는 Assignment의 정보를 보여줍니다. Assignment 정보에는 Assignment에 있는 문제, 문제 별 test case의 수가 포함됩니다. 다음 사진은 Assignment 1의 정보를 보여줍니다.

```
(base) /auto_grading_release$ ./run.sh
< hw1 scoring system >
> # of problem : 5
> Problem 1:   case 1  case 2  case 3  case 4  case 5
> Problem 2_A: case 1  case 2  case 3  case 4
> Problem 2_B: case 1  case 2  case 3  case 4
> Problem 2_C: case 1  case 2  case 3  case 4
> Problem 3:   case 1  case 2  case 3  case 4
```

`run.sh`는 실행되는 동안 진행 상황을 다음과 같이 보여줍니다. 만약 컴파일 과정에서 segmentation fault가 일어났다면, 해당 학생이 제출한 코드 중 하나가 segmentation fault가 일어났다는 것을 의미합니다. 채점 시 참고하면 됩니다.

```
1. Unzip submissions
Progress : [#####] 100%

2. Change submitted files into correct format
Progress : [#####] 100%

3. combine submission and cases
Progress : [#####] 100%

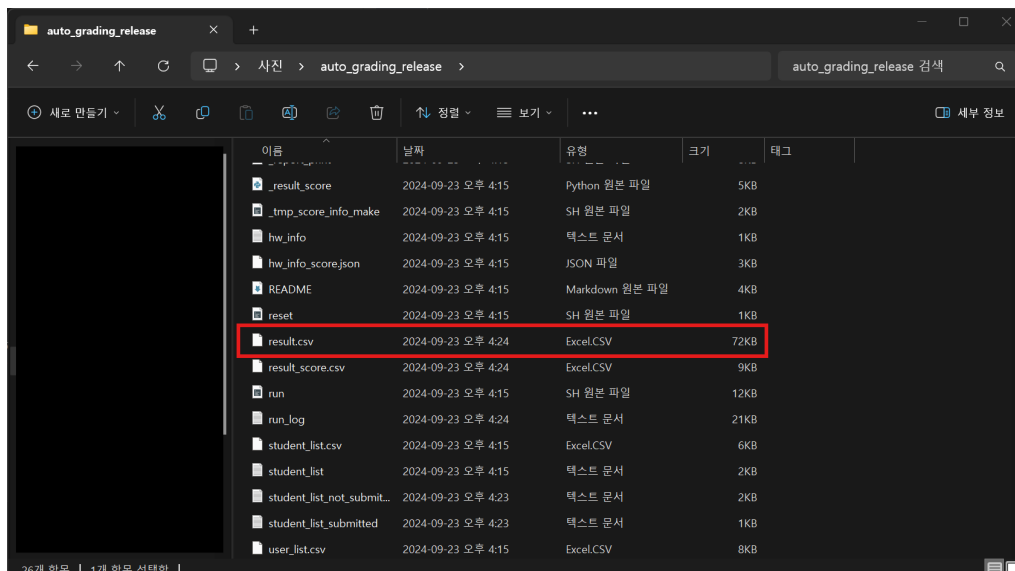
4. compile cases and make outputs, make diff file

> student id: / Progress (1/28)
> student id: / Progress (2/28)
> student id: / Progress (3/28)
> student id: / Progress (4/28)
> student id: / Progress (5/28)
> student id: / Progress (6/28)
> student id: / Progress (7/28)
> student id: / Progress (8/28)
> student id: / Progress (9/28)
> student id: / Progress (10/28) ./run.sh: line 300: 370394 Segmentation fault "$output_file".out" > "$output_file)_output.txt"
```

6. 결과 확인

`run.sh`는 다음 파일들을 생성합니다.

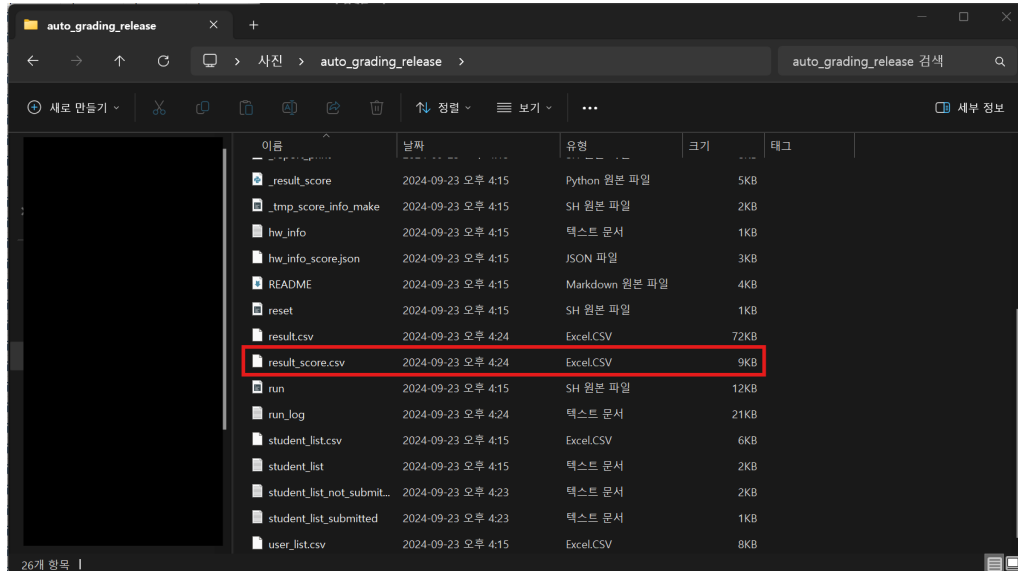
- 모든 학생의 테스트 케이스 통과 여부를 담고 있는 `result.csv`



엑셀로 열면 다음과 같이 학생별로 테스트 케이스의 결과 정보들을 담고 있습니다.

[illegible]

2. 모든 학생의 테스트 케이스 별 점수 여부와 총 점을 담고 있는 `result_score.csv`

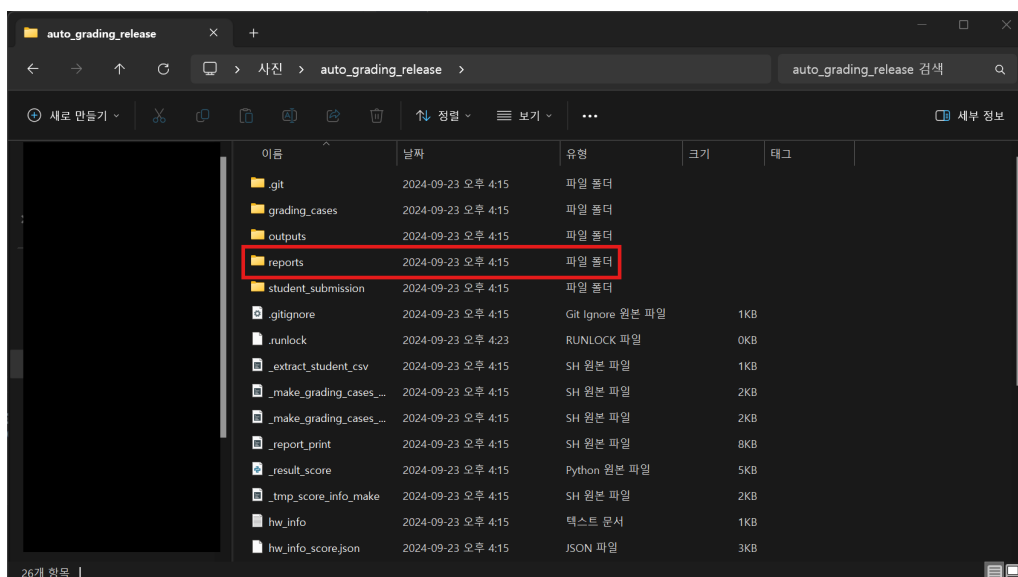


엑셀로 열면 다음과 같이 학생별로 테스트 케이스에서 얻은 점수와 총점을 담고 있습니다.

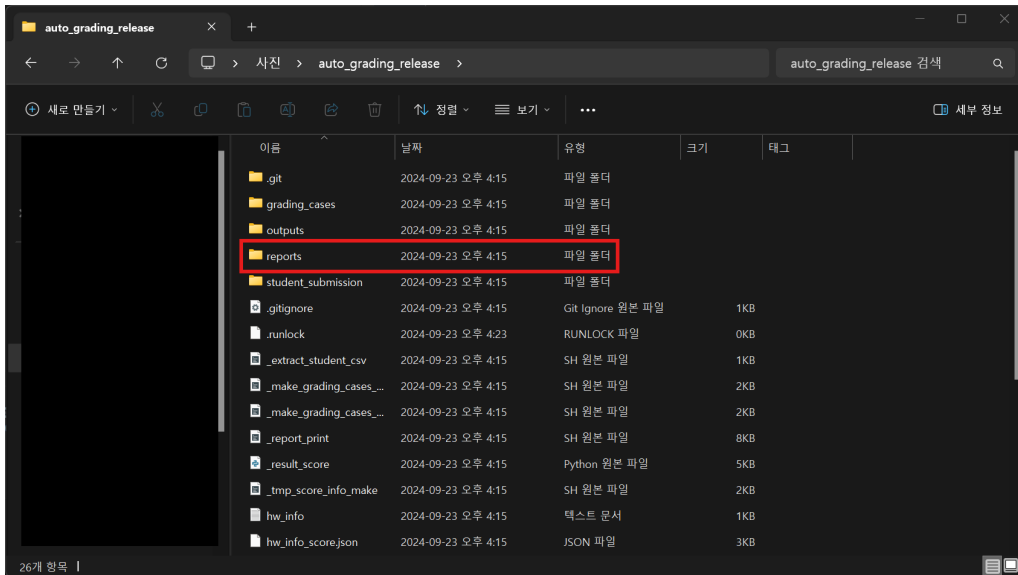
[illegible]

3. 학생 별 평가 개요, 제출한 코드와 컴파일 결과, 출력 결과를 보여주는 마크다운 리포트

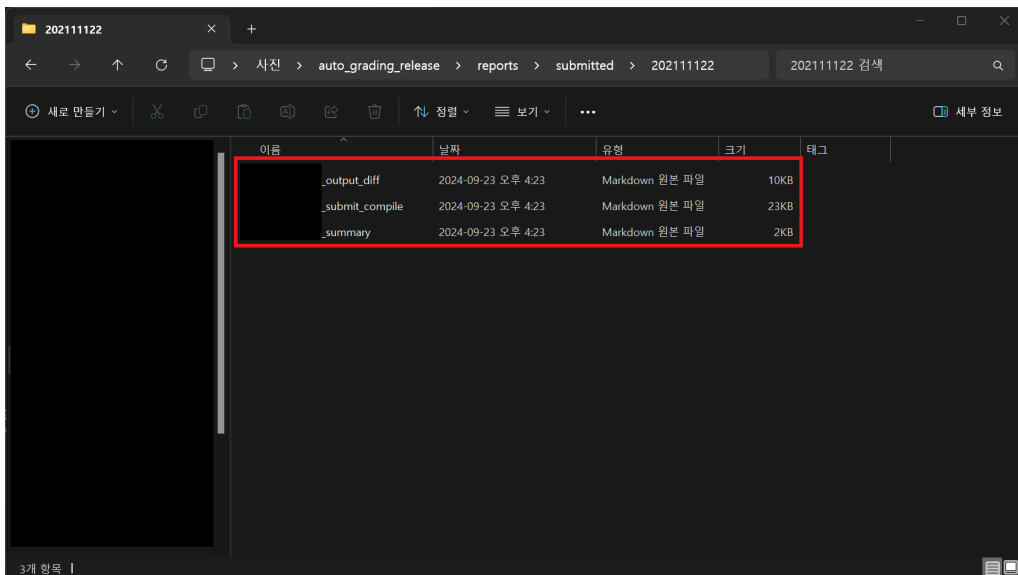
다음 폴더에 마크다운 리포트가 저장되어 있습니다.



`not_submitted`에는 zip 파일을 제출 안 한 학생들의 리포트가, `submitted` 폴더에는 zip 파일을 제출 한 학생들의 리포트가 있습니다.



`not_submitted`, `submitted` 폴더에는 학생들의 학번별로 폴더가 생성되어 있는데, 폴더 속에 학생별 리포트 파일이 있습니다. 마크다운 파일들은 vscode를 이용해 열면 렌더링 된 상태로 볼 수 있습니다.



`_summary.md`는 학생의 채점 결과와 학생이 제출한 zip 파일 속 내용물에 대한 정보를 볼 수 있습니다.

hw1 scoring report - summary

Result

```
{
  "1": "file-submitted",
  "1-case-1" : "fail",
  "1-case-2" : "fail",
  "1-case-3" : "fail",
  "1-case-4" : "fail",
  "1-case-5" : "fail",
  "2_A": "file-submitted",
  "2_A-case-1" : "pass",
  "2_A-case-2" : "pass",
  "2_A-case-3" : "pass",
  "2_A-case-4" : "pass",
  "2_B": "file-submitted",
  "2_B-case-1" : "compile-error",
  "2_B-case-2" : "compile-error",
  "2_B-case-3" : "compile-error",
  "2_B-case-4" : "compile-error",
  "2_C": "file-submitted",
  "2_C-case-1" : "pass",
  "2_C-case-2" : "pass",
  "2_C-case-3" : "pass",
  "2_C-case-4" : "pass",
  "3": "file-submitted",
  "3-case-1" : "pass",
  "3-case-2" : "pass",
  "3-case-3" : "pass",
  "3-case-4" : "pass",
  "dummy" : "dummy"
}
```

Inside .zip

```
Archive: ./student_submission/hw1_...zip
  Length      Date    Time    Name
-----
  427          2024-09-27 10:10    hw1_1_...
  282          2024-09-27 10:10    hw1_2_A_...
  288          2024-09-27 10:10    hw1_2_B_...
  352          2024-09-27 10:10    hw1_2_C_...
  1395         2024-09-27 10:10    hw1_3_...
  1532034      2024-09-27 10:10    hw1_...pdf
-----
  1534778             6 files
```

submit_compile.md는 학생이 제출한 소스코드와 이를 바탕으로 컴파일 한 소스코드, 컴파일 결과를 볼 수 있습니다. 컴파일 결과에 아무것도 없으면 컴파일이 정상적으로 진행되었다는 의미이고, 컴파일 결과에 내용이 있다면 컴파일 에러가 일어났다는 의미입니다.

hw1 scoring report - submit_compile

Submitted Source Code

submitted problem-1 source code

```
#include <iostream>
using namespace std;
```

compiled problem-1-case-3 code

```
#include <iostream>
using namespace std;
```

compiled problem-1-case-3 result

output_diff.md는 테스트 케이스 별 학생 코드의 출력 결과, 정답 출력 결과와의 **diff** 결과를 보여줍니다. 정답 출력 결과는 두 종류로 파일의 맨 끝에 EOL 문자가 있는 정답과 EOL 문자가 없는 정답 두 가지로 나뉩니

다. 둘 중 하나라도 동일하다면 정답 처리 합니다.

```
problem-2_A-case-4 output result
*****
****
***
**
*

problem-2_A-case-4 output 1 diff result - without EOL

problem-2_A-case-4 output 2 diff result - with EOL
--- ./grading_cases/hw1_2_A_case_4_output_2.txt 2024-09-23 16:15:36.432282200 +0900
+++                                     2024-09-23 16:23:24.411809400 +0900
@@ -2,4 +2,4 @@
****
***
**
.*
\ No newline at end of file
+*
```

출력 형식을 제대로 지킨 경우에는 제대로 채점이 되지만, 간혹 의미적으로는 맞지만 출력 형식을 지키지 않았거나, 제출 파일 이름 형식을 지키지 않아 채점이 되지 않을 수 있습니다. 이 경우에는 결과 파일을 바탕으로 직접 채점을 해야 할 수 있습니다.

4. 학생들간의 코드 유사도를 측정한 결과를 담고 있는 `result_moss.md`

유사도는 [MOSS](#)를 이용해 측정한 결과입니다. 파일 안에 각 문제 별 유사도 측정 결과를 볼 수 있는 링크가 있습니다. 간단한 과제의 경우 유사도가 의미 없을 수 있지만, 팀 프로젝트 같이 복잡한 과제의 경우는 유사도가 중요하게 작용할 수 있습니다.

다음은 assignment 1의 경우를 표시한 경우입니다.

```
1  # MOSS result of all problems in hw1
2
3  click the link below to see each problem's MOSS report
4
5  1 : http://moss.stanford.edu/
6  2_A : http://moss.stanford.edu/
7  2_B : http://moss.stanford.edu/
8  2_C : http://moss.stanford.edu/
9  3 : http://moss.stanford.edu/
10
```

각 링크로 들어가면, 어떤 파일이 얼마나 유사한지를 퍼센트로 표시하고 있습니다. 정렬은 유사도가 높은 순서대로 입니다.

Moss Results

Options -l cc -m 10

[[How to Read the Results](#) | [Tips](#) | [FAQ](#) | [Contact](#) | [Submission Scripts](#) | [Credits](#)]

File 1	File 2	Lines Matched
./submission_by_problem	.cpp (90%) ./submission_by_problem	.cpp (91%) 25
./submission_by_problem	.cpp (82%) ./submission_by_problem	.cpp (83%) 28
./submission_by_problem	.cpp (64%) ./submission_by_problem	.cpp (83%) 28
./submission_by_problem	.cpp (64%) ./submission_by_problem	.cpp (82%) 23
./submission_by_problem	.cpp (81%) ./submission_by_problem	.cpp (90%) 26
./submission_by_problem	.cpp (64%) ./submission_by_problem	.cpp (90%) 28
./submission_by_problem	.cpp (78%) ./submission_by_problem	.cpp (78%) 27
./submission_by_problem	.cpp (80%) ./submission_by_problem	.cpp (80%) 21
./submission_by_problem	.cpp (59%) ./submission_by_problem	.cpp (70%) 19
./submission_by_problem	.cpp (73%) ./submission_by_problem	.cpp (73%) 24
./submission_by_problem	.cpp (72%) ./submission_by_problem	.cpp (72%) 25
./submission_by_problem	.cpp (65%) ./submission_by_problem	.cpp (71%) 28
./submission_by_problem	.cpp (71%) ./submission_by_problem	.cpp (65%) 28
./submission_by_problem	.cpp (64%) ./submission_by_problem	.cpp (71%) 20
./submission_by_problem	.cpp (50%) ./submission_by_problem	.cpp (71%) 22
./submission_by_problem	.cpp (68%) ./submission_by_problem	.cpp (68%) 24
./submission_by_problem	.cpp (46%) ./submission_by_problem	.cpp (46%) 20
./submission_by_problem	.cpp (50%) ./submission_by_problem	.cpp (52%) 14
./submission_by_problem	.cpp (65%) ./submission_by_problem	.cpp (72%) 14
./submission_by_problem	.cpp (72%) ./submission_by_problem	.cpp (64%) 21
./submission_by_problem	.cpp (62%) ./submission_by_problem	.cpp (62%) 21

여기서 파일을 클릭하면, 두 소스코드가 어떤 부분에서 유사한지를 보여줍니다. 이 정보들을 종합해서 채점 시 판단하면 됩니다.

./submission_by_problem/ (90%) .cpp

8-20

2-7

./submission_by_problem/ (91%) .cpp

9-25

1-8

./submission_by_problem/ .cpp

char** SecretCode(const char* sentences[], int num_sentences) {

./submission_by_problem/ .cpp

char** SecretCode(const char* sentences[], int num_sentences)

{

char** SecretCode(const char* sentences[], int num_sentences)

{

- 학생들의 compile error, fail 여부 등을 담고 있는 `result_compile_error.md`, `result_fail_list.md`, `result_file_not_submitted_list.md`, `result_zip_file_not_submitted.md`

채점 시 확인하기 편하도록 이번에 추가했습니다. 각 목록을 보고 채점시 참고하면 됩니다.

다음은 `result_compile_error.md`의 예시로, 학생의 학번과 컴파일 에러가 난 문제 및 케이스를 보여줍니다.

result_compile_error.md

1	- []	: 2_B-case-1
2	- []	: 2_B-case-2
3	- []	: 2_B-case-3
4	- []	: 2_B-case-4
5	- []	: 3-case-1
6	- []	: 3-case-2
7	- []	: 3-case-3
8	- []	: 3-case-4
9	- []	: 1-case-1
10	- []	: 1-case-2
11	- []	: 1-case-3
12	- []	: 1-case-4
13	- []	: 1-case-5

7. 재실행

채점 프로그램을 다시 실행하기 위해서는 `./reset.sh`를 실행 한 후 `./run.sh`를 실행해야 합니다.

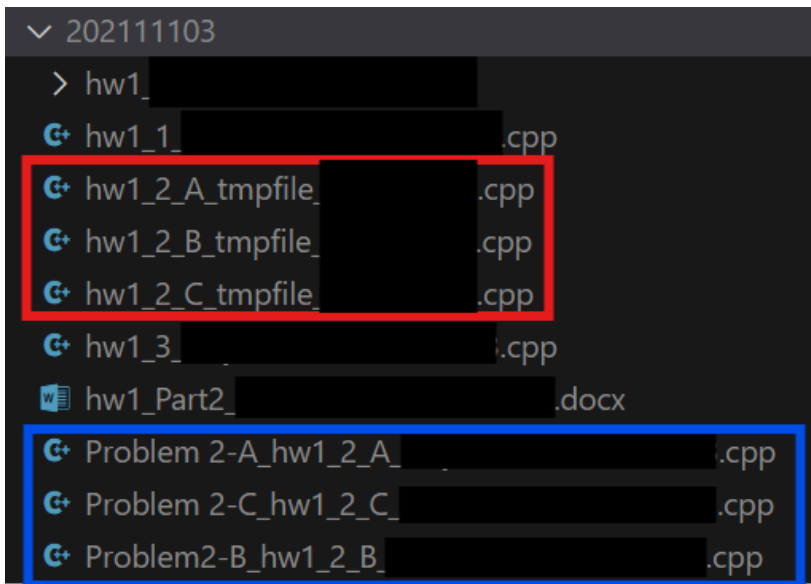
```
./reset.sh
```

또는, `./run.sh`를 실행한 이후, 특정 학생이 제출한 코드를 `./student_submission/학생학번`에서 수정한 뒤 다시 채점하고자 한다면, `./student_submission/학생학번/수정하고자하는파일.cpp`에서 소스 코드를 수정한 다음 `./run_extra` 학생학번을 실행하면 됩니다. 이러면 새로 수정한 소스코드를 바탕으로 해당 학생만 새로 채점합니다.

```
./run_extra.sh 학생학번
```

이때, 소스코드 중 `_tmpfile_`이라 표시된 소스코드가 있다면, 해당 소스코드를 수정해야 합니다.

아래 예시를 보면, 실제 학생이 제출한 코드는 **파란색 사각형** 안 코드입니다. 만약 해당 문제의 소스코드를 수정하고자 한다면, **파란색 사각형** 속 소스코드를 수정하는 것이 아니라 **빨간색 사각형** 안 소스코드를 수정해야 합니다.



Troubleshooting

만약 `/bin/bash^M`을 찾을 수 없다는 오류 메시지가 `./setup.sh`를 실행할 때 나온다면, 다음 명령어를 입력해 주세요.

```
dos2unix ./setup.sh
```