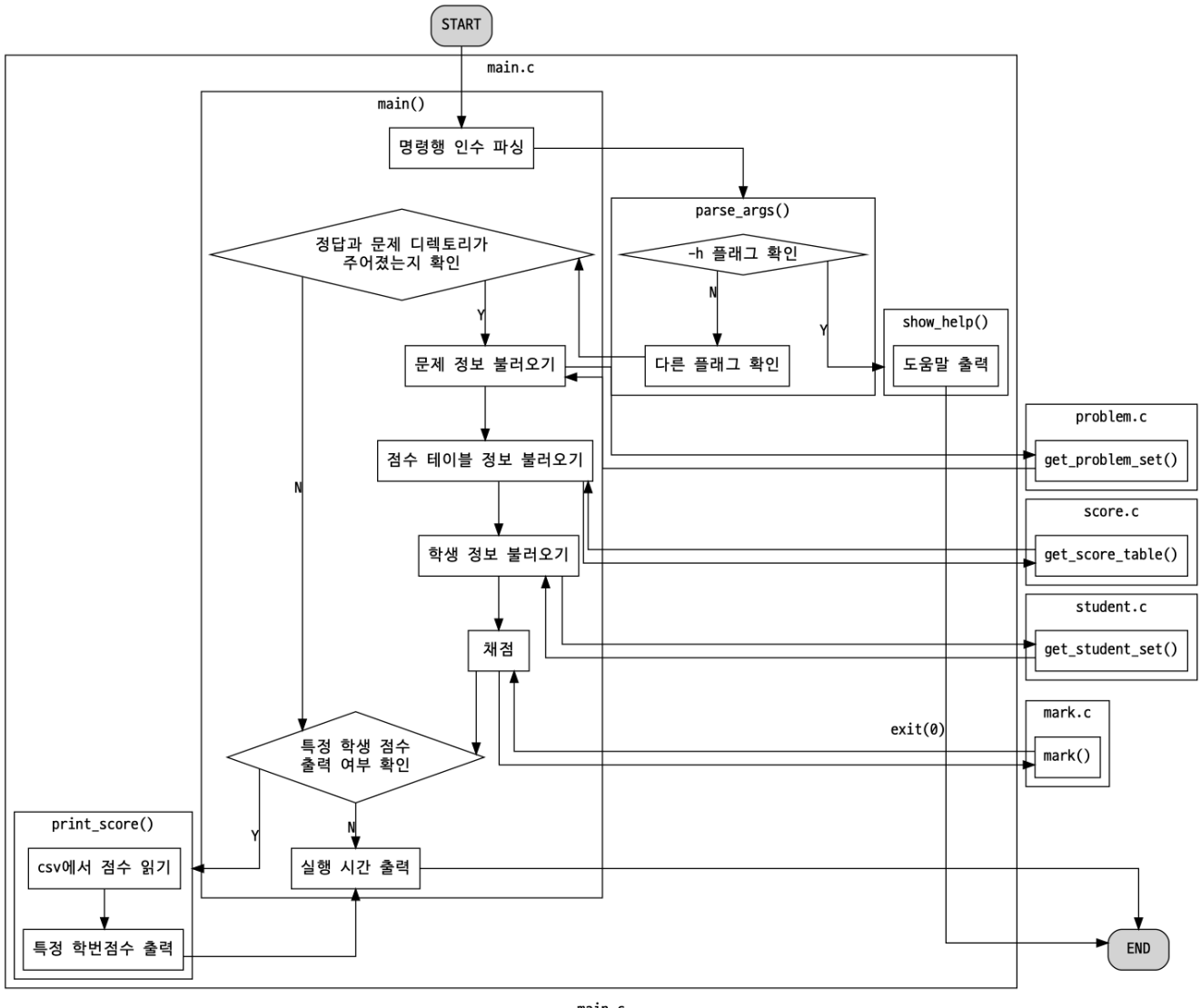
컴퓨터학부 20142314 김승호

1. 개요

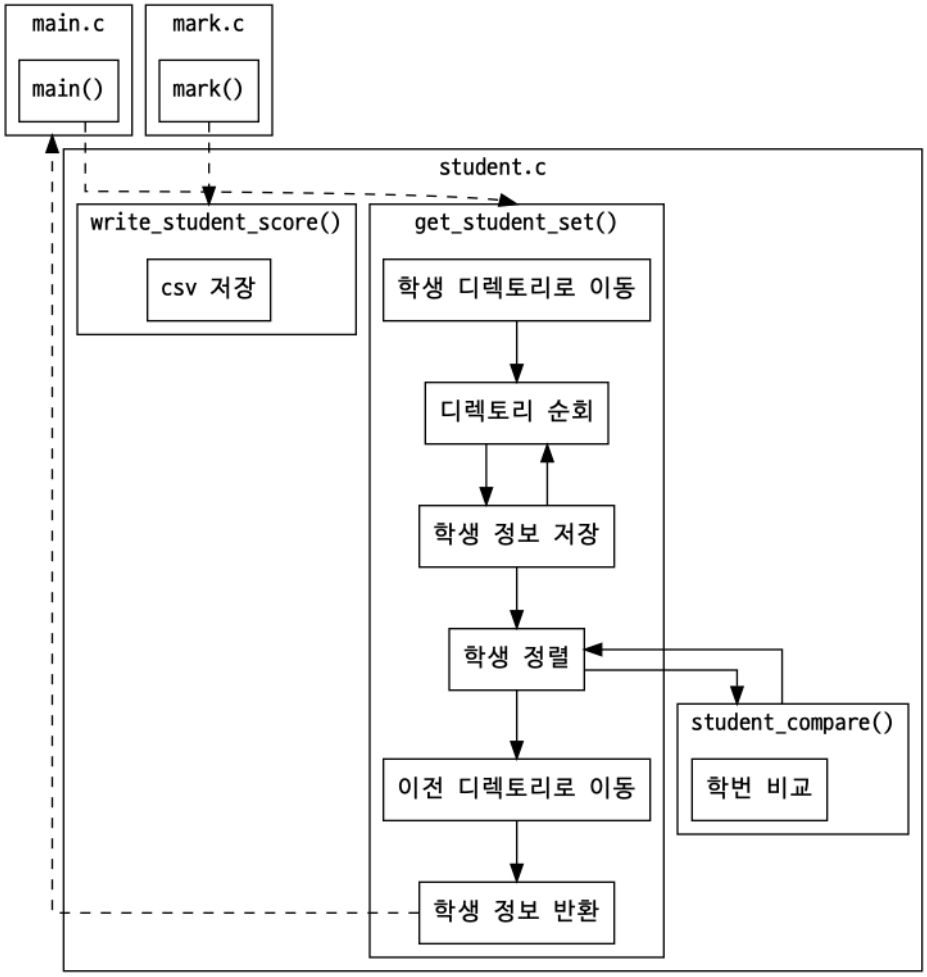
ssu_score는 정답 파일을 기준으로 학생들이 제출한 답안 파일을 채점하는 프로그램이다.
ssu_score가 채점할 수 있는 정답 파일의 종류는 두 가지인데, 바로 텍스트 파일과 코드이다.
텍스트 파일을 채점할 때에는 단순 비교뿐 아니라 연산자 우선 순위 등을 고려하여 같은 의미를 갖고 있다면 정답으로 처리되도록 하여야 한다.
코드의 경우 컴파일한 뒤, 컴파일의 성공 여부, 워닝 또는 에러 발생 여부, 정답을 출력하는지 등의 여부를 종합적으로 고려하여 점수에 반영하는 프로그램이다.
프로그램은 다양한 플래그를 명령행 인수로 받아 추가적인 기능을 수행할 수 있다.

2. 설계



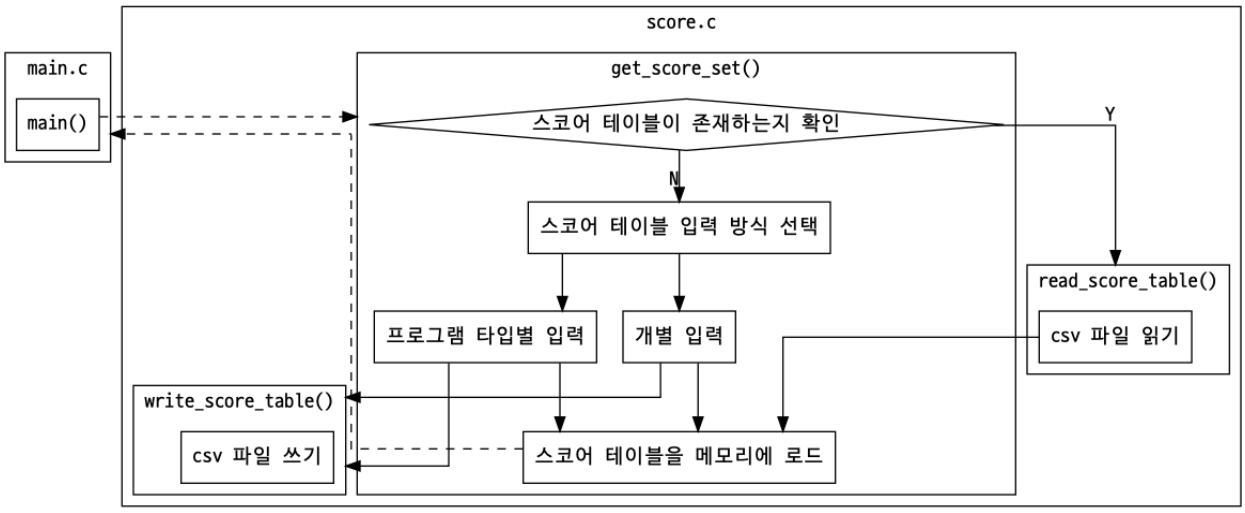
main.c

main.c에서는 프로그램의 전체적인 흐름과 명령행 인수의 처리를 담당한다.



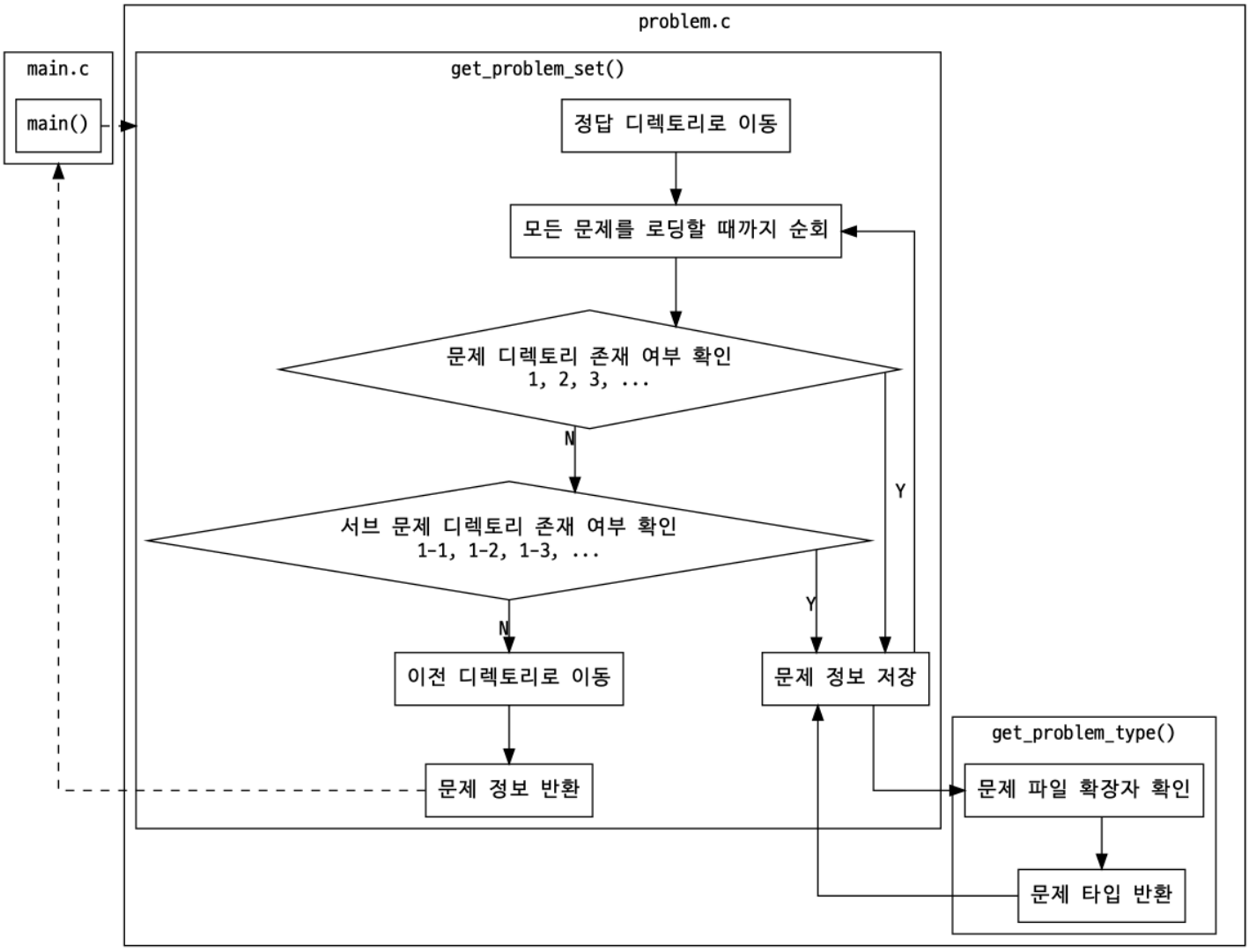
student.c

student.c는 디렉토리에서 학생의 정보를 읽어들인다.



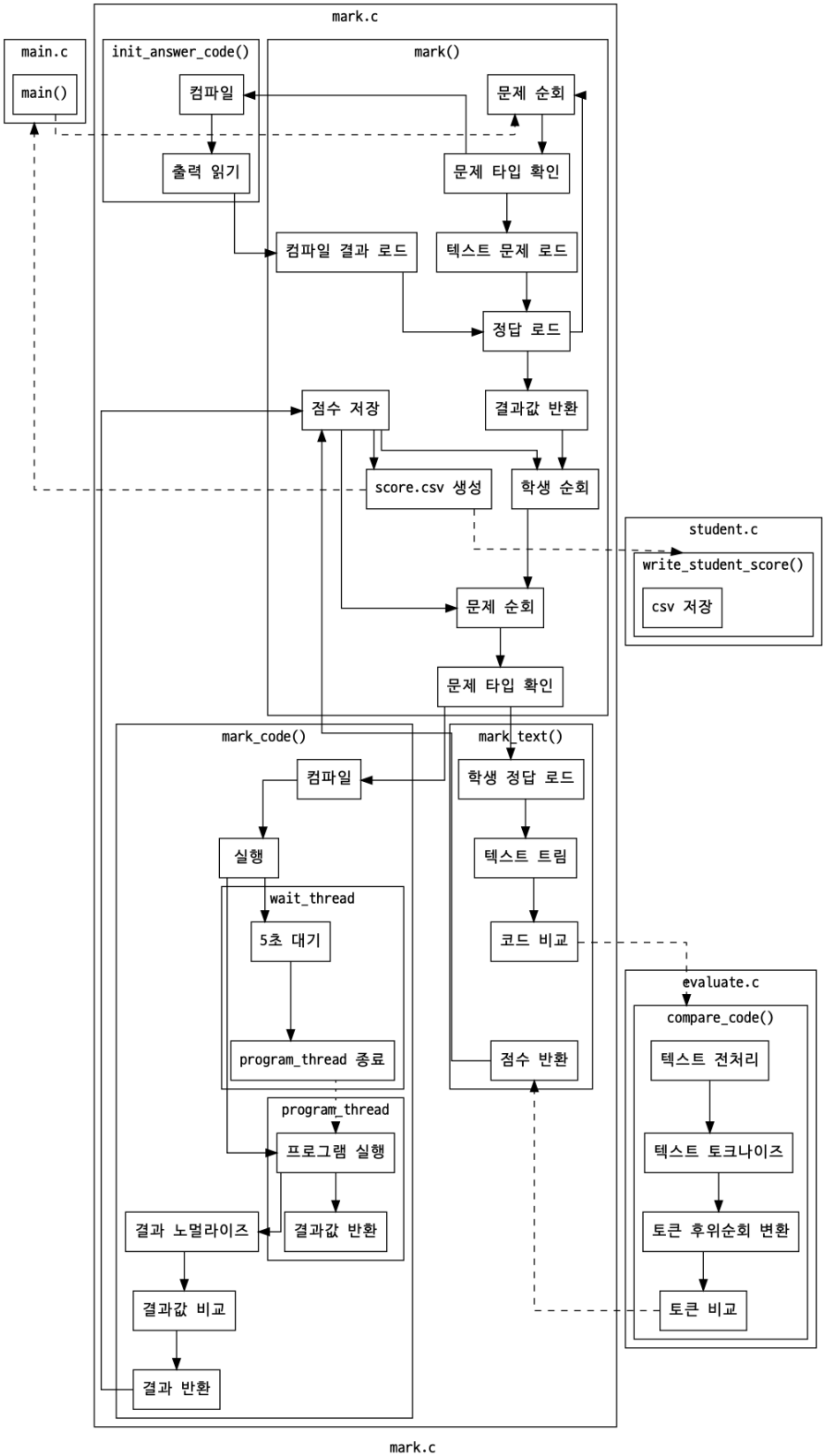
score.c

score.c는 스코어 테이블을 읽어오고 저장한다.



problem.c

problem는 디렉토리에서 문제에 대한 정보를 읽어들여 반환한다.



mark.c는 앞에서 말한 문제와 학생, 점수에 대한 정보를 읽어 프로그램을 채점한다,
컴파일을 하는 데에는 쓰레드가 이용되며, 텍스트를 검사하는 데에는 evaluate.c의 코드가 사용된다.

3. 구현

<main.c>
```
void parse_args(int argc, char *const argv[]); //명령행 인수 파싱
void print_score(void); //점수 표시(-c 옵션)
void show_help(void); //도움말 표시
int main(int argc, char *const argv[]);
```

<evaluate.c>
```
//(Postfix로의 변환을 위한) 스택 관련 연산자
void token_stack_init(void);
const char *token_stack_push(const char *token);
const char *token_stack_pop(void);
bool token_stack_is_empty(void);
const char *token_stack_top(void);

int token_stack_get_priority(const char *token); //연산자 우선순위 얻기
bool token_is_operator(const char *token); //연산자인지 확인
token_set_type *token_infix_to_postfix(token_set_type *token_set); //postfix로 변환

char *preprocess_code(const char *str); //코드 전처리
token_set_type *get_token_set(const char *str); // 토큰 분리
token_set_type *get_norm_token_set(const token_set_type *token_set); // 토큰 노멀라이즈
char *get_norm_text(const char *str); // 토큰을 텍스트로 바꾸기
bool compare_code(const char *str1, const char *str2); // 두 텍스트 비교
```

<mark.c>
```
void mark(const char *student_dir, const char *true_dir, const char *error_dir,
          const problem_set_type *problem_set,
          const score_table_type *score_table, student_set_type *student_set,
          const char qnames[5][16], const int qname_count, bool print_score); // 문제 채점

char *init_answer_code(const char *true_dir, const problem_type *problem,
                       const bool use_lpthread); // 정답 코드 초기화

double mark_text(const char *student_dir, const char *student_id,
                 const problem_type *problem, const char *ans,
                 const double problem_score); // 텍스트 채점

run_program_result_type mark_code(const char *student_dir,
```

```
                                const char *student_id,
                                const problem_type *problem,
                                const char *error_dir,
                                const bool use_lpthread); // 코드 채점

void *run_program(void *arg); // 프로그램 실행
void *wait_program(void *arg); // 프로그램 5초 대기후 run_program 캔슬
```

<problem.c>

```
ENUM_PROBLEM_TYPE get_problem_type(const char *problem_name); // 디렉토리에서 문제의 타입을 알아낸다.
problem_set_type *get_problem_set(const char *ans_dir_name); // 문제 세트를 가져온다.
```

<score.c>

```
score_table_type *read_score_table(const char *score_table_path); // 스코어 테이블을 읽는다.
void write_score_table(const char *score_table_path,
                       const score_table_type *score_table); // 스코어 테이블을 기록한다.
score_table_type *get_score_table(const char *true_dir,
                                const problem_set_type *problem_set); //스코어 테이블을 가져온다.
```

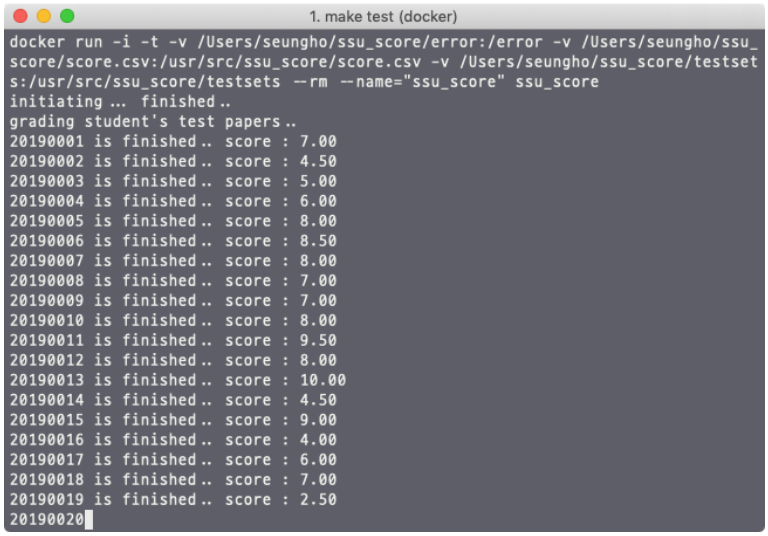
<student.c>

```
int student_compare(const void *s1, const void *s2); //두 학생의 학번을 비교한다.
student_set_type *get_student_set(const char *student_dir_name,
                                const problem_set_type *problem_set); // 학생 정보를 가져온다.
void write_student_score(const problem_set_type *problem_set,
                         const student_set_type *student_set); // score.csv를 기록한다.
```

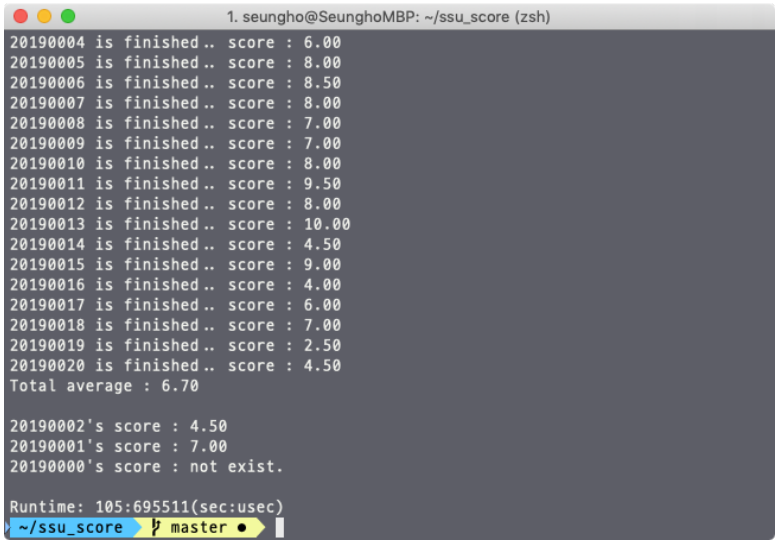
<util.c>

```
void ssu_runtime(struct timeval *begin_t, struct timeval *end_t); // 실행 시간 체크
bool file_exists(const char *pathname); // 파일이 존재하는지 확인
char *get_file_data(const char *pathname); // 파일을 읽어서 가져온다.
bool is_directory(const char *pathname); // 디렉토리인지 확인
bool string_ends_with(const char *str, const char *suffix); // 문자열이 다른 문자열로 끝나는지 확인
int string_count(const char *str, const char *substr); // 문자열이 몇 번 등장하는지 체크
char *string_trim(const char *str); // 양쪽 공백 제거
char *string_normalize_program_output(const char *str); // 프로그램 출력 노멀라이즈
bool string_compare_program_output(const char *str1, const char *str2); // 두 프로그램 출력 비교
void fatal_error_print(const char *format, ...); // 에러가 발생했을 때 strerr출력 + 비정상 종료
```

4. 테스트 및 결과

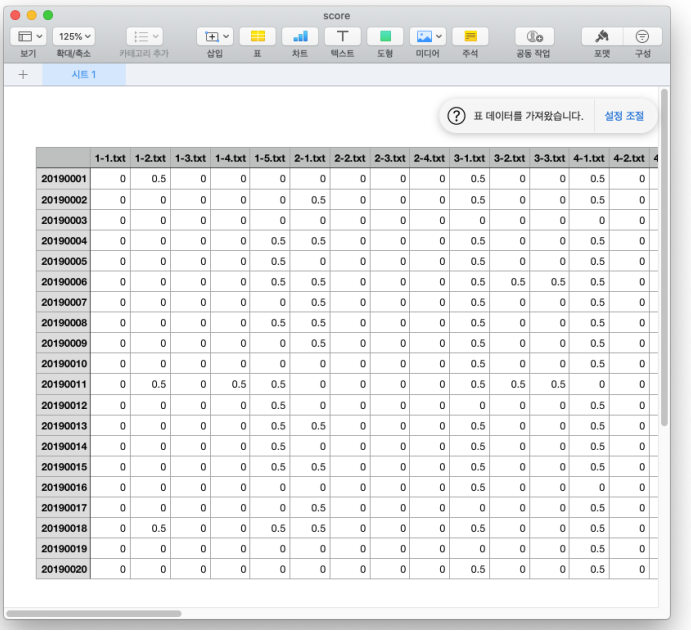


채점이 진행중인 화면
-p 옵션을 주어 점수가 출력되는 화면이다.
-t 옵션을 통해 12번 문제에 lpthread를 적용하여 실행중이다.



-c 옵션을 통해 csv 파일에서 원하는 학번의 점수를 읽어 들일 수 있다.
존재하지 않는 학번인 경우에는 에러 메시지를 출력한다.
프로그램이 종료되면 수행 시간을 출력한다.

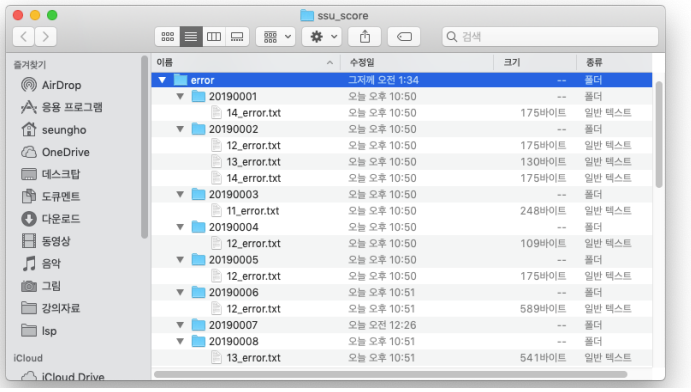
프로그램을 컴파일하여 비교하는 것은 명세에 맞게 잘 비교한다고 할 수 있으나, 텍스트 비교의 경우에는 문제가 존재한다.
프로그램의 토큰의 비교가 완전하지 못해서 채점되어야 할 점수보다 점수가 낮다.
토큰화와 노멀라이즈의 개선이 요구된다.



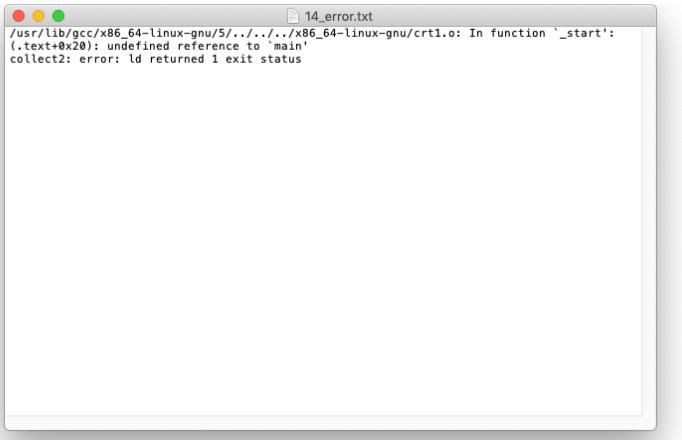
	1-1.txt	1-2.txt	1-3.txt	1-4.txt	1-5.txt	2-1.txt	2-2.txt	2-3.txt	2-4.txt	3-1.txt	3-2.txt	3-3.txt	4-1.txt	4-2.txt
20190001	0	0.5	0	0	0	0	0	0	0	0.5	0	0	0.5	0
20190002	0	0	0	0	0	0.5	0	0	0	0.5	0	0	0.5	0
20190003	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20190004	0	0	0	0	0.5	0.5	0	0	0	0.5	0	0	0.5	0
20190005	0	0	0	0	0.5	0	0	0	0	0.5	0	0	0.5	0
20190006	0	0	0	0	0.5	0.5	0	0	0	0.5	0.5	0.5	0.5	0
20190007	0	0	0	0	0	0.5	0	0	0	0.5	0	0	0.5	0
20190008	0	0	0	0	0.5	0.5	0	0	0	0.5	0	0	0.5	0
20190009	0	0	0	0	0	0.5	0	0	0	0.5	0	0	0.5	0
20190010	0	0	0	0	0	0	0	0	0	0.5	0	0	0.5	0
20190011	0	0.5	0	0.5	0.5	0	0	0	0	0.5	0.5	0.5	0	0
20190012	0	0	0	0	0.5	0	0	0	0	0	0	0	0.5	0
20190013	0	0	0	0	0.5	0.5	0	0	0	0.5	0	0	0.5	0
20190014	0	0	0	0	0.5	0	0	0	0	0.5	0	0	0.5	0
20190015	0	0	0	0	0.5	0.5	0	0	0	0.5	0	0	0.5	0
20190016	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0
20190017	0	0	0	0	0	0.5	0	0	0	0	0	0	0.5	0
20190018	0	0.5	0	0	0.5	0.5	0	0	0	0.5	0	0	0.5	0
20190019	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0
20190020	0	0	0	0	0	0	0	0	0	0.5	0	0	0.5	0

score.csv 파일 출력 화면

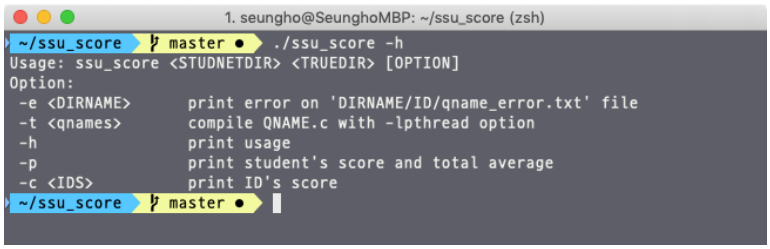
프로그램이 끝나고 나면 score.csv 파일이 출력된다.



-e 옵션을 통해 에러가 출력된 모습이다.



다음과 같이 컴파일 도중 발생한 메시지가 저장된다.



-h 옵션을 통해 도움말을 읽어들일 수 있다.

5. 소스코드

<<evaluate.c>>

```
#include "evaluate.h"

const char *token_stack[200];
int _token_stack_top;

// postfix 전환에 사용되는 스택 연산자

void token_stack_init(void) { _token_stack_top = -1; }

const char *token_stack_push(const char *token)
{
  if (_token_stack_top >= 199) {
    return NULL;
  }
  token_stack[++_token_stack_top] = token;
  return token;
}

const char *token_stack_pop(void)
{
  if (_token_stack_top < 0) {
    return NULL;
  }
  return token_stack[_token_stack_top--];
}

bool token_stack_is_empty(void) { return (_token_stack_top < 0); }
```

```
const char *token_stack_top(void)
{
  return token_stack_is_empty() ? NULL : token_stack[_token_stack_top];
}

bool token_is_operator(const char *token)
{
  const char *op[] = {"++",   "--",   ".",   "->", "!",   "~",   "&",   "sizeof",
                      "*",    "/",    "%",   "+",   "-",   "<<", ">>", "<",
                      ">",    ">=",   "<=", "==", "!=", "&",   "^",   "|",
                      "&&",   "||",   "=",   "+=", "-=", "*=", "/=", "%=",
                      "<<=", ">>=", "&=", "^=", "|=", ","};
  for (int i = 0; i < 38; ++i) {
    if (strcmp(token, op[i]) == 0)
      return true;
  }
  return false;
}

int token_stack_get_priority(const char *token)
{
  const char *braces[] = {"(", "["};
  for (int i = 0; i < 2; ++i) {
    if (strcmp(token, braces[i]) == 0)
      return 0;
  }
  const char *p0[] = {"++", "--", ".", "->"};
  for (int i = 0; i < 4; ++i) {
    if (strcmp(token, p0[i]) == 0)
      return 14;
  }
  const char *p1[] = {"!", "~", "&", "sizeof"};
  for (int i = 0; i < 4; ++i) {
    if (strcmp(token, p1[i]) == 0)
      return 13;
  }
  const char *p2[] = {"*", "/", "%"};
  for (int i = 0; i < 3; ++i) {
    if (strcmp(token, p2[i]) == 0)
      return 12;
  }
  const char *p3[] = {"+", "-"};
  for (int i = 0; i < 2; ++i) {
    if (strcmp(token, p3[i]) == 0)
```

```
      return 11;
  }
  const char *p4[] = {"<<", ">>"};
  for (int i = 0; i < 2; ++i) {
    if (strcmp(token, p4[i]) == 0)
      return 10;
  }
  const char *p5[] = {"<", ">", ">=", "<="};
  for (int i = 0; i < 4; ++i) {
    if (strcmp(token, p5[i]) == 0)
      return 9;
  }
  const char *p6[] = {"==", "!="};
  for (int i = 0; i < 2; ++i) {
    if (strcmp(token, p6[i]) == 0)
      return 8;
  }
  if (strcmp(token, "&") == 0)
    return 7;
  if (strcmp(token, "^") == 0)
    return 6;
  if (strcmp(token, "|") == 0)
    return 5;
  if (strcmp(token, "&&") == 0)
    return 4;
  if (strcmp(token, "||") == 0)
    return 3;
  const char *p12[] = {
      "=", "+=", "-=", "*=", "/=", "%=", "<<=", ">>=", "&=", "^=", "|="};
  for (int i = 0; i < 11; ++i) {
    if (strcmp(token, p12[i]) == 0)
      return 2;
  }
  if (strcmp(token, ",") == 0)
    return 1;
  return 100;
}

token_set_type *token_infix_to_postfix(token_set_type *token_set)
{
  // 스택을 이용하여 postfix로 변환한다.
  token_set_type *ret = (token_set_type *)malloc(sizeof(token_set_type));
  ret->size = 0;
  token_stack_init();
```

```
  int i, j;
  for (i = 0, j = 0; i < token_set->size; ++i) {
    const char *cur = token_set->token[i];
    if (strcmp(cur, "(") == 0) {
      token_stack_push(cur);
    }
    else if (strcmp(cur, ")") == 0) {
      while (!token_stack_is_empty() && strcmp(token_stack_top(), "(") != 0) {
        strcpy(ret->token[j], token_stack_pop());
        ret->size = ++j;
      }
      token_stack_pop();
    }
    else if (strcmp(cur, "[") == 0) {
      token_stack_push(cur);
    }
    else if (strcmp(cur, "]") == 0) {
      while (strcmp(token_stack_top(), "[") != 0) {
        strcpy(ret->token[j], token_stack_pop());
        ret->size = ++j;
      }
      token_stack_pop();
    }
    else if (token_is_operator(cur)) {
      while (!token_stack_is_empty() &&
             (token_stack_get_priority(token_stack_top()) >=
              token_stack_get_priority(cur))) {
        strcpy(ret->token[j], token_stack_pop());
        ret->size = ++j;
      }
      token_stack_push(cur);
    }
    else {
      token_stack_push(cur);
    }
  }

  while (!token_stack_is_empty()) {
    strcpy(ret->token[j], token_stack_pop());
    ret->size = ++j;
  }
  return ret;
}
```

```
char *preprocess_code(const char *str)
{
  // 문자열 안 공백 제거
  char *ret = (char *)malloc(strlen(str) + 1);
  bool dqoute = false;
  size_t i, j;
  for (i = 0, j = 0; i < strlen(str); i++, j++) {
    if (str[i] == '"') {
      if (i == 0 || str[i - 1] != '\\') {
        dqoute = !dqoute;
      }
    }
    if (str[i] != ' ' || !dqoute)
      ret[j] = str[i];
    else
      j--;
  }
  ret[j] = '\0';
  return ret;
}

token_set_type *get_token_set(const char *str)
{
  // strtok_r을 통해 문자열 토크나이징하여 키워드 셋에 저장
  keyword_set_type keyword_set;
  keyword_set.size = 0;
  const char *code_delim = "+-*/!%^|&()<>[],.:';\\=\"~ ";
  char *tmp = strdup(str);
  char *lasts;
  char *token = strtok_r(tmp, code_delim, &lasts);
  while (token != NULL) {
    strcpy(keyword_set.keyword[keyword_set.size], token);
    keyword_set.size++;
    token = strtok_r(NULL, code_delim, &lasts);
  }
  free(tmp);
  tmp = NULL;

  const char *cur = str;

  // 키워드를 다시 매칭하여 토큰셋으로 저장
  int keyword_index = 0;
  token_set_type *token_set = (token_set_type *)malloc(sizeof(token_set_type));
  token_set->size = 0;
```

```
  while (*cur != '\0') {
    if (*cur == ' ') {
      ++cur;
      continue;
    }
    if (keyword_index < keyword_set.size) {
      char *keyword = keyword_set.keyword[keyword_index];
      if (strncmp(cur, keyword, strlen(keyword)) == 0) {
        strcpy(token_set->token[token_set->size], keyword);
        token_set->size++;
        cur += strlen(keyword);
        ++keyword_index;
        continue;
      }
    }
    if (strchr(code_delim, *cur) != NULL) {
      sprintf(token_set->token[token_set->size], "%c", *cur);
      token_set->size++;
    }
    ++cur;
  }
  return token_set;
}

token_set_type *get_norm_token_set(const token_set_type *token_set)
{
  // 2자 이상의 연산자를 하나의 토큰으로 합친다.
  token_set_type *ret = (token_set_type *)malloc(sizeof(token_set_type));
  int i, j;
  for (i = 0, j = 0; i < token_set->size; i++, j++) {
    const char *cur = token_set->token[i];
    const char *prev = ret->token[j - 1];
    char *repeats[] = {"&", "|", "+", "-", ">", "<", "="};
    char *equals[] = {"+", "-",  "*",  "/", "%", "!", "<",
                      ">", "<<", ">>", "^", "|", "~"};
    bool matched = false;
    for (int k = 0; k < 7; ++k) {
      char *sym = repeats[k];
      if (j > 0 && strcmp(cur, sym) == 0 && strcmp(prev, sym) == 0) {
        matched = true;
        --j;
        strcat(ret->token[j], sym);
        ret->size = j + 1;
        break;
```

```
      }
    }
    if (matched)
      continue;
    for (int k = 0; k < 13; ++k) {
      char *sym = equals[k];
      if (j > 0 && strcmp(cur, "=") == 0 && strcmp(prev, sym) == 0) {
        matched = true;
        --j;
        strcat(ret->token[j], "=");
        ret->size = j + 1;
        break;
      }
    }
    if (matched)
      continue;
    if (i > 0 && strcmp(cur, ">") == 0 && strcmp(prev, "-") == 0) {
      --j;
      strcat(ret->token[j], ">");
      ret->size = j + 1;
    }
    else {
      strcpy(ret->token[j], token_set->token[i]);
      ret->size = j + 1;
    }
  }
  return ret;
}

char *get_norm_text(const char *str)
{
  // 문자열을 토큰화하여 postfix로 바꾸고 다시 문자열로 변환
  char *str_preprocessed = preprocess_code(str);
  token_set_type *token_set = get_token_set(str_preprocessed);
  free(str_preprocessed);
  token_set_type *norm_token_set = get_norm_token_set(token_set);
  token_set_type *post_token_set = token_infix_to_postfix(norm_token_set);

  char *ret = (char *)malloc(sizeof(char) * 1024 * 200);
  strcpy(ret, "");
  for (int i = 0; i < post_token_set->size; ++i) {
    strcat(ret, post_token_set->token[i]);
  }
```

```
  free(token_set);
  free(norm_token_set);
  free(post_token_set);
  return ret;
}

bool compare_code(const char *str1, const char *str2)
{
  // 두 코드 비교
  bool ret;
  char *norm_str1 = get_norm_text(str1);
  char *norm_str2 = get_norm_text(str2);
  ret = strcmp(norm_str1, norm_str2) == 0;
  free(norm_str1);
  free(norm_str2);
  return ret;
}
```

<<evaluate.h>>

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

// 키워드 저장 타입
typedef struct _keyword_set_type {
  int size;
  char keyword[100][1024];
} keyword_set_type;

// 토큰 저장 타입
typedef struct _token_set_type {
  int size;
  char token[200][1024];
} token_set_type;

//(Postfix로의 변환을 위한) 스택 관련 연산자
void token_stack_init(void);
const char *token_stack_push(const char *token);
```

```
const char *token_stack_pop(void);
bool token_stack_is_empty(void);
const char *token_stack_top(void);

//연산자 우선순위 얻기
int token_stack_get_priority(const char *token);
//연산자인지 확인
bool token_is_operator(const char *token);
// postfix로 변환
token_set_type *token_infix_to_postfix(token_set_type *token_set);

//코드 전처리
char *preprocess_code(const char *str);
// 토큰 분리
token_set_type *get_token_set(const char *str);
// 토큰 노멀라이즈
token_set_type *get_norm_token_set(const token_set_type *token_set);
// 토큰을 텍스트로 바꾸기
char *get_norm_text(const char *str);
// 두 텍스트 비교
bool compare_code(const char *str1, const char *str2);
```

<<main.c>>

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>

#include "util.h"
#include "problem.h"
#include "score.h"
#include "student.h"
#include "mark.h"

// 명령행 인자  관련 변수
bool flag_e = false, flag_t = false, flag_p = false, flag_c = false;
int qname_count = 0, id_count = 0;
char *student_dir = NULL, *true_dir = NULL, *error_dir = NULL;
char qnames[5][16], ids[5][16];
```

```
void parse_args(int argc, char *const argv[]); //명령행 인수 파싱
void print_score(void);                        //점수 표시(-c 옵션)
void show_help(void);                          //도움말 표시

int main(int argc, char *const argv[])
{
  struct timeval begin_t, end_t;
  gettimeofday(&begin_t, NULL);

  parse_args(argc, argv);

  if (error_dir != NULL) {
    if (strcmp(error_dir, student_dir) == 0 ||
        strcmp(error_dir, true_dir) == 0) {
      fatal_error_print("error directory is not valid!");
    }
  }

  // 디렉토리 정보가 온전한 경우만 채점을 실시한다.
  if (student_dir != NULL && true_dir != NULL) {
    // 문제, 점수, 학생 정보 가져오기
    problem_set_type *problem_set = get_problem_set(true_dir);
    score_table_type *score_table = get_score_table(true_dir, problem_set);
    student_set_type *student_set = get_student_set(student_dir, problem_set);
    // 채점
    mark(student_dir, true_dir, error_dir, problem_set, score_table,
         student_set, qnames, qname_count, flag_p);
    // csv출력
    write_student_score(problem_set, student_set);

    free(problem_set);
    free(score_table);
    free(student_set);
  }

  // -c 옵션을 받은 경우
  if (id_count > 0) {
    print_score();
  }

  gettimeofday(&end_t, NULL);
  ssu_runtime(&begin_t, &end_t);
```

```
  exit(0);
  return 0;
}

void parse_args(int argc, char *const argv[])
{
  // -h 옵션을 받았으면 도움말을 출력하고 바로 종료
  for (int i = 0; i < argc; ++i) {
    if (strcmp(argv[i], "-h") == 0 || strcmp(argv[i], "--help") == 0) {
      show_help();
      exit(0);
    }
  }

  extern int optind, opterr;
  extern char *optarg;
  // -c 옵션이 처음 올 때
  if (argc > 2 && (strcmp(argv[1], "-c") == 0)) {
    int c;
    while ((c = getopt(argc, argv, "c:")) != -1) {
      switch (c) {
      case 'c':
        flag_c = true;
        optind -= 1;
        for (; optind < argc && *argv[optind] != '-'; ++optind) {
          if (*argv[optind] == '-')
            break;
          if (id_count >= 5)
            continue;
          strcpy(ids[id_count], argv[optind]);
          id_count += 1;
        }
        break;
      case '?':
        fatal_error_print("invalid option: %s", argv[optind]);
        break;
      }
    }
    return;
  }

  // 디렉토리가 먼저 주어지는 경우
  if (argc < 3) {
    fprintf(stderr, "Usage: ssu_score <STUDNETDIR> <TRUEDIR> [OPTION]\n");
```

```
  exit(1);
}

optind += 2;
opterr = 0;
int c;
// 명령행 인자를 받아온다.
while ((c = getopt(argc, argv, "e:t:pc:")) != -1) {
  switch (c) {
  case 'e':
    flag_e = true;
    error_dir = optarg;
    break;
  case 't':
    flag_t = true;
    optind -= 1;
    for (; optind < argc && *argv[optind] != '-'; ++optind) {
      if (qname_count >= 5)
        continue;
      strcpy(qnames[qname_count], argv[optind]);
      qname_count += 1;
    }
    break;
  case 'p':
    flag_p = true;
    break;
  case 'c':
    flag_c = true;
    optind -= 1;
    for (; optind < argc && *argv[optind] != '-'; ++optind) {
      if (*argv[optind] == '-')
        break;
      if (id_count >= 5)
        continue;
      strcpy(ids[id_count], argv[optind]);
      id_count += 1;
    }
    break;
  case '?':
    fatal_error_print("invalid option: %s", argv[optind]);
    break;
  }
};
student_dir = argv[1];
```

```
  true_dir = argv[2];
}

void print_score(void)
{
  printf("\n");
  char *fname = "score.csv";
  if (!file_exists(fname)) {
    fatal_error_print("score.csv file not exist.");
  }
  // csv에서 점수를 가져온다.
  FILE *score_csv;
  if ((score_csv = fopen(fname, "r")) == NULL) {
    fatal_error_print("file open error for score.csv");
  }
  // 학번을 순회하며 점수를 출력한다.
  for (int i = 0; i < id_count; ++i) {
    const char *id = ids[i];
    char buf[1024];
    bool matched = false;
    while (!feof(score_csv)) {
      fgets(buf, 1024, score_csv);

      if (strncmp(id, buf, strlen(id)) == 0 && buf[strlen(id)] == ',') {
        matched = true;
        char *score = strrchr(buf, ',') + 1;
        printf("%s's score : %s", id, score);
        fseek(score_csv, 0, SEEK_SET);
        break;
      }
    }
    if (!matched) {
      printf("%s's score : not exist.\n", id);
    }
  }
  printf("\n");
}

void show_help(void)
{
  const char *help_msg =
      "Usage: ssu_score <STUDNETDIR> <TRUEDIR> [OPTION]\n"
      "Option:\n"
      " -e <DIRNAME>        print error on 'DIRNAME/ID/qname_error.txt' file\n"
```

```
      " -t <qnames>        compile QNAME.c with -lpthread option\n"
      " -h                 print usage\n"
      " -p                 print student's score and total average\n"
      " -c <IDS>           print ID's score\n";
  printf("%s", help_msg);
}
```

<<mark.c>>

```
#include "mark.h"

void mark(const char *student_dir, const char *true_dir, const char *error_dir,
          const problem_set_type *problem_set,
          const score_table_type *score_table, student_set_type *student_set,
          const char qnames[5][16], const int qname_count, bool print_score)
{
  double avg_score = 0;
  printf("initiating...");
  fflush(stdout);

  answer_set_type answer_set;
  answer_set.size = problem_set->size;

  if (error_dir != NULL) {
    if (!file_exists(error_dir)) {
      if (mkdir(error_dir, 0755) < 0) {
        fatal_error_print("mkdir error for %s", error_dir);
      }
    }
  }

  // 문제 순회
  for (int i = 0; i < problem_set->size; ++i) {
    const problem_type *problem = &(problem_set->problem[i]);
    if (problem->type == PROBLEM_CODE) {
      bool use_lpthread = false;
      for (int k = 0; k < qname_count; ++k) {
        if (strcmp(problem->name, qnames[k]) == 0) {
          use_lpthread = true;
          break;
        }
      }
```

```
    // 정답 컴파일
    answer_set.answer[i] = init_answer_code(true_dir, problem, use_lpthread);
  }
  else {
    char ans_path[PATH_MAX];
    sprintf(ans_path, "%s/%s/%s.txt", true_dir, problem->name, problem->name);
    answer_set.answer[i] = get_file_data(ans_path);
  }
}

printf(" finished..\n");
printf("grading student's test papers..\n");

// 학생 채점
for (int i = 0; i < student_set->size; ++i) {
  student_type *const student = &(student_set->students[i]);
  printf("%s", student->id);
  fflush(stdout);
  if (error_dir != NULL) {
    char student_error_dir[PATH_MAX];
    sprintf(student_error_dir, "%s/%s", error_dir, student->id);
    if (!file_exists(student_error_dir)) {
      if (mkdir(student_error_dir, 0755) < 0) {
        fatal_error_print("mkdir error for %s", student_error_dir);
      }
    }
  }

  for (int j = 0; j < problem_set->size; ++j) {
    const char *answer = answer_set.answer[j];
    const problem_type *problem = &(problem_set->problem[j]);
    const double problem_score = score_table->row[j].score;
    double *const student_score = &(student->score.row[j].score);
    // 텍스트인 경우
    if (problem->type == PROBLEM_TEXT) {
      *student_score =
          mark_text(student_dir, student->id, problem, answer, problem_score);
    }
    // 코드인 경우
    else if (problem->type == PROBLEM_CODE) {
      bool use_lpthread = false;
      for (int k = 0; k < qname_count; ++k) {
        if (strcmp(problem->name, qnames[k]) == 0) {
          use_lpthread = true;
```

```
            break;
          }
        }
        *student_score = PROGRAM_ERROR_SCORE;
        run_program_result_type program_result = mark_code(
            student_dir, student->id, problem, error_dir, use_lpthread);
        if (program_result.status == PROGRAM_SUCCESS) {
          if (string_compare_program_output(answer, program_result.output)) {
            *student_score = problem_score;
          }
        }
        if (program_result.error != NULL) {
          *student_score -= string_count(program_result.error, "warning:") *
                            PROGRAM_WARNING_DEMERIT;
        }
        if (*student_score < 0) {
          *student_score = 0;
        }
        if (program_result.error != NULL) {
          free(program_result.error);
          program_result.error = NULL;
        }
        if (program_result.output != NULL) {
          free(program_result.output);
          program_result.output = NULL;
        }
      }
      else {
        fatal_error_print("data is not valid.");
      }
    }
    printf(" is finished..");
    if (print_score) {
      double score = 0;
      for (int j = 0; j < student->score.size; ++j) {
        score += student->score.row[j].score;
      }
      avg_score += score;
      printf(" score : %.2lf", score);
    }
    printf("\n");
  }
  avg_score /= student_set->size;
  printf("Total average : %.2lf\n", avg_score);
```

```
}

char *init_answer_code(const char *true_dir, const problem_type *problem,
                       const bool use_lpthread)
{
  char cwd[PATH_MAX];
  getcwd(cwd, PATH_MAX);

  char ans_path[PATH_MAX];
  sprintf(ans_path, "%s/%s", true_dir, problem->name);
  if (chdir(ans_path) == -1) {
    fatal_error_print("chdir error for %s", ans_path);
  }

  char answer_path[PATH_MAX];

  sprintf(answer_path, "%s.stdout", problem->name);
  if (!file_exists(answer_path)) {
    int stderr_backup = dup(2);
    int fd_err = open("/dev/null", O_WRONLY, 0664);
    if (fd_err < 0) {
      fatal_error_print("file open error for /dev/null");
    }
    dup2(fd_err, 2);

    int stdout_backup = dup(1);
    int fd_out = open("/dev/null", O_WRONLY, 0664);
    if (fd_out < 0) {
      fatal_error_print("file open error for /dev/null");
    }
    dup2(fd_out, 1);

    char cmd[LINE_MAX];
    sprintf(cmd, "gcc %s.c -o %s.exe %s", problem->name, problem->name,
            use_lpthread ? "-lpthread" : "");
    system(cmd);

    dup2(stdout_backup, 1);
    close(fd_out);

    fd_out = open(answer_path, O_WRONLY | O_CREAT | O_TRUNC, 0664);
    if (fd_out < 0) {
      fatal_error_print("file open error for %s", answer_path);
    }
```

```
    dup2(fd_out, 1);

    sprintf(cmd, "./%s.exe", problem->name);
    system(cmd);

    dup2(stdout_backup, 1);
    close(stdout_backup);
    close(fd_out);

    dup2(stderr_backup, 2);
    close(stderr_backup);
    close(fd_err);
  }
  char *ret = get_file_data(answer_path);
  chdir(cwd);
  return ret;
}

double mark_text(const char *student_dir, const char *student_id,
                 const problem_type *problem, const char *ans,
                 const double problem_score)
{
  char student_ans_path[PATH_MAX];
  sprintf(student_ans_path, "%s/%s/%s.txt", student_dir, student_id,
          problem->name);

  char *student_ans = get_file_data(student_ans_path);
  if (student_ans == NULL) {
    return 0;
  }

  char *ans_dup = strdup(ans);
  char *lasts;
  char *token = strtok_r(ans_dup, ":", &lasts);
  char *student_answer = string_trim(student_ans);
  while (token != NULL) {
    char *answer = string_trim(token);
    if (strcmp(student_answer, answer) == 0) {
      return problem_score;
    }
    if (compare_code(student_answer, answer)) {
      free(ans_dup);
      ans_dup = NULL;
      return problem_score;
```

```
    }
    free(answer);
    answer = NULL;
    token = strtok_r(NULL, ":", &lasts);
  }
  free(student_answer);
  student_answer = NULL;
  free(ans_dup);
  ans_dup = NULL;
  return 0;
}

run_program_result_type mark_code(const char *student_dir,
                                  const char *student_id,
                                  const problem_type *problem,
                                  const char *error_dir,
                                  const bool use_lpthread)
{
  run_program_result_type ret = {PROGRAM_SUCCESS, NULL, NULL};

  char cwd[PATH_MAX];
  getcwd(cwd, PATH_MAX);

  char error_out_path[PATH_MAX] = "/dev/null";
  sprintf(error_out_path, "%s/%s/%s_error.txt", error_dir, student_id,
          problem->name);

  char problem_path[PATH_MAX];
  sprintf(problem_path, "%s/%s", student_dir, student_id);
  if (chdir(problem_path) == -1) {
    fatal_error_print("chdir error for %s", problem_path);
  }
  // 코드 경로
  char source_path[PATH_MAX];
  sprintf(source_path, "%s.c", problem->name);
  // 출력 경로
  char stdout_path[PATH_MAX];
  sprintf(stdout_path, "%s.stdout", problem->name);
  // 바이너리 경로
  char stdexe_path[PATH_MAX];
  sprintf(stdexe_path, "%s.stdexe", problem->name);
  // 에러 경로
  char stderr_path[PATH_MAX];
  sprintf(stderr_path, "%s.stderr", problem->name);
```

```
if (!file_exists(source_path)) {
  chdir(cwd);
  ret.status = PROGRAM_NOT_EXIST;
  return ret;
}

int stderr_backup = dup(2);
int fd = open(stderr_path, O_WRONLY | O_CREAT | O_TRUNC, 0664);
if (fd < 0) {
  fatal_error_print("file open error for &s", stderr_path);
}
dup2(fd, 2);

char cmd[LINE_MAX];
sprintf(cmd, "gcc %s.c -o %s.stdexe %s", problem->name, problem->name,
        use_lpthread ? "-lpthread" : "");
system(cmd);

dup2(stderr_backup, 2);
close(stderr_backup);
close(fd);

if (file_exists(stdexe_path)) {
  int stdout_backup = dup(1);
  int fd = open(stdout_path, O_WRONLY | O_CREAT | O_TRUNC, 0664);
  if (fd < 0) {
    fatal_error_print("file open error for &s", stdout_path);
  }
  dup2(fd, 1);

  int stderr_backup = dup(2);
  int fd_err = open("/dev/null", O_WRONLY, 0664);
  if (fd_err < 0) {
    fatal_error_print("file open error for /dev/null");
  }
  dup2(fd_err, 2);

  // 실행 쓰레드
  pthread_t program_thread;
  run_program_arg_type run_arg;
  strcpy(run_arg.problem_name, problem->name);
  run_program_status *program_status = NULL;
  pthread_create(&program_thread, NULL, run_program, &run_arg);
```

```
  // 대기 쓰레드
  pthread_t wait_thread;
  wait_program_arg_type wait_arg;
  wait_arg.tid = &program_thread;
  pthread_create(&wait_thread, NULL, wait_program, &wait_arg);

  pthread_join(program_thread, (void **)&program_status);
  if (pthread_kill(wait_thread, 0) == 0) {
    pthread_cancel(wait_thread);
  }
  pthread_join(wait_thread, NULL);

  dup2(stderr_backup, 2);
  close(stderr_backup);
  close(fd_err);

  dup2(stdout_backup, 1);
  close(stdout_backup);
  close(fd);

  if (program_status != PTHREAD_CANCELED) {
    ret.status = *program_status;
    free(program_status);
  }
  else {
    // 제한 시간 경과
    ret.status = PROGRAM_TIMEOUT;
    system("pkill -9 stdexe");
  }
}
else {
  ret.status = PROGRAM_COMPILE_ERROR;
}
// 출력, 워닝 리턴
ret.output = get_file_data(stdout_path);
ret.error = get_file_data(stderr_path);
remove(stderr_path);
chdir(cwd);
if (error_dir != NULL) {
  if (file_exists(error_out_path)) {
    remove(error_out_path);
  }
  if (ret.error != NULL) {
```

```
        int fd_err;
        if ((fd_err = open(error_out_path, O_WRONLY | O_CREAT | O_TRUNC, 0666)) <
            0) {
          fatal_error_print("open error for %s", error_out_path);
        }
        write(fd_err, ret.error, strlen(ret.error));
        close(fd_err);
      }
    }
    return ret;
}

void *run_program(void *arg)
{
    run_program_status *ret = malloc(sizeof(run_program_status));
    *ret = PROGRAM_SUCCESS;
    const char *problem_name = ((run_program_arg_type *)arg)->problem_name;
    char cmd[LINE_MAX];
    sprintf(cmd, "./%s.stdexe", problem_name);
    if (system(cmd) != 0) {
      *ret = PROGRAM_RUNTIME_ERROR;
    }
    return ret;
}

void *wait_program(void *arg)
{
    pthread_t tid = *(((wait_program_arg_type *)arg)->tid);
    sleep(PROGRAM_TIMEOUT_SEC);
    pthread_cancel(tid);

    return NULL;
}
```

<<mark.h>>

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
```

```
#include <string.h>
#include <pthread.h>
#include <signal.h>

#include "util.h"
#include "problem.h"
#include "score.h"
#include "student.h"
#include "evaluate.h"

// 프로그램 타임아웃 초
#define PROGRAM_TIMEOUT_SEC 5
// 워닝 감점 점수
#define PROGRAM_WARNING_DEMERIT 0.1
// 에러 발생했을 시 점수
#define PROGRAM_ERROR_SCORE 0

// 정답 셋
typedef struct _answer_set_type {
  int size;
  char *answer[100];
} answer_set_type;

// 프로그램 실행 타입
typedef enum _run_program_status {
  PROGRAM_SUCCESS,
  PROGRAM_NOT_EXIST,
  PROGRAM_COMPILE_ERROR,
  PROGRAM_RUNTIME_ERROR,
  PROGRAM_TIMEOUT
} run_program_status;

// 프로그램 출력
typedef struct _run_program_result_type {
  run_program_status status;
  char *output;
  char *error;
} run_program_result_type;

// 프로그램 실행 쓰레드에 전달되는  인자
typedef struct _run_program_arg_type {
  char problem_name[16];
} run_program_arg_type;
```

```
// 대기 쓰레드에 전달되는 인자
typedef struct _wait_program_arg_type {
  pthread_t *tid;
} wait_program_arg_type;

void mark(const char *student_dir, const char *true_dir, const char *error_dir,
          const problem_set_type *problem_set,
          const score_table_type *score_table, student_set_type *student_set,
          const char qnames[5][16], const int qname_count,
          bool print_score); // 문제 채점

char *init_answer_code(const char *true_dir, const problem_type *problem,
                       const bool use_lpthread); // 정답 코드 초기화

double mark_text(const char *student_dir, const char *student_id,
                 const problem_type *problem, const char *ans,
                 const double problem_score); // 텍스트 채점

run_program_result_type mark_code(const char *student_dir,
                                  const char *student_id,
                                  const problem_type *problem,
                                  const char *error_dir,
                                  const bool use_lpthread); // 코드 채점

void *run_program(void *arg);  // 프로그램 실행
void *wait_program(void *arg); // 프로그램 5초 대기후 run_program 캔슬
```

<<problem.c>>

```
#include "problem.h"

ENUM_PROBLEM_TYPE get_problem_type(const char *problem_name)
{
  char answer_code[32], answer_text[32];
  sprintf(answer_code, "%s/%s.c", problem_name, problem_name);
  sprintf(answer_text, "%s/%s.txt", problem_name, problem_name);
  if (file_exists(answer_code)) {
    return PROBLEM_CODE;
  }
  else if (file_exists(answer_text)) {
    return PROBLEM_TEXT;
  }
```

```
  else {
    fatal_error_print("answerset not valid");
  }
  return PROBLEM_TEXT;
}

problem_set_type *get_problem_set(const char *ans_dir_name)
{
  char cwd[PATH_MAX];
  getcwd(cwd, PATH_MAX);

  if (chdir(ans_dir_name) == -1) {
    fatal_error_print("cannot open TRUEDIR(%s)", ans_dir_name);
  }

  problem_set_type *problem_set =
      (problem_set_type *)malloc(sizeof(problem_set_type));

  problem_set->size = 0;

  // 디렉토리를 순회하며 디렉토리 존재 여부를 통해 문제 정보를 가져온다.
  for (int i = 1; i <= 100; ++i) {
    char pathname[16];
    sprintf(pathname, "%d", i);
    if (file_exists(pathname)) {
      problem_set->size += 1;
      problem_set->problem[problem_set->size - 1].type =
          get_problem_type(pathname);
      strcpy(problem_set->problem[problem_set->size - 1].name, pathname);
    }
    else {
      bool dir_exists = false;
      for (int j = 1; j <= 100; ++j) {
        sprintf(pathname, "%d-%d", i, j);
        if (file_exists(pathname)) {
          problem_set->size += 1;
          problem_set->problem[problem_set->size - 1].type =
              get_problem_type(pathname);
          strcpy(problem_set->problem[problem_set->size - 1].name, pathname);
          dir_exists = true;
        }
      }
      if (!dir_exists)
        break;
```

```
    }
  }

  chdir(cwd);

  return problem_set;
}
```

<<problem.h>>

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>

#include "util.h"

// 문제의 종류
typedef enum _ENUM_PROBLEM_TYPE {
  PROBLEM_TEXT,
  PROBLEM_CODE
} ENUM_PROBLEM_TYPE;

// 문제 타입
typedef struct _problem_type {
  ENUM_PROBLEM_TYPE type;
  char name[16];
} problem_type;

// 문제 셋 타입
typedef struct _problem_set_type {
  int size;
  problem_type problem[100];
} problem_set_type;

ENUM_PROBLEM_TYPE get_problem_type(
    const char *problem_name); // 디렉토리에서 문제의 타입을 알아낸다.
problem_set_type *
```

get_problem_set(const char *ans_dir_name); // 문제 세트를 가져온다.

<<score.c>>

```
#include "score.h"

score_table_type *read_score_table(const char *score_table_path)
{
  score_table_type *score_table =
      (score_table_type *)calloc(1, sizeof(score_table_type));
  score_table->size = 0;
  FILE *fp;
  if ((fp = fopen(score_table_path, "r")) == NULL) {
    fatal_error_print("fp is null");
  }

  while (!feof(fp)) {
    char filename[16];
    int i = score_table->size;
    fscanf(fp, "%[^,],%lf\n", filename, &(score_table->row[i].score));
    if (string_ends_with(filename, ".c")) {
      score_table->row[i].type = PROBLEM_CODE;
    }
    else if (string_ends_with(filename, ".txt")) {
      score_table->row[i].type = PROBLEM_TEXT;
    }
    else {
      fatal_error_print("\"score_table.csv\" not valid.");
    }
    strcpy(score_table->row[i].name, strtok(filename, "."));
    score_table->size += 1;
  }

  fclose(fp);

  return score_table;
};

void write_score_table(const char *score_table_path,
                       const score_table_type *score_table)
{
  FILE *fp = fopen(score_table_path, "a");
```

```
  if (fp == NULL) {
    fatal_error_print("fp is NULL");
  }
  for (int i = 0; i < score_table->size; ++i) {
    char filename[16];
    strcpy(filename, score_table->row[i].name);
    strcat(filename, score_table->row[i].type == PROBLEM_TEXT ? ".txt" : ".c");
    fprintf(fp, "%s,%.2lf\n", filename, score_table->row[i].score);
  }
  fclose(fp);
}

score_table_type *get_score_table(const char *true_dir,
                                  const problem_set_type *problem_set)
{
  score_table_type *score_table = NULL;
  char score_table_path[PATH_MAX];
  strcpy(score_table_path, true_dir);
  strcat(score_table_path, "/score_table.csv");
  // 파일이 존재하면 불러온다.
  if (file_exists(score_table_path)) {
    score_table = read_score_table(score_table_path);
    // 문제 수나 항목이 다르면 올바르지 않은 스코어 테이블이다.
    if (score_table->size != problem_set->size) {
      fatal_error_print("score_table.csv is not valid.");
    }
    for (int i = 0; i < score_table->size; ++i) {
      if (strcmp(score_table->row[i].name, problem_set->problem[i].name) != 0) {
        fatal_error_print("score_table.csv is not valid.");
      }
    }
  }
  else {
    score_table = (score_table_type *)malloc(sizeof(score_table_type));
    score_table->size = problem_set->size;
    printf("score_table.csv file does't exist in \"%s\"!\n", true_dir);
    while (true) {
      // csv파일이 없으면 만든다.
      printf(
          "1. input blank question and program question's score ex) 0.5 1\n");
      printf("2. input all question's score ex) Onput value of 1-1: 0.1\n");
      int choice;
      scanf("%d", &choice);
      if (choice == 1) {
```

```
        // 유형당 점수 부여
        double blank, program;
        printf("Input value of blank question: ");
        scanf("%lf", &blank);
        printf("Input value of program question: ");
        scanf("%lf", &program);
        for (int i = 0; i < score_table->size; ++i) {
          score_table->row[i].score =
              problem_set->problem[i].type == PROBLEM_TEXT ? blank : program;
          score_table->row[i].type = problem_set->problem[i].type;
          strcpy(score_table->row[i].name, problem_set->problem[i].name);
        }
        write_score_table(score_table_path, score_table);
        break;
      }
      else if (choice == 2) {
        // 개별 점수 부여
        for (int i = 0; i < score_table->size; ++i) {
          score_table->row[i].type = problem_set->problem[i].type;
          strcpy(score_table->row[i].name, problem_set->problem[i].name);
          printf("Input of %s.%s:", score_table->row[i].name,
                 score_table->row[i].type == PROBLEM_TEXT ? "txt" : "c");
          scanf("%lf", &score_table->row[i].score);
        }
        write_score_table(score_table_path, score_table);
        break;
      }
      else {
        printf("error: input is not valid.\n");
      }
    }
  }
  return score_table;
}
```

<<score.h>>

```
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```
#include "util.h"
#include "problem.h"

// 점수 타입의 열
typedef struct _score_row_type {
  char name[16];
  double score;
} score_row_type;

// 점수 타입
typedef struct _score_type {
  int size;
  score_row_type row[100];
} score_type;

// 스코어 테이블 타입의 열
typedef struct _score_table_row_type {
  ENUM_PROBLEM_TYPE type;
  char name[16];
  double score;
} score_table_row_type;

// 스코어 테이블 타입
typedef struct _score_table_type {
  int size;
  score_table_row_type row[100];
} score_table_type;

score_table_type *
read_score_table(const char *score_table_path); // 스코어 테이블을 읽는다.
void write_score_table(
    const char *score_table_path,
    const score_table_type *score_table); // 스코어 테이블을 기록한다.
score_table_type *get_score_table(
    const char *true_dir,
    const problem_set_type *problem_set); //스코어 테이블을 가져온다.
```

<<student.c>>

```
#include "student.h"
```

```
int student_compare(const void *s1, const void *s2)
{
  return strcmp(((const student_type *)s1)->id, ((const student_type *)s2)->id);
}

student_set_type *get_student_set(const char *student_dir_name,
                                  const problem_set_type *problem_set)
{
  student_set_type *student_set =
      (student_set_type *)malloc(sizeof(student_set_type));
  student_set->size = 0;

  struct dirent *dentry;
  DIR *dirp;
  char cwd[1024];
  getcwd(cwd, 1024);

  // 디렉토리를 순회하며 학생 정보를 가져온다.
  if ((dirp = opendir(student_dir_name)) == NULL ||
      chdir(student_dir_name) == -1) {
    fatal_error_print("opendir, chdir error for %s", student_dir_name);
  }
  while ((dentry = readdir(dirp)) != NULL) {
    if (dentry->d_ino == 0)
      continue;
    if (!is_directory(dentry->d_name))
      continue;
    if (strcmp(".", dentry->d_name) == 0 || strcmp("..", dentry->d_name) == 0)
      continue;
    student_type *const student = &(student_set->students[student_set->size]);
    strcpy(student->id, dentry->d_name);
    student->score.size = problem_set->size;
    for (int i = 0; i < problem_set->size; ++i) {
      strcpy(student->score.row[i].name, problem_set->problem[i].name);
      student->score.row[i].score = 0;
    }
    student_set->size += 1;
  }
  qsort(student_set->students, student_set->size, sizeof(student_type),
        student_compare);

  chdir(cwd);
  closedir(dirp);
```

```
  return student_set;
}

void write_student_score(const problem_set_type *problem_set,
                         const student_set_type *student_set)
{
  // answer.csv 출력
  char *fname = "score.csv";
  FILE *score_csv;
  if ((score_csv = fopen(fname, "w+")) == NULL) {
    fatal_error_print("file open error for score.csv");
  };
  // 헤더 출력
  fprintf(score_csv, ",");
  for (int i = 0; i < problem_set->size; ++i) {
    fprintf(score_csv, "%s%s,", problem_set->problem[i].name,
            (problem_set->problem[i].type == PROBLEM_CODE) ? ".c" : ".txt");
  }
  fprintf(score_csv, "sum\n");
  // 내용 출력
  for (int i = 0; i < student_set->size; ++i) {
    fprintf(score_csv, "%s,", student_set->students[i].id);
    double score_sum = 0;
    for (int j = 0; j < problem_set->size; ++j) {
      const double score = student_set->students[i].score.row[j].score;
      score_sum += score;
      fprintf(score_csv, "%.2lf,", score);
    }
    fprintf(score_csv, "%.2lf\n", score_sum);
  }
  fclose(score_csv);
}
```

<<student.h>>

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <fcntl.h>
```

```
#include <sys/stat.h>

#include "score.h"
#include "score.h"

#define STUDENT_MAX 100

typedef struct _student_type {
  char id[16];
  score_type score;
} student_type;

typedef struct _student_set_type {
  int size;
  student_type students[STUDENT_MAX];
} student_set_type;

int student_compare(const void *s1,
                    const void *s2); //두 학생의 학번을 비교한다.
student_set_type *
get_student_set(const char *student_dir_name,
                const problem_set_type *problem_set); // 학생 정보를 가져온다.
void write_student_score(
    const problem_set_type *problem_set,
    const student_set_type *student_set); // score.csv를 기록한다.
```

<<util.c>>

```
#include "util.h"

void ssu_runtime(struct timeval *begin_t, struct timeval *end_t)
{
  end_t->tv_sec -= begin_t->tv_sec;
  if (end_t->tv_usec < begin_t->tv_usec) {
    end_t->tv_sec--;
    end_t->tv_usec += SECOND_TO_MICRO;
  }
  end_t->tv_usec -= begin_t->tv_usec;
  printf("Runtime: %ld:%06ld(sec:usec)\n", end_t->tv_sec, end_t->tv_usec);
}

char *get_file_data(const char *pathname)
```

```
{
  // 파일 크기만큼 메모리를 동적 할당해 파일을 전부 가져온다.
  if (!file_exists(pathname)) {
    return NULL;
  }

  int fd;
  if ((fd = open(pathname, O_RDONLY)) < 0) {
    fatal_error_print("file open error for %s", pathname);
  }

  off_t fsize;
  if ((fsize = lseek(fd, 0, SEEK_END)) < 0) {
    fatal_error_print("lseek error");
  }

  char *buf = (char *)malloc(fsize + 1);

  if (lseek(fd, 0, SEEK_SET) < 0) {
    fatal_error_print("lseek error");
  }

  ssize_t len;

  if ((len = read(fd, buf, fsize)) < 0) {
    fatal_error_print("file read error for %s", pathname);
  }

  close(fd);

  buf[len] = 0;

  if (strlen(buf) == 0) {
    return NULL;
  }

  return buf;
}

bool file_exists(const char *pathname) { return access(pathname, F_OK) == 0; }

bool is_directory(const char *pathname)
{
  struct stat statbuf;
```

```
  if (stat(pathname, &statbuf) != 0)
    return false;
  return S_ISDIR(statbuf.st_mode);
}

bool string_ends_with(const char *str, const char *suffix)
{
  const int len_diff = (int)strlen(str) - (int)strlen(suffix);
  return (len_diff >= 0) && (strcmp(str + len_diff, suffix) == 0);
}

int string_count(const char *str, const char *substr)
{
  int count = 0;
  const char *tmp = str;
  while ((tmp = strstr(tmp, substr)) != NULL) {
    count++;
    tmp++;
  }
  return count;
}

char *string_trim(const char *str)
{
  // 문자열을 복제하여 양 쪽의 공백을 제거하여 반환한다.
  char *tmp = strdup(str);
  char *cur = tmp;
  while (*cur == ' ')
    cur++;
  if (*tmp == 0) {
    free(tmp);
    tmp = NULL;
    return NULL;
  }
  char *end = cur + strlen(cur) - 1;
  while (end > str && *end == ' ')
    end--;
  end[1] = '\0';
  char *ret = strdup(cur);
  free(tmp);
  tmp = NULL;
  return ret;
}
```

```
char *string_normalize_program_output(const char *str)
{
  char *ret = (char *)malloc(strlen(str) + 1);
  size_t i, j;
  for (i = 0, j = 0; i < strlen(str); i++, j++) {
    if (str[i] != ' ')
      ret[j] = str[i];
    else
      j--;
  }
  ret[j] = '\0';
  return ret;
}

bool string_compare_program_output(const char *str1, const char *str2)
{
  if (str1 == NULL || str2 == NULL)
    return 0;
  char *s1 = string_normalize_program_output(str1),
       *s2 = string_normalize_program_output(str2);
  bool isEqual = (strcasecmp(s1, s2) == 0);
  free(s1);
  free(s2);
  return isEqual;
}

void fatal_error_print(const char *format, ...)
{
  fprintf(stderr, "error: ");
  va_list args;
  va_start(args, format);
  vfprintf(stderr, format, args);
  va_end(args);
  fprintf(stderr, "\n");
  exit(1);
}
```

<<util.h>>

```
#pragma once

#include <stdio.h>
```

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdarg.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <dirent.h>
#include <limits.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>

#define SECOND_TO_MICRO 1000000

void ssu_runtime(struct timeval *begin_t,
                 struct timeval *end_t);   // 실행 시간 체크
bool file_exists(const char *pathname);    // 파일이 존재하는지 확인
char *get_file_data(const char *pathname); // 파일을 읽어서 가져온다.
bool is_directory(const char *pathname);   // 디렉토리인지 확인
bool string_ends_with(
    const char *str,
    const char *suffix); // 문자열이 다른 문자열로 끝나는지 확인
int string_count(const char *str,
                 const char *substr); // 문자열이 몇 번 등장하는지 체크
char *string_trim(const char *str); // 양쪽 공백 제거
char *
string_normalize_program_output(const char *str); // 프로그램 출력 노멀라이즈
bool string_compare_program_output(const char *str1,
                                   const char *str2); // 두 프로그램 출력 비교
void fatal_error_print(const char *format,
                       ...); // 에러가 발생했을 때 strerr출력 + 비정상 종료
```