컴퓨터학부 20160548 이승현

(비밀번호 : 333333)

1. **과제 개요**

ssu\_score는 학생들이 제출한 답안 파일을 정답 파일과 비교하여 채점하는 프로그램이다. 문제는 두 종류, 빈칸 채우기 문제와 프로그램 문제로 이루어져 있다. 점수의 배점은 프로그램 실행 시 사용자가 원하는 대로 설정할 수 있다. 프로그램을 이용해 채점한 후에는 채점 결과 테이블 파일, 에러 메시지 출력 파일 등을 통해 채점 결과를 확인할 수 있다. 이 과제의 목표는 주어진 ssu\_score 프로그램의 소스 코드를 분석하여 프로그램이 어떻게 작동하는지 파악하고, 코드를 수정하여 과제의 명세서에 맞게 기능을 추가, 수정하는 것이다.

1. **분석**

**<main.c>**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/time.h>

#include "ssu\_score.h"

#define SECOND\_TO\_MICRO 1000000

void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t);

int main(int argc, char \*argv[])

{

struct timeval begin\_t, end\_t;

gettimeofday(&begin\_t, NULL); // 시작 시간 기록

ssu\_score(argc, argv); // 채점

gettimeofday(&end\_t, NULL); // 종료 시간 기록

ssu\_runtime(&begin\_t, &end\_t); // 프로그램 실행 시간 계산, 출력

exit(0);

}

void ssu\_runtime(struct timeval \*begin\_t, struct timeval \*end\_t)

{

// 시작시간과 종료시간의 차이 계산

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); // 프로그램 실행에 걸린 시간 출력

}

**<ssu\_score.h>**

#ifndef MAIN\_H\_

#define MAIN\_H\_

#ifndef true

#define true 1

#endif

#ifndef false

#define false 0

#endif

#ifndef STDOUT

#define STDOUT 1

#endif

#ifndef STDERR

#define STDERR 2

#endif

#ifndef TEXTFILE

#define TEXTFILE 3

#endif

#ifndef CFILE

#define CFILE 4

#endif

#ifndef OVER

#define OVER 5 // 학생들이 제출한 프로그램의 실행이 여기에 정의된 시간(초단위) 이상 걸리면 0점처리

#endif

#ifndef WARNING

#define WARNING -0.1 // 컴파일 WARNING 발생한 문제 감점 점수

#endif

#ifndef ERROR

#define ERROR 0 // 컴파일 에러 발생시 점수

#endif

#define FILELEN 64

#define BUFLEN 1024

#define SNUM 100

#define QNUM 100

#define ARGNUM 5 // 가변인자를 받는 옵션의 경우, 최대로 받을 수 있는 가변인자의 개수는 5개로 제한

struct ssu\_scoreTable{ // 문제별 정보를 담아놓는 구조체

char qname[FILELEN]; // 문제 번호

double score; // 배점

};

void ssu\_score(int argc, char \*argv[]);

int check\_option(int argc, char \*argv[]); // 프로그램 실행 시 전달된 옵션을 체크하는 함수

void print\_usage(); // 프로그램 사용법 출력 (-h 옵션)

void score\_students();// 채점하는 함수

double score\_student(int fd, char \*id);// 한 학생에 대하여 채점을 하는 함수, 리턴값은 해당 학생의 총점

void write\_first\_row(int fd);// score.csv의 첫번째 열을 write하는 함수

char \*get\_answer(int fd, char \*result);// 답안 파일에서 내용을 읽어와 result에 저장하는 함수

int score\_blank(char \*id, char \*filename);// 빈칸 문제를 채점하는 함수, 리턴값은 정답 여부 또는 감점된 점수

double score\_program(char \*id, char \*filename);// 프로그램 문제를 채점하는 함수, 리턴값은 정답 여부 또는 감점된 점수

double compile\_program(char \*id, char \*filename);// 프로그램 문제를 컴파일하는 함수

int execute\_program(char \*id, char \*filname);// 프로그램 문제를 실행하는 함수

pid\_t inBackground(char \*name);// 인자로 전달된 프로세스가 실행중인지 확인하는 함수

double check\_error\_warning(char \*filename);// 컴파일 에러 내용이 저장된 파일을 이용해 error인지 warning인지 확인해서 결과 점수를 리턴하는 함수

int compare\_resultfile(char \*file1, char \*file2);// 프로그램을 실행한 결과로 나온 파일을 비교하는 함수

void do\_cOption(char (\*ids)[FILELEN]);// 선택한 학번의 점수를 출력하는 옵션 -> 필요 없음

int is\_exist(char (\*src)[FILELEN], char \*target);// src 문장열 배열 안에 target 문자열이 들어있는지 확인하는 함수

int is\_thread(char \*qname);// -t(lpthread 사용) 옵션으로 지정된 문제인지 확인하는 함수

void redirection(char \*command, int newfd, int oldfd);// new 파일디스크립터를 old에 복사한 뒤 command를 실행한다. 실행한 후에는 다시 원래 old에 있던 값으로 복구

int get\_file\_type(char \*filename);// 파일의 확장자를 확인하는 함수

void rmdirs(const char \*path);// 디렉터리 삭제하는 함수

void to\_lower\_case(char \*c);// 대문자 알파벳을 소문자로 변환하는 함수

void set\_scoreTable(char \*ansDir);// 각 문제별 점수를 저장해 놓는 score\_table 구조체 배열을 setting하는 함수

void read\_scoreTable(char \*path);// 파일에서 문제의 정보를 읽어와 score\_table 구조체 배열에 저장하는 함수

void make\_scoreTable(char \*ansDir);// score\_table 구조체 배열에 문제 번호와 점수를 저장해 점수 테이블을 만드는 함수

void write\_scoreTable(char \*filename);// score\_table 구조체 배열의 내용을 csv 형식의 파일에 출력하는 함수

void set\_idTable(char \*stuDir);// 학생들의 학번을 저장해 놓는 id\_table 배열을 setting 하는 함수

int get\_create\_type();// 사용자가 문제 번호와 점수를 어떤 식으로 저장할지 선택하도록 하는 함수

void sort\_idTable(int size);// 학생들의 학번이 저장되어 있는 id\_table을 정렬하는 함수

void sort\_scoreTable(int size);// 각 문제들의 점수가 저장되어있는 score\_table 구조체 배열을 정렬하는 함수

void get\_qname\_number(char \*qname, int \*num1, int \*num2);// 문자열로 되어있는 문제번호를 인자로 받아 int형으로 변환하는 함수, num1은 상위 문제번호, num2는 하위 문제번호를 뜻함

void do\_mOption();// m옵션 수행하는 함수

void do\_iOption(char (\*ids)[FILELEN]);// i옵션 수행하는 함수

#endif

**<ssu\_score.c>**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <signal.h>

#include <string.h>

#include <sys/types.h>

#include <dirent.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h>

#include "ssu\_score.h"

#include "blank.h"

extern struct ssu\_scoreTable score\_table[QNUM];

extern char id\_table[SNUM][10];

struct ssu\_scoreTable score\_table[QNUM];

char id\_table[SNUM][10];

char stuDir[BUFLEN]; // 학생들이 제출한 답안들이 들어있는 디렉토리

char ansDir[BUFLEN]; // 정답 파일들이 들어있는 디렉토리

char errorDir[BUFLEN];

char threadFiles[ARGNUM][FILELEN];

// char cIDs[ARGNUM][FILELEN];

char iIDs[ARGNUM][FILELEN]; // -i 옵션에 사용할 학번

// option flags

int eOption = false;

int tOption = false;

// p옵션, c옵션 필요 없으므로 수정 필요 //////////////////////////////////////////////////

//int pOption = false;

//int cOption = false;

/////////////////////////////////////////////////////////////////////////////////////////

int mOption = false;

int iOption = false;

void ssu\_score(int argc, char \*argv[])

{

char saved\_path[BUFLEN];

int i; // for문에 사용할 인덱스 변수

for(i = 0; i < argc; i++){

if(!strcmp(argv[i], "-h")){ // -h 옵션이 적용되어 실행되면

print\_usage(); // 사용법 출력

return; // ssu\_score 종료

}

}

memset(saved\_path, 0, BUFLEN); // saved\_path을 0으로 초기화

if(argc >= 3 && strcmp(argv[1], "-c") != 0){ // \*\*\*\*\*\*\*\*\*\* -c옵션이 무엇인지?

strcpy(stuDir, argv[1]); // <STUDENTDIR> 저장

strcpy(ansDir, argv[2]); // <TRUESETDIR> 저장

}

if(!check\_option(argc, argv))

exit(1);

// p옵션, c옵션 없으므로 수정 필요 //////////////////////////////////////////////////

if(!eOption && !tOption && !mOption && iOption && !strcmp(argv[1], "-i")){ // -c 옵션만 적용됐으면 --------------------> i옵션만 적용된 경우에는 채점이 안됨 수정 필요함

//do\_cOption(cIDs);

do\_iOption(iIDs);

return; // ssu\_score 종료

}

getcwd(saved\_path, BUFLEN); // 프로세스의 현재 위치 절대경로 저장

if(chdir(stuDir) < 0){ // cd <STUDENTDIR>, cd 실패했다면

fprintf(stderr, "%s doesn't exist\n", stuDir); //에레메세지 출력

return; // ssu\_score 종료

}

getcwd(stuDir, BUFLEN); // stuDir에 <STUDENTDIR>의 절대경로 저장

chdir(saved\_path); // 다시 프로세스가 실행된 디렉토리로 이동

if(chdir(ansDir) < 0){ // cd <TRUESETDIR>, cd 실패했다면

fprintf(stderr, "%s doesn't exist\n", ansDir); //에러메세지 출력

return; // ssu\_score 종료

}

getcwd(ansDir, BUFLEN); // ansDir에 <TRUESETDIR>의 절대경로 저장

chdir(saved\_path); // 다시 프로세스가 실행된 디렉토리로 이동

set\_scoreTable(ansDir); // 문제별 점수들이 저장될 score\_table 구조체 배열을 setting

set\_idTable(stuDir); // 학생들의 학번이 저장될 id\_table 배열을 setting

//

if(mOption)

do\_mOption();

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

score\_students();

// c옵션 없으므로 수정 필요 //////////////////////////////////////////////////

// if(cOption)

// do\_cOption(cIDs);

//////////////////////////////////////////////////////////////////////////////

if(iOption)

do\_iOption(iIDs);

return;

}

int check\_option(int argc, char \*argv[]) // 프로그램 실행 시 전달된 옵션을 체크하는 함수

{

int i, j; // 반복문에서 사용하는 인덱스

int c; // 옵션으로 전달된 알파벳

// 옵션 p, c가 필요 없으므로 수정 필요 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////

while((c = getopt(argc, argv, "e:thmi")) != -1)

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

{

switch(c){

case 'e': // 옵션 e

eOption = true;

strcpy(errorDir, optarg); // 옵션에 전달된 인자(에러 메세지가 출력될 디렉토리)를 errorDir에 복사해 놓는다

if(access(errorDir, F\_OK) < 0) // 디렉터리에 접근이 불가능하면

mkdir(errorDir, 0755);// 디렉터리 생성

else{

rmdirs(errorDir); // 기존 디렉터리 제거

mkdir(errorDir, 0755); // 새 디렉터리 생성

}

break;

case 't': // 옵션 t

tOption = true;

i = optind; // 프로그램 전달인자 인덱스

j = 0; // 옵션 가변인자 인덱스

while(i < argc && argv[i][0] != '-'){ // t 옵션에 전달된 가변인자들 확인을 위한 반복문

if(j >= ARGNUM) // 가변인자를 받는 옵션이므로 가변인자의 개수가 최대 개수를 넘지 않았는지 확인한다

printf("Maximum Number of Argument Exceeded. :: %s\n", argv[i]);

else // 옵션에 전달된 인자를 threadFiles에 복사해 놓는다

strcpy(threadFiles[j], argv[i]);

i++;

j++;

}

break;

case 'm':

mOption = true;

case 'i':

iOption = true;

i = optind; // 프로그램 전달인자 인덱스

j = 0; // 옵션에 전달된 가변인자 인덱스

while(i < argc && argv[i][0] != '-'){ // i 옵션에 전달된 가변인자들 확인을 위한 반복문

if(j >= ARGNUM) // 가변인자를 받는 옵션이므로 가변인자의 개수가 최대 개수를 넘지 않았는지 확인

printf("Maximum Number of Argument Exceeded. :: %s\n", argv[i]);

else

strcpy(iIDs[j], argv[i]); // 옵션에 전달된 인자를 iIDs에 복사해 놓는다

i++;

j++;

}

break;

// // 옵션 p - 항상 수행되어야 하므로 수정 필요 //////////////////////////////////////////////////////////////////////////

// case 'p':

// pOption = true;

// break;

// ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// 옵션 c - 필요없는 옵션 /////////////////////////////////////////////////////////////////////////////////////////////

// case 'c':

// cOption = true;

// i = optind; // 프로그램 전달인자 인덱스

// j = 0; // 옵션에 전달된 가변인자 인덱스

//

// while(i < argc && argv[i][0] != '-'){ // c 옵션에 전달된 가변인자들 확인을 위한 반복문

//

// if(j >= ARGNUM) // 가변인자를 받는 옵션이므로 가변인자의 개수가 최대 개수를 넘지 않았는지 확인

// printf("Maximum Number of Argument Exceeded. :: %s\n", argv[i]);

// else

// strcpy(cIDs[j], argv[i]); // 옵션에 전달된 인자를 cIDs에 복사해 놓는다

// i++;

// j++;

// }

// break;

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

case '?': // 파라미터가 빠진 채로 옵션이 전달된 경우

printf("Unkown option %c\n", optopt);

return false;

}

}

return true;

}

void do\_cOption(char (\*ids)[FILELEN]) // 선택한 학번의 점수를 출력하는 옵션 -> 필요 없음

{

FILE \*fp;

char tmp[BUFLEN];

int i = 0;

char \*p, \*saved;

if((fp = fopen("score.csv", "r")) == NULL){

fprintf(stderr, "file open error for score.csv\n");

return;

}

fscanf(fp, "%s\n", tmp);

while(fscanf(fp, "%s\n", tmp) != EOF)

{

p = strtok(tmp, ",");

if(!is\_exist(ids, tmp))

continue;

printf("%s's score : ", tmp);

while((p = strtok(NULL, ",")) != NULL)

saved = p;

printf("%s\n", saved);

}

fclose(fp);

}

void do\_iOption(char (\*ids)[FILELEN]) // i옵션 수행하는 함수

{

FILE \*fp;

char tmp[BUFLEN];

char numbers[BUFLEN];

int i = 0;

char \*p, \*saved, \*np;

int isFirstWrongAnswer = true; // 두번째 오답부터는 앞에 콤마를 찍기 위해 사용하는 플래그

if((fp = fopen("score.csv", "r")) == NULL){ // 점수파일 오픈

fprintf(stderr, "file open error for score.csv\n");

return;

}

fscanf(fp, "%s\n", numbers); // 파일에서 첫번째 줄(문제 번호들) 읽어들임

while(fscanf(fp, "%s\n", tmp) != EOF) // 한 학생씩 채점 결과 읽어들임

{

isFirstWrongAnswer = true;

np = numbers;

p = strtok(tmp, ",");

if(!is\_exist(ids, tmp)) // i 옵션을 지정한 학생중에 현재 읽어들인 학생이 있다면

continue;

// 틀린 문제들 출력

printf("%s's wrong answer : \n", tmp);

while((p = strtok(NULL, ",")) != NULL) {

np = strchr(np, ',') + 1;

if (!strcmp(p, "0")) { // 해당 문제를 틀렸다면

// 문제번호 출력

if(!isFirstWrongAnswer) printf(", ");

else isFirstWrongAnswer = false;

while(\*np != ',') {

printf("%c", \*np);

++np;

}

}

}

printf("\n");

}

fclose(fp);

}

void do\_mOption()// m옵션 수행하는 함수

{

int i;

double newScore;

char filename[FILELEN];

char qname[FILELEN]; // 문제 번호를 저장할 배열

char inputqname[FILELEN];

while(true) {

printf("Input question's number to modify >> ");

scanf("%s", inputqname); // 수정할 문제 번호 입력받음

if(!strcmp(inputqname, "no")) break; // 입력된 문제 번호가 no라면 수정 종료

i = 0;

while(score\_table[i].score != 0) {

memset(qname, 0, sizeof(qname));

memcpy(qname, score\_table[i].qname, strlen(score\_table[i].qname) - strlen(strrchr(score\_table[i].qname, '.'))); // qname에 확장자 명을 뺀 파일 이름(문제 번호)을 넣음

if(strcmp(qname, inputqname)) { // 문제 번호가 서로 일치하지 않는다면

++i;

continue;

}

// 문제 번호가 서로 일치하면

printf("Current score : %.2f\n", score\_table[i].score);

printf("New score : ");

scanf("%lf", &newScore); // 변경할 배점 입력받음

score\_table[i].score = newScore; // score\_table 구조체 배열에 변경된 점수 기록

break;

}

}

sprintf(filename, "%s", "score\_table.csv"); // 점수 테이블 파일이 생성될 경로를 생성해 filename에 저장

write\_scoreTable(filename); // 변경된 score\_table 구조체 배열의 내용을 score.csv에 출력

}

int is\_exist(char (\*src)[FILELEN], char \*target) // src 문장열 배열 안에 target 문자열이 들어있는지 확인하는 함수

{

int i = 0;

while(1)

{

if(i >= ARGNUM)

return false;

else if(!strcmp(src[i], ""))

return false;

else if(!strcmp(src[i++], target))

return true;

}

return false;

}

void set\_scoreTable(char \*ansDir) // 각 문제별 점수를 저장해 놓는 score\_table 구조체 배열을 setting하는 함수

{

char filename[FILELEN];

// 점수 테이블 파일은 "./score\_table.csv" 이름으로, 현재 실행 위치에 존재해야 하기 때문에 수정 필요 ////////////////////////////////////////////

// sprintf(filename, "%s/%s", ansDir, "score\_table.csv"); // 점수 테이블 파일이 생성될 경로를 생성해 filename에 저장

sprintf(filename, "%s", "score\_table.csv"); // 점수 테이블 파일이 생성될 경로를 생성해 filename에 저장

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

if(access(filename, F\_OK) == 0) // 이미 점수 테이블 파일이 존재한다면

read\_scoreTable(filename); // 기존의 파일에서 문제 번호와 점수들을 불러온다

else{ // 점수 테이블 파일이 존재하지 않는다면

make\_scoreTable(ansDir);

write\_scoreTable(filename);

}

}

void read\_scoreTable(char \*path) // 파일에서 문제의 정보를 읽어와 score\_table 구조체 배열에 저장하는 함수

{

FILE \*fp;

char qname[FILELEN]; // 문제 번호를 임시 저장할 배열

char score[BUFLEN]; // 점수를 임시 저장할 배열

int idx = 0; // score\_table 구조체 배열의 인덱스

if((fp = fopen(path, "r")) == NULL){ // 점수 테이블 오픈

fprintf(stderr, "file open error for %s\n", path);

return ;

}

while(fscanf(fp, "%[^,],%s\n", qname, score) != EOF){ // 파일에서 문제 번호와 점수 읽어 들임

strcpy(score\_table[idx].qname, qname); // 문제 번호를 score\_table 구조체 배열에 저장

score\_table[idx++].score = atof(score); // 해당 문제의 점수를 score\_table 구조체 배열에 저장

}

fclose(fp);

}

void make\_scoreTable(char \*ansDir) // score\_table 구조체 배열에 문제 번호와 점수를 저장해 점수 테이블을 만드는 함수

{

int type, num; // type - 파일의 확장자 type을 저장해 놓을 변수

double score, bscore, pscore;

struct dirent \*dirp, \*c\_dirp;

DIR \*dp, \*c\_dp;

char tmp[BUFLEN];

int idx = 0; // 문제 총 개수 저장할 변수

int i;

num = get\_create\_type(); // 사용자에게 점수를 어떤식으로 입력받을 것인지 선택하도록 한다

if(num == 1) // 점수 일괄 입력 선택 시

{

printf("Input value of blank question : ");

scanf("%lf", &bscore); // 빈칸 문제의 점수 입력받음

printf("Input value of program question : ");

scanf("%lf", &pscore); // 프로그램 문제의 점수 입력받음

}

if((dp = opendir(ansDir)) == NULL){ // 디렉터리 open

fprintf(stderr, "open dir error for %s\n", ansDir);

return;

}

while((dirp = readdir(dp)) != NULL)

{

if(!strcmp(dirp->d\_name, ".") || !strcmp(dirp->d\_name, "..")) // 디렉터리 이름이 . 과 .. 이라면 pass

continue;

sprintf(tmp, "%s/%s", ansDir, dirp->d\_name); // 다음에 확인할 디렉터리 경로 tmp에 저장

// if((c\_dp = opendir(tmp)) == NULL){ // 확인 할 디렉터리 open

// fprintf(stderr, "open dir error for %s\n", tmp);

// return;

// }

//

// while((c\_dirp = readdir(c\_dp)) != NULL) // 문제별 디렉터리 확인

// {

// if(!strcmp(c\_dirp->d\_name, ".") || !strcmp(c\_dirp->d\_name, ".."))

// continue;

//

// if((type = get\_file\_type(c\_dirp->d\_name)) < 0) // 파일의 확장자가 .txt or .c 가 아니라면 pass

// continue;

//

// strcpy(score\_table[idx++].qname, c\_dirp->d\_name);

// }

//

// closedir(c\_dp);

// 답안 디렉터리의 바로 아래에 정답 파일들이 들어있도록 바뀌었으므로 이 부분을 수정

if((type = get\_file\_type(dirp->d\_name)) < 0) // 파일의 확장자가 .txt or .c 가 아니라면 pass

continue;

strcpy(score\_table[idx++].qname, dirp->d\_name);

}

closedir(dp);

sort\_scoreTable(idx);

for(i = 0; i < idx; i++) // 모든 문제에 대하여

{

type = get\_file\_type(score\_table[i].qname); // 파일의 확장자가 무엇인지 확인해 type에 저장

if(num == 1) // 점수 일괄 입력 선택 시

{

if(type == TEXTFILE) // .txt 파일이라면 (빈칸문제)

score = bscore;

else if(type == CFILE) // .c 파일이라면 (프로그램 문제)

score = pscore;

}

else if(num == 2) // 점수 각각 입력 선택 시

{

printf("Input of %s: ", score\_table[i].qname); // 점수 입력받을 문제 번호 출력

scanf("%lf", &score); // 해당 문제의 점수 입력받음

}

score\_table[i].score = score; // score\_table 구조체 배열에 문제 점수 저장

}

}

void write\_scoreTable(char \*filename) // score\_table 구조체 배열의 내용을 csv 형식의 파일에 출력하는 함수

{

int fd;

char tmp[BUFLEN];

int i;

int num = sizeof(score\_table) / sizeof(score\_table[0]); // 문제 총 개수

if((fd = creat(filename, 0666)) < 0){ // 새로운 csv파일 생성

fprintf(stderr, "creat error for %s\n", filename);

return;

}

for(i = 0; i < num; i++) // 모든 문제에 대하여

{

if(score\_table[i].score == 0) // score\_table에 저장된 점수가 0점이라면 끝 //////////////////////////////////////////////////////////// score\_table 끝에 0 어디서 넣었는지? -> 전역 변수 0 초기화됨

break;

sprintf(tmp, "%s,%.2f\n", score\_table[i].qname, score\_table[i].score); // 문제 번호와 점수를 csv형식의 문자열로 만들어 tmp에 저장

write(fd, tmp, strlen(tmp)); // tmp의 내용을 파일에 write

}

close(fd);

}

void set\_idTable(char \*stuDir) // 학생들의 학번을 저장해 놓는 id\_table 배열을 setting 하는 함수

{

struct stat statbuf;

struct dirent \*dirp;

DIR \*dp;

char tmp[BUFLEN];

int num = 0;

if((dp = opendir(stuDir)) == NULL){

fprintf(stderr, "opendir error for %s\n", stuDir);

exit(1);

}

while((dirp = readdir(dp)) != NULL){

if(!strcmp(dirp->d\_name, ".") || !strcmp(dirp->d\_name, "..")) // 디렉터리 이름이 . 과 .. 이라면 pass

continue;

sprintf(tmp, "%s/%s", stuDir, dirp->d\_name); // 확인할 디렉터리의 경로를 tmp에 저장

stat(tmp, &statbuf);

if(S\_ISDIR(statbuf.st\_mode)) // 디렉터리 파일이라면

strcpy(id\_table[num++], dirp->d\_name); // id\_table에 디렉터리 이름(학번) 복사

else

continue;

}

sort\_idTable(num); // id\_table 정렬

}

void sort\_idTable(int size) // 학생들의 학번이 저장되어 있는 id\_table을 정렬하는 함수

{

int i, j;

char tmp[10]; // swap에 사용하는 임시 배열

for(i = 0; i < size - 1; i++){

for(j = 0; j < size - 1 -i; j++){

if(strcmp(id\_table[j], id\_table[j+1]) > 0){

strcpy(tmp, id\_table[j]);

strcpy(id\_table[j], id\_table[j+1]);

strcpy(id\_table[j+1], tmp);

}

}

}

}

void sort\_scoreTable(int size) // 각 문제들의 점수가 저장되어있는 score\_table 구조체 배열을 정렬하는 함수

{

int i, j;

struct ssu\_scoreTable tmp;

int num1\_1, num1\_2; // 비교할 첫번째 문제의 상위 문제 번호와 하위 문제 번호

int num2\_1, num2\_2; // 비교할 두번째 문제의 상위 문제 번호와 하위 문제 번호

for(i = 0; i < size - 1; i++){

for(j = 0; j < size - 1 - i; j++){

get\_qname\_number(score\_table[j].qname, &num1\_1, &num1\_2); // 비교할 첫번째 문제 번호를 int형으로 변환

get\_qname\_number(score\_table[j+1].qname, &num2\_1, &num2\_2); // 비교할 두번째 문제 번호를 int형으로 변환

if((num1\_1 > num2\_1) || ((num1\_1 == num2\_1) && (num1\_2 > num2\_2))){ // 첫번째 문제 번호가 두번째 문제 번호보다 큰 경우 SWAP

memcpy(&tmp, &score\_table[j], sizeof(score\_table[0]));

memcpy(&score\_table[j], &score\_table[j+1], sizeof(score\_table[0]));

memcpy(&score\_table[j+1], &tmp, sizeof(score\_table[0]));

}

}

}

}

void get\_qname\_number(char \*qname, int \*num1, int \*num2) // 문자열로 되어있는 문제번호를 인자로 받아 int형으로 변환하는 함수, num1은 상위 문제번호, num2는 하위 문제번호를 뜻함

{

char \*p;

char dup[FILELEN];

strncpy(dup, qname, strlen(qname)); // 인자로 받은 문제 번호를 dup에 복사한다

\*num1 = atoi(strtok(dup, "-.")); // '-', '.'을 기준으로 문자열을 분할, 분할한 문자열을 int형으로 변환

p = strtok(NULL, "-."); // '-', '.'을 기준으로 위의 문자열을 이어서 분할

if(p == NULL) // 여기서 p가 NULL이면 하위 문제가 없는 문제

\*num2 = 0;

else // 하위 문제가 있는 문제일 경우

\*num2 = atoi(p); // 하위 문제 번호를 int형으로 변환해서 num2에 저장

}

int get\_create\_type() // 사용자가 문제 번호와 점수를 어떤 식으로 저장할지 선택하도록 하는 함수

{

int num;

while(1)

{

printf("score\_table.csv file doesn't exist!\n");

printf("1. input blank question and program question's score. ex) 0.5 1\n"); // 빈칸 문제와 프로그램 문제의 점수를 각각 일괄적으로 입력하려면 1 선택

printf("2. input all question's score. ex) Input value of 1-1: 0.1\n"); // 각각의 문제에 대해서 따로 따로 점수를 입력하려면 2 선택

printf("select type >> ");

scanf("%d", &num); // 사용자의 선택을 읽어들인다

if(num != 1 && num != 2) // 사용자의 입력이 1도 아니고 2도 아닐 떄

printf("not correct number!\n"); //잘못된 입력이므로 다시 입력받음

else

break;

}

return num; // 사용자가 선택한 숫자 리턴

}

void score\_students() // 채점하는 함수

{

double score = 0; // 모든 학생의 총점을 저장할 변수

int num;

int fd;

char tmp[BUFLEN];

int size = sizeof(id\_table) / sizeof(id\_table[0]); // 전체 학생 수

if((fd = creat("score.csv", 0666)) < 0){ // 채점 결과를 저장할 score.csv 파일 생성

fprintf(stderr, "creat error for score.csv");

return;

}

write\_first\_row(fd); // score.csv의 첫번째 열에 문제 번호 등 출력

for(num = 0; num < size; num++) // 전체 학생 수 만큼 반복

{

if(!strcmp(id\_table[num], ""))

break;

// 학번을 score.csv에 출력

sprintf(tmp, "%s,", id\_table[num]);

write(fd, tmp, strlen(tmp));

score += score\_student(fd, id\_table[num]); // 해당 학생에 대하여 채점을 한 뒤 학생의 총점을 전체 총점에 더함

}

// if(pOption) // p옵션이 설정되어 있다면 -> 항상 수행되도록 해야함

// printf("Total average : %.2f\n", score / num); // 전체 평균 점수 출력

//

printf("Total average : %.2f\n", score / num); // 전체 평균 점수 출력

close(fd);

}

double score\_student(int fd, char \*id) // 한 학생에 대하여 채점을 하는 함수, 리턴값은 해당 학생의 총점

{

int type; // 해당 문제가 빈칸 문제인지, 프로그램 문제인지 저장할 변수

double result; // 해당 문제에 대한 정답 여부 또는 감점된 점수를 담을 변수

double score = 0; // 해당 학생의 총점을 저장할 변수

int i;

char tmp[BUFLEN]; // 파일에 write하기 전에 임시로 담아 놓는 배열

int size = sizeof(score\_table) / sizeof(score\_table[0]); // 전체 문항 수

for(i = 0; i < size ; i++) // 전체 문항 수만큼 반복

{

if(score\_table[i].score == 0) // 해당 문제의 배점이 0점이라면 채점 중단

break;

sprintf(tmp, "%s/%s/%s", stuDir, id, score\_table[i].qname); // 해당 학생의 해당 문제 디렉터리로 이동

if(access(tmp, F\_OK) < 0) // 해당 문제 디렉터리에 접근이 불가하다면

result = false;

else

{

if((type = get\_file\_type(score\_table[i].qname)) < 0) // 파일의 확장자 명으로 빈칸 문제인지, 프로그램 문제인지 확인

continue;

if(type == TEXTFILE) // .txt 파일(빈칸 문제)이라면

result = score\_blank(id, score\_table[i].qname); // 빈칸문제 채점

else if(type == CFILE) // .c 파일(프로그램 문제)이라면

result = score\_program(id, score\_table[i].qname); // 프로그램 문제 채점

}

if(result == false) // 해당 학생이 문제를 틀렸을 때

write(fd, "0,", 2); // 0점 부여, score.csv에 해당 문제 0점이라고 출력

else{

if(result == true){ // 해당 학생이 문제를 맞혔을 때

score += score\_table[i].score; // 해당 문제의 배점을 학생의 점수에 더함

sprintf(tmp, "%.2f,", score\_table[i].score); // tmp에 해당 문제에 대하여 학생이 받은 점수를 기록

}

else if(result < 0){ // result 값이 0보다 작다면 감점

score = score + score\_table[i].score + result; // 감점된 점수를 반영하여 학생의 점수에 더함

sprintf(tmp, "%.2f,", score\_table[i].score + result); // tmp에 해당 문제에 대하여 학생이 받은 점수를 기록

}

write(fd, tmp, strlen(tmp)); // score.csv에 tmp에 기록해 놨던 점수 출력

}

}

// 항상 p옵션이 수행되어야 하므로 수정 //////////////////////////////////////////////////////////////////////////////////////////////////

// if(pOption) // p옵션이 설정되어 있다면 -> 항상 수행되도록 해야함

// printf("%s is finished.. score : %.2f\n", id, score); // 해당 학생의 총점 출력

// else

// printf("%s is finished..\n", id);

printf("%s is finished. score : %.2f\n", id, score); // 해당 학생의 총점 출력

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

sprintf(tmp, "%.2f\n", score); // 학생의 총점 tmp에 기록

write(fd, tmp, strlen(tmp)); // tmp에 있는 학생의 총점 score.csv에 출력

return score; // 해당 학생의 총점 return

}

void write\_first\_row(int fd) // score.csv의 첫번째 열을 write하는 함수

{

int i;

char tmp[BUFLEN];

int size = sizeof(score\_table) / sizeof(score\_table[0]); // 전체 문항 수

write(fd, ",", 1);

for(i = 0; i < size; i++){

if(score\_table[i].score == 0)

break;

// score.csv에 문제 번호 출력

sprintf(tmp, "%s,", score\_table[i].qname);

write(fd, tmp, strlen(tmp));

}

write(fd, "sum\n", 4);

}

char \*get\_answer(int fd, char \*result) // 답안 파일에서 내용을 읽어와 result에 저장하는 함수

{

char c;

int idx = 0;

memset(result, 0, BUFLEN); // 전달인자로 받은 result를 0으로 초기화

while(read(fd, &c, 1) > 0) // fd에서 한문자씩 읽어온다

{

if(c == ':') // 읽은 문자가 ':' 이라면

break; // 반복 종료

result[idx++] = c; // result배열에 읽은 문자 넣는다

}

if(result[strlen(result) - 1] == '\n') // result에 들어간 문자열의 마지막 문자가 개행문자라면

result[strlen(result) - 1] = '\0'; // 널문자로 바꿈

return result;

}

int score\_blank(char \*id, char \*filename) // 빈칸 문제를 채점하는 함수, 리턴값은 정답 여부 또는 감점된 점수

{

char tokens[TOKEN\_CNT][MINLEN];

node \*std\_root = NULL, \*ans\_root = NULL;

int idx, start;

char tmp[BUFLEN];

char s\_answer[BUFLEN], a\_answer[BUFLEN];

char qname[FILELEN]; // 문제 번호를 저장할 배열

int fd\_std, fd\_ans; // fd\_std는 학생의 답안파일의 파일디스크립터, fd\_ans는 정답 파일의 파일 디스크립터

int result = true; // 정답인지 오답인지

int has\_semicolon = false; // 학생의 답 맨 끝에 세미콜론이 있었는지 기록해 놓을 변수

memset(qname, 0, sizeof(qname)); // qname 배열 0 초기화

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.'))); // qname에 확장자 명을 뺀 파일 이름(문제 번호)을 넣음

sprintf(tmp, "%s/%s/%s", stuDir, id, filename); // 현재 문제 경로 tmp에 저장

fd\_std = open(tmp, O\_RDONLY); // fd\_std에 학생의 답안 파일 파일디스크립터 저장

strcpy(s\_answer, get\_answer(fd\_std, s\_answer)); // 학생이 제출한 답안 파일에서 답을 읽어와 s\_answer에 저장

if(!strcmp(s\_answer, "")){ // 학생의 답이 비어있다면

close(fd\_std);

return false; // 오답

}

if(!check\_brackets(s\_answer)){ // 여는 괄호, 닫는 괄호의 짝이 맞지 않으면

close(fd\_std);

return false; // 오답

}

strcpy(s\_answer, ltrim(rtrim(s\_answer))); // 학생의 답 앞뒤에 있는 white space를 제거하여 다시 s\_answer에 담는다

if(s\_answer[strlen(s\_answer) - 1] == ';'){ // 학생의 답 제일 뒤에 ;이 있다면

has\_semicolon = true; // 세미콜론이 있었다고 기록하고

s\_answer[strlen(s\_answer) - 1] = '\0'; // 널문자를 넣는다

}

if(!make\_tokens(s\_answer, tokens)){ // 학생의 답을 토큰들로 분해, 토큰으로 나누는 과정에서 오답임이 밝혀지면

close(fd\_std);

return false; // false 리턴

}

idx = 0;

std\_root = make\_tree(std\_root, tokens, &idx, 0); // 위에서 생성한 토큰들을 트리에 넣는다

// 답안 디렉터리의 바로 아래에 답안 파일이 있으므로 수정 필요 //////////////////////////////////////////////////////////////////////

// sprintf(tmp, "%s/%s/%s", ansDir, qname, filename);

sprintf(tmp, "%s/%s", ansDir, filename);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

fd\_ans = open(tmp, O\_RDONLY); // 답안 파일 open

while(1)

{

ans\_root = NULL;

result = true; // 결과값에 true 넣어놓는다

for(idx = 0; idx < TOKEN\_CNT; idx++)

memset(tokens[idx], 0, sizeof(tokens[idx])); // 토큰들 배열 0초기화

strcpy(a\_answer, get\_answer(fd\_ans, a\_answer)); // 답안 파일에서 답을 읽어와 a\_answer에 저장

if(!strcmp(a\_answer, "")) // a\_answer가 null string이면 반복 종료

break;

strcpy(a\_answer, ltrim(rtrim(a\_answer))); // a\_answer 앞뒤의 공백 제거

if(has\_semicolon == false){ // 학생의 답안에 세미콜론이 없었다면

if(a\_answer[strlen(a\_answer) -1] == ';') // 답안에는 세미콜론이 있다면

continue; // 다음 답안 확인

}

else if(has\_semicolon == true) // 학생의 답안에 세미콜론이 있었다면

{

if(a\_answer[strlen(a\_answer) - 1] != ';') // 답안에는 세미콜론이 없다면

continue; // 다음 답안 확인

else // 세미콜론 있다면

a\_answer[strlen(a\_answer) - 1] = '\0'; // 널문자로 바꿈

}

if(!make\_tokens(a\_answer, tokens)) // 답안을 토큰으로 나눈다

continue; // 토큰으로 나누는 과정에서 잘못됐다면 다음 답안으로 이동

idx = 0;

ans\_root = make\_tree(ans\_root, tokens, &idx, 0); // 나눈 토큰들을 트리에 넣는다

compare\_tree(std\_root, ans\_root, &result); // 학생이 제출한 답으로 만든 트리와 정답으로 만든 트리를 비교해서 정답인지 확인한다

if(result == true){ // 결과가 true라면

close(fd\_std);

close(fd\_ans);

if(std\_root != NULL)

free\_node(std\_root);

if(ans\_root != NULL)

free\_node(ans\_root);

return true; // true 리턴

}

}

close(fd\_std);

close(fd\_ans);

if(std\_root != NULL)

free\_node(std\_root);

if(ans\_root != NULL)

free\_node(ans\_root);

return false; // 오답 리턴

}

double score\_program(char \*id, char \*filename) // 프로그램 문제를 채점하는 함수, 리턴값은 정답 여부 또는 감점된 점수

{

double compile; // 컴파일 결과를 저장할 변수

int result;

compile = compile\_program(id, filename); // 컴파일 수행

if(compile == ERROR || compile == false) // 컴파일 실패했다면

return false; // false 리턴

result = execute\_program(id, filename); // 정답 파일과 학생의 답안 파일을 실행하고, 결과 파일을 비교하여 정답인지 오답인지 확인한다.

if(!result) // 오답이라면

return false; // false 리턴

if(compile < 0) // 감점됐다면

return compile; // 감점된 점수 리턴

return true; // 정답이면 true 리턴

}

int is\_thread(char \*qname) // -t(lpthread 사용) 옵션으로 지정된 문제인지 확인하는 함수

{

int i;

int size = sizeof(threadFiles) / sizeof(threadFiles[0]); // -lpthread 옵션으로 실행할 프로그램들의 목록이 저장된 threadFiles배열의 크기 계산

for(i = 0; i < size; i++){

if(!strcmp(threadFiles[i], qname)) // 인자로 전달된 qname이 threadFiles에 있다면

return true; // true 리턴

}

return false; // qname이 threadFiles에 없으면 false 리턴

}

double compile\_program(char \*id, char \*filename) // 프로그램 문제를 컴파일하는 함수

{

int fd;

char tmp\_f[BUFLEN], tmp\_e[BUFLEN];

char command[BUFLEN];

char qname[FILELEN];

int isthread;

off\_t size;

double result;

memset(qname, 0, sizeof(qname)); // qname 0초기화

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.'))); // qname에 확장자명 제외한 파일명 복사

isthread = is\_thread(qname); // -lpthread 옵션으로 컴파일을 할것인지 확인

// 답안 디렉터리 바로 아래에 답안 파일들이 있으므로 수정 필요 //////////////////////////////////////////////////////////////////////////////

// sprintf(tmp\_f, "%s/%s/%s", ansDir, qname, filename); // 컴파일 할 파일 명

// sprintf(tmp\_e, "%s/%s/%s.exe", ansDir, qname, qname); // 컴파일 결과로 나올 실행파일의 파일 명

sprintf(tmp\_f, "%s/%s", ansDir, filename); // 컴파일 할 파일 명

sprintf(tmp\_e, "%s/%s.exe", ansDir, qname); // 컴파일 결과로 나올 실행파일의 파일 명

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

if(tOption && isthread) // -lpthread 옵션으로 컴파일할 파일이라면

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f); // -lpthread 옵션으로 컴파일 할 때 사용할 문자열

else

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f); // 그냥 컴파일 할 때 사용할 문자열

// 답안 디렉터리 바로 아래에 답안 파일들이 있으므로 수정 필요 //////////////////////////////////////////////////////////////////////////////

// sprintf(tmp\_e, "%s/%s/%s\_error.txt", ansDir, qname, qname); // 컴파일 에러를 출력할 파일명 생성

sprintf(tmp\_e, "%s/%s\_error.txt", ansDir, qname); // 컴파일 에러를 출력할 파일명 생성

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

fd = creat(tmp\_e, 0666); // 에러 출력할 파일 생성

redirection(command, fd, STDERR); // command를 실행하고 실행하는 동안 STDERR에 출력될 내용을 fd(에러 출력할 파일)에 출력함

size = lseek(fd, 0, SEEK\_END); // 에러 출력된 파일의 크기 저장

close(fd); // 에러 출력된 파일 close

unlink(tmp\_e); // 에러 출력된 파일 삭제 - 학생 답안 파일이 아닌 정답 파일 컴파일 시 에러 내용이므로 저장할 필요 없음

if(size > 0) // 컴파일 에러 발생했다면

return false; // false 리턴

sprintf(tmp\_f, "%s/%s/%s", stuDir, id, filename); // 컴파일 할 학생 답안 파일

sprintf(tmp\_e, "%s/%s/%s.stdexe", stuDir, id, qname); // 컴파일 결과로 나올 실행파일의 파일 명

if(tOption && isthread) // -lpthread 옵션으로 컴파일할 파일이라면

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f); // -lpthread 옵션으로 컴파일 할 때 사용할 command

else

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f); // 그냥 컴파일 할 때 사용할 command

sprintf(tmp\_f, "%s/%s/%s\_error.txt", stuDir, id, qname); // 컴파일 에러 내용을 저장할 파일 명

fd = creat(tmp\_f, 0666); // 에러 저장할 파일 생성

redirection(command, fd, STDERR); // command를 실행하고 실행하는 동안 STDERR에 출력될 내용을 fd(에러 출력할 파일)에 출력함

size = lseek(fd, 0, SEEK\_END); // 에러 출력된 파일의 크기 저장

close(fd); // 에러 출력된 파일 close

if(size > 0){ // 컴파일 에러 발생했다면

if(eOption) // e 옵션이 지정되어 있다면

{

sprintf(tmp\_e, "%s/%s", errorDir, id); // 에러 파일 저장할 경로

if(access(tmp\_e, F\_OK) < 0) // 해당 경로에 접근 불가하면

mkdir(tmp\_e, 0755); // 디렉터리 생성

sprintf(tmp\_e, "%s/%s/%s\_error.txt", errorDir, id, qname); // 에러 파일명

rename(tmp\_f, tmp\_e); // 위에서 만들어둔 에러파일의 이름을 제대로 된 에러 파일명으로 바꾼다

result = check\_error\_warning(tmp\_e); // 에러 내용을 확인해 결과 점수를 계산한다

}

else{

result = check\_error\_warning(tmp\_f); // 에러 내용을 확인해 결과 점수를 계산한다

unlink(tmp\_f); // 에러내용 저장했던 파일 삭제

}

return result; // 결과 점수 리턴

}

unlink(tmp\_f); // 에러 내용 파일 삭제

return true; // 컴파일 성공 했으므로 true 리턴

}

double check\_error\_warning(char \*filename) // 컴파일 에러 내용이 저장된 파일을 이용해 error인지 warning인지 확인해서 결과 점수를 리턴하는 함수

{

FILE \*fp;

char tmp[BUFLEN];

double warning = 0;

if((fp = fopen(filename, "r")) == NULL){ // 파일을 읽기 모드로 open

fprintf(stderr, "fopen error for %s\n", filename);

return false; // 파일을 열지 못했으면 false 리턴

}

while(fscanf(fp, "%s", tmp) > 0){ // 파일 내용 확인

if(!strcmp(tmp, "error:")) // error: 라는 문자열이 들어있으면

return ERROR; // ERROR 리턴

else if(!strcmp(tmp, "warning:")) // warning: 이라는 문자열이 들어있으면

warning += WARNING; // WARNING 점수를 누적해서 저장

}

return warning; // 점수 리턴

}

int execute\_program(char \*id, char \*filename) // 프로그램 문제를 실행하는 함수

{

char std\_fname[BUFLEN], ans\_fname[BUFLEN];

char tmp[BUFLEN];

char qname[FILELEN];

time\_t start, end;

pid\_t pid;

int fd;

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 /////////////////////////////////////////////////////////////////////////////////

int tmpSTDERR;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

memset(qname, 0, sizeof(qname)); // qname 0초기화

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.'))); // qname에 문제 번호 저장

// 답안 디렉터리 하위에 바로 답안 파일들이 있으므로 수정 필요 ///////////////////////////////////////////////////////////////

// sprintf(ans\_fname, "%s/%s/%s.stdout", ansDir, qname, qname); // 정답 실행파일의 실행결과를 저장할 파일

sprintf(ans\_fname, "%s/%s.stdout", ansDir, qname); // 정답 실행파일의 실행결과를 저장할 파일

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

fd = creat(ans\_fname, 0666); // 실행결과 저장파일 생성

// 답안 디렉터리 하위에 바로 답안 파일들이 있으므로 수정 필요 ///////////////////////////////////////////////////////////////

// sprintf(tmp, "%s/%s/%s.exe", ansDir, qname, qname); // 정답 실행 파일

sprintf(tmp, "%s/%s.exe", ansDir, qname); // 정답 실행 파일

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 /////////////////////////////////////////////////////////////////////////////////

tmpSTDERR = dup(STDERR);

dup2(fd, STDERR);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

redirection(tmp, fd, STDOUT); // 정답 실행 파일을 실행시키고 그 결과를 위에서 만든 실행 결과 저장 파일에 저장

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 /////////////////////////////////////////////////////////////////////////////////

dup2(tmpSTDERR, STDERR);

close(tmpSTDERR);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

close(fd); // 실행 결과 저장 파일 close

sprintf(std\_fname, "%s/%s/%s.stdout", stuDir, id, qname); // 학생의 답안을 실행한 결과를 저장할 파일

fd = creat(std\_fname, 0666); // 학생 답안 실행결과 저장파일 생성

sprintf(tmp, "%s/%s/%s.stdexe &", stuDir, id, qname); // 학생 답안 실행 파일

start = time(NULL); // 학생 답안 실행 시작시간 저장

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 /////////////////////////////////////////////////////////////////////////////////

tmpSTDERR = dup(STDERR);

dup2(fd, STDERR);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

redirection(tmp, fd, STDOUT);// 학생 답안을 실행하고 결과를 저장

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 /////////////////////////////////////////////////////////////////////////////////

dup2(tmpSTDERR, STDERR);

close(tmpSTDERR);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

sprintf(tmp, "%s.stdexe", qname); // 학생 답안 실행파일 파일명

while((pid = inBackground(tmp)) > 0){ // 학생 답안 실행파일이 실행되는 동안 반복하며 체크

end = time(NULL); // 얼마동안 실행중인지 저장

if(difftime(end, start) > OVER){ // 실행 제한 시간(5초)을 초과했다면

kill(pid, SIGKILL); // 종료시킴

close(fd);

return false; // false 리턴

}

}

close(fd);

return compare\_resultfile(std\_fname, ans\_fname); // 정답과 학생 답안의 결과를 비교해 리턴

}

pid\_t inBackground(char \*name) // 인자로 전달된 프로세스가 실행중인지 확인하는 함수

{

pid\_t pid;

char command[64];

char tmp[64];

int fd;

off\_t size;

memset(tmp, 0, sizeof(tmp));

fd = open("background.txt", O\_RDWR | O\_CREAT | O\_TRUNC, 0666);

sprintf(command, "ps | grep %s", name);

redirection(command, fd, STDOUT); // ps | grep <name> 실행 뒤 그 결과를 background.txt에 저장

lseek(fd, 0, SEEK\_SET);

read(fd, tmp, sizeof(tmp)); // 위의 실행 결과를 읽어옴

if(!strcmp(tmp, "")){ // 아무것도 출력되지 않았다면

unlink("background.txt");

close(fd);

return 0;

}

pid = atoi(strtok(tmp, " ")); // pid 저장

close(fd);

unlink("background.txt");

return pid;

}

int compare\_resultfile(char \*file1, char \*file2) // 프로그램을 실행한 결과로 나온 파일을 비교하는 함수

{

int fd1, fd2;

char c1, c2;

int len1, len2;

fd1 = open(file1, O\_RDONLY); // 첫번째 파일 open

fd2 = open(file2, O\_RDONLY); // 두번째 파일 open

while(1)

{

while((len1 = read(fd1, &c1, 1)) > 0){ // 첫번째 파일 공백문자 제거

if(c1 == ' ')

continue;

else

break;

}

while((len2 = read(fd2, &c2, 1)) > 0){ // 두번째 파일 공백문자 제거

if(c2 == ' ')

continue;

else

break;

}

if(len1 == 0 && len2 == 0) // 파일의 끝이라면

break; // 반복 종료

to\_lower\_case(&c1); // c1에 저장된 문자가 대문자 알파벳인 경우 소문자로 변환

to\_lower\_case(&c2); // c2에 저장된 문자가 대문자 알파벳인 경우 소문자로 변환

if(c1 != c2){ // c1과 c2가 다르다면 프로그램 수행 결과가 다르다는 뜻이므로

close(fd1);

close(fd2);

return false; // 오답

}

}

close(fd1);

close(fd2);

return true; // 정답

}

void redirection(char \*command, int new, int old) // new 파일디스크립터를 old에 복사한 뒤 command를 실행한다. 실행한 후에는 다시 원래 old에 있던 값으로 복구

{

int saved;

saved = dup(old);

dup2(new, old);

system(command);

dup2(saved, old);

close(saved);

}

int get\_file\_type(char \*filename) // 파일의 확장자를 확인하는 함수

{

char \*extension = strrchr(filename, '.'); // filename에서 '.'이 있는 위치의 포인터를 extension에 저장

if(!strcmp(extension, ".txt")) // 파일 확장자가 .txt 라면

return TEXTFILE;

else if (!strcmp(extension, ".c")) // 파일 확장자가 .c 라면

return CFILE;

else // .c도, .txt도 아니라면

return -1;

}

void rmdirs(const char \*path) // 디렉터리 삭제하는 함수

{

struct dirent \*dirp;

struct stat statbuf;

DIR \*dp;

char tmp[50];

if((dp = opendir(path)) == NULL) // opendir 실패시

return;

while((dirp = readdir(dp)) != NULL) // 디렉토리 내용 확인

{

if(!strcmp(dirp->d\_name, ".") || !strcmp(dirp->d\_name, "..")) // . .. 디렉토리는 pass

continue;

sprintf(tmp, "%s/%s", path, dirp->d\_name);

if(lstat(tmp, &statbuf) == -1) // stat구조체 가져옴

continue;

if(S\_ISDIR(statbuf.st\_mode)) // 디렉토리 파일일 경우

rmdirs(tmp); // 재귀호출

else // 아닌경우

unlink(tmp); // unlink

}

closedir(dp);

rmdir(path);

}

void to\_lower\_case(char \*c) // 대문자 알파벳을 소문자로 변환하는 함수

{

if(\*c >= 'A' && \*c <= 'Z') // 대문자 알파벳 일때만

\*c = \*c + 32; // 소문자로 변환

}

void print\_usage() // 프로그램 사용법 출력

{

printf("Usage : ssu\_score <STUDENTDIR> <TRUEDIR> [OPTION]\n");

printf("Option : \n");

// m 옵션 추가 /////////////////////////////////////////////////////////////////////////////////////

printf(" -m modify question's score\n");

/////////////////////////////////////////////////////////////////////////////////////////////////////////

printf(" -e <DIRNAME> print error on 'DIRNAME/ID/qname\_error.txt' file \n");

printf(" -t <QNAMES> compile QNAME.C with -lpthread option\n");

// i 옵션 추가 /////////////////////////////////////////////////////////////////////////////////////

printf(" -i <IDS> print ID's wrong questions\n");

/////////////////////////////////////////////////////////////////////////////////////////////////////////

printf(" -h print usage\n");

// 항상 p옵션이 수행되어야 하므로 수정 필요 ////////////////////////////////////////////////////////////

//printf(" -p print student's score and total average\n");

/////////////////////////////////////////////////////////////////////////////////////////////////////////

// c 옵션은 필요 없음 ///////////////////////////////////////////////////////////////////////////////////

//printf(" -c <IDS> print ID's score\n");

/////////////////////////////////////////////////////////////////////////////////////////////////////////

}

**<blank.h>**

#ifndef BLANK\_H\_

#define BLANK\_H\_

#ifndef true

#define true 1

#endif

#ifndef false

#define false 0

#endif

#ifndef BUFLEN

#define BUFLEN 1024

#endif

#define OPERATOR\_CNT 24 // 연산자 개수

#define DATATYPE\_SIZE 35 // 데이터 타입 개수

#define MINLEN 64

#define TOKEN\_CNT 50 // 토큰 최대 개수

typedef struct node{ // 트리의 노드

int parentheses; // 몇번째 괄호 안에 있는 토큰인지

char \*name; // 토큰

struct node \*parent; // 부모 노드

struct node \*child\_head; // 자식 노드

struct node \*prev; // 앞쪽 형제 노드

struct node \*next; // 뒤쪽 형제 노드

}node;

typedef struct operator\_precedence{ // 연산자, 연산자 우선순위 함께 저장된 구조체

char \*operator; // 연산자

int precedence; // 연산자 우선순위

}operator\_precedence;

void compare\_tree(node \*root1, node \*root2, int \*result);// 두 트리(답안)을 비교하여 같은 내용인지, 다른 내용인지 알아내는 함수

node \*make\_tree(node \*root, char (\*tokens)[MINLEN], int \*idx, int parentheses);// tokens에 들어있는 토큰들을 트리에 넣는 함수

node \*change\_sibling(node \*parent);// 전달인자로 받은 노드의 자식 노드들이 저장된 순서를 바꾸는 함수

node \*create\_node(char \*name, int parentheses);// 새로운 노드를 생성하는 함수

int get\_precedence(char \*op);// 연산자의 우선순위를 구하는 함수

int is\_operator(char \*op);// 전달인자로 받은 문자가 연산자인지 확인하는 함수

void print(node \*cur); // 노드 정보 출력

node \*get\_operator(node \*cur); // 해당 노드에 대한 연산자 찾는 함수

node \*get\_root(node \*cur); // 트리의 루트를 찾아 리턴하는 함수

node \*get\_high\_precedence\_node(node \*cur, node \*new);// 더 우선순위가 높은 노드를 찾는 함수

node \*get\_most\_high\_precedence\_node(node \*cur, node \*new);// 가장 우선순위가 높은 연산자를 구하는 함수

node \*insert\_node(node \*old, node \*new);// 새로운 노드를 old의 자리에 삽입하는 함수

node \*get\_last\_child(node \*cur);// 제일 끝의 자식 노드 찾는 함수

void free\_node(node \*cur);// 인자로 받은 노드 삭제하는 함수 (자식 노드, 뒤에 있는 형제노드들도 삭제)

int get\_sibling\_cnt(node \*cur);// 형제 노드의 개수를 세는 함수

int make\_tokens(char \*str, char tokens[TOKEN\_CNT][MINLEN]);// 답안을 토큰으로 분해하는 함수

int is\_typeStatement(char \*str);// type이 맨 앞에 있는 구문(선언문?)인지 확인하는 함수

int find\_typeSpecifier(char tokens[TOKEN\_CNT][MINLEN]);// 토큰들 중에서 형식 지정자를 찾는 함수

int find\_typeSpecifier2(char tokens[TOKEN\_CNT][MINLEN]);// 토큰에서 struct 형식 지정자를 찾는 함수

int is\_character(char c);// c가 알파벳, 숫자인지 체크하는 함수

int all\_star(char \*str);// 이 문자열이 전부 '\*' 문자로 이루어졌는지 확인하는 함수

int all\_character(char \*str);// 인자로 받은 문자열이 전부 숫자나 영어 알파벳으로만 이루어져 있는지 확인하는 함수

int reset\_tokens(int start, char tokens[TOKEN\_CNT][MINLEN]);// 토큰들 정리하는 함수

void clear\_tokens(char tokens[TOKEN\_CNT][MINLEN]);// 전달인자로 받은 token 배열을 0으로 초기화하는 함수

int get\_token\_cnt(char tokens[TOKEN\_CNT][MINLEN]);// 토큰의 총 개수 리턴하는 함수

char \*rtrim(char \*\_str);// 문자열 오른쪽의 white space 제거

char \*ltrim(char \*\_str);// 문자열 왼쪽의 white space 제거

void remove\_space(char \*str);// 문자열 내에 있는 공백문자 지우는 함수

int check\_brackets(char \*str);// 괄호가 짝을 맞춰 제대로 있는지 검사하는 함수

char\* remove\_extraspace(char \*str); // 의미없는 공백 지우는 함수

#endif

**<blank.c>**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <ctype.h>

#include "blank.h"

char datatype[DATATYPE\_SIZE][MINLEN] = {"int", "char", "double", "float", "long"

, "short", "ushort", "FILE", "DIR","pid"

,"key\_t", "ssize\_t", "mode\_t", "ino\_t", "dev\_t"

, "nlink\_t", "uid\_t", "gid\_t", "time\_t", "blksize\_t"

, "blkcnt\_t", "pid\_t", "pthread\_mutex\_t", "pthread\_cond\_t", "pthread\_t"

, "void", "size\_t", "unsigned", "sigset\_t", "sigjmp\_buf"

, "rlim\_t", "jmp\_buf", "sig\_atomic\_t", "clock\_t", "struct"}; // 자료형들

operator\_precedence operators[OPERATOR\_CNT] = {

{"(", 0}, {")", 0}

,{"->", 1}

,{"\*", 4} ,{"/", 3} ,{"%", 2}

,{"+", 6} ,{"-", 5}

,{"<", 7} ,{"<=", 7} ,{">", 7} ,{">=", 7}

,{"==", 8} ,{"!=", 8}

,{"&", 9}

,{"^", 10}

,{"|", 11}

,{"&&", 12}

,{"||", 13}

,{"=", 14} ,{"+=", 14} ,{"-=", 14} ,{"&=", 14} ,{"|=", 14}

}; // 연산자와 연산자 우선순위

void compare\_tree(node \*root1, node \*root2, int \*result) // 두 트리(답안)을 비교하여 같은 내용인지, 다른 내용인지 알아내는 함수

{

node \*tmp;

int cnt1, cnt2;

if(root1 == NULL || root2 == NULL){ // root1 root2중 NULL인 것이 있다면

\*result = false;

return;

}

if(!strcmp(root1->name, "<") || !strcmp(root1->name, ">") || !strcmp(root1->name, "<=") || !strcmp(root1->name, ">=")){ // 루트1의 토큰이 비교연산자라면

if(strcmp(root1->name, root2->name) != 0){ // 두 트리의 루트의 토큰이 서로 다르다면

if(!strncmp(root2->name, "<", 1)) // 루트2의 토큰이 < 라면

strncpy(root2->name, ">", 1); // 루트2에 > 복사

else if(!strncmp(root2->name, ">", 1)) // 루트2의 토큰이 >라면

strncpy(root2->name, "<", 1); // 루트2에 < 복사

else if(!strncmp(root2->name, "<=", 2)) // 루트2의 토큰이 <=라면

strncpy(root2->name, ">=", 2); // 루트2에 >= 복사

else if(!strncmp(root2->name, ">=", 2)) // 루트2의 토큰이 >=라면

strncpy(root2->name, "<=", 2); // 루트2에 <= 복사

root2 = change\_sibling(root2); // 루트2의 자식 노드들의 순서를 바꾼다

}

}

if(strcmp(root1->name, root2->name) != 0){ // 루트1의 토큰과 루트2의 토큰이 서로 다르다면

\*result = false; // 오답

return;

}

if((root1->child\_head != NULL && root2->child\_head == NULL) // 루트1과 루트2의 자식 노드 존재 유무가 서로 다르다면

|| (root1->child\_head == NULL && root2->child\_head != NULL)){

\*result = false; // 오답

return;

}

else if(root1->child\_head != NULL){ // 루트1에 자식 노드가 있다면

if(get\_sibling\_cnt(root1->child\_head) != get\_sibling\_cnt(root2->child\_head)){ // 루트1과 루트2의 자식 노드 개수가 서로 다르다면

\*result = false; // 오답

return;

}

if(!strcmp(root1->name, "==") || !strcmp(root1->name, "!=")) // 루트1의 토큰이 == 또는 != 이라면

{

compare\_tree(root1->child\_head, root2->child\_head, result); // 재귀호출하여 루트1과 루트2의 자식노드를 비교

if(\*result == false) // 위의 재귀 호출 결과가 flase라면

{

\*result = true; // 결과를 true로 바꾸고

root2 = change\_sibling(root2); // 루트 2의 자식 노드들의 순서를 바꿔놓고

compare\_tree(root1->child\_head, root2->child\_head, result); // 다시 비교

}

}

else if(!strcmp(root1->name, "+") || !strcmp(root1->name, "\*") // 루트1의 토큰이 +, \*, |, ||, &, && 연산자라면

|| !strcmp(root1->name, "|") || !strcmp(root1->name, "&")

|| !strcmp(root1->name, "||") || !strcmp(root1->name, "&&"))

{

if(get\_sibling\_cnt(root1->child\_head) != get\_sibling\_cnt(root2->child\_head)){ // 루트1과 루트2의 자식 노드의 개수가 서로 다르다면

\*result = false; // 오답

return;

}

tmp = root2->child\_head; // tmp에 루트2의 자식노드 포인터 저장

while(tmp->prev != NULL) // 루트2의 자식노드의 앞쪽에 다른 형제 노드가 있다면

tmp = tmp->prev; // 앞쪽 형제노드로 이동

while(tmp != NULL)

{

compare\_tree(root1->child\_head, tmp, result); // 재귀호출하여 위에서 구한 루트2의 자식노드와 루트1의 자식노드를 비교

if(\*result == true) // 결과가 true라면

break; // 반복 그만

else{ // 결과가 true가 아니라면

if(tmp->next != NULL) // tmp의 뒤에 다른 형제노드가 있다면

\*result = true; // 결과에 true를 넣어놓고

tmp = tmp->next; // 다음 형제노드로 이동

}

}

}

else{

compare\_tree(root1->child\_head, root2->child\_head, result); // 재귀호출로 루트1의 자식노드와 루트2의 자식노드를 다시 비교한다

}

}

if(root1->next != NULL){ // 루트 1의 뒤에 다른 형제노드가 있다면

if(get\_sibling\_cnt(root1) != get\_sibling\_cnt(root2)){ // 루트1과 루트2의 자식 노드의 개수가 서로 다르다면

\*result = false; // 오답

return;

}

if(\*result == true) // 결과값에 true가 들어있다면

{

tmp = get\_operator(root1); // root1에 대한 연산자 노드 찾는다

if(!strcmp(tmp->name, "+") || !strcmp(tmp->name, "\*") // 찾은 연산자가 +, \*, |, ||, &, && 중 하나라면

|| !strcmp(tmp->name, "|") || !strcmp(tmp->name, "&")

|| !strcmp(tmp->name, "||") || !strcmp(tmp->name, "&&"))

{

tmp = root2;

while(tmp->prev != NULL) // 루트2의 형제 노드들 중 제일 앞에 있는 노드로 이동

tmp = tmp->prev;

while(tmp != NULL)

{

compare\_tree(root1->next, tmp, result); // 재귀호출하여 위에서 구한 루트2의 형제노드와 루트1의 형제노드를 비교

if(\*result == true) // 결과가 true라면

break; // 반복 그만

else{ // false라면

if(tmp->next != NULL) // 다음 형제 노드가 있다면

\*result = true; // 결과값으로 true를 넣어놓고

tmp = tmp->next; // 다음 형제 노드로 이동

}

}

}

else

compare\_tree(root1->next, root2->next, result); // 재귀호출하여 루트1의 다음 형제노드와 루트2의 다음 형제노드를 비교

}

}

}

int make\_tokens(char \*str, char tokens[TOKEN\_CNT][MINLEN]) // 답안을 토큰으로 분해하는 함수

{

char \*start, \*end;

char tmp[BUFLEN];

char str2[BUFLEN];

char \*op = "(),;><=!|&^/+-\*\""; // 연산자들 모음

int row = 0; // 몇번째 토큰인지?

int i;

int isPointer;

int lcount, rcount;

int p\_str;

clear\_tokens(tokens); // token 배열 0으로 초기화

start = str; // start에 str의 시작 위치 저장

if(is\_typeStatement(str) == 0) // 맨 앞에 type이 써있는 구문이라면

return false;

while(1)

{

if((end = strpbrk(start, op)) == NULL) // 문자열(start)에 연산자가 하나도 포함되어 있지 않으면

break;

if(start == end){ // 다음 토큰이 연산자라면

if(!strncmp(start, "--", 2) || !strncmp(start, "++", 2)){ // -- 또는 ++라면

if(!strncmp(start, "++++", 4)||!strncmp(start,"----",4)) // ++++ 또는 ----이라면

return false;

if(is\_character(\*ltrim(start + 2))){ // -- 또는 ++ 뒤의 공백문자 제거 후 뒤에 오는 문자가 알파벳이나 숫자 문자라면 (전위 연산자)

if(row > 0 && /\*!\*/is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1])) // 이전 토큰의 마지막 문자가 알파벳이나 숫자 문자가 아니라면 ////////////////////////////

return false;

end = strpbrk(start + 2, op); // 다음 연산자의 위치를 end에 저장

if(end == NULL) // 뒤에 더이상 연산자가 없다면

end = &str[strlen(str)]; // end에 문자열의 끝 위치를 저장

while(start < end) { // 문자열의 끝에 다다를 때까지

if(\*(start - 1) == ' ' && is\_character(tokens[row][strlen(tokens[row]) - 1]))

return false;

else if(\*start != ' ') // \*start가 공백문자가 아니라면

strncat(tokens[row], start, 1); // tokens[row]에 전위 연산자를 포함한 토큰 넣음

start++;

}

}

else if(row>0 && is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1])){ // -- 또는 ++ 앞에 있던 토큰의 마지막 문자가 숫자 또는 알파벳 이라면 (후위 연산자)

if(strstr(tokens[row - 1], "++") != NULL || strstr(tokens[row - 1], "--") != NULL) // 바로 앞의 토큰이 ++ 또는 --를 포함하고 있다면

return false;

memset(tmp, 0, sizeof(tmp)); // tmp배열 0초기화

strncpy(tmp, start, 2); //tmp에 -- 또는 ++ 연산자 복사

strcat(tokens[row - 1], tmp); // 앞의 토큰 뒤에 -- 또는 ++ 연산자 갖다 붙임

start += 2; // 다음 토큰으로 이동

row--;

}

else{

memset(tmp, 0, sizeof(tmp)); //tmp 배열 0초기화

strncpy(tmp, start, 2); // tmp 배열에 ++ 또는 -- 복사

strcat(tokens[row], tmp); // tokens[row]의 뒤에 ++ 또는 -- 덧붙임

start += 2; // 다음 토큰으로 이동

}

}

else if(!strncmp(start, "==", 2) || !strncmp(start, "!=", 2) || !strncmp(start, "<=", 2)

|| !strncmp(start, ">=", 2) || !strncmp(start, "||", 2) || !strncmp(start, "&&", 2)

|| !strncmp(start, "&=", 2) || !strncmp(start, "^=", 2) || !strncmp(start, "!=", 2)

|| !strncmp(start, "|=", 2) || !strncmp(start, "+=", 2) || !strncmp(start, "-=", 2)

|| !strncmp(start, "\*=", 2) || !strncmp(start, "/=", 2)){ // 이 연산자들로 시작하는 문자열이라면

strncpy(tokens[row], start, 2); // 이 연산자들을 tokens[row]에 복사

start += 2; // 다음 토큰으로 이동

}

else if(!strncmp(start, "->", 2)) // -> 연산자라면

{

end = strpbrk(start + 2, op); // 이 뒤에 다른 연산자가 있는지 검사

if(end == NULL) // 뒤에 다른 연산자가 없다면

end = &str[strlen(str)];

while(start < end){ // 다음 연산자 까지 or 뒤에 더이상 연산자가 없다면 문자열의 끝까지

if(\*start != ' ') // \*start가 공백문자가 아니라면

strncat(tokens[row - 1], start, 1); // 앞의 토큰(tokens[row - 1]) 에 덧붙임

start++; // 다음 문자로 이동

}

row--; // 앞의 토큰에 덧붙였으므로 row 하나 감소

}

else if(\*end == '&') // &연산자라면

{

if(row == 0 || (strpbrk(tokens[row - 1], op) != NULL)){ // 이 & 연산자가 첫번째 토큰이거나 이전 토큰에 연산자가 포함되어 있다면 (주소 연산자 &)

end = strpbrk(start + 1, op); // end에 다음 연산자의 위치 저장

if(end == NULL) // 뒤에 더이상 연산자가 없다면

end = &str[strlen(str)]; // end에 문자열의 끝 위치 저장

strncat(tokens[row], start, 1); // token[row]에 & 연산자 저장

start++;

while(start < end){

if(\*(start - 1) == ' ' && tokens[row][strlen(tokens[row]) - 1] != '&') // & 연산자와 뒤의 피연산자 사이에 공백이 있다면 에러

return false;

else if(\*start != ' ')

strncat(tokens[row], start, 1); // tokens[row]에 한문자씩 덧붙임

start++;

}

}

else{ // 비트연산자 &

strncpy(tokens[row], start, 1); // tokens[row] 에 & 저장

start += 1; // 다음 토큰으로 이동

}

}

else if(\*end == '\*') // \*연산자라면

{

isPointer=0; // 포인터 연산자인지 곱셈 연산자인지를 저장해 놓을 flag 변수

if(row > 0) // 첫번째 토큰이 아니라면

{

for(i = 0; i < DATATYPE\_SIZE; i++) {

if(strstr(tokens[row - 1], datatype[i]) != NULL){ // 이전 토큰이 데이터 타입 이라면 (이 \* 연산자가 포인터 연산자 \*인 경우)

strcat(tokens[row - 1], "\*"); // 앞의 토큰(데이터 타입)의 뒤에 \* 덧붙임

start += 1; // 다음 토큰으로 이동

isPointer = 1; // 이 연산자가 포인터 연산자임을 저장해놓는다

break;

}

}

if(isPointer == 1) // 이 연산자가 포인터 연산자라면 다음 토큰으로 이동

continue;

if(\*(start+1) !=0) // 문자열의 끝이 아니라면

end = start + 1; // 다음 문자로 이동

if(row>1 && !strcmp(tokens[row - 2], "\*") && (all\_star(tokens[row - 1]) == 1)){ // 이 토큰 앞앞의 토큰이 \*이고, 앞의 토큰이 전부 '\*'로 이루어져 있다면(다중 포인터)

strncat(tokens[row - 1], start, end - start); // 앞의 토큰 뒤에 덧붙임

row--;

}

else if(is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1]) == 1){ // 앞의 토큰의 마지막 문자가 알파벳이나 숫자라면

strncat(tokens[row], start, end - start); // tokens[row]의 뒤에 덧붙임

}

else if(strpbrk(tokens[row - 1], op) != NULL){ // 이전 토큰에 연산자가 포함되어 있다면

strncat(tokens[row] , start, end - start); // tokens[row]의 뒤에 덧붙임

}

else

strncat(tokens[row], start, end - start); // tokens[row]의 뒤에 덧붙임

start += (end - start); // 다음 토큰으로 이동

}

else if(row == 0) // 첫번째 토큰이라면

{

if((end = strpbrk(start + 1, op)) == NULL){ // start+1 문자열에 연산자가 포함되어 있지 않다면

strncat(tokens[row], start, 1); // tokens[row] 뒤에 덧붙임

start += 1;

}

else{

while(start < end){

if(\*(start - 1) == ' ' && is\_character(tokens[row][strlen(tokens[row]) - 1]))

return false;

else if(\*start != ' ')

strncat(tokens[row], start, 1);

start++;

}

if(all\_star(tokens[row]))

row--;

}

}

}

else if(\*end == '(') // ( 연산자라면

{

lcount = 0;

rcount = 0;

if(row>0 && (strcmp(tokens[row - 1],"&") == 0 || strcmp(tokens[row - 1], "\*") == 0)){ // 앞의 토큰이 &이나 \*이 아니라면

while(\*(end + lcount + 1) == '(') // 다음 '('의 위치 찾는다

lcount++;

start += lcount;

end = strpbrk(start + 1, ")"); // 다음 ')'의 위치 찾는다

if(end == NULL) // 다음 ')'을 찾지 못했다면

return false;

else{

while(\*(end + rcount +1) == ')') // 다음 ')'의 위치 찾는다

rcount++;

end += rcount;

if(lcount != rcount)

return false;

if( (row > 1 && !is\_character(tokens[row - 2][strlen(tokens[row - 2]) - 1])) || row == 1){ // 앞앞의 토큰의 마지막 문자가 알파벳, 숫자가 아니거나, 이번이 두번째 토큰인 경우

strncat(tokens[row - 1], start + 1, end - start - rcount - 1); // 앞의 토큰 뒤에 //////////////////////////////////

row--;

start = end + 1;

}

else{

strncat(tokens[row], start, 1);

start += 1;

}

}

}

else{

strncat(tokens[row], start, 1);

start += 1;

}

}

else if(\*end == '\"') // \" 문자라면 (문자열)

{

end = strpbrk(start + 1, "\""); // 뒤쪽에 또다른 \"이 있는지 확인

if(end == NULL) // 없다면 짝이 맞지 않으므로

return false;

else{

strncat(tokens[row], start, end - start + 1); // tokens[row]에 \"으로 둘러싸인 토큰 넣는다

start = end + 1; // 다음 토큰으로 이동

}

}

else{ // 그 외의 연산자라면

if(row > 0 && !strcmp(tokens[row - 1], "++")) // 마지막 토큰이 ++라면 짝이 맞지 않으므로

return false;

if(row > 0 && !strcmp(tokens[row - 1], "--")) // 마지막 토큰이 ++라면 짝이 맞지 않으므로

return false;

strncat(tokens[row], start, 1); // tokens[row]에 연산자 넣는다

start += 1; // 다음 토큰으로 이동

if(!strcmp(tokens[row], "-") || !strcmp(tokens[row], "+") || !strcmp(tokens[row], "--") || !strcmp(tokens[row], "++")){ // -, +, --, ++ 연산자라면

if(row == 0) // 첫번째 토큰이었다면

row--;

else if(!is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1])){ // 앞 토큰의 마지막 문자가 알파벳이나 숫자가 아니었다면

if(strstr(tokens[row - 1], "++") == NULL && strstr(tokens[row - 1], "--") == NULL) // 앞의 토큰이 ++ 또는 --를 포함하고 있었다면

row--;

}

}

}

}

else{ // 그 외 연산자들

if(all\_star(tokens[row - 1]) && row > 1 && !is\_character(tokens[row - 2][strlen(tokens[row - 2]) - 1])) // 이전 토큰이 전부 '\*'로 이루어져 있고 앞앞 토큰의 마지막 문자가 숫자나 알파벳이 아니라면

row--;

if(all\_star(tokens[row - 1]) && row == 1) // 앞의 토큰이 전부 '\*'로 이루어져 있고, 이번 토큰이 두번째 토큰이라면 (다중 포인터)

row--;

for(i = 0; i < end - start; i++){

if(i > 0 && \*(start + i) == '.'){ // 다음 문자가 .이면

strncat(tokens[row], start + i, 1); // 덧붙임

while( \*(start + i +1) == ' ' && i< end - start ) // 공백 제거

i++;

}

else if(start[i] == ' '){ // 공백 skip

while(start[i] == ' ')

i++;

break;

}

else

strncat(tokens[row], start + i, 1); // 뒤에 토큰 덧붙임

}

if(start[0] == ' '){ // 첫분자가 공백이라면

start += i; // 다음 문자로 이동

continue;

}

start += i;

}

strcpy(tokens[row], ltrim(rtrim(tokens[row]))); // 토큰의 앞, 뒤로 있는 공백들 제거

if(row > 0 && is\_character(tokens[row][strlen(tokens[row]) - 1])

&& (is\_typeStatement(tokens[row - 1]) == 2

|| is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1])

|| tokens[row - 1][strlen(tokens[row - 1]) - 1] == '.' ) ){ // 새로 구한 토큰이 첫번째 토큰이 아니고, 마지막 문자가 숫자나 알파벳 이고, 변수 선언문이거나, 앞 토큰의 마지막 문자가 숫자나 알파벳이거나, 앞 토큰의 마지막 문자가 '.'이라면

if(row > 1 && strcmp(tokens[row - 2],"(") == 0) // 앞앞 토큰이 '(' 이라면

{

if(strcmp(tokens[row - 1], "struct") != 0 && strcmp(tokens[row - 1],"unsigned") != 0) // 앞 토큰이 struct, unsigned 둘 다 아니라면

return false;

}

else if(row == 1 && is\_character(tokens[row][strlen(tokens[row]) - 1])) { // 이번 토큰이 두번째 토큰이고, 마지막 문자가 숫자나 알파벳이라면

if(strcmp(tokens[0], "extern") != 0 && strcmp(tokens[0], "unsigned") != 0 && is\_typeStatement(tokens[0]) != 2)// 첫번째 토큰이 extern, unsigned, gcc가 아니라면

return false;

}

else if(row > 1 && is\_typeStatement(tokens[row - 1]) == 2){ // 앞 토큰이 gcc라면

if(strcmp(tokens[row - 2], "unsigned") != 0 && strcmp(tokens[row - 2], "extern") != 0) // 앞앞 토큰이 unsigned, extern이 아니라면

return false;

}

}

if((row == 0 && !strcmp(tokens[row], "gcc")) ){ // 이번에 구한 토큰이 gcc라면

clear\_tokens(tokens); // 모든 token들을 0으로 초기확

strcpy(tokens[0], str); // tokens[0]에 str을 넣고

return 1; // 1 리턴

}

row++; // 다음 토큰을 얻기 위해 row + 1

}

if(all\_star(tokens[row - 1]) && row > 1 && !is\_character(tokens[row - 2][strlen(tokens[row - 2]) - 1])) // 앞의 토큰이 전부 '\*'로 되어있고, 그 앞의 토큰의 마지막 문자가 숫자나 알파벳이 아니라면

row--;

if(all\_star(tokens[row - 1]) && row == 1) // 앞의 토큰이 전부 '\*'로 되어있고, 이번에 구한 토큰이 두번째 토큰이라면

row--;

for(i = 0; i < strlen(start); i++)

{

if(start[i] == ' ') // 공백문자라면

{

while(start[i] == ' ') // 공백문자 동안 pass

i++;

if(start[0]==' ') { // 첫 문자가 공백문자라면

start += i; // 공백문자가 아닌 첫번째 문자로 start를 이동시킨다

i = 0;

}

else // 첫 문자가 공백문자가 아니라면

row++; // 다음 토큰

i--;

}

else // 공백문자가 아니라면

{

strncat(tokens[row], start + i, 1); // token[row]에 덧붙임

if( start[i] == '.' && i<strlen(start)){ // 덧붙인 문자가 '.'라면

while(start[i + 1] == ' ' && i < strlen(start)) // 뒤에 있는 공백들 pass

i++;

}

}

strcpy(tokens[row], ltrim(rtrim(tokens[row]))); // tokens[row] 앞, 뒤의 공백문자 제거

if(!strcmp(tokens[row], "lpthread") && row > 0 && !strcmp(tokens[row - 1], "-")){ // 토큰이 -lpthread 라면

strcat(tokens[row - 1], tokens[row]); // 앞의 토큰 뒤에 현재 토큰을 덧붙임

memset(tokens[row], 0, sizeof(tokens[row])); // 현재 토큰 저장했던 부분 0으로 초기화

row--;

}

else if(row > 0 && is\_character(tokens[row][strlen(tokens[row]) - 1])

&& (is\_typeStatement(tokens[row - 1]) == 2

|| is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1])

|| tokens[row - 1][strlen(tokens[row - 1]) - 1] == '.') ){ // 이번에 구한 토큰이 첫번째 토큰이 아니고, 마지막 문자가 숫자나 알파벳이고, 앞 토큰이 gcc이거나, 앞 토큰의 마지막 문자가 숫자나 알파벳이거나, '.'이라면

if(row > 1 && strcmp(tokens[row-2],"(") == 0) // 앞앞 토큰이 (라면

{

if(strcmp(tokens[row-1], "struct") != 0 && strcmp(tokens[row-1], "unsigned") != 0) // 앞 토큰이 struct와 unsigned가 아니라면

return false;

}

else if(row == 1 && is\_character(tokens[row][strlen(tokens[row]) - 1])) { // 이 토큰이 두번째 토큰이고, 토큰의 마지막 문자가 숫자나 알파벳이라면

if(strcmp(tokens[0], "extern") != 0 && strcmp(tokens[0], "unsigned") != 0 && is\_typeStatement(tokens[0]) != 2)// 첫번째 토큰이 extern이나 unsigned, gcc가 아니라면

return false;

}

else if(row > 1 && is\_typeStatement(tokens[row - 1]) == 2){ // 앞 토큰이 gcc라면

if(strcmp(tokens[row - 2], "unsigned") != 0 && strcmp(tokens[row - 2], "extern") != 0) // 앞앞 토큰이 unsigned, extern이 아니라면

return false;

}

}

}

if(row > 0) // 첫번째 토큰이 아니라면

{

if(strcmp(tokens[0], "#include") == 0 || strcmp(tokens[0], "include") == 0 || strcmp(tokens[0], "struct") == 0){ // 첫번째 토큰이 #include, include, struct라면

clear\_tokens(tokens); // 토큰 초기화

strcpy(tokens[0], remove\_extraspace(str)); // 토큰에 공백 제거한 str을 넣는다

}

}

if(is\_typeStatement(tokens[0]) == 2 || strstr(tokens[0], "extern") != NULL){ // 첫번째 토큰이 gcc이거나 extern이라면

for(i = 1; i < TOKEN\_CNT; i++){ // 모든 토큰들 확인

if(strcmp(tokens[i],"") == 0) // 토큰이 null string이라면 반복 종료

break;

if(i != TOKEN\_CNT -1 ) // 마지막 토큰이 아니라면

strcat(tokens[0], " "); // 첫번째 토큰 뒤에 공백 추가

strcat(tokens[0], tokens[i]); // 첫번째 토큰뒤에 tokens[i] 덧붙임

memset(tokens[i], 0, sizeof(tokens[i])); // tokens[i] 0초기화

}

}

while((p\_str = find\_typeSpecifier(tokens)) != -1){ // 토큰들 중 형식 지정자가 있다면

if(!reset\_tokens(p\_str, tokens)) // 토큰들을 정리한다

return false; // 잘못된게 발견되면 false 리턴

}

while((p\_str = find\_typeSpecifier2(tokens)) != -1){ // 토큰들 중 struct 형식 지정자가 있다면

if(!reset\_tokens(p\_str, tokens)) // 토큰들을 정리한다

return false; // 잘못된게 발견되면 false 리턴

}

return true;

}

node \*make\_tree(node \*root, char (\*tokens)[MINLEN], int \*idx, int parentheses) // tokens에 들어있는 토큰들을 트리에 넣는 함수

{

node \*cur = root;

node \*new;

node \*saved\_operator;

node \*operator;

int fstart;

int i;

while(1)

{

if(strcmp(tokens[\*idx], "") == 0) // 토큰이 null string이라면

break;

if(!strcmp(tokens[\*idx], ")")) // 토큰이 ) 라면

return get\_root(cur);

else if(!strcmp(tokens[\*idx], ",")) // 토큰이 , 라면

return get\_root(cur);

else if(!strcmp(tokens[\*idx], "(")) // 토큰이 ( 라면

{

if(\*idx > 0 && !is\_operator(tokens[\*idx - 1]) && strcmp(tokens[\*idx - 1], ",") != 0){ // 이전 토큰이 연산자가 아니고, ','도 아니라면

fstart = true;

while(1)

{

\*idx += 1; // 인덱스 증가

if(!strcmp(tokens[\*idx], ")")) // 토큰이 ) 라면

break; // 반복 종료

new = make\_tree(NULL, tokens, idx, parentheses + 1); // 재귀 호출

if(new != NULL){ // 위의 결과가 NULL이 아니라면

if(fstart == true){

cur->child\_head = new; // 새로 만든 트리를 자식노드로 넣음

new->parent = cur;

fstart = false; // 새 트리 생성 종료

}

else{

cur->next = new; // 형제 노드에 새 트리 추가

new->prev = cur;

}

cur = new; // 새로 만든 트리로 이동

}

if(!strcmp(tokens[\*idx], ")")) // 토큰이 ) 라면

break; // 반복 종료

}

}

else{ // 이전 토큰이 연산자 이거나 ',' 라면

\*idx += 1; // 인덱스 증가

new = make\_tree(NULL, tokens, idx, parentheses + 1); // 트리 생성 재귀 호출

if(cur == NULL)

cur = new;

else if(!strcmp(new->name, cur->name)){ // new의 name과 cur의 name이 같다면

if(!strcmp(new->name, "|") || !strcmp(new->name, "||")

|| !strcmp(new->name, "&") || !strcmp(new->name, "&&"))// 이 노드가 |, ||, &, && 연산자라면

{

cur = get\_last\_child(cur); // 제일 끝에 있는 자식노드로 이동

if(new->child\_head != NULL){ // 새로운 트리에 자식 노드가 존재한다면

new = new->child\_head; // 그 자식 노드로 이동

// 새로 만든 트리에서 root 노드를 제거

new->parent->child\_head = NULL;

new->parent = NULL;

// 새로 생성된 트리의 앞쪽에 cur를 추가함

new->prev = cur;

cur->next = new;

}

}

else if(!strcmp(new->name, "+") || !strcmp(new->name, "\*")) // 이 노드가 +, \* 연산자라면

{

i = 0;

while(1)

{

if(!strcmp(tokens[\*idx + i], "")) // 마지막 토큰이라면 break

break;

if(is\_operator(tokens[\*idx + i]) && strcmp(tokens[\*idx + i], ")") != 0) // 토큰이 연산자이고, )가 아니라면 break

break;

i++; // 다음 토큰으로 이동

}

if(get\_precedence(tokens[\*idx + i]) < get\_precedence(new->name)) // 다음 토큰의 우선순위가 new 보다 더 낮다면

{

cur = get\_last\_child(cur); // 현재 노드의 가장 마지막 자식 노드 찾아온다

cur->next = new; // 가장 마지막 자식 노드의 다음 형제 노드에 new를 추가

new->prev = cur;

cur = new; // new로 이동

}

else // 다음 토큰의 우선순위다 new 보다 더 높다면

{

cur = get\_last\_child(cur); // cur의 가장 마지막 자식 노드 찾아온다

if(new->child\_head != NULL){ // new에 자식 노드가 있다면

new = new->child\_head; // new의 루트 노드 제거

// cur의 맨 마지막 자식 노드로 new 를 넣는다

new->parent->child\_head = NULL;

new->parent = NULL;

new->prev = cur;

cur->next = new;

}

}

}

else{

cur = get\_last\_child(cur); // cur의 가장 마지막 자식 노드 찾아온다

cur->next = new; // cur의 다음 노드에 new를 추가한다

new->prev = cur;

cur = new;

}

}

else

{

cur = get\_last\_child(cur); // cur의 가장 마지막 자식 노드 찾아온다

cur->next = new; // cur의 다음 노드에 new를 추가한다

new->prev = cur;

cur = new;

}

}

}

else if(is\_operator(tokens[\*idx])) // 토큰이 연산자라면

{

if(!strcmp(tokens[\*idx], "||") || !strcmp(tokens[\*idx], "&&")

|| !strcmp(tokens[\*idx], "|") || !strcmp(tokens[\*idx], "&")

|| !strcmp(tokens[\*idx], "+") || !strcmp(tokens[\*idx], "\*")) // 토큰이 ||, &&, |, &, +, \* 연산자라면

{

if(is\_operator(cur->name) == true && !strcmp(cur->name, tokens[\*idx])) // cur가 연산자이고, 현재 토큰과 동일하다면

operator = cur;

else

{

new = create\_node(tokens[\*idx], parentheses); // 새로운 노드 생성

operator = get\_most\_high\_precedence\_node(cur, new); // 가장 우선순위가 높은 연산자를 찾는다

if(operator->parent == NULL && operator->prev == NULL){ // 가장 우선순위가 높은 연산자 노드가 부모가 없고, 앞에 다른 형제도 없다면

if(get\_precedence(operator->name) < get\_precedence(new->name)){ // 새로 만든 트리의 연산자 우선순위가 더 높다면

cur = insert\_node(operator, new); // operator 노드 자리에 new를 삽입

}

else if(get\_precedence(operator->name) > get\_precedence(new->name)) // operator 노드의 우선순위가 더 높다면

{

if(operator->child\_head != NULL){ // operator 노드가 자식 노드를 갖고 있다면

operator = get\_last\_child(operator); // 가장 마지막 자식 노드를 구한다

cur = insert\_node(operator, new); // 가장 마지막 자식 노드 자리에 new를 삽입

}

}

else

{

operator = cur;

while(1)

{

if(is\_operator(operator->name) == true && !strcmp(operator->name, tokens[\*idx])) // operator가 연산자이고, 현재 토큰과 같다면

break; // 반복 종료

if(operator->prev != NULL) // operator 앞에 다른 형제 노드가 있다면

operator = operator->prev; // 앞쪽 형제 노드로 이동

else // 앞에 다른 형제 노드가 없다면

break; // 반복 종료

}

if(strcmp(operator->name, tokens[\*idx]) != 0) // operator의 토큰과 일치하지 않는다면

operator = operator->parent; // operator의 부모 노드로 이동

if(operator != NULL){ // operator가 NULL이 아니라면

if(!strcmp(operator->name, tokens[\*idx])) // operator의 토큰과 일치한다면

cur = operator;

}

}

}

else

cur = insert\_node(operator, new); // operator 위치에 새로운 노드 삽입

}

}

else

{

new = create\_node(tokens[\*idx], parentheses); // 새로운 노드 생성

if(cur == NULL)

cur = new;

else

{

operator = get\_most\_high\_precedence\_node(cur, new); // 가장 우선순위가 높은 연산자를 찾는다

if(operator->parentheses > new->parentheses) //

cur = insert\_node(operator, new); // operator자리에 새로운 노드 삽입

else if(operator->parent == NULL && operator->prev == NULL){ // operator가 루트노드이면

if(get\_precedence(operator->name) > get\_precedence(new->name)) // operator가 new보다 연산자 우선순위가 높으면

{

if(operator->child\_head != NULL){ // operator에 자식 노드가 있다면

operator = get\_last\_child(operator); // operator의 마지막 자식 노드를 구한다

cur = insert\_node(operator, new); // 마지막 자식노드 위치에 새 노드를 삽입

}

}

else

cur = insert\_node(operator, new); // operator자리에 새 노드를 삽입

}

else

cur = insert\_node(operator, new); // operator자리에 새 노드를 삽입

}

}

}

else

{

new = create\_node(tokens[\*idx], parentheses); // 새로운 노드 생성

if(cur == NULL)

cur = new;

else if(cur->child\_head == NULL){ // cur 노드에 자식 노드가 없다면

cur->child\_head = new; // new를 자식 노드로 넣는다

new->parent = cur;

cur = new; // 새 노드로 이동

}

else{

cur = get\_last\_child(cur); // cur의 마지막 자식 노드로 이동

cur->next = new; // cur의 마지막 자식 노드의 형제 노드로 새 노드를 넣는다

new->prev = cur;

cur = new; // 새 노드로 이동

}

}

\*idx += 1; // 인덱스 증가시킴

}

return get\_root(cur); // 트리의 루트노드 구해서 리턴

}

node \*change\_sibling(node \*parent) // 전달인자로 받은 노드의 자식 노드들이 저장된 순서를 바꾸는 함수

{

node \*tmp;

tmp = parent->child\_head;

// 두번째 자식 노드를 첫번째 자식 노드 위치로 옮김

parent->child\_head = parent->child\_head->next;

parent->child\_head->parent = parent;

parent->child\_head->prev = NULL;

// 첫번째 자식 노드를 두번째 자식 노드 위치로 옮김

parent->child\_head->next = tmp;

parent->child\_head->next->prev = parent->child\_head;

parent->child\_head->next->next = NULL;

parent->child\_head->next->parent = NULL;

return parent;

}

node \*create\_node(char \*name, int parentheses) // 새로운 노드를 생성하는 함수

{

node \*new;

new = (node \*)malloc(sizeof(node)); // node 동적 할당

new->name = (char \*)malloc(sizeof(char) \* (strlen(name) + 1)); // node의 name 동적할당

strcpy(new->name, name); // node에 name을 넣는다

// node의 내용 초기화

new->parentheses = parentheses;

new->parent = NULL;

new->child\_head = NULL;

new->prev = NULL;

new->next = NULL;

return new; // 생성된 노드 리턴

}

int get\_precedence(char \*op) // 연산자의 우선순위를 구하는 함수

{

int i;

for(i = 2; i < OPERATOR\_CNT; i++){

if(!strcmp(operators[i].operator, op)) // 전달인자와 같은 연산자를 찾는다

return operators[i].precedence; // 해당 연산자의 우선순위를 리턴한다

}

return false; // 전달인자가 연산자가 아니었다면 false를 리턴한다

}

int is\_operator(char \*op) // 전달인자로 받은 문자가 연산자인지 확인하는 함수

{

int i;

for(i = 0; i < OPERATOR\_CNT; i++)

{

if(operators[i].operator == NULL) // 모든 문자열들과 비교했다면

break; // 반복 중지

if(!strcmp(operators[i].operator, op)){ // 전달인자와 일치하는 연산자가 있다면

return true; // true 리턴

}

}

return false;// 전달인자와 일치하는 연산자가 없었다면 false 리턴

}

void print(node \*cur) // 노드 정보 출력

{

if(cur->child\_head != NULL){ // 자식노드가 있다면

print(cur->child\_head); // 자식노드 출력

printf("\n");

}

if(cur->next != NULL){ // 형제노드가 있다면

print(cur->next); // 형제노드 출력

printf("\t");

}

printf("%s", cur->name); // 갖고있는 토큰 출력

}

node \*get\_operator(node \*cur) // 해당 노드에 대한 연산자 찾는 함수

{

if(cur == NULL)

return cur;

if(cur->prev != NULL) // 앞쪽에 형제노드가 있다면

while(cur->prev != NULL) // 더이상 앞에 형제노드가 없을 때까지

cur = cur->prev; // 앞으로 이동

return cur->parent; // 부모노드(연산자) 리턴

}

node \*get\_root(node \*cur) // 트리의 루트를 찾아 리턴하는 함수

{

if(cur == NULL)

return cur;

while(cur->prev != NULL) // 제일 앞쪽 노드로 이동 (제일 앞 노드에서 부모에게 갈 수 있음)

cur = cur->prev;

if(cur->parent != NULL) // 부모 노드로 이동

cur = get\_root(cur->parent);

return cur; // 루트노드 리턴

}

node \*get\_high\_precedence\_node(node \*cur, node \*new) // 더 우선순위가 높은 노드를 찾는 함수

{

if(is\_operator(cur->name))

if(get\_precedence(cur->name) < get\_precedence(new->name)) // cur의 연산자가 우선순위가 더 높다면

return cur; // cur 리턴

if(cur->prev != NULL){ // cur 앞에 형제노드가 있다면

while(cur->prev != NULL){ // 맨 앞의 형제 노드까지 이동

cur = cur->prev;

return get\_high\_precedence\_node(cur, new); // cur와 비교하며 재귀호출

}

if(cur->parent != NULL) // 부모노드가 있다면

return get\_high\_precedence\_node(cur->parent, new); // 부모노드와 비교하며 재귀호출

}

if(cur->parent == NULL) // 루트 노드라면

return cur; // cur 리턴

}

node \*get\_most\_high\_precedence\_node(node \*cur, node \*new) // 가장 우선순위가 높은 연산자를 구하는 함수

{

node \*operator = get\_high\_precedence\_node(cur, new);

node \*saved\_operator = operator;

while(1)

{

if(saved\_operator->parent == NULL)

break;

if(saved\_operator->prev != NULL) // 저장해둔 노드의 앞에 다른 형제노드들이 있다면

operator = get\_high\_precedence\_node(saved\_operator->prev, new); // 앞쪽의 형제노드와 새 노드 중 우선순위가 높은 노드 저장

else if(saved\_operator->parent != NULL) // 저장해둔 노드에게 부모노드가 있다면

operator = get\_high\_precedence\_node(saved\_operator->parent, new); // 부모노드와 새 노드 중 우선순위가 높은 노드 저장

saved\_operator = operator;

}

return saved\_operator; // 가장 연산자 우선순위 높은 노드 리턴

}

node \*insert\_node(node \*old, node \*new) // 새로운 노드를 old의 자리에 삽입하는 함수

{

if(old->prev != NULL){ // old노드의 앞쪽에 다른 형제노드가 있다면

// new 노드의 앞쪽에 그 형제노드를 넣는다

new->prev = old->prev;

// 그 형제노드의 뒤쪽에 new 노드를 넣는다

old->prev->next = new;

old->prev = NULL;

}

new->child\_head = old;// old노드를 new 노드의 자식으로 넣는다

old->parent = new; // old의 부모 노드의 자식노드를 old노드에서 new 노드로 바꾼다

return new;

}

node \*get\_last\_child(node \*cur) // 제일 끝의 자식 노드 찾는 함수

{

if(cur->child\_head != NULL) // 자식노드가 있다면

cur = cur->child\_head; // 자식노드로 이동

while(cur->next != NULL) // 다음 형제노드가 있다면

cur = cur->next; // 다음 형제노드로 이동

return cur; // 찾은 노드 리턴

}

int get\_sibling\_cnt(node \*cur) // 형제 노드의 개수를 세는 함수

{

int i = 0;

while(cur->prev != NULL) // 제일 앞쪽 형제 노드로 이동

cur = cur->prev;

while(cur->next != NULL){ // 뒤쪽으로 이동하며 형제 노드의 수를 센다

cur = cur->next;

i++;

}

return i; // 형제 노드의 수 리턴

}

void free\_node(node \*cur) // 인자로 받은 노드 삭제하는 함수 (자식 노드, 뒤에 있는 형제노드들도 삭제)

{

if(cur->child\_head != NULL) // 자식 노드가 있었다면

free\_node(cur->child\_head); // 자식 노드에 대하여 free\_node() 호출

if(cur->next != NULL) // 뒤에 다른 형제노드가 있다면

free\_node(cur->next); // 뒤에있던 형제노드들에 대하여 free\_node() 호출

if(cur != NULL){

// 현재 노드를 free한다

cur->prev = NULL;

cur->next = NULL;

cur->parent = NULL;

cur->child\_head = NULL;

free(cur);

}

}

int is\_character(char c) // c가 알파벳, 숫자인지 체크하는 함수

{

return (c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'); // 알파벳 대소문자 또는 숫자로 된 문자라면 1 리턴

}

int is\_typeStatement(char \*str) // type이 맨 앞에 있는 구문(선언문?)인지 확인하는 함수

{

char \*start;

char str2[BUFLEN] = {0};

char tmp[BUFLEN] = {0};

char tmp2[BUFLEN] = {0};

int i;

start = str;

strncpy(str2,str,strlen(str)); // str2에 str 복사

remove\_space(str2); // str2에 있는 공백문자들 제거

while(start[0] == ' ') // start 앞에 있는 공백문자들 제거

start += 1;

if(strstr(str2, "gcc") != NULL) // str2에 "gcc"라는 문자열이 포함되어 있다면

{

strncpy(tmp2, start, strlen("gcc")); // tmp2에 start의 앞에서 세문자(strlen("gcc")) 복사

if(strcmp(tmp2,"gcc") != 0) // tmp2(start의 맨 앞 문자 세개)가 gcc가 아니라면

return 0; // 0 리턴

else // gcc라면

return 2; // 2 리턴

}

for(i = 0; i < DATATYPE\_SIZE; i++)

{

if(strstr(str2,datatype[i]) != NULL) // str2에 특정 데이터 타입이 포함되어 있다면

{

strncpy(tmp, str2, strlen(datatype[i])); // str2의 맨 앞에 있는 단어 tmp로 복사

strncpy(tmp2, start, strlen(datatype[i])); // start의 맨 앞에 있는 단어 tmp로 복사

if(strcmp(tmp, datatype[i]) == 0) // 위에서 복사한 단어가 해당 데이터 타입 이라면

if(strcmp(tmp, tmp2) != 0) // tmp와 tmp2가 같다면

return 0; // 0 리턴

else // tmp와 tmp2가 다르다면

return 2; // 2 리턴

}

}

return 1; // 위의 것들 중 아무것도 아니라면 1 리턴

}

int find\_typeSpecifier(char tokens[TOKEN\_CNT][MINLEN]) // 토큰들 중에서 형식 지정자를 찾는 함수

{

int i, j;

for(i = 0; i < TOKEN\_CNT; i++) // 모든 토큰들 확인

{

for(j = 0; j < DATATYPE\_SIZE; j++) // 모든 자료형 확인

{

if(strstr(tokens[i], datatype[j]) != NULL && i > 0) // 토큰에 datatype이 포함되어 있다면

{

if(!strcmp(tokens[i - 1], "(") && !strcmp(tokens[i + 1], ")") // 앞 토큰이 ( 이고, 뒤 토큰이 ) 이고

&& (tokens[i + 2][0] == '&' || tokens[i + 2][0] == '\*'

|| tokens[i + 2][0] == ')' || tokens[i + 2][0] == '('

|| tokens[i + 2][0] == '-' || tokens[i + 2][0] == '+'

|| is\_character(tokens[i + 2][0]))) // 이 토큰 뒤에 있는 ) 바로 뒤에 &,\*,),(,-,+,알파벳이나 숫자가 있으면

return i; // 토큰의 인덱스 리턴

}

}

}

return -1; // 못찾았으면 -1 리턴

}

int find\_typeSpecifier2(char tokens[TOKEN\_CNT][MINLEN]) // 토큰에서 struct 형식 지정자를 찾는 함수

{

int i, j;

for(i = 0; i < TOKEN\_CNT; i++) // 모든 토큰들 확인

{

for(j = 0; j < DATATYPE\_SIZE; j++) // 모든 자료형 확인

{

if(!strcmp(tokens[i], "struct") && (i+1) <= TOKEN\_CNT && is\_character(tokens[i + 1][strlen(tokens[i + 1]) - 1])) // 토큰이 struct이고, 다음 토큰의 마지막 문자가 알파벳이나 숫자라면

return i; // 토큰의 인덱스 리턴

}

}

return -1; // 찾지 못했다면 -1 리턴

}

int all\_star(char \*str) // 이 문자열이 전부 '\*' 문자로 이루어졌는지 확인하는 함수

{

int i;

int length= strlen(str);

if(length == 0)

return 0;

for(i = 0; i < length; i++)

if(str[i] != '\*')

return 0; // 이 문자열에 하나라도 '\*'이 아닌 문자가 포함되어 있다면 0 리턴

return 1; // 문자열이 모두 '\*' 문자로 이루어져 있다면 1 리턴

}

int all\_character(char \*str) // 인자로 받은 문자열이 전부 숫자나 영어 알파벳으로만 이루어져 있는지 확인하는 함수

{

int i;

for(i = 0; i < strlen(str); i++)

if(is\_character(str[i])) // 문자가 숫자나 알파벳으로 이루어져 있다면

return 1; // 바로 1 리턴 -> ???

return 0;

}

int reset\_tokens(int start, char tokens[TOKEN\_CNT][MINLEN])

{

int i;

int j = start - 1;

int lcount = 0, rcount = 0; // lcount : start 토큰 왼쪽의 여는 괄호 개수, rcount : 오른쪽의 닫는 괄호 개수

int sub\_lcount = 0, sub\_rcount = 0;

if(start > -1){ // start가 음수가 아니라면

if(!strcmp(tokens[start], "struct")) { // start 토큰이 struct 라면

strcat(tokens[start], " "); // start 토큰(형식지정자) 뒤에 공백 추가

strcat(tokens[start], tokens[start+1]); // start 토큰 뒤에 start + 1 토큰 덧붙임

for(i = start + 1; i < TOKEN\_CNT - 1; i++){ // start 토큰(형식지정자) 다음 토큰부터 마지막 토큰까지

// 토큰들 앞으로 한칸씩 이동

strcpy(tokens[i], tokens[i + 1]);

memset(tokens[i + 1], 0, sizeof(tokens[0]));

}

}

else if(!strcmp(tokens[start], "unsigned") && strcmp(tokens[start+1], ")") != 0) { // start 토큰이 unsigned이고, 그 다음 토큰이 가 아니라면

strcat(tokens[start], " "); // start 토큰 뒤에 형식 지정자 추가

strcat(tokens[start], tokens[start + 1]); // 다음 토큰 start 토큰에 덧붙임

strcat(tokens[start], tokens[start + 2]); // 다다음 토큰 strat 토큰에 덧붙임

for(i = start + 1; i < TOKEN\_CNT - 1; i++){ // start 다음 토큰부터 마지막 토큰까지 ---

// 토큰들 앞으로 한칸씩 이동

strcpy(tokens[i], tokens[i + 1]);

memset(tokens[i + 1], 0, sizeof(tokens[0]));

}

}

j = start + 1; // j에 start 다음 토큰의 인덱스를 넣는다

while(!strcmp(tokens[j], ")")){ // j번째 토큰이 ) 라면

rcount ++; // 오른쪽 ) 괄호 개수 + 1

if(j==TOKEN\_CNT) // 토큰 끝까지 전부 확인했다면

break; // 반복 종료

j++; // 다음 토큰으로

}

j = start - 1; // j에 start 이전 토큰의 인덱스를 넣는다

while(!strcmp(tokens[j], "(")){ // j번째 토큰이 ( 라면

lcount ++;// 왼쪽 ( 괄호 개수 + 1

if(j == 0) // 모든 토큰 확인했다면

break; // 반복 종료

j--; // 다음 토큰으로

}

if( (j!=0 && is\_character(tokens[j][strlen(tokens[j])-1]) ) || j==0) // j가 첫번째 토큰이 아니고 마지막 문자가 알파벳이나 숫자이거나 또는 첫번째 토큰이라면

lcount = rcount; // lcount에 rcount를 넣는다

if(lcount != rcount ) // 괄호의 짝이 맞지 않는다면

return false;

if( (start - lcount) >0 && !strcmp(tokens[start - lcount - 1], "sizeof")){ // start 토큰 왼쪽에 ( 가 아닌 문자가 있고, 그 (들 바로 앞의 토큰이 sizeof라면

return true; // true 리턴

}

else if((!strcmp(tokens[start], "unsigned") || !strcmp(tokens[start], "struct")) && strcmp(tokens[start+1], ")")) { //start 뒤의 토큰이 )이고, start토큰이 unsigned 또는 struct이면

strcat(tokens[start - lcount], tokens[start]); // start - lcount 번째 토큰에 start 토큰을 덧붙임 ('(' 괄호들 삭제)

strcat(tokens[start - lcount], tokens[start + 1]); // start - lcount 번째 토큰에 start + 1 번째 토큰을 덧붙임

strcpy(tokens[start - lcount + 1], tokens[start + rcount]); // start - lount 번째 토큰의 다음 토큰에 start + rcount번째 토큰을 덧붙임

for(int i = start - lcount + 1; i < TOKEN\_CNT - lcount -rcount; i++) {

strcpy(tokens[i], tokens[i + lcount + rcount]); // i + lcount + rcount 번째 토큰부터 제일 끝에 있는 토큰까지 i번째 토큰으로 앞당긴다

memset(tokens[i + lcount + rcount], 0, sizeof(tokens[0])); // 앞으로 당겨온 토큰들 0초기화

}

}

else{

if(tokens[start + 2][0] == '('){ // start 다다음 토큰이 ( 라면

j = start + 2;

while(!strcmp(tokens[j], "(")){ // (가 아닌 토큰이 나올때까지 반복한다

sub\_lcount++; // sub\_lcount 하나씩 증가시킨다

j++;

}

if(!strcmp(tokens[j + 1],")")){ // j + 1번째 토큰이 )라면

j = j + 1;

while(!strcmp(tokens[j], ")")){ // )가 아닌 토큰이 나올때까지 반복한더

sub\_rcount++; // sub\_rcount를 증가시킨다

j++;

}

}

else

return false;

if(sub\_lcount != sub\_rcount) // sub\_lcount와 sub\_rcount가 다르다면

return false;

strcpy(tokens[start + 2], tokens[start + 2 + sub\_lcount]); // 토큰 앞으로 당겨온다

for(int i = start + 3; i<TOKEN\_CNT; i++)

memset(tokens[i], 0, sizeof(tokens[0])); // 뒤에 있던 토큰들 0초기화

}

strcat(tokens[start - lcount], tokens[start]); // 왼쪽에 있던 괄호 앞 토큰에 start 토큰 덧붙임

strcat(tokens[start - lcount], tokens[start + 1]); // start 토큰의 바로 뒤에 있던 토큰 덧붙임

strcat(tokens[start - lcount], tokens[start + rcount + 1]); // 오른쪽에 있던 괄호 뒤의 토큰 덧붙임

for(int i = start - lcount + 1; i < TOKEN\_CNT - lcount -rcount -1; i++) { // start 토큰 주변에 있던 괄호들을 없앤다

strcpy(tokens[i], tokens[i + lcount + rcount +1]);

memset(tokens[i + lcount + rcount + 1], 0, sizeof(tokens[0]));

}

}

}

return true;

}

void clear\_tokens(char tokens[TOKEN\_CNT][MINLEN]) // 전달인자로 받은 token 배열을 0으로 초기화하는 함수

{

int i;

for(i = 0; i < TOKEN\_CNT; i++)

memset(tokens[i], 0, sizeof(tokens[i]));

}

char \*rtrim(char \*\_str) // 문자열 오른쪽의 white space 제거

{

char tmp[BUFLEN];

char \*end;

strcpy(tmp, \_str);

end = tmp + strlen(tmp) - 1; // tmp의 마지막 문자 위치를 가리키는 포인터

while(end != \_str && isspace(\*end)) // 문자열 끝(오른쪽)부터 앞쪽으로 이동하며 해당 문자가 white space가 아닐 때까지 이동

--end;

\*(end + 1) = '\0';

\_str = tmp; // 전달인자로 받은 \_str에 tmp를 넣어서

return \_str; // 리턴

}

char \*ltrim(char \*\_str)// 문자열 왼쪽의 white space 제거

{

char \*start = \_str; // \_str의 맨 앞 문자를 가리키는 포인터

while(\*start != '\0' && isspace(\*start)) // 문자열 맨 앞(왼쪽)부터 뒤쪽으로 이동하며 해당 문자가 white space 또는 널문자가 아닐 때까지 이동

++start;

\_str = start;

return \_str;

}

char\* remove\_extraspace(char \*str) // 의미없는 공백 지우는 함수

{

int i;

char \*str2 = (char\*)malloc(sizeof(char) \* BUFLEN); // 결과 저장할 문자열 동적할당

char \*start, \*end;

char temp[BUFLEN] = "";

int position;

if(strstr(str,"include<")!=NULL){ // 전달인자로 받은 문자열에 "include<"가 포함되어 있다면

start = str; // 시작주소 설정

end = strpbrk(str, "<"); // 끝주소 설정

position = end - start;

strncat(temp, str, position); // str에서 position 길이만큼 temp에 갖다붙임

strncat(temp, " ", 1); // temp뒤에 공백 추가

strncat(temp, str + position, strlen(str) - position + 1); // temp 뒤에 str의 나머지 부분 갖다 붙임

str = temp; // str을 temp로 바꿈

}

for(i = 0; i < strlen(str); i++) // 문자열 내 모든 문자 확인

{

if(str[i] ==' ') // 공백 문자라면

{

if(i == 0 && str[0] ==' ') // 첫번째 문자가 공백문자라면

while(str[i + 1] == ' ') // 공백문자가 아닌 문자가 나올때까지

i++; // 다음 문자로 이동

else{

if(i > 0 && str[i - 1] != ' ') // 앞문자가 공백문자가 아니라면

str2[strlen(str2)] = str[i]; // str2에 넣는다

while(str[i + 1] == ' ')// 공백문자가 아닌 문자가 나올때까지

i++; // 다음 문자로 이동

}

}

else

str2[strlen(str2)] = str[i]; // str2에 넣는다

}

return str2; // 쓸데없는 공백들 제거한 새로운 문자열 return

}

void remove\_space(char \*str) // 문자열 내에 있는 공백문자 지우는 함수

{

char\* i = str;

char\* j = str;

while(\*j != 0)

{

\*i = \*j++;

if(\*i != ' ')

i++;

}

\*i = 0;

}

int check\_brackets(char \*str) // 괄호가 짝을 맞춰 제대로 있는지 검사하는 함수

{

char \*start = str;

int lcount = 0, rcount = 0; // lcount는 여는 괄호 개수, rcount는 닫는 괄호 개수

while(1){

if((start = strpbrk(start, "()")) != NULL){ // '(',')'이 있는 위치 리턴

if(\*(start) == '(') // 여는 괄호라면

lcount++;

else // 닫는 괄호라면

rcount++;

start += 1; // 뒤쪽 문자열 검사를 위해 + 1

}

else // 괄호 더이상 찾지 못했다면 반복 종료

break;

}

if(lcount != rcount) // 여는 괄호와 닫는 괄호의 개수가 다르면 잘못된 것

return 0; // 괄호가 잘못됐으면 0 리턴

else

return 1; // 괄호 개수가 일치하면 1 리턴

}

int get\_token\_cnt(char tokens[TOKEN\_CNT][MINLEN]) // 토큰의 총 개수 리턴하는 함수

{

int i;

for(i = 0; i < TOKEN\_CNT; i++)

if(!strcmp(tokens[i], "")) // 토큰이 nullstring이라면

break; // 반복 종료

return i; // 반복한 횟수 리턴

}

**수정이 필요한 함수:**

void print\_usage() -> 새로운 옵션 사용법 추가, 사용하지 않는 옵션 사용법 삭제

int execute\_program(char \*id, char \*filename) -> 답안 파일 위치 수정, 실행 결과 저장 위치 수정, 채점 중 에러 메세지가 stdout으로 출력되지 않도록 수정

double compile\_program(char \*id, char \*filename) -> 답안 파일 위치 수정

int score\_blank(char \*id, char \*filename) -> 답안 파일 위치 수정

double score\_student(int fd, char \*id) -> 항상 채점 결과가 출력되도록 수정

void make\_scoreTable(char \*ansDir) -> score\_table.csv 가 저장되어 있는 위치 수정

void set\_scoreTable(char \*ansDir) -> score\_table.csv의 저장 위치 수정

int check\_option(int argc, char \*argv[]) -> -m 옵션과 -i 옵션도 체크하도록 수정하고, 각 옵션 수행 전에 사전 세팅 하도록 수정

void ssu\_score(int argc, char \*argv[]) -> m 옵션과 i 옵션을 수행할 수 있도록 수정

**추가해야 할 함수:**

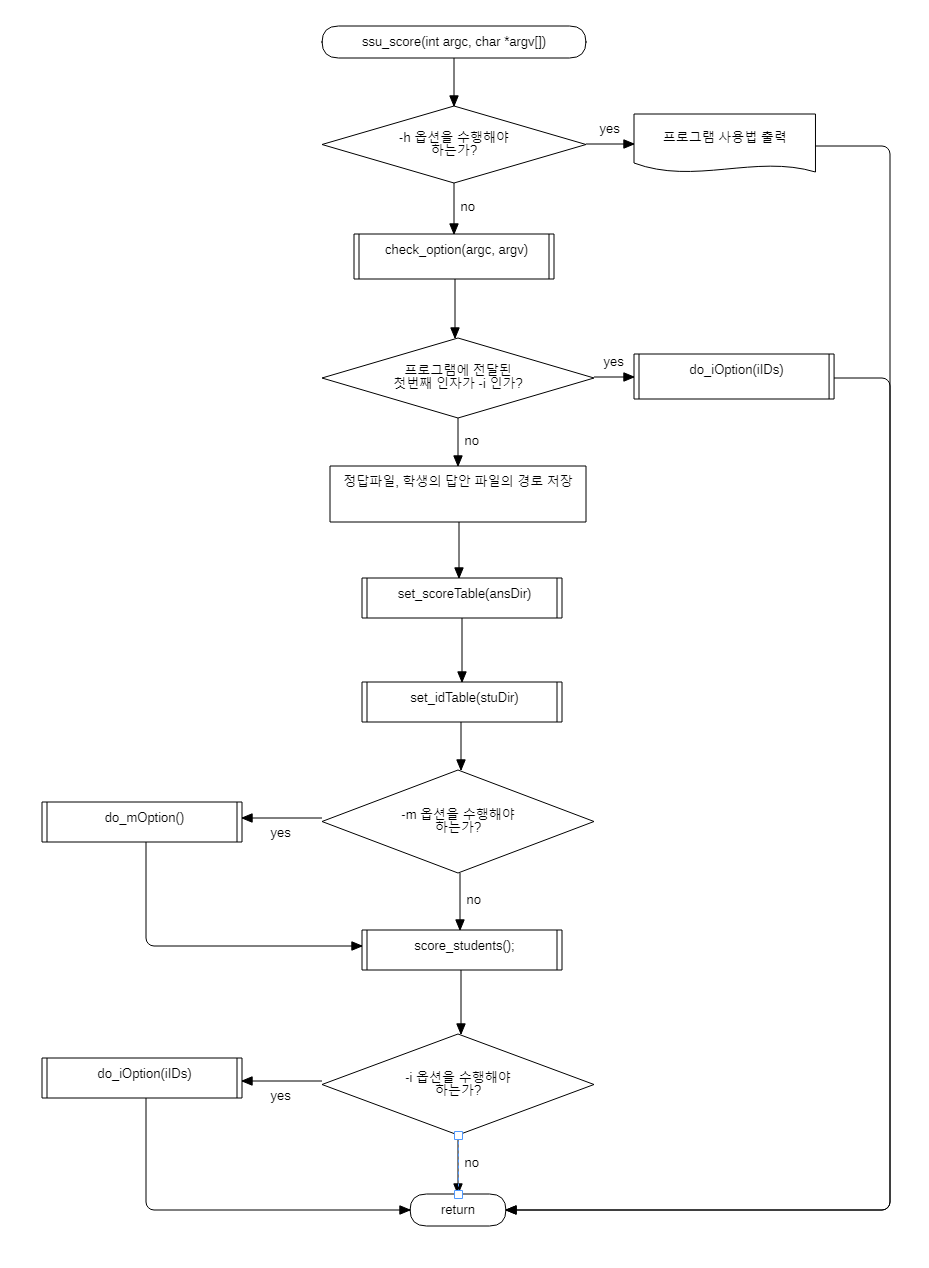
void do\_mOption() -> -m 옵션을 수행하는 기능 추가

void do\_iOption(char (\*ids)[FILELEN]) -> -i 옵션을 수행하는 기능 추가

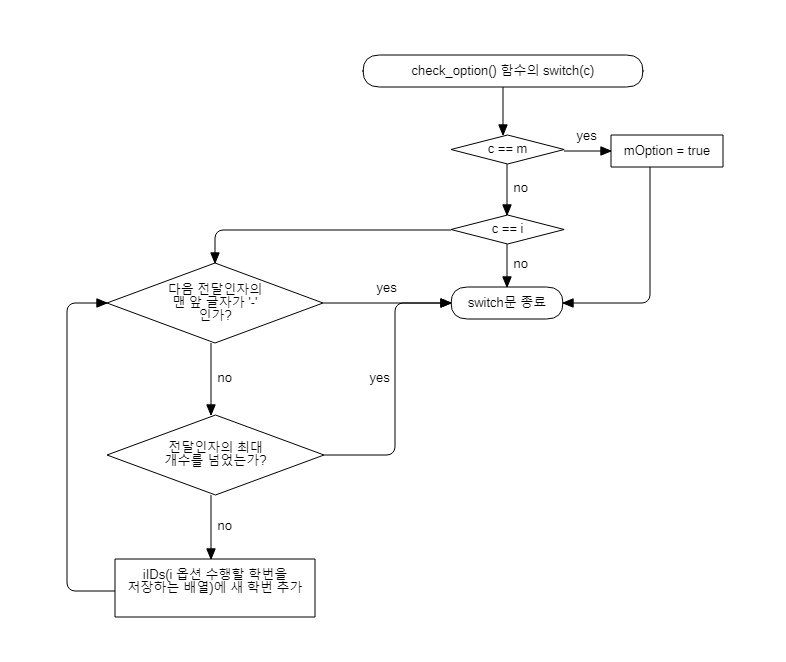
1. **설계**

코드 수정으로 인해 흐름이 변경된 함수, 새롭게 추가된 함수의 흐름도입니다.

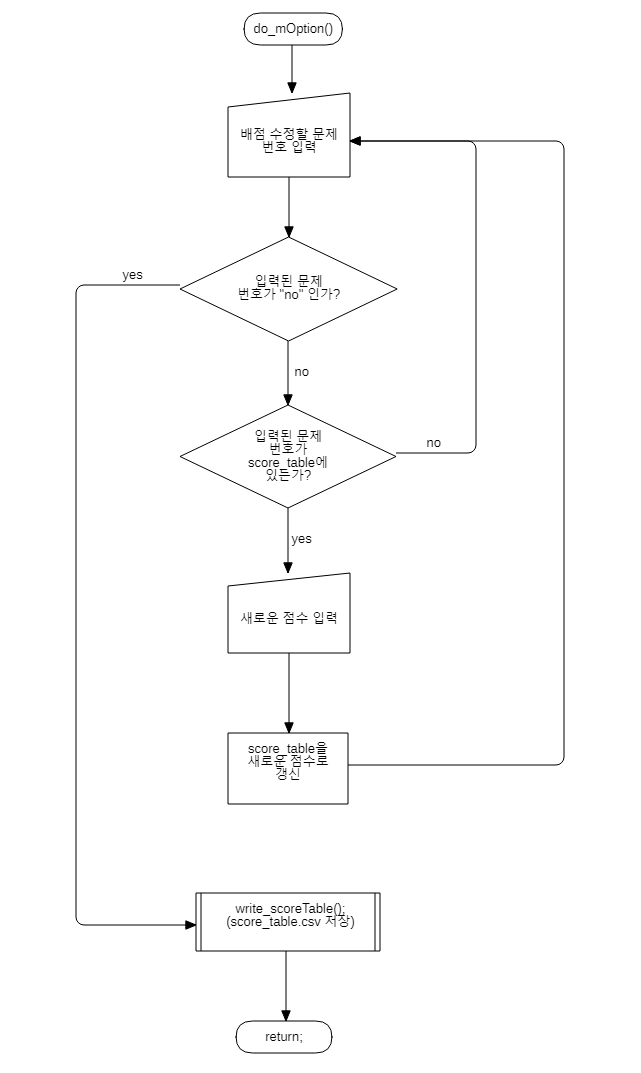
**void ssu\_score(int argc, char \*argv[])**

****

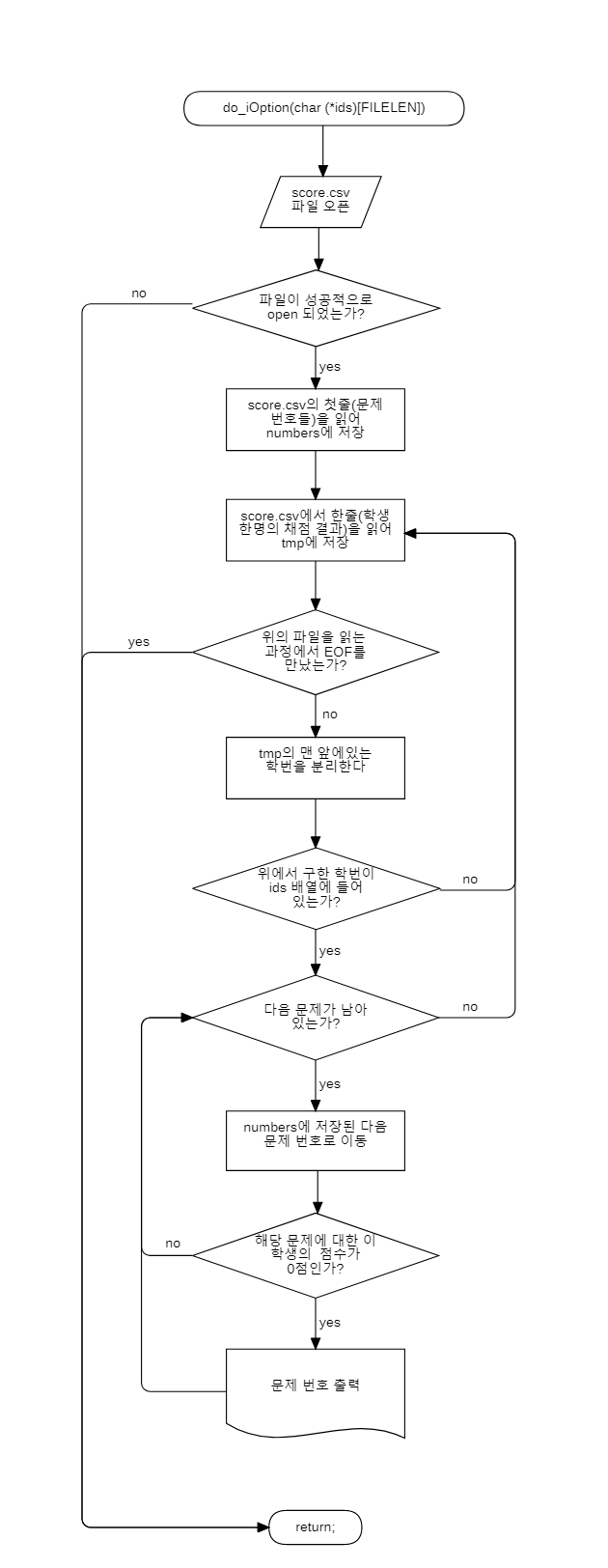
**int check\_option(int argc, char \*argv[]) 내부 switch문의 case ‘m’, case ‘i’**

****

**void do\_mOption();**



**void do\_iOption(char (\*ids)[FILELEN]);**



1. **구현**

**void do\_mOption();**

함수 명: do\_mOption

전달 인자: 없음

리턴 값: 없음

함수 설명: -m 옵션을 수행하는 함수. -m 옵션은 채점 전에 원하는 문제의 점수를 수정하는 옵션이다. 점수를 수정할 문제의 번호를 사용자로부터 입력 받는다. 입력받은 문제를 score\_table 구조체 배열에서 찾아 문제의 배점을 수정한다. 위 과정을 사용자가 문제 번호로 no를 입력할 때까지 반복한다. 반복하면서 수정되는 내용들은 score\_table에 계속해서 기록한다. 사용자가 no를 입력해 수정이 종료되면 write\_scoreTable() 함수를 호출하여 수정된 score\_table을 이용하여 새로운 score\_table.csv 파일을 생성한다.

**void do\_iOption(char (\*ids)[FILELEN]);**

함수 명: do\_iOption

전달 인자: 틀린 문제 파일명을 출력할 학생들의 id(학번)가 들어있는 배열 char (\*ids)[FILELEN]

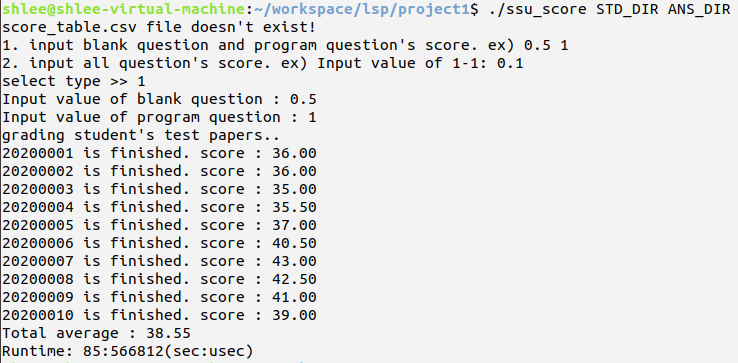
리턴 값: 없음

함수 설명:

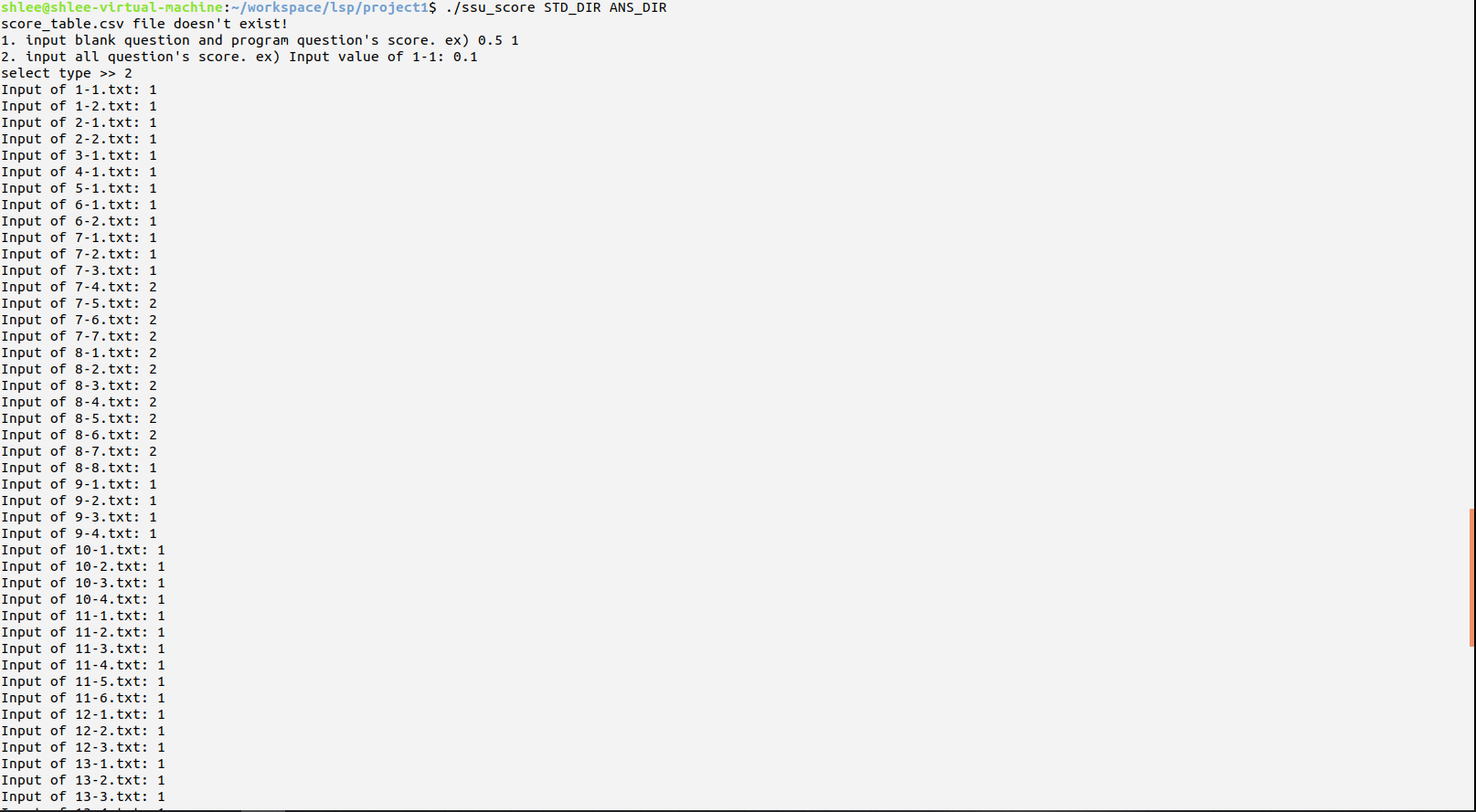
-i 옵션을 수행하는 함수. -i 옵션은 채점결과 파일이 있는 경우 해당 학생들의 틀린 문제 파일을 출력하는 옵션이다. 이 함수는 틀린 문제 파일을 출력할 학생들의 id(학번)가 담긴 배열을 인자로 받는다. 인자로 받은 학번이 담긴 배열과 채점 결과 파일(score.csv)을 이용하여 학생들이 틀린 문제 파일들을 출력한다.

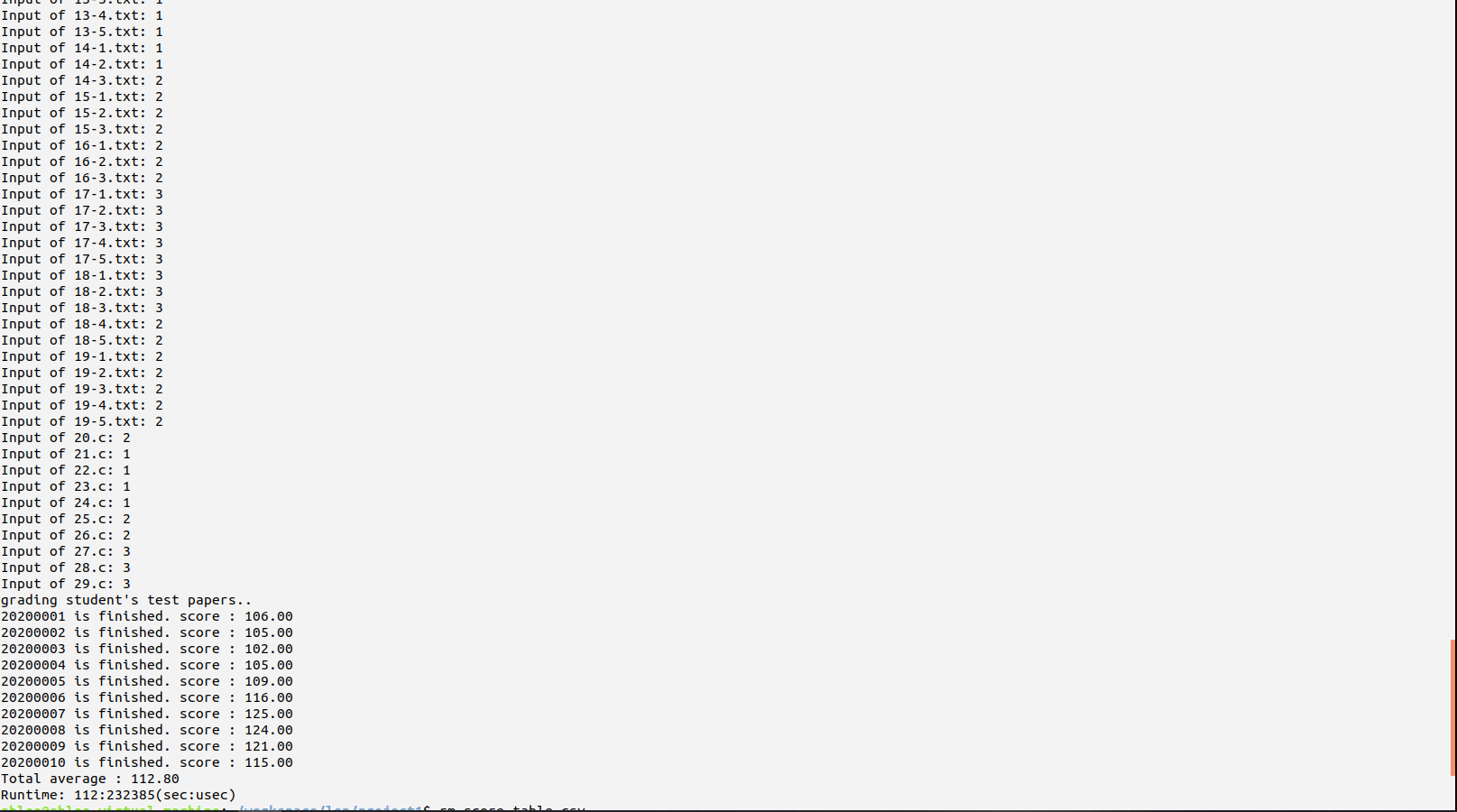
1. **테스트 및 결과**

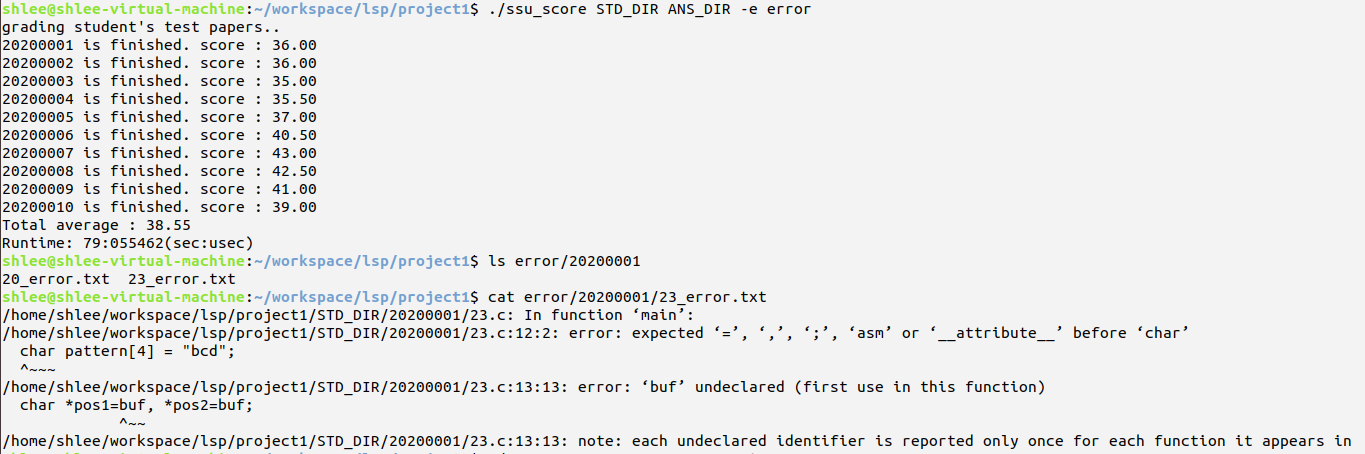
* 그냥 실행, 점수 일괄 입력

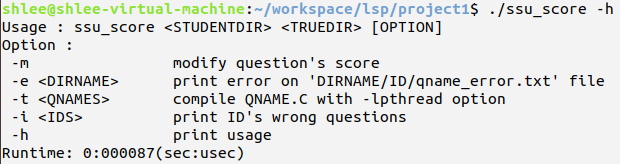


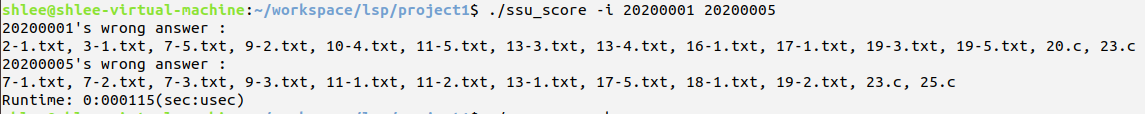
* 그냥 실행, 점수 개별 입력

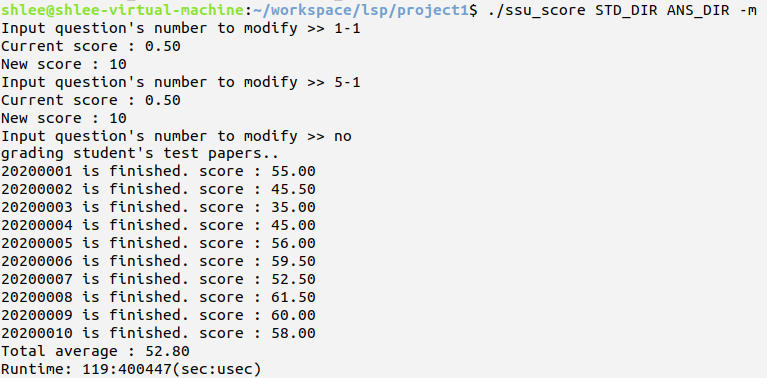




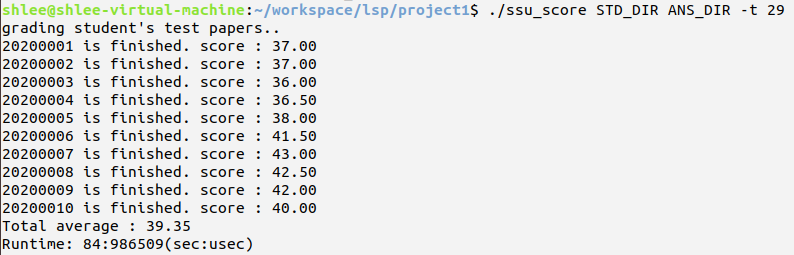
* -e 옵션
* -h 옵션



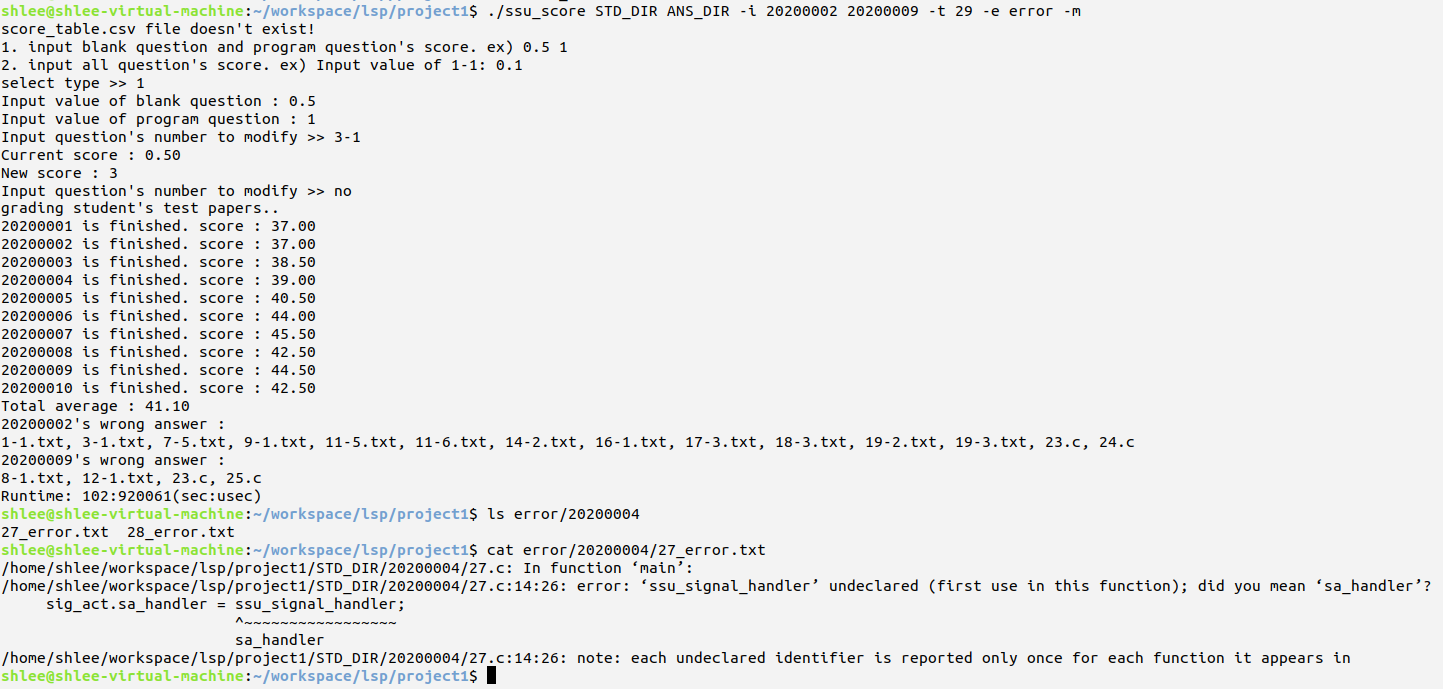
* -i 옵션
* -m 옵션



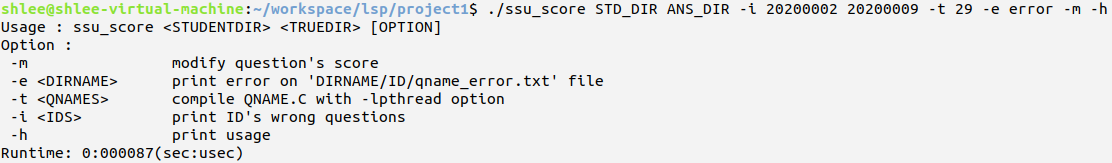
* -t 옵션



* 여러 옵션 1



* 여러 옵션 2



1. **소스코드**

<ssu\_score.c>

* 추가한 함수

**void do\_iOption(char (\*ids)[FILELEN])**

{

FILE \*fp;

char tmp[BUFLEN];

char numbers[BUFLEN];

int i = 0;

char \*p, \*saved, \*np;

int isFirstWrongAnswer = true; // 두번째 오답부터는 앞에 콤마를 찍기 위해 사용하는 플래그

if((fp = fopen("score.csv", "r")) == NULL){ // 점수파일 오픈

fprintf(stderr, "file open error for score.csv\n");

return;

}

fscanf(fp, "%s\n", numbers); // 파일에서 첫번째 줄(문제 번호들) 읽어들임

while(fscanf(fp, "%s\n", tmp) != EOF) // 한 학생씩 채점 결과 읽어들임

{

isFirstWrongAnswer = true;

np = numbers;

p = strtok(tmp, ",");

if(!is\_exist(ids, tmp)) // i 옵션을 지정한 학생중에 현재 읽어들인 학생이 있다면

continue;

// 틀린 문제들 출력

printf("%s's wrong answer : \n", tmp);

while((p = strtok(NULL, ",")) != NULL) {

np = strchr(np, ',') + 1;

if (!strcmp(p, "0")) { // 해당 문제를 틀렸다면

// 문제번호 출력

if(!isFirstWrongAnswer) printf(", ");

else isFirstWrongAnswer = false;

while(\*np != ',') {

printf("%c", \*np);

++np;

}

}

}

printf("\n");

}

fclose(fp);

}

**void do\_mOption()**

{

int i;

double newScore;

char filename[FILELEN];

char qname[FILELEN]; // 문제 번호를 저장할 배열

char inputqname[FILELEN];

while(true) {

printf("Input question's number to modify >> ");

scanf("%s", inputqname); // 수정할 문제 번호 입력받음

if(!strcmp(inputqname, "no")) break; // 입력된 문제 번호가 no라면 수정 종료

i = 0;

while(score\_table[i].score != 0) {

memset(qname, 0, sizeof(qname));

memcpy(qname, score\_table[i].qname, strlen(score\_table[i].qname) - strlen(strrchr(score\_table[i].qname, '.'))); // qname에 확장자 명을 뺀 파일 이름(문제 번호)을 넣음

if(strcmp(qname, inputqname)) { // 문제 번호가 서로 일치하지 않는다면

++i;

continue;

}

// 문제 번호가 서로 일치하면

printf("Current score : %.2f\n", score\_table[i].score);

printf("New score : ");

scanf("%lf", &newScore); // 변경할 배점 입력받음

score\_table[i].score = newScore; // score\_table 구조체 배열에 변경된 점수 기록

break;

}

}

sprintf(filename, "%s", "score\_table.csv"); // 점수 테이블 파일이 생성될 경로를 생성해 filename에 저장

write\_scoreTable(filename); // 변경된 score\_table 구조체 배열의 내용을 score.csv에 출력

}

* 수정된 함수

**void ssu\_score(int argc, char \*argv[])**

{

char saved\_path[BUFLEN];

int i; // for문에 사용할 인덱스 변수

for(i = 0; i < argc; i++){

if(!strcmp(argv[i], "-h")){ // -h 옵션이 적용되어 실행되면

print\_usage(); // 사용법 출력

return; // ssu\_score 종료

}

}

memset(saved\_path, 0, BUFLEN); // saved\_path을 0으로 초기화

if(argc >= 3 && strcmp(argv[1], "-c") != 0){ // \*\*\*\*\*\*\*\*\*\* -c옵션이 무엇인지?

strcpy(stuDir, argv[1]); // <STUDENTDIR> 저장

strcpy(ansDir, argv[2]); // <TRUESETDIR> 저장

}

if(!check\_option(argc, argv))

exit(1);

if(!eOption && !tOption && !mOption && iOption && !strcmp(argv[1], "-i")){ // 채점할 디렉터리 없이 i옵션만 적용된 경우

//do\_cOption(cIDs);

do\_iOption(iIDs);

return; // ssu\_score 종료

}

//////////////////////////////////////////////////////////////////////

getcwd(saved\_path, BUFLEN); // 프로세스의 현재 위치 절대경로 저장

if(chdir(stuDir) < 0){ // cd <STUDENTDIR>, cd 실패했다면

fprintf(stderr, "%s doesn't exist\n", stuDir); //에레메세지 출력

return; // ssu\_score 종료

}

getcwd(stuDir, BUFLEN); // stuDir에 <STUDENTDIR>의 절대경로 저장

chdir(saved\_path); // 다시 프로세스가 실행된 디렉토리로 이동

if(chdir(ansDir) < 0){ // cd <TRUESETDIR>, cd 실패했다면

fprintf(stderr, "%s doesn't exist\n", ansDir); //에러메세지 출력

return; // ssu\_score 종료

}

getcwd(ansDir, BUFLEN); // ansDir에 <TRUESETDIR>의 절대경로 저장

chdir(saved\_path); // 다시 프로세스가 실행된 디렉토리로 이동

set\_scoreTable(ansDir); // 문제별 점수들이 저장될 score\_table 구조체 배열을 setting

set\_idTable(stuDir); // 학생들의 학번이 저장될 id\_table 배열을 setting

//

if(mOption)

do\_mOption();

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

score\_students();

// c옵션 없으므로 수정 필요 //////////////////////////////////////////////////

// if(cOption)

// do\_cOption(cIDs);

//////////////////////////////////////////////////////////////////////////////

if(iOption) // i옵션 수행

do\_iOption(iIDs);

return;

}

**int check\_option(int argc, char \*argv[])**

{

int i, j; // 반복문에서 사용하는 인덱스

int c; // 옵션으로 전달된 알파벳

// 옵션 p, c가 필요 없으므로 수정 필요 /////////////////////////////////////////////////////////////////////////

while((c = getopt(argc, argv, "e:thmi")) != -1)

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

{

switch(c){

case 'e': // 옵션 e

eOption = true;

strcpy(errorDir, optarg); // 옵션에 전달된 인자(에러 메세지가 출력될 디렉토리)를 errorDir에 복사해 놓는다

if(access(errorDir, F\_OK) < 0) // 디렉터리에 접근이 불가능하면

mkdir(errorDir, 0755);// 디렉터리 생성

else{

rmdirs(errorDir); // 기존 디렉터리 제거

mkdir(errorDir, 0755); // 새 디렉터리 생성

}

break;

case 't': // 옵션 t

tOption = true;

i = optind; // 프로그램 전달인자 인덱스

j = 0; // 옵션 가변인자 인덱스

while(i < argc && argv[i][0] != '-'){ // t 옵션에 전달된 가변인자들 확인을 위한 반복문

if(j >= ARGNUM) // 가변인자를 받는 옵션이므로 가변인자의 개수가 최대 개수를 넘지 않았는지 확인한다

printf("Maximum Number of Argument Exceeded. :: %s\n", argv[i]);

else // 옵션에 전달된 인자를 threadFiles에 복사해 놓는다

strcpy(threadFiles[j], argv[i]);

i++;

j++;

}

break;

case 'm':

mOption = true;

case 'i':

iOption = true;

i = optind; // 프로그램 전달인자 인덱스

j = 0; // 옵션에 전달된 가변인자 인덱스

while(i < argc && argv[i][0] != '-'){ // i 옵션에 전달된 가변인자들 확인을 위한 반복문

if(j >= ARGNUM) // 가변인자를 받는 옵션이므로 가변인자의 개수가 최대 개수를 넘지 않았는지 확인

printf("Maximum Number of Argument Exceeded. :: %s\n", argv[i]);

else

strcpy(iIDs[j], argv[i]); // 옵션에 전달된 인자를 iIDs에 복사해 놓는다

i++;

j++;

}

break;

// // 옵션 p - 항상 수행되어야 하므로 수정 필요 ///////////////////////////////////////////

// case 'p':

// pOption = true;

// break;

// ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// 옵션 c - 필요없는 옵션 ///////////////////////////////////////////////////////////////////

// case 'c':

// cOption = true;

// i = optind; // 프로그램 전달인자 인덱스

// j = 0; // 옵션에 전달된 가변인자 인덱스

//

// while(i < argc && argv[i][0] != '-'){ // c 옵션에 전달된 가변인자들 확인을 위한 반복문

//

// if(j >= ARGNUM) // 가변인자를 받는 옵션이므로 가변인자의 개수가 최대 개수를 넘지 않았는지 확인

// printf("Maximum Number of Argument Exceeded. :: %s\n", argv[i]);

// else

// strcpy(cIDs[j], argv[i]); // 옵션에 전달된 인자를 cIDs에 복사해 놓는다

// i++;

// j++;

// }

// break;

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

case '?': // 파라미터가 빠진 채로 옵션이 전달된 경우

printf("Unkown option %c\n", optopt);

return false;

}

}

return true;

}

**void set\_scoreTable(char \*ansDir)** // 각 문제별 점수를 저장해 놓는 score\_table 구조체 배열을 setting하는 함수

{

char filename[FILELEN];

// 점수 테이블 파일은 "./score\_table.csv" 이름으로, 현재 실행 위치에 존재해야 하기 때문에 수정 필요

// sprintf(filename, "%s/%s", ansDir, "score\_table.csv"); // 점수 테이블 파일이 생성될 경로를 생성해 filename에 저장

sprintf(filename, "%s", "score\_table.csv"); // 점수 테이블 파일이 생성될 경로를 생성해 filename에 저장

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

if(access(filename, F\_OK) == 0) // 이미 점수 테이블 파일이 존재한다면

read\_scoreTable(filename); // 기존의 파일에서 문제 번호와 점수들을 불러온다

else{ // 점수 테이블 파일이 존재하지 않는다면

make\_scoreTable(ansDir);

write\_scoreTable(filename);

}

}

**void make\_scoreTable(char \*ansDir)** // score\_table 구조체 배열에 문제 번호와 점수를 저장해 점수 테이블을 만드는 함수

{

int type, num; // type - 파일의 확장자 type을 저장해 놓을 변수

double score, bscore, pscore;

struct dirent \*dirp, \*c\_dirp;

DIR \*dp, \*c\_dp;

char tmp[BUFLEN];

int idx = 0; // 문제 총 개수 저장할 변수

int i;

num = get\_create\_type(); // 사용자에게 점수를 어떤식으로 입력받을 것인지 선택하도록 한다

if(num == 1) // 점수 일괄 입력 선택 시

{

printf("Input value of blank question : ");

scanf("%lf", &bscore); // 빈칸 문제의 점수 입력받음

printf("Input value of program question : ");

scanf("%lf", &pscore); // 프로그램 문제의 점수 입력받음

}

if((dp = opendir(ansDir)) == NULL){ // 디렉터리 open

fprintf(stderr, "open dir error for %s\n", ansDir);

return;

}

while((dirp = readdir(dp)) != NULL)

{

if(!strcmp(dirp->d\_name, ".") || !strcmp(dirp->d\_name, "..")) // 디렉터리 이름이 . 과 .. 이라면 pass

continue;

sprintf(tmp, "%s/%s", ansDir, dirp->d\_name); // 다음에 확인할 디렉터리 경로 tmp에 저장

// if((c\_dp = opendir(tmp)) == NULL){ // 확인 할 디렉터리 open

// fprintf(stderr, "open dir error for %s\n", tmp);

// return;

// }

//

// while((c\_dirp = readdir(c\_dp)) != NULL) // 문제별 디렉터리 확인

// {

// if(!strcmp(c\_dirp->d\_name, ".") || !strcmp(c\_dirp->d\_name, ".."))

// continue;

//

// if((type = get\_file\_type(c\_dirp->d\_name)) < 0) // 파일의 확장자가 .txt or .c 가 아니라면 pass

// continue;

//

// strcpy(score\_table[idx++].qname, c\_dirp->d\_name);

// }

//

// closedir(c\_dp);

// 답안 디렉터리의 바로 아래에 정답 파일들이 들어있도록 바뀌었으므로 이 부분을 수정

if((type = get\_file\_type(dirp->d\_name)) < 0) // 파일의 확장자가 .txt or .c 가 아니라면 pass

continue;

strcpy(score\_table[idx++].qname, dirp->d\_name);

}

closedir(dp);

sort\_scoreTable(idx);

for(i = 0; i < idx; i++) // 모든 문제에 대하여

{

type = get\_file\_type(score\_table[i].qname); // 파일의 확장자가 무엇인지 확인해 type에 저장

if(num == 1) // 점수 일괄 입력 선택 시

{

if(type == TEXTFILE) // .txt 파일이라면 (빈칸문제)

score = bscore;

else if(type == CFILE) // .c 파일이라면 (프로그램 문제)

score = pscore;

}

else if(num == 2) // 점수 각각 입력 선택 시

{

printf("Input of %s: ", score\_table[i].qname); // 점수 입력받을 문제 번호 출력

scanf("%lf", &score); // 해당 문제의 점수 입력받음

}

score\_table[i].score = score; // score\_table 구조체 배열에 문제 점수 저장

}

}

**double score\_student(int fd, char \*id)** // 한 학생에 대하여 채점을 하는 함수, 리턴값은 해당 학생의 총점

{

int type; // 해당 문제가 빈칸 문제인지, 프로그램 문제인지 저장할 변수

double result; // 해당 문제에 대한 정답 여부 또는 감점된 점수를 담을 변수

double score = 0; // 해당 학생의 총점을 저장할 변수

int i;

char tmp[BUFLEN]; // 파일에 write하기 전에 임시로 담아 놓는 배열

int size = sizeof(score\_table) / sizeof(score\_table[0]); // 전체 문항 수

for(i = 0; i < size ; i++) // 전체 문항 수만큼 반복

{

if(score\_table[i].score == 0) // 해당 문제의 배점이 0점이라면 채점 중단

break;

sprintf(tmp, "%s/%s/%s", stuDir, id, score\_table[i].qname); // 해당 학생의 해당 문제 디렉터리로 이동

if(access(tmp, F\_OK) < 0) // 해당 문제 디렉터리에 접근이 불가하다면

result = false;

else

{

if((type = get\_file\_type(score\_table[i].qname)) < 0) // 파일의 확장자 명으로 빈칸 문제인지, 프로그램 문제인지 확인

continue;

if(type == TEXTFILE) // .txt 파일(빈칸 문제)이라면

result = score\_blank(id, score\_table[i].qname); // 빈칸문제 채점

else if(type == CFILE) // .c 파일(프로그램 문제)이라면

result = score\_program(id, score\_table[i].qname); // 프로그램 문제 채점

}

if(result == false) // 해당 학생이 문제를 틀렸을 때

write(fd, "0,", 2); // 0점 부여, score.csv에 해당 문제 0점이라고 출력

else{

if(result == true){ // 해당 학생이 문제를 맞혔을 때

score += score\_table[i].score; // 해당 문제의 배점을 학생의 점수에 더함

sprintf(tmp, "%.2f,", score\_table[i].score); // tmp에 해당 문제에 대하여 학생이 받은 점수를 기록

}

else if(result < 0){ // result 값이 0보다 작다면 감점

score = score + score\_table[i].score + result; // 감점된 점수를 반영하여 학생의 점수에 더함

sprintf(tmp, "%.2f,", score\_table[i].score + result); // tmp에 해당 문제에 대하여 학생이 받은 점수를 기록

}

write(fd, tmp, strlen(tmp)); // score.csv에 tmp에 기록해 놨던 점수 출력

}

}

// 항상 p옵션이 수행되어야 하므로 수정 ///////////////////////////////////////////////////////////////////////

// if(pOption) // p옵션이 설정되어 있다면 -> 항상 수행되도록 해야함

// printf("%s is finished.. score : %.2f\n", id, score); // 해당 학생의 총점 출력

// else

// printf("%s is finished..\n", id);

printf("%s is finished. score : %.2f\n", id, score); // 해당 학생의 총점 출력

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

sprintf(tmp, "%.2f\n", score); // 학생의 총점 tmp에 기록

write(fd, tmp, strlen(tmp)); // tmp에 있는 학생의 총점 score.csv에 출력

return score; // 해당 학생의 총점 return

}

**int score\_blank(char \*id, char \*filename)** // 빈칸 문제를 채점하는 함수, 리턴값은 정답 여부 또는 감점된 점수

{

char tokens[TOKEN\_CNT][MINLEN];

node \*std\_root = NULL, \*ans\_root = NULL;

int idx, start;

char tmp[BUFLEN];

char s\_answer[BUFLEN], a\_answer[BUFLEN];

char qname[FILELEN]; // 문제 번호를 저장할 배열

int fd\_std, fd\_ans; // fd\_std는 학생의 답안파일의 파일디스크립터, fd\_ans는 정답 파일의 파일 디스크립터

int result = true; // 정답인지 오답인지

int has\_semicolon = false; // 학생의 답 맨 끝에 세미콜론이 있었는지 기록해 놓을 변수

memset(qname, 0, sizeof(qname)); // qname 배열 0 초기화

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.'))); // qname에 확장자 명을 뺀 파일 이름(문제 번호)을 넣음

sprintf(tmp, "%s/%s/%s", stuDir, id, filename); // 현재 문제 경로 tmp에 저장

fd\_std = open(tmp, O\_RDONLY); // fd\_std에 학생의 답안 파일 파일디스크립터 저장

strcpy(s\_answer, get\_answer(fd\_std, s\_answer)); // 학생이 제출한 답안 파일에서 답을 읽어와 s\_answer에 저장

if(!strcmp(s\_answer, "")){ // 학생의 답이 비어있다면

close(fd\_std);

return false; // 오답

}

if(!check\_brackets(s\_answer)){ // 여는 괄호, 닫는 괄호의 짝이 맞지 않으면

close(fd\_std);

return false; // 오답

}

strcpy(s\_answer, ltrim(rtrim(s\_answer))); // 학생의 답 앞뒤에 있는 white space를 제거하여 다시 s\_answer에 담는다

if(s\_answer[strlen(s\_answer) - 1] == ';'){ // 학생의 답 제일 뒤에 ;이 있다면

has\_semicolon = true; // 세미콜론이 있었다고 기록하고

s\_answer[strlen(s\_answer) - 1] = '\0'; // 널문자를 넣는다

}

if(!make\_tokens(s\_answer, tokens)){ // 400줄짜리 함수

close(fd\_std);

return false;

}

idx = 0;

std\_root = make\_tree(std\_root, tokens, &idx, 0);

// 답안 디렉터리의 바로 아래에 답안 파일이 있으므로 수정 필요 //////////////////////////////////////////////////////////////////////

// sprintf(tmp, "%s/%s/%s", ansDir, qname, filename);

sprintf(tmp, "%s/%s", ansDir, filename);

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

fd\_ans = open(tmp, O\_RDONLY);

while(1)

{

ans\_root = NULL;

result = true;

for(idx = 0; idx < TOKEN\_CNT; idx++)

memset(tokens[idx], 0, sizeof(tokens[idx]));

strcpy(a\_answer, get\_answer(fd\_ans, a\_answer));

if(!strcmp(a\_answer, ""))

break;

strcpy(a\_answer, ltrim(rtrim(a\_answer)));

if(has\_semicolon == false){

if(a\_answer[strlen(a\_answer) -1] == ';')

continue;

}

else if(has\_semicolon == true)

{

if(a\_answer[strlen(a\_answer) - 1] != ';')

continue;

else

a\_answer[strlen(a\_answer) - 1] = '\0';

}

if(!make\_tokens(a\_answer, tokens))

continue;

idx = 0;

ans\_root = make\_tree(ans\_root, tokens, &idx, 0);

compare\_tree(std\_root, ans\_root, &result);

if(result == true){

close(fd\_std);

close(fd\_ans);

if(std\_root != NULL)

free\_node(std\_root);

if(ans\_root != NULL)

free\_node(ans\_root);

return true;

}

}

close(fd\_std);

close(fd\_ans);

if(std\_root != NULL)

free\_node(std\_root);

if(ans\_root != NULL)

free\_node(ans\_root);

return false;

}

**double compile\_program(char \*id, char \*filename)**

{

int fd;

char tmp\_f[BUFLEN], tmp\_e[BUFLEN];

char command[BUFLEN];

char qname[FILELEN];

int isthread;

off\_t size;

double result;

memset(qname, 0, sizeof(qname)); // qname 0초기화

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.'))); // qname에 확장자명 제외한 파일명 복사

isthread = is\_thread(qname); // -lpthread 옵션으로 컴파일을 할것인지 확인

// 답안 디렉터리 바로 아래에 답안 파일들이 있으므로 수정 필요 //////////////////////////////////////////////////////////////////////////////

// sprintf(tmp\_f, "%s/%s/%s", ansDir, qname, filename); // 컴파일 할 파일 명

// sprintf(tmp\_e, "%s/%s/%s.exe", ansDir, qname, qname); // 컴파일 결과로 나올 실행파일의 파일 명

sprintf(tmp\_f, "%s/%s", ansDir, filename); // 컴파일 할 파일 명

sprintf(tmp\_e, "%s/%s.exe", ansDir, qname); // 컴파일 결과로 나올 실행파일의 파일 명

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

if(tOption && isthread) // -lpthread 옵션으로 컴파일할 파일이라면

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f); // -lpthread 옵션으로 컴파일 할 때 사용할 문자열

else

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f); // 그냥 컴파일 할 때 사용할 문자열

// 답안 디렉터리 바로 아래에 답안 파일들이 있으므로 수정 필요 //////////////////////////////////////////

// sprintf(tmp\_e, "%s/%s/%s\_error.txt", ansDir, qname, qname); // 컴파일 에러를 출력할 파일명 생성

sprintf(tmp\_e, "%s/%s\_error.txt", ansDir, qname); // 컴파일 에러를 출력할 파일명 생성

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

fd = creat(tmp\_e, 0666); // 에러 출력할 파일 생성

redirection(command, fd, STDERR); // command를 실행하고 실행하는 동안 STDERR에 출력될 내용을 fd(에러 출력할 파일)에 출력함

size = lseek(fd, 0, SEEK\_END); // 에러 출력된 파일의 크기 저장

close(fd); // 에러 출력된 파일 close

unlink(tmp\_e); // 에러 출력된 파일 삭제 - 학생 답안 파일이 아닌 정답 파일 컴파일 시 에러 내용이므로 저장할 필요 없음

if(size > 0) // 컴파일 에러 발생했다면

return false; // false 리턴

sprintf(tmp\_f, "%s/%s/%s", stuDir, id, filename); // 컴파일 할 학생 답안 파일

sprintf(tmp\_e, "%s/%s/%s.stdexe", stuDir, id, qname); // 컴파일 결과로 나올 실행파일의 파일 명

if(tOption && isthread) // -lpthread 옵션으로 컴파일할 파일이라면

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f); // -lpthread 옵션으로 컴파일 할 때 사용할 command

else

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f); // 그냥 컴파일 할 때 사용할 command

sprintf(tmp\_f, "%s/%s/%s\_error.txt", stuDir, id, qname); // 컴파일 에러 내용을 저장할 파일 명

fd = creat(tmp\_f, 0666); // 에러 저장할 파일 생성

redirection(command, fd, STDERR); // command를 실행하고 실행하는 동안 STDERR에 출력될 내용을 fd(에러 출력할 파일)에 출력함

size = lseek(fd, 0, SEEK\_END); // 에러 출력된 파일의 크기 저장

close(fd); // 에러 출력된 파일 close

if(size > 0){ // 컴파일 에러 발생했다면

if(eOption) // e 옵션이 지정되어 있다면

{

sprintf(tmp\_e, "%s/%s", errorDir, id); // 에러 파일 저장할 경로

if(access(tmp\_e, F\_OK) < 0) // 해당 경로에 접근 불가하면

mkdir(tmp\_e, 0755); // 디렉터리 생성

sprintf(tmp\_e, "%s/%s/%s\_error.txt", errorDir, id, qname); // 에러 파일명

rename(tmp\_f, tmp\_e); // 위에서 만들어둔 에러파일의 이름을 제대로 된 에러 파일명으로 바꾼다

result = check\_error\_warning(tmp\_e); // 에러 내용을 확인해 결과 점수를 계산한다

}

else{

result = check\_error\_warning(tmp\_f); // 에러 내용을 확인해 결과 점수를 계산한다

unlink(tmp\_f); // 에러내용 저장했던 파일 삭제

}

return result; // 결과 점수 리턴

}

unlink(tmp\_f); // 에러 내용 파일 삭제

return true; // 컴파일 성공 했으므로 true 리턴

}

**int execute\_program(char \*id, char \*filename)**

{

char std\_fname[BUFLEN], ans\_fname[BUFLEN];

char tmp[BUFLEN];

char qname[FILELEN];

time\_t start, end;

pid\_t pid;

int fd;

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 /////////////////////////////////////////////////////////////////////////////////

int tmpSTDERR;

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

memset(qname, 0, sizeof(qname)); // qname 0초기화

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.'))); // qname에 문제 번호 저장

// 답안 디렉터리 하위에 바로 답안 파일들이 있으므로 수정 필요 ///////////////////////////////////////////////////////////////

// sprintf(ans\_fname, "%s/%s/%s.stdout", ansDir, qname, qname); // 정답 실행파일의 실행결과를 저장할 파일

sprintf(ans\_fname, "%s/%s.stdout", ansDir, qname); // 정답 실행파일의 실행결과를 저장할 파일

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

fd = creat(ans\_fname, 0666); // 실행결과 저장파일 생성

// 답안 디렉터리 하위에 바로 답안 파일들이 있으므로 수정 필요 ///////////////////////////////////////////

// sprintf(tmp, "%s/%s/%s.exe", ansDir, qname, qname); // 정답 실행 파일

sprintf(tmp, "%s/%s.exe", ansDir, qname); // 정답 실행 파일

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 ///////////////////////////////////////////////

tmpSTDERR = dup(STDERR);

dup2(fd, STDERR);

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

redirection(tmp, fd, STDOUT); // 정답 실행 파일을 실행시키고 그 결과를 위에서 만든 실행 결과 저장 파일에 저장

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 //////////////////////////////////////////////

dup2(tmpSTDERR, STDERR);

close(tmpSTDERR);

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

close(fd); // 실행 결과 저장 파일 close

sprintf(std\_fname, "%s/%s/%s.stdout", stuDir, id, qname); // 학생의 답안을 실행한 결과를 저장할 파일

fd = creat(std\_fname, 0666); // 학생 답안 실행결과 저장파일 생성

sprintf(tmp, "%s/%s/%s.stdexe &", stuDir, id, qname); // 학생 답안 실행 파일

start = time(NULL); // 학생 답안 실행 시작시간 저장

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 //////////////////////////////////////////////

tmpSTDERR = dup(STDERR);

dup2(fd, STDERR);

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

redirection(tmp, fd, STDOUT);// 학생 답안을 실행하고 결과를 저장

// 정답 파일 실행 중 발생한 에러가 출력되지 않기 위해 수정 ////////////////////////////////////////////////

dup2(tmpSTDERR, STDERR);

close(tmpSTDERR);

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

sprintf(tmp, "%s.stdexe", qname); // 학생 답안 실행파일 파일명

while((pid = inBackground(tmp)) > 0){ // 학생 답안 실행파일이 실행되는 동안 반복하며 체크

end = time(NULL); // 얼마동안 실행중인지 저장

if(difftime(end, start) > OVER){ // 실행 제한 시간(5초)을 초과했다면

kill(pid, SIGKILL); // 종료시킴

close(fd);

return false; // false 리턴

}

}

close(fd);

return compare\_resultfile(std\_fname, ans\_fname); // 정답과 학생 답안의 결과를 비교해 리턴

}

**void print\_usage()** // 프로그램 사용법 출력

{

printf("Usage : ssu\_score <STUDENTDIR> <TRUEDIR> [OPTION]\n");

printf("Option : \n");

// m 옵션 추가 /////////////////////////////////////////////////////////////////////////////////////

printf(" -m modify question's score\n");

/////////////////////////////////////////////////////////////////////////////////////////////////////////

printf(" -e <DIRNAME> print error on 'DIRNAME/ID/qname\_error.txt' file \n");

printf(" -t <QNAMES> compile QNAME.C with -lpthread option\n");

// i 옵션 추가 /////////////////////////////////////////////////////////////////////////////////////

printf(" -i <IDS> print ID's wrong questions\n");

/////////////////////////////////////////////////////////////////////////////////////////////////////////

printf(" -h print usage\n");

// 항상 p옵션이 수행되어야 하므로 수정 필요 ////////////////////////////////////////////////////////////

//printf(" -p print student's score and total average\n");

/////////////////////////////////////////////////////////////////////////////////////////////////////////

// c 옵션은 필요 없음 ///////////////////////////////////////////////////////////////////////////////////

//printf(" -c <IDS> print ID's score\n");

/////////////////////////////////////////////////////////////////////////////////////////////////////////

}