리눅스 시스템 프로그래밍 설계 과제 1 : ssu\_score

20162481 나반

컴퓨터학부 안승훈

비밀번호 : SUZUME

1. 개요

- ssu\_score 프로그램은 정답 파일을 기준으로 학생들이 제출한 답안 파일을 채점하는 프로그램이다.

- 빈칸을 채우는 방식의 문제와 코딩을 통한 프로그램 문제의 채점이 가능하도록 한다.

- 단순하게 채점을 하는 기능 이외에도 다양한 옵션을 제공하도록 한다.

- 옵션 선택을 통해, 틀린 문제 출력, 틀린 이유 저장 등 채점 전후로 필요한 다양한 기능을 제공한다.

- 채점의 용이성을 위해 학생의 점수를 파일로 출력할 수 있도록 하고, 각 문제의 배점 또한 파일에 저장하고, 쉽게 수정 가능하도록 한다.

- 다양한 정답이 가능한 경우, 모든 정답의 경우를 고려할 수 있도록 프로그램을 작성한다.

2. 분석

- 소스코드에 주석을 통해 분석을 진행하였다.

**<main.c>**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/time.h>

#include "ssu\_score.h"

#define SECOND\_TO\_MICRO 1000000 // 1초는 백만마이크로초

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); //시작시간 저장

ssu\_score(argc, argv);

gettimeofday(&end\_t, NULL); //끝시간 저장

ssu\_runtime(&begin\_t, &end\_t); //시간계산, 끝시간 - 처음 시간

exit(0); //프로그램 종료

}

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

{

end\_t->tv\_sec -= begin\_t->tv\_sec; //끝난 시간에서 처음시간을 뺌

if(end\_t->tv\_usec < begin\_t->tv\_usec){

end\_t->tv\_sec--;

end\_t->tv\_usec += SECOND\_TO\_MICRO;

} //나중의 us값이 작을경우 1초를 뺴서 us값이 더해줌

end\_t->tv\_usec -= begin\_t->tv\_usec;

printf("Runtime: %ld:%06ld(sec:usec)\n", end\_t->tv\_sec, end\_t->tv\_usec);

} //프로그램 총 소요시간 계산 후 출력하는 함수

**<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]; //학생정답폴더의 디렉터리를 저장

char ansDir[BUFLEN]; //정답폴더의 디렉터리를 저장

char errorDir[BUFLEN]; //e옵션이 적용 되었을 때, 에러를 저장할 디렉터리 이름을 저장

char threadFiles[ARGNUM][FILELEN]; //t옵션 적용 시, -lpthread 옵션을 통해 컴파일 할 문제 번호를 저장함

char cIDs[ARGNUM][FILELEN]; //옵션의 조건으로 입력되는 데이터를 저장하기 위한 배열

int eOption = false;

int tOption = false;

int iOption = false;

int hOption = false;

int mOption = false;

//각 옵션들을 위한 스위치

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

{

char saved\_path[BUFLEN];

int i, j, log;

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

if(!strcmp(argv[i], "-h")){

print\_usage();

return;

}

} // h옵션이 있을 경우 h옵션을 출력하고 프로그램 종료

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

if(argc >= 3 && (log = strcmp(argv[1], "-i")) != 0){

strcpy(stuDir, argv[1]);

strcpy(ansDir, argv[2]);

} //옵션 i가 아닐경우 분리

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

exit(1);

//입력된 옵션을 체크함

j = 0;

if(!eOption && !tOption && iOption && !log) {

do\_iOption(cIDs);

return;

} //i옵션만 활성화되어있고, 사용자가 문제 디렉터리를 입력하지 않았을 경우

//i옵션과 관련된 함수 실행

getcwd(saved\_path, BUFLEN);

//현재 작업디렉터리의 경로를 얻어옴

if(chdir(stuDir) < 0){

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

return;

} //현재 작업디렉터리 변경

getcwd(stuDir, BUFLEN);

//학생 디렉터리의 정보 저장

chdir(saved\_path); //정답 디렉터리 확인

if(chdir(ansDir) < 0){

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

return;

} //정답디렉터리의 존재 유무 확인

getcwd(ansDir, BUFLEN);

//정답디렉터리 불러옴

chdir(saved\_path);

//정답디렉터리 정보저장

set\_scoreTable(ansDir); //문제의 점수를 저장하는 csv파일을 생성

set\_idTable(stuDir); //학생의 득점을 저장하기 위한 csv 파일 생성

if(mOption) {

do\_mOption();

} //m옵션을 입력하였을 경우, m옵션 함수 실행

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

score\_students(); //학생들의 답안 채점

if(iOption)

do\_iOption(cIDs); //다른 조건들을 실행하고 i옵션이 있을 경우 실행함

return;

}

void do\_mOption(void) {

int i, ft = 0;

char tmp[30];

int cnt;

double ns;

while(1) {

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

scanf("%s", tmp);

if(!strcmp(tmp, "no"))

break;

//no라고 입력한 경우 문제 배점 수정 종료

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

cnt = strncmp(tmp, score\_table[i].qname, strlen(tmp));

if(!cnt) {

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

printf("New score : ");

scanf("%lf", &ns);

score\_table[i].score = ns;

ft = 1;

}

} //문제의 새 배점을 입력받아 출력함

if(!ft) //사용자가 없는 문제 번호를 입력할 경우, 에러 출력 후 새 입력을 대기함

printf("Wrong question number!!\n");

} // break조건이 나올떄 까지 반복하여 입력받음

write\_scoreTable("score\_table.csv"); //바뀐 배점으로 score\_table을 새로 작성

return;

} //m옵션이 주어졌을 경우 문제의 배점을 수정하는 함수

int check\_option(int argc, char \*argv[])

{ //입력된 옵션을 체크 하는 함수

int i, j;

int c;

while((c = getopt(argc, argv, "me:ti:")) != -1)

{ //지정된 옵션이 입력되었는지 확인함

switch(c){

case 'm': //m옵션시 옵션 스위치를 켬

mOption = true;

break;

case 'e': //e옵션이 입렫외었을 경우 지정된 이름의 error 디렉터리를 생성함

eOption = true;

strcpy(errorDir, optarg);

if(access(errorDir, F\_OK) < 0)

mkdir(errorDir, 0755);

else{

rmdirs(errorDir);

mkdir(errorDir, 0755);

}

break;

case 't': //t옵션의 스위치를 켜고 어느 문제를 thread를 이용하여 채점할지 기억함

tOption = true;

i = optind;

j = 0;

while(i < argc && argv[i][0] != '-'){

if(j >= ARGNUM)

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

else

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

i++;

j++;

}

break;

case 'i': //i옵션 인 경우 어느 학생의 틀린 문제를 출력할지 저장함

iOption = true;

i = optind - 1;

j = 0;

while(i < argc && argv[i][0] != '-'){

if(j >= ARGNUM)

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

else

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

i++;

j++;

}

break;

case '?': //그 외의 지정되지 않은 옵션이 들어올 경우 판단불가능을 반환함

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

return false;

}

} //모든 옵션을 판단한 후 while문 종료

return true;

}

void do\_iOption(char (\*ids)[FILELEN])

{ //학생의 틀린 문제 번호를 출력하는 함수 -i 옵션을 위함

FILE \*fp;

char tmp[BUFLEN];

char ques\_num[BUFLEN];

char ques\_num\_backup[BUFLEN];

int i = 0;

char \*p, \*saved, \*qn, \*nextptr;

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

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

return;

} //score.csv를 오픈함, 실패시 에러출력

fscanf(fp, "%s\n", ques\_num);

strcpy(ques\_num\_backup, ques\_num);

//문제번호 줄 읽어옴

//printf("%s\n", ques\_num);

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

{

p = strtok(tmp, ",");

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

continue;

//학생의 학번이 올바른지 판단함

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

strcpy(ques\_num, ques\_num\_backup);

qn = strtok\_r(ques\_num, ",", &nextptr);

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

if(!strcmp(p, "0"))

printf("%s ", qn);

qn = strtok\_r(NULL, ",", &nextptr);

} //학생의 점수가 0인 문제의 반호를 문제의 끝까지 출력함

printf("\n");

}

fclose(fp);

}

int is\_exist(char (\*src)[FILELEN], char \*target)

{ //입력된 학생의 학번이 실제로 목록에 존재하는지 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;

} //파일의 첫 줄부터 돌면서 학생이 존재하는지 확인함

return false; //확인되지 않을 경우 false 반환

}

void set\_scoreTable(char \*ansDir)

{ //문제의 배점을 저장하는 함수

char filename[FILELEN];

sprintf(filename, "./%s", "score\_table.csv"); //I changed

//현재 디렉터리에 문제 배점 저장하는 파일 생성

if(access(filename, F\_OK) == 0)

read\_scoreTable(filename); //배점 파일을 읽어옴

else{

make\_scoreTable(ansDir); //배점 파일을 만듦

write\_scoreTable(filename); //배점 파일을 저장함

}

}

void read\_scoreTable(char \*path)

{ //배점 파일을 읽어오는 함수

FILE \*fp;

char qname[FILELEN];

char score[BUFLEN];

int idx = 0;

if((fp = fopen(path, "r")) == NULL){

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

return ;

} // 배점 파일 열기

while(fscanf(fp, "%[^,],%s\n", qname, score) != EOF){

strcpy(score\_table[idx].qname, qname);

score\_table[idx++].score = atof(score);

} //문제 배점 파일을 한 줄 씩 읽어 문제 번호와 점수를 score\_table 배열에 저장함

fclose(fp);

}

void make\_scoreTable(char \*ansDir)

{ //score\_table.csv 초기에 배점을 사용자에게 입력받는 함수

int type, num;

double score, bscore, pscore;

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

DIR \*dp, \*c\_dp;

char tmp[BUFLEN];

int idx = 0;

int i;

num = get\_create\_type();

//사용자의 메뉴 입력을 받아옴

if(num == 1)

{

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

scanf("%lf", &bscore);

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

scanf("%lf", &pscore);

} //빈칸 문제와 프로그램 문제의 점수를 설정함

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

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

return;

} //dp는 정답 폴더의 디렉터리 포인터

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

{

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

continue;

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

if(get\_file\_type(dirp->d\_name) != -1)

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

}//정답 폴더에 있는 문제 번호를 불러옴, 어떠한 문제 번호들을 분석함

closedir(dp);

sort\_scoreTable(idx); //문제번호 순으로 문제 정렬

for(i = 0; i < idx; i++)

{

type = get\_file\_type(score\_table[i].qname);

//문제 번호를 보고 문제가 텍스트 문제인지 c프로그램 파일인지 구분함

if(num == 1)

{ //사용자가 1번을 선택하였을 경우, 모든 빈칸문제는 bscore로, 모든 프로그램 문제는 pscore로 설정

if(type == TEXTFILE)

score = bscore;

else if(type == CFILE)

score = pscore;

}

else if(num == 2)

{ //사용자가 2번을 선택한 경우, 모든 문제에 대해 입력받음

printf("Input of %s: ", score\_table[i].qname);

scanf("%lf", &score);

}

//입력받은 점수를 score\_table배열에 저장

score\_table[i].score = score;

}

}

void write\_scoreTable(char \*filename)

{ //배열에 저장해 두었던 문제별 배점데이터를 csv 파일에 쓰는 함수

int fd;

char tmp[BUFLEN];

int i;

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

if((fd = creat(filename, 0666)) < 0){

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

return;

} //score\_table 파일을 생성함

for(i = 0; i < num; i++)

{

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

break;

sprintf(tmp, "%s,%.2f\n", score\_table[i].qname, score\_table[i].score);

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

} //파일에 문제 번호와 점수를 저장함

close(fd);

}

void set\_idTable(char \*stuDir)

{ //학생 디렉터리를 읽고 학번을 추출해서 저장하는 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);

} //학생들의 정답이 있는 폴더을 오픈함

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

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

continue;

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

stat(tmp, &statbuf);

if(S\_ISDIR(statbuf.st\_mode))

strcpy(id\_table[num++], dirp->d\_name);

else

continue;

} //폴더안을 돌면서 학생의 학번을 수집하여 배열에 저장함

sort\_idTable(num); // 학번순으로 정렬

}

void sort\_idTable(int size)

{ //불러와 저장한 학번을 정렬하는 함수

int i, j;

char tmp[10];

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)

{ //문제번호 순서대로 문제배점 배열을 정렬하는 함수

int i, j;

struct ssu\_scoreTable tmp;

int num1\_1, num1\_2;

int num2\_1, num2\_2;

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);

get\_qname\_number(score\_table[j+1].qname, &num2\_1, &num2\_2);

//비교를 위한 두개의 문제번호를 불러옴

if((num1\_1 > num2\_1) || ((num1\_1 == num2\_1) && (num1\_2 > num2\_2))){

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]));

} // 문제번호순대로 배열의 원소를 교환함

}

}//버블 정렬을 이용하여 문제번호 순으로 정렬

}

void get\_qname\_number(char \*qname, int \*num1, int \*num2)

{ //베열에서 문제번호를 불러오는 함수

char \*p;

char dup[FILELEN];

strncpy(dup, qname, strlen(qname));

\*num1 = atoi(strtok(dup, "-."));

//문제 번호를 - 앞까지 끊어서 저장

p = strtok(NULL, "-.");

if(p == NULL)

\*num2 = 0;

else

\*num2 = atoi(p);

} // - 이후로도 문제번호가 있을경우 별도로 .txt 또는 .c 앞까지 읽어서 저장

int get\_create\_type()

{ //초기에 문제배점 테이블이 없을때 사용자로부터 입력을 받기 위한 메뉴 함수

int num;

while(1)

{ //1번 모든 빈칸 문제 프로그램 문제를 한번에 설정함, 2번 각각의 문제에 대해 설정함

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

printf("1. input blank question and program question's score. ex) 0.5 1\n");

printf("2. input all question's score. ex) Input value of 1-1: 0.1\n");

printf("select type >> ");

scanf("%d", &num);

if(num != 1 && num != 2)

printf("not correct number!\n"); //1 또는 2가 들어올때 까지 반복

else

break;

}

return num; //사용자의 입력 반환

}

void score\_students()

{ //학생들의 득점을 저장하는 함수

double score = 0;

int num;

int fd;

char tmp[BUFLEN];

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

if((fd = creat("score.csv", 0666)) < 0){

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

return;

} //학생들의 점수를 저장할 파일을 생성함

write\_first\_row(fd); //score.csv에 문제번호 입력

for(num = 0; num < size; num++)

{

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

break;

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

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

//처음에 학생의 학번 입력

score += score\_student(fd, id\_table[num]);

} //학생의 득점을 순차적으로 입력함 , score 변수에 평균 계산을 위해 학생의 총 득점을 더해둠

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

//학생들의 총 평균 출력

close(fd); //학생들의 득점 파일 close

}

double score\_student(int fd, char \*id)

{ //학생들의 점수를 실질적으로 계산하는 함수

int type;

double result;

double score = 0;

int i;

char tmp[BUFLEN];

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

//문제 갯수만큼 for문을 돌리기 위해 문제 갯수 계산

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

{

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

break;

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

//학생들의 정답 위치로 이동

if(access(tmp, F\_OK) < 0)

result = false;

else

{

if((type = get\_file\_type(score\_table[i].qname)) < 0)

continue;

//텍스트파일 또는 프로그램 파일이 아닐 경우 무시

if(type == TEXTFILE)

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

else if(type == CFILE)

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

} //빈칸문제일 경우 score\_blank에서 점수 계산

//프로그램 문제일 경우 score\_program에서 점수 계산

if(result == false)

write(fd, "0,", 2); //틀렸을 경우 0점 입력

else{

if(result == true){

score += score\_table[i].score;

sprintf(tmp, "%.2f,", score\_table[i].score);

}//맞았을 경우 배열에 점수 저장, 학생의 총점 계산을 위해 score에 저장

else if(result < 0){

score = score + score\_table[i].score + result;

sprintf(tmp, "%.2f,", score\_table[i].score + result);

} //result가 0보다 작을 경우 감점도 점수에 더해줌

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

} //계산된 점수를 씀

}

printf("%s is finished.. score : %.2f\n", id, score);

//점수 계산이 끝난 후 학생별 점수 출력

sprintf(tmp, "%.2f\n", score);

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

//학생의 총점을 파일에 쓰고 반환함

return score;

}

void write\_first\_row(int fd)

{ // 학생 배점 파일 첫줄에 문제 번호를 작성하는 함수

int i;

char tmp[BUFLEN];

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

write(fd, ",", 1);

//가장 왼쪽 위의 칸은 빈칸

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

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

break;

//문제의 끝에 도달하면 읽기 종료

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

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

}

write(fd, "sum\n", 4); // 문제 번호를 입력하고 마지막에 sum 입력칸을 만들어줌

}

char \*get\_answer(int fd, char \*result)

{ //답안에서 정답을 뽑아오는 함수

char c;

int idx = 0;

memset(result, 0, BUFLEN);

while(read(fd, &c, 1) > 0)

{

if(c == ':')

break;

//정답을 콜론으로 구분하여 하나씩 추출함

result[idx++] = c;

}

if(result[strlen(result) - 1] == '\n')

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

//개행 문자를 널문자로 교체하여 문장의 끝 표시

return result;

}

int score\_blank(char \*id, char \*filename)

{ //빈칸 문제 채점 함수

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];

int fd\_std, fd\_ans;

int result = true;

int has\_semicolon = false;

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

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.')));

sprintf(tmp, "%s/%s/%s", stuDir, id, filename);

fd\_std = open(tmp, O\_RDONLY);

strcpy(s\_answer, get\_answer(fd\_std, s\_answer));

//학생답을 불러옴

if(!strcmp(s\_answer, "")){

close(fd\_std);

return false;

} //학생 답이 없으면 0점 처리

if(!check\_brackets(s\_answer)){

close(fd\_std);

return false;

} //괄호 갯수가 맞지 않으면 0점 처리

strcpy(s\_answer, ltrim(rtrim(s\_answer)));

//왼쪽 오른쪽 공백 지우기

if(s\_answer[strlen(s\_answer) - 1] == ';'){

has\_semicolon = true; //세미콜론 유무 확인

s\_answer[strlen(s\_answer) - 1] = '\0'; //세미콜론 삭제

}

if(!make\_tokens(s\_answer, tokens)){

close(fd\_std);

return false;

} //토큰 만들기

idx = 0;

std\_root = make\_tree(std\_root, tokens, &idx, 0);

//학생 정답을 이용해서 트리를 생성함

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

fd\_ans = open(tmp, O\_RDONLY); //정답파일 불러오기

while(1)

{

ans\_root = NULL; //정답이 여러개일수 있으니 NULL

result = true;

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

memset(tokens[idx], 0, sizeof(tokens[idx]));

//토큰 초기화

strcpy(a\_answer, get\_answer(fd\_ans, a\_answer));

if(!strcmp(a\_answer, ""))

break;

//정답 파일이 비었을 경우 break

strcpy(a\_answer, ltrim(rtrim(a\_answer)));

//정답의 양쪽 공백 제거

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; //정답 파일에 대한 토큰 생성

idx = 0;

ans\_root = make\_tree(ans\_root, tokens, &idx, 0);

//정답파일의 토큰을 이용해서 트리 생성

compare\_tree(std\_root, ans\_root, &result);

// 학생의 정답 트리와 정답지의 트리를 서로 비교한다

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;

//트리가 일치 할 경우 정답처리

}

}

close(fd\_std);

close(fd\_ans);

if(std\_root != NULL)

free\_node(std\_root);

if(ans\_root != NULL)

free\_node(ans\_root);

return false;

} //트리가 같지 않을 경우 오답 처리

double score\_program(char \*id, char \*filename)

{ //프로그램 문제를 채점하는 함수

double compile;

int result;

compile = compile\_program(id, filename);

//컴파일 한 결과를 받아와서 판단함

if(compile == ERROR || compile == false)

return false;

//컴파일이 안되거나 에러가 나올 경우 오답처리

result = execute\_program(id, filename);

//컴파일이 된 프로그램 문제에 대해 실행 여부 확인

if(!result)

return false;

//결과가 다를 경우 오답 처리

if(compile < 0)

return compile;

//오류 발생시 감점 적용

return true; //맞았을 경우 정답 처리

}

int is\_thread(char \*qname)

{ //t옵션을 적용해야 하는 문제인지 배열에서 검색하는 함수

int i;

int size = sizeof(threadFiles) / sizeof(threadFiles[0]);

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

if(!strcmp(threadFiles[i], qname))

return true;

}

return false;

} //t 옵션으로 지정된 문제 번호에 컴파일시 -lpthread 옵션 추가

double compile\_program(char \*id, char \*filename)

{ //프로그램 문제를 컴파일 하는 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));

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.')));

isthread = is\_thread(qname);

//-t 옵션으로 지정된 문제인지 판단

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

sprintf(tmp\_e, "%s/%s.exe", ansDir, qname);

if(tOption && isthread)

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f);

else

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f);

//t옵션이 적용된 문제의 경우 -lpthread 옵션 추가

sprintf(tmp\_e, "%s/%s/%s\_error.txt", ansDir, qname, qname);

fd = creat(tmp\_e, 0666);

//e옵션이 적용되어있는 경우, 오류메시지를 지정한 디렉터리에 텍스트 파일로 저장함

redirection(command, fd, STDERR);

size = lseek(fd, 0, SEEK\_END);

close(fd);

unlink(tmp\_e);

if(size > 0)

return false;

sprintf(tmp\_f, "%s/%s/%s", stuDir, id, filename);

sprintf(tmp\_e, "%s/%s/%s.stdexe", stuDir, id, qname);

if(tOption && isthread)

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f);

else

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f);

// t옵션의 유무에 따라 컴파일을 다르게 적용함

sprintf(tmp\_f, "%s/%s/%s\_error.txt", stuDir, id, qname);

fd = creat(tmp\_f, 0666);

//에러 파일은 다른 곳로 출력함

redirection(command, fd, STDERR); // STDERR의 fd를 다른 파일로 옮김

size = lseek(fd, 0, SEEK\_END); //파일 끝으로 가서 파일의 사이즈를 판단함

close(fd);

if(size > 0){

if(eOption)

{ //e옵션이 있을 경우 오류메시지를 사용자가 지정한 폴더로 내보냄

sprintf(tmp\_e, "%s/%s", errorDir, id);

if(access(tmp\_e, F\_OK) < 0)

mkdir(tmp\_e, 0755);

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

rename(tmp\_f, tmp\_e);

result = check\_error\_warning(tmp\_e);

}

else{ //e옵션이 없을 경우 에러나 경고의 갯수만을 판단함

result = check\_error\_warning(tmp\_f);

unlink(tmp\_f);

}

return result;

}

unlink(tmp\_f);

return true;

}

double check\_error\_warning(char \*filename)

{ //학생들의 프로그램에서 에러 또는 경고의 갯수를 계산하는 함수

FILE \*fp;

char tmp[BUFLEN];

double warning = 0;

if((fp = fopen(filename, "r")) == NULL){

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

return false;

}

while(fscanf(fp, "%s", tmp) > 0){

if(!strcmp(tmp, "error:"))

return ERROR;

else if(!strcmp(tmp, "warning:"))

warning += WARNING;

}

//warning의 갯수만큼 -0.1점함, error 발생시 0점

return warning;

}

int execute\_program(char \*id, char \*filename)

{ //학생들의 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;

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

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.')));

sprintf(ans\_fname, "%s/%s.stdout", ansDir, qname);

fd = creat(ans\_fname, 0666);

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

redirection(tmp, fd, STDOUT); //프로그램 문제 정답 프로그램의 표준 출력을 임시 파일로 돌림

close(fd);

sprintf(std\_fname, "%s/%s/%s.stdout", stuDir, id, qname);

fd = creat(std\_fname, 0666);

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

start = time(NULL);

redirection(tmp, fd, STDOUT); //학생들의 프로그램 문제의 표준 출력을 임시 파일로 돌림

sprintf(tmp, "%s.stdexe", qname);

while((pid = inBackground(tmp)) > 0){ //백그라운드에서 돌고 있는 학생 프로그램의 프로세스 ID를 가져옴

end = time(NULL);

if(difftime(end, start) > OVER){ //5초이상 소요되었을 경우 프로세스를 KILL하고 0점 부여

kill(pid, SIGKILL);

close(fd);

return false;

}

}

close(fd);

//마지막으로 결과 파일을 비교하여 점수를 부여함

return compare\_resultfile(std\_fname, ans\_fname);

}

pid\_t inBackground(char \*name)

{//백그라운드에서 실행중인 학생 또는 정답 프로그램의 프로세스 ID를 반환함

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);

//백그라운드에서 돌아가는 프로그램의 pid를 파일 하나에 저장해둠

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

redirection(command, fd, STDOUT); //프로그램의 표준 출력을 다른 파일로 변경함

lseek(fd, 0, SEEK\_SET);

read(fd, tmp, sizeof(tmp));

if(!strcmp(tmp, "")){

unlink("background.txt");

close(fd);

return 0;

}

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

close(fd);

unlink("background.txt"); //파일 삭제

return pid;

}

int compare\_resultfile(char \*file1, char \*file2)

{ //결과물 파일 두개를 서로 비교함

int fd1, fd2;

char c1, c2;

int len1, len2;

fd1 = open(file1, O\_RDONLY);

fd2 = open(file2, O\_RDONLY);

while(1)

{

while((len1 = read(fd1, &c1, 1)) > 0){

if(c1 == ' ')

continue;

else

break;

}

while((len2 = read(fd2, &c2, 1)) > 0){

if(c2 == ' ')

continue;

else

break;

} //각 파일에서 한글자씩 읽어옴

if(len1 == 0 && len2 == 0)

break;

//파일이 끝났을 경우 종료

to\_lower\_case(&c1);

to\_lower\_case(&c2);

//각 글자를 소문자로 변환한다

if(c1 != c2){

close(fd1);

close(fd2);

return false;

} //글자가 서로 다를 경우 false를 리턴

}

close(fd1);

close(fd2);

return true;

} //글자가 다른 경우가 없을 경우 파일을 닫고 true를 리턴

void redirection(char \*command, int new, int old)

{ //파일 디스크립터를 덮어썼다가 다시 복구 시켜주는 함수

int saved, saved2;

saved = dup(old);

saved2 = dup(STDERR);

dup2(new, old);

dup2(new, STDERR);

system(command);

dup2(saved, old);

close(saved);

close(saved2);

} // 학생 및 정답 프로그램 파일에서 오류나 출력값이 나올때 화면이 아닌 임의의 파일로 출력하는데 사용

int get\_file\_type(char \*filename)

{

char \*extension = strrchr(filename, '.');

if(!strcmp(extension, ".txt"))

return TEXTFILE;

else if (!strcmp(extension, ".c"))

return CFILE;

else

return -1;

} //파일의 확장자를 통해 텍스트파일과 C파일을 구분함

void rmdirs(const char \*path)

{ //입력된 디렉터리를 삭제하는데에 사용하는 함수

struct dirent \*dirp;

struct stat statbuf;

DIR \*dp;

char tmp[1024];

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

return;

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

{

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

continue;

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

if(lstat(tmp, &statbuf) == -1)

continue;

if(S\_ISDIR(statbuf.st\_mode))

rmdirs(tmp);

else

unlink(tmp);

} //디렉터리를 읽어서 내부에 디렉터리가 존재할 경우 디렉터리를 삭제함

closedir(dp);

rmdir(path);

}

void to\_lower\_case(char \*c)

{ //모두 소문자로 전환함

if(\*c >= 'A' && \*c <= 'Z')

\*c = \*c + 32;

}

void print\_usage() //-h 옵션일때 사용 방법 출력

{

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

printf("Option : \n");

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

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

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

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

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

}

**<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"};

//연산자 우선순위 저장

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}

};

void compare\_tree(node \*root1, node \*root2, int \*result)

{ // 정답과 학생의 답으로 만든 트리를 서로 비교하는 함수

node \*tmp;

int cnt1, cnt2;

if(root1 == NULL || root2 == NULL){

\*result = false;

return;

} //트리가 없으면 끝

if(!strcmp(root1->name, "<") || !strcmp(root1->name, ">") || !strcmp(root1->name, "<=") || !strcmp(root1->name, ">=")){

if(strcmp(root1->name, root2->name) != 0){

if(!strncmp(root2->name, "<", 1))

strncpy(root2->name, ">", 1);

else if(!strncmp(root2->name, ">", 1))

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

else if(!strncmp(root2->name, "<=", 2))

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

else if(!strncmp(root2->name, ">=", 2))

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

root2 = change\_sibling(root2);

}

} //첫 노드가 부등호일 경우 두 트리의 부등호 뱡향을 통일 시키기 위해 자식 노드를 교환함

if(strcmp(root1->name, root2->name) != 0){

\*result = false;

return;

} //트리 시작이 다르면 false 리턴

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

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

\*result = false;

return;

} //자식 노드의 시작이 달라도 false 리턴

else if(root1->child\_head != NULL){

if(get\_sibling\_cnt(root1->child\_head) != get\_sibling\_cnt(root2->child\_head)){

\*result = false;

return;

} //자식 노드의 갯수가 다르면 false 리턴

if(!strcmp(root1->name, "==") || !strcmp(root1->name, "!="))

{// 트리의 시작이 비교 연산자일 경우

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

//루트 노드의 자식 노드를 비교함

if(\*result == false)

{

\*result = true;

root2 = change\_sibling(root2);

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

} //false가 나왔을 경우 자식 노드의 순서를 바꿔서 다시 확인함

}

else if(!strcmp(root1->name, "+") || !strcmp(root1->name, "\*")

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

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

{//트리의 시작이 + \* | || & && 일 경우

if(get\_sibling\_cnt(root1->child\_head) != get\_sibling\_cnt(root2->child\_head)){

\*result = false;

return;

} //자식 노드 갯수가 다르면 false

tmp = root2->child\_head;

while(tmp->prev != NULL)

tmp = tmp->prev;

//마지막 자식노드로 이동

while(tmp != NULL)

{ // 학생 트리와 정답 트리의 같은 깊이의 자식들을 서로 비교함

compare\_tree(root1->child\_head, tmp, result);

if(\*result == true)

break;

else{

if(tmp->next != NULL)

\*result = true;

tmp = tmp->next;

}

}

}

else{ //루트 다음 단계의 노드들끼리의 비교를 시작함

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

}

}

if(root1->next != NULL){

//최상위 노드의 형제 노드가 있을 경우

if(get\_sibling\_cnt(root1) != get\_sibling\_cnt(root2)){

\*result = false;

return;

} //형제 노드의 갯수를 비교하고 갯수가 다를 경우 false 리턴

if(\*result == true)

{

tmp = get\_operator(root1);

//최상위 노드의 연산자를 가져옴

if(!strcmp(tmp->name, "+") || !strcmp(tmp->name, "\*")

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

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

{ //최상위 노드의 연산자가 위의 종류중에 있을경우

tmp = root2;

while(tmp->prev != NULL)

tmp = tmp->prev;

//형제 노드의 마지막으로 이동

while(tmp != NULL)

{

compare\_tree(root1->next, tmp, result);

if(\*result == true)

break;

else{

if(tmp->next != NULL)

\*result = true;

tmp = tmp->next;

}

} // 그 노드의 형제 노드들을 비교함

}

else //또는 그 다음 단계의 노드를 비교하기 시작함

compare\_tree(root1->next, root2->next, result);

}

}

}

int make\_tokens(char \*str, char tokens[TOKEN\_CNT][MINLEN])

{ //학생의 답과 빈칸의 정답을 토큰으로 분해하는 함수

char \*start, \*end;

char tmp[BUFLEN];

char str2[BUFLEN];

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

int row = 0;

int i;

int isPointer;

int lcount, rcount;

int p\_str;

clear\_tokens(tokens);

//시작전 토큰 NULL로 초기화

start = str;

if(is\_typeStatement(str) == 0)

return false;

//명령어는 빼버림

while(1)

{

if((end = strpbrk(start, op)) == NULL)

break;

//더이상 연산자가 없는 경우 종료

if(start == end){

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

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

return false;

//문자열의 시작이 ++, --, ++++, ----일 경우 false 반환

if(is\_character(\*ltrim(start + 2))){ //왼쪽 공백을 지운것이 문자일 경우

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

return false;

//

end = strpbrk(start + 2, op); //연산자가 나오는 앞까지 end로 저장

if(end == NULL) //문자열에 연산자가 없는경우

end = &str[strlen(str)];

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++;

}

}

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

if(strstr(tokens[row - 1], "++") != NULL || strstr(tokens[row - 1], "--") != NULL)

return false;

//row열에 있는 토큰이 문자일 경우

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

strncpy(tmp, start, 2);

strcat(tokens[row - 1], tmp);// 토큰으로 만들어서 저장

start += 2;

row--;

}

else{

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

strncpy(tmp, start, 2);

strcat(tokens[row], tmp);

start += 2;

}

}

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)){

//문자열이 위의 연산자와 같은경우

strncpy(tokens[row], start, 2);

start += 2;

}//토큰으로 저장하고 2칸 전진

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

{//멤버 포인터 연산자일 경우

end = strpbrk(start + 2, op);

//연산자가 나올때까지 끊어서 end에 저장함

if(end == NULL)

end = &str[strlen(str)];

while(start < end){

if(\*start != ' ')

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

start++;

}

row--;

}

else if(\*end == '&')

{ //'&'연산자 일때

if(row == 0 || (strpbrk(tokens[row - 1], op) != NULL)){

end = strpbrk(start + 1, op);

if(end == NULL)

end = &str[strlen(str)];

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

start++;

//start를 토큰으로 저장

while(start < end){

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

return false;

else if(\*start != ' ')

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

start++;

}//직전에 만든 토큰이 &가 아닐때 새로 토큰을 만들어 저장

}

else{

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

start += 1;

}

}

else if(\*end == '\*')

{ //end가 \*를 가리킬 떄

isPointer=0;

if(row > 0)

{

for(i = 0; i < DATATYPE\_SIZE; i++) { //DS는 35

if(strstr(tokens[row - 1], datatype[i]) != NULL){

strcat(tokens[row - 1], "\*");

start += 1;

isPointer = 1;

break;

}//datatype과 같지 않으면 새 토큰으로 \* 추가

}

if(isPointer == 1)

continue;

if(\*(start+1) !=0)

end = start + 1;

if(row>1 && !strcmp(tokens[row - 2], "\*") && (all\_star(tokens[row - 1]) == 1)){

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

row--;

}//토큰에 start의 길이만큼 복사해 넣음

else if(is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1]) == 1){

strncat(tokens[row], start, end - start);

}//전 토큰의 마지막이 문자일때, 새 토큰으로 하나 생성

else if(strpbrk(tokens[row - 1], op) != NULL){

strncat(tokens[row] , start, end - start);

}// 전 토큰에 연산자 전까지 NULL이 아니라면 새 토큰 생성

else

strncat(tokens[row], start, end - start);

//새 토큰 생성

start += (end - start);

} //문자열 길이만큼 진행

else if(row == 0)

{ //토큰이 하나도 없을 때

if((end = strpbrk(start + 1, op)) == NULL){

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

start += 1;

} // 문자열에 연산자 전으로 아무것도 없을 때, 한바이트 만큼 토큰 생성

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++;

}//공백이 아닐 경우 문자 하나만큼 토큰으로 만듬

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

row--;

//토큰이 \*문자일 경우 row 하나 감소

}

}

}

else if(\*end == '(')

{ //토큰의 시작이 '('일경우

lcount = 0;

rcount = 0;

if(row>0 && (strcmp(tokens[row - 1],"&") == 0 || strcmp(tokens[row - 1], "\*") == 0)){

while(\*(end + lcount + 1) == '(')

lcount++;

start += lcount;

end = strpbrk(start + 1, ")");

//닫는 괄호 나올때까지 end에 저장

if(end == NULL)

return false; //닫는 괄호 없으면 문제 있는거

else{

while(\*(end + rcount +1) == ')')

rcount++;

end += rcount;

if(lcount != rcount)

return false;

//여는 괄호와 닫는 괄호의 갯수가 달라도 문제 있는 것

if( (row > 1 && !is\_character(tokens[row - 2][strlen(tokens[row - 2]) - 1])) || row == 1){

strncat(tokens[row - 1], start + 1, end - start - rcount - 1);

row--;

start = end + 1;

}//1번 줄이거나 row가 1보다 클때, 전 토큰의 마지막이 문자일떄

else{

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

start += 1;

}//새 토큰 생성

}

}

else{

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

start += 1;

}

}

else if(\*end == '\"')

{//문자열이 큰 따옴표일 때

end = strpbrk(start + 1, "\"");

//다시 큰따옴표가 나올때까지 end에 저장

if(end == NULL)

return false;

//큰따옴표가 없으면 문제 있는 것

else{

strncat(tokens[row], start, end - start + 1);

start = end + 1;

} //따옴표를 제외한 부분을 새 토큰으로 생성

}

else{ //그 외의 문자열일때

if(row > 0 && !strcmp(tokens[row - 1], "++"))

return false;

//전의 토큰이 ++ 이였으면 false

if(row > 0 && !strcmp(tokens[row - 1], "--"))

return false;

//전의 토큰이 --였으면 false

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

start += 1;

//start에서 토큰으로 한 문자를 불러옴

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

//현재 토큰이 -, +, --, ++일경우

if(row == 0)

row--;

else if(!is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1])){

if(strstr(tokens[row - 1], "++") == NULL && strstr(tokens[row - 1], "--") == NULL)

row--;

}//토큰이 ++ 또는 --가 아닌 경우 전의 토큰으로 올라감

}

}

}

else{ //start == end가 아닐떄

if(all\_star(tokens[row - 1]) && row > 1 && !is\_character(tokens[row - 2][strlen(tokens[row - 2]) - 1]))

row--;

if(all\_star(tokens[row - 1]) && row == 1)

row--;

//row가 1이고 첫번째 토큰이 \* 문자일때

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

if(i > 0 && \*(start + i) == '.'){

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

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

i++;

}

else if(start[i] == ' '){

while(start[i] == ' ')

i++;

break;

} //start에서 공백을 제외한 부분을 토큰으로 저장함

else

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

}

if(start[0] == ' '){

start += i;

continue;

}

start += i;

}

strcpy(tokens[row], ltrim(rtrim(tokens[row])));

//토큰의 좌우에서 공백까지 제거함

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] == '.' ) ){

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

{//여는 괄호가 있을떄

if(strcmp(tokens[row - 1], "struct") != 0 && strcmp(tokens[row - 1],"unsigned") != 0)

return false;

}//괄호 안에 struct나 unsigned가 있으면 false리턴

else if(row == 1 && is\_character(tokens[row][strlen(tokens[row]) - 1])) {

if(strcmp(tokens[0], "extern") != 0 && strcmp(tokens[0], "unsigned") != 0 && is\_typeStatement(tokens[0]) != 2)

return false;

}//토큰의 마지막이 문자일때 가장 처음의 토큰이 extern 또는 unsigned 일 경우 false 리턴

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

if(strcmp(tokens[row - 2], "unsigned") != 0 && strcmp(tokens[row - 2], "extern") != 0)

return false;

}// 전 토큰이 명령어 일 때, 그 전의 토큰이 unsigned 또는 extern일때, false 리턴

}

if((row == 0 && !strcmp(tokens[row], "gcc")) ){

clear\_tokens(tokens);

strcpy(tokens[0], str);

return 1;

} //토큰이 gcc 일 경우 토큰을 초기화하고 str을 첫번째 토큰으로 복사함

row++;

}

if(all\_star(tokens[row - 1]) && row > 1 && !is\_character(tokens[row - 2][strlen(tokens[row - 2]) - 1]))

row--;

if(all\_star(tokens[row - 1]) && row == 1)

row--;

//start의 길이 만큼 start를 분석함

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

{

if(start[i] == ' ')

{

while(start[i] == ' ')

i++;

if(start[0]==' ') {

start += i;

i = 0;

}//처음 공백 제거

else

row++;

i--;

}

else

{

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

if( start[i] == '.' && i<strlen(start)){

while(start[i + 1] == ' ' && i < strlen(start))

i++;

//공백이 아닐때까지 start 포인터를 이동시켜서 한 문자를 토큰으로 가져옴

}

}

strcpy(tokens[row], ltrim(rtrim(tokens[row])));

//토큰의 좌우에서 공백까지 제거함

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

strcat(tokens[row - 1], tokens[row]);

memset(tokens[row], 0, sizeof(tokens[row]));

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] == '.') ){

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

{//여는 괄호가 있을떄

if(strcmp(tokens[row-1], "struct") != 0 && strcmp(tokens[row-1], "unsigned") != 0)

return false;

}//괄호 안에 struct나 unsigned가 있으면 false리턴

else if(row == 1 && is\_character(tokens[row][strlen(tokens[row]) - 1])) {

if(strcmp(tokens[0], "extern") != 0 && strcmp(tokens[0], "unsigned") != 0 && is\_typeStatement(tokens[0]) != 2)

return false;

}//토큰의 마지막이 문자일때 가장 처음의 토큰이 extern 또는 unsigned 일 경우 false 리턴

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

if(strcmp(tokens[row - 2], "unsigned") != 0 && strcmp(tokens[row - 2], "extern") != 0)

return false;

}// 전 토큰이 명령어 일 때, 그 전의 토큰이 unsigned 또는 extern일때, false 리턴

}

}

if(row > 0)

{ //기존에 민들어진 토큰이 있을 때

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

clear\_tokens(tokens);

strcpy(tokens[0], remove\_extraspace(str));

}//토큰이 위와 같은 경우, str에서 빈칸을 지운 문자열을 토큰으로 만듬

}

//문자열이 명령어이거나, extern이 첫번째 토큰일 떄

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

for(i = 1; i < TOKEN\_CNT; i++){

if(strcmp(tokens[i],"") == 0)

break;

if(i != TOKEN\_CNT -1 )

strcat(tokens[0], " ");

strcat(tokens[0], tokens[i]);

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

}//0번 토큰 뒤에 i번 토큰을 붙이고 i번 토큰 초기화

}

while((p\_str = find\_typeSpecifier(tokens)) != -1){

if(!reset\_tokens(p\_str, tokens))

return false;

} // 남은 문자열에 토큰으로 만들게 있으면 false리턴

while((p\_str = find\_typeSpecifier2(tokens)) != -1){

if(!reset\_tokens(p\_str, tokens))

return false;

} // 남은 문자열에 토큰으로 만들게 있으면 false리턴

return true;

}

node \*make\_tree(node \*root, char (\*tokens)[MINLEN], int \*idx, int parentheses)

{ //토큰을 바탕으로 트리를 생성하는 함수

node \*cur = root;

node \*new;

node \*saved\_operator;

node \*operator;

int fstart;

int i;

while(1)

{

if(strcmp(tokens[\*idx], "") == 0)

break;

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

return get\_root(cur);

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

return get\_root(cur);

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

{ //여는 괄호부터 닫는 괄호가 나올때 까지 내부로 트리를 만들어감

if(\*idx > 0 && !is\_operator(tokens[\*idx - 1]) && strcmp(tokens[\*idx - 1], ",") != 0){

fstart = true;

//토큰이 연산자가 아니고 콤마가 아닐때

while(1)

{

\*idx += 1;

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

break;

//닫는 괄호가 나오면 종료

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

//재귀호출로 자식노드들의 트리 생성

if(new != NULL){

if(fstart == true){

cur->child\_head = new;

new->parent = cur;

//자식 노드 생성 후 부모 노드와 연결

fstart = false;

}

else{

cur->next = new;

new->prev = cur;

}

cur = new;

} //자식노드의 트리가 있을떄 형제 노드들을 생성함

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

break; //닫는 괄호가 나오면 종료

}

}

else{ //토큰이 연산자이거나 콤마일때

\*idx += 1;

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

if(cur == NULL)

cur = new;

else if(!strcmp(new->name, cur->name)){

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

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

{ //and or 등의 연산자일떄

cur = get\_last\_child(cur);

if(new->child\_head != NULL){

new = new->child\_head;

//새로운 자식노드를 시작함

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

new->parent = NULL;

new->prev = cur;

cur->next = new;

}

}

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

{ //더하기 또는 곱하기일때

i = 0;

while(1)

{

if(!strcmp(tokens[\*idx + i], ""))

break;

//토큰이 공백이면 break

if(is\_operator(tokens[\*idx + i]) && strcmp(tokens[\*idx + i], ")") != 0)

break;

//토큰이 닫는 괄호거나 연산자일 경우 break

i++;

}

if(get\_precedence(tokens[\*idx + i]) < get\_precedence(new->name))

{ //연산자의 우선순위를 고려하여 자식노드와 부모노드를 결정함

cur = get\_last\_child(cur);

cur->next = new;

new->prev = cur;

cur = new;

}

else

{

cur = get\_last\_child(cur);

if(new->child\_head != NULL){

new = new->child\_head;

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

new->parent = NULL;

new->prev = cur;

cur->next = new;

} //마지막 자식노드로 추가함

}

}

else{ //더하기나 곱하기가 아닐때

cur = get\_last\_child(cur);

cur->next = new;

new->prev = cur;

cur = new;

}

}

else

{ //새 노드와 현재 노드가 같은 문자일때

cur = get\_last\_child(cur);

cur->next = new;

new->prev = cur;

cur = new;

}

}

}

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

{ //토큰이 연산자일때

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

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

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

{//토큰이 위와 같은 연산자일때

if(is\_operator(cur->name) == true && !strcmp(cur->name, tokens[\*idx]))

operator = cur;

//현재 노드 저장

else

{

new = create\_node(tokens[\*idx], parentheses);

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

//둘의 가장 상위 노드를 불러와 저장

if(operator->parent == NULL && operator->prev == NULL){

//연산자의 부모와 형제가 모두 없을떄

if(get\_precedence(operator->name) < get\_precedence(new->name)){

cur = insert\_node(operator, new);

} //두 연산자의 우선순위를 비교하여 새 노드 삽입

else if(get\_precedence(operator->name) > get\_precedence(new->name))

{

if(operator->child\_head != NULL){

operator = get\_last\_child(operator);

cur = insert\_node(operator, new);

}

}//연산자 우선순위를 비교하여 자식노드의 마지막에 새 노드 생성

else

{

operator = cur;

while(1)

{

if(is\_operator(operator->name) == true && !strcmp(operator->name, tokens[\*idx]))

break;

//operator가 토큰과 같아지면 탈출

if(operator->prev != NULL)

operator = operator->prev;

else

break;

}//operator의 마지막 형제 노드까지 이동하고 루프 탈출

if(strcmp(operator->name, tokens[\*idx]) != 0)

operator = operator->parent;

//토큰과 operator가 같지 않다면 부모노드로 이동

if(operator != NULL){

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

cur = operator;

}//연산자가 null이 아니고 토큰과 같을떄 현재 노드를 operator로 덮어씀

}

}

else

cur = insert\_node(operator, new);

}//연산자의 부모 현제노드가 없는 경우가 아닐 경우 새 노드 생성 후 트리에 추가함

}

else

{//토큰이 위와 같은 연산자가 아닐때

new = create\_node(tokens[\*idx], parentheses);

if(cur == NULL)

cur = new;

else

{

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

//현재 노드중 최상위 노드를 불러옴

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

cur = insert\_node(operator, new);

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

//operator의 부모 형제 노드 모두 없을때

if(get\_precedence(operator->name) > get\_precedence(new->name))

{ //연산자 순위 비교

if(operator->child\_head != NULL){

operator = get\_last\_child(operator);

cur = insert\_node(operator, new);

}//operator의 자식노드가 있는 경우 자식노드의 가장 마지막에 추가함

}

else

cur = insert\_node(operator, new);

}//operator의 prev에 새노드 추가

else

cur = insert\_node(operator, new);

} //operator의 prev에 새노드 추가

}

}

else //토큰이 여는 괄호가 아니고 연산자도 아닐때

{

new = create\_node(tokens[\*idx], parentheses);

//새 노드 생성

if(cur == NULL)

cur = new;

//현재 노드가 없을 경우 현재노드가 됨

else if(cur->child\_head == NULL){

cur->child\_head = new;

new->parent = cur;

cur = new;

} //현재노드의 자식만 없을 경우 자식 노드가 됨

else{

//그 외의 경우 마지막 자식노드의 prev 노드가 됨

cur = get\_last\_child(cur);

cur->next = new; //왼쪽 노드

new->prev = cur; //오른쪽 노드

cur = new;

}

}

\*idx += 1;

}//다음 토큰으로 이동

return get\_root(cur);

}//루트를 리턴함

node \*change\_sibling(node \*parent)

{ //자식 노드의 순서를 바꾸는 함수

node \*tmp;

tmp = parent->child\_head;

//원래의 child\_head 저장

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

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

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

//원래의 child\_head의 next를 새 child\_head로 지정

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;

//새 child\_head의 next를 원래의 child\_head로 지정

return parent;

}

node \*create\_node(char \*name, int parentheses)

{ //트리의 노드를 새로 생성하는 함수

node \*new;

new = (node \*)malloc(sizeof(node));

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

strcpy(new->name, name);

new->parentheses = parentheses;

new->parent = NULL;

new->child\_head = NULL;

new->prev = NULL;

new->next = NULL;

return new;

}

int get\_precedence(char \*op)

{ //지정한 연산자 우선순위를 검색함

int i;

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

if(!strcmp(operators[i].operator, op))

return operators[i].precedence;

}

return false;

} //찾은 연산자 우선순위를 반환함

int is\_operator(char \*op)

{ //입력된 문자가 연산자인지 확인하는 함수

int i;

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

{

if(operators[i].operator == NULL)

break;

if(!strcmp(operators[i].operator, op)){

return true;

}

}

return false;

}

void print(node \*cur)

{ //트리를 하나씩 출력하는 함수

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

print(cur->child\_head);

printf("\n");

}

if(cur->next != NULL){

print(cur->next);

printf("\t");

}

printf("%s", cur->name);

}

node \*get\_operator(node \*cur)

{ // 트리 탐색중 연산자를 불러오기 위한 함수

if(cur == NULL)

return cur;

if(cur->prev != NULL)

while(cur->prev != NULL)

cur = cur->prev;

return cur->parent;

} //child\_head의 부모노드는 연산자이므로 이를 반환함

node \*get\_root(node \*cur)

{ // 트리 탐색중 root 노드를 확인하기 위한 함수

if(cur == NULL)

return cur;

while(cur->prev != NULL)

cur = cur->prev;

if(cur->parent != NULL)

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

return cur;

} //루트 노드에 닿을때까지 반복한다

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

{ //상위에 있는 노드를 탐색하기 위한 함수

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

if(get\_precedence(cur->name) < get\_precedence(new->name))

return cur;

if(cur->prev != NULL){

while(cur->prev != NULL){

cur = cur->prev;

return get\_high\_precedence\_node(cur, new);

}

//가장 상단에 도착할 떄까지 재귀 순환

if(cur->parent != NULL)

return get\_high\_precedence\_node(cur->parent, new);

}

if(cur->parent == NULL)

return cur;

}

node \*get\_most\_high\_precedence\_node(node \*cur, node \*new)

{ //가장 상위에 있는 노드를 탐색하기 위한 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)

operator = get\_high\_precedence\_node(saved\_operator->prev, new);

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

operator = get\_high\_precedence\_node(saved\_operator->parent, new);

saved\_operator = operator;

} //get\_high\_precedence\_node() 함수를 재귀적으로 호출하여 최상위의 노드를 탐색함

return saved\_operator;

}

node \*insert\_node(node \*old, node \*new)

{ // 트리에 새 노드를 추가하는 함수

if(old->prev != NULL){

new->prev = old->prev;

old->prev->next = new;

old->prev = NULL;

}

new->child\_head = old;

old->parent = new;

return new;

}

node \*get\_last\_child(node \*cur)

{ //노드의 가장 마지막 자식 노드를 탐색하는 함수

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

cur = cur->child\_head;

while(cur->next != NULL)

cur = cur->next;

return cur;

}

int get\_sibling\_cnt(node \*cur)

{ //자식노드의 총 갯수를 탐색하는 함수

int i = 0;

while(cur->prev != NULL)

cur = cur->prev;

while(cur->next != NULL){

cur = cur->next;

i++;

}

return i;

} //형제 노드의 가장 마지막에서부터 head로 오면서 갯수를 셈

void free\_node(node \*cur)

{ //트리를 만들면서 지정한 메모리를 free 해주는 함수

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

free\_node(cur->child\_head);

//자식노드가 있는 경우 자식노드 해제

if(cur->next != NULL)

free\_node(cur->next);

//형제 노드가 있는 경우 형제 노드 해제

if(cur != NULL){

cur->prev = NULL;

cur->next = NULL;

cur->parent = NULL;

cur->child\_head = NULL;

free(cur);

}

} //트리 전체를 순환하며 남는 것이 없게 메모리를 해제해줌

int is\_character(char c)

{ //문자인지 확인하는 함수

return (c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z');

}

int is\_typeStatement(char \*str)

{ //명령어를 구분해 내는 함수

char \*start;

char str2[BUFLEN] = {0};

char tmp[BUFLEN] = {0};

char tmp2[BUFLEN] = {0};

int i;

start = str;

strncpy(str2,str,strlen(str));

remove\_space(str2);

while(start[0] == ' ')

start += 1;

//gcc 명령어가 포함되어있을 경우

if(strstr(str2, "gcc") != NULL)

{

strncpy(tmp2, start, strlen("gcc"));

if(strcmp(tmp2,"gcc") != 0)

return 0;

else

return 2;

}//start에서 tmp2로 3개만큼 문자열 복사

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

{

if(strstr(str2,datatype[i]) != NULL)

{

strncpy(tmp, str2, strlen(datatype[i]));

strncpy(tmp2, start, strlen(datatype[i]));

//문자열을 tmp, tmp2로 지정한 문자열의 길이만큼 복사

if(strcmp(tmp, datatype[i]) == 0)

if(strcmp(tmp, tmp2) != 0)

return 0;

else

return 2;

} //복사한 문자열들을 비교 판단함

}

return 1;

}

int find\_typeSpecifier(char tokens[TOKEN\_CNT][MINLEN])

{ //문자의 type을 결정하는 인자를 추출하는 함수

int i, j;

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

{

for(j = 0; j < DATATYPE\_SIZE; j++)

{

if(strstr(tokens[i], datatype[j]) != NULL && i > 0)

{

if(!strcmp(tokens[i - 1], "(") && !strcmp(tokens[i + 1], ")")

&& (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])))

return i;

}

}

}

return -1;

}

int find\_typeSpecifier2(char tokens[TOKEN\_CNT][MINLEN])

{ //변수의 type을 결정하는 인자를 추출해내는 함수

int i, j;

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

{

for(j = 0; j < DATATYPE\_SIZE; j++)

{

if(!strcmp(tokens[i], "struct") && (i+1) <= TOKEN\_CNT && is\_character(tokens[i + 1][strlen(tokens[i + 1]) - 1]))

return i;

}

}

return -1;

}

int all\_star(char \*str)

{ // 문자열이 모두 \*인지 확인하는 함수

int i;

int length= strlen(str);

if(length == 0)

return 0;

for(i = 0; i < length; i++)

if(str[i] != '\*')

return 0;

return 1;

}

int all\_character(char \*str)

{ //문자열이 모두 문자인지 확인하는 힘수

int i;

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

if(is\_character(str[i]))

return 1;

return 0;

}

int reset\_tokens(int start, char tokens[TOKEN\_CNT][MINLEN])

{ //분해한 토큰을 다시 하나의 문자열로 합치는 함수

int i;

int j = start - 1;

int lcount = 0, rcount = 0;

int sub\_lcount = 0, sub\_rcount = 0;

if(start > -1){

if(!strcmp(tokens[start], "struct")) {

strcat(tokens[start], " ");

strcat(tokens[start], tokens[start+1]);

for(i = start + 1; i < TOKEN\_CNT - 1; i++){

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) {

strcat(tokens[start], " ");

strcat(tokens[start], tokens[start + 1]);

strcat(tokens[start], tokens[start + 2]);

for(i = start + 1; i < TOKEN\_CNT - 1; i++){

strcpy(tokens[i], tokens[i + 1]);

memset(tokens[i + 1], 0, sizeof(tokens[0]));

}

}

j = start + 1;

while(!strcmp(tokens[j], ")")){

rcount ++;

if(j==TOKEN\_CNT)

break;

j++;

}

j = start - 1;

while(!strcmp(tokens[j], "(")){

lcount ++;

if(j == 0)

break;

j--;

}

if( (j!=0 && is\_character(tokens[j][strlen(tokens[j])-1]) ) || j==0)

lcount = rcount;

if(lcount != rcount )

return false;

if( (start - lcount) >0 && !strcmp(tokens[start - lcount - 1], "sizeof")){

return true;

}

else if((!strcmp(tokens[start], "unsigned") || !strcmp(tokens[start], "struct")) && strcmp(tokens[start+1], ")")) {

strcat(tokens[start - lcount], tokens[start]);

strcat(tokens[start - lcount], tokens[start + 1]);

strcpy(tokens[start - lcount + 1], tokens[start + rcount]);

for(int i = start - lcount + 1; i < TOKEN\_CNT - lcount -rcount; i++) {

strcpy(tokens[i], tokens[i + lcount + rcount]);

memset(tokens[i + lcount + rcount], 0, sizeof(tokens[0]));

}

}

else{

if(tokens[start + 2][0] == '('){

j = start + 2;

while(!strcmp(tokens[j], "(")){

sub\_lcount++;

j++;

}

if(!strcmp(tokens[j + 1],")")){

j = j + 1;

while(!strcmp(tokens[j], ")")){

sub\_rcount++;

j++;

}

}//괄호의 갯수를 맞추기 위해 여는 괄호와 닫는 괄호의 갯수를 비교함

else

return false;

if(sub\_lcount != sub\_rcount)

return false;

strcpy(tokens[start + 2], tokens[start + 2 + sub\_lcount]);

for(int i = start + 3; i<TOKEN\_CNT; i++)

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

}

strcat(tokens[start - lcount], tokens[start]);

strcat(tokens[start - lcount], tokens[start + 1]);

strcat(tokens[start - lcount], tokens[start + rcount + 1]);

for(int i = start - lcount + 1; i < TOKEN\_CNT - lcount -rcount -1; i++) {

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])

{

int i;

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

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

} //토큰들을 모두 널문자로 초기화 하는 함수

char \*rtrim(char \*\_str)

{// 문자열의 뒤쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수

char tmp[BUFLEN];

char \*end;

strcpy(tmp, \_str);

end = tmp + strlen(tmp) - 1;

while(end != \_str && isspace(\*end))

--end;

\*(end + 1) = '\0';

\_str = tmp;

return \_str;

}

char \*ltrim(char \*\_str)

{// 문자열의 앞쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수

char \*start = \_str;

while(\*start != '\0' && isspace(\*start))

++start;

\_str = start;

return \_str;

}

char\* remove\_extraspace(char \*str)

{ //문자열에 공백이 1개 이상 연속될 때, 1개만 남기고 공백을 지우는 함수

int i;

char \*str2 = (char\*)malloc(sizeof(char) \* BUFLEN);

char \*start, \*end;

char temp[BUFLEN] = "";

int position;

if(strstr(str,"include<")!=NULL){

start = str;

end = strpbrk(str, "<");

position = end - start;

strncat(temp, str, position);

strncat(temp, " ", 1);

strncat(temp, str + position, strlen(str) - position + 1);

str = temp;

} //문자열의 시작이 include 일 경우 뒤에 오는 헤더파일을 구분하기 위해 문자열을 자름

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

{

if(str[i] ==' ')

{

if(i == 0 && str[0] ==' ')

while(str[i + 1] == ' ')

i++;

else{

if(i > 0 && str[i - 1] != ' ')

str2[strlen(str2)] = str[i];

while(str[i + 1] == ' ')

i++;

}

} //문자열 중간에 있는 공백을 제거함

else

str2[strlen(str2)] = str[i];

}

return str2;

}

void remove\_space(char \*str)

{ //문자열의 공백을 제거하는 함수

char\* i = str;

char\* j = str;

while(\*j != 0)

{

\*i = \*j++;

if(\*i != ' ')

i++;

}//공백문자를 뛰어넘는 방식으로 공백을 없앰

\*i = 0;

}

int check\_brackets(char \*str)

{ //여는괄호 닫는 괄호 갯수가 일치하는지 확인하는 함수

char \*start = str;

int lcount = 0, rcount = 0;

while(1){

if((start = strpbrk(start, "()")) != NULL){

if(\*(start) == '(')

lcount++;

else

rcount++;

start += 1;

}

else

break;

}

if(lcount != rcount)

return 0;

else

return 1;

}

int get\_token\_cnt(char tokens[TOKEN\_CNT][MINLEN])

{ //토큰의 갯수를 세는 함수

int i;

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

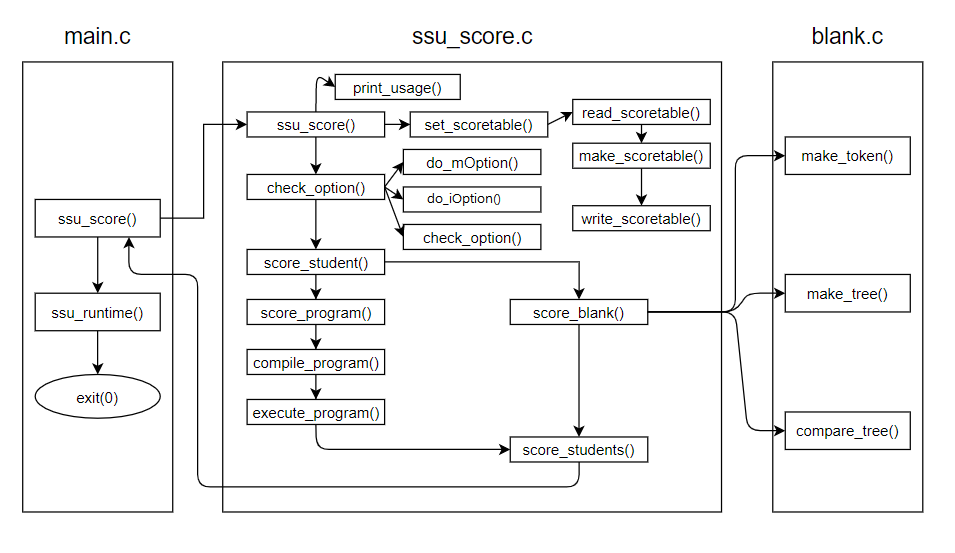
if(!strcmp(tokens[i], ""))

break;

return i;

}

- 기본 소스 코드에 대한 분석을 토대로, 실질적으로 작동에 크게 관여 하는 주요 함수를 추려 흐름도를 작성하였다.



- 흐름도에 나타나지 않는 다수의 함수도 프로그램 상에 존재한다.

- ssu\_score.c 에서는 전반적인 파일 생성, 문제 배점 저장 계산, 프로그램 문제 채점 등의 기능을 한다.

- blank.c 에서는 빈칸 문제를 채점하는데에 특화되어있고, 무엇보다 다수의 답이 정답이 되는 경우를 모두 계산하기 위해, 학생의 답과 정답을 최소 단위인 토큰으로 분해하여 트리를 생성하여 트리의 구조를 통해 답안을 비교 채점한다.

3. 설계

- 전체 프로그램 코드의 함수에 대한 설계

**<ssu\_score.c>**

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

//main함수의 호출로 불러져 전체 채점을 시작하게 하는 함수, 표준 입력으로 디렉터리와 각종 옵션을 불러옴

int check\_option(int argc, char \*argv[]);

//사용자로부터 인자로 받은 옵션을 확인하여 옵션에 알맞은 함수를 실행시키는 함수

//표준 입력으로 받은 문자열에서 옵션을 구분해내어 선택된 옵션을 반환함

void print\_usage();

//-h 옵션 사용법 출력에 사용하는 함수

void score\_students();

//학생들의 득점을 파일에 저장하고 총점을 계산하는 함수

double score\_student(int fd, char \*id);

//실질적으로 학생들의 점수를 계산하는 함수, 학번과 파일위치를 받아서 점수를 리턴함

void write\_first\_row(int fd);

//학생 배점 파일 첫줄에 문제 번호를 작성하는 함수, 파일 디스크립터를 받아서 파일에 작성함

char \*get\_answer(int fd, char \*result);

//학생 파일 또는 정답 파일에서 정답 텍스트만을 추출하는 함수

//콜론으로 구분된 정답파일에서 하나씩 정답을 추출함

int score\_blank(char \*id, char \*filename);

//빈칸문제를 채점하는 함수

//학번과 파일이름을 받아와 정답과 오답을 반환함

double score\_program(char \*id, char \*filename);

//프로그램 문제를 채점하는 함수, 학번과 파일이름을 받아와 정답인지 오답인지를 반환함

double compile\_program(char \*id, char \*filename);

//학생 및 정답의 프로그램 문제를 컴파일하는 함수, 컴파일한 프로그램이 정답인지 오답인지 반환함

int execute\_program(char \*id, char \*filname);

// 컴파일된 프로그램 문제를 직접 실행시키는 함수, 실행한 프로그램이 정답인지 오답인지를 반환함

pid\_t inBackground(char \*name);

//백그라운드에서 실행중인 학생 또는 정답 프로그램의 프로세스 ID를 반환함

//프로그램의 이름으로 프로세스 ID를 판단함

double check\_error\_warning(char \*filename);

// 학생 프로그램의 에러 또는 경고의 개수를 계산하는 함수, 파일이름을 받아와 에러 경고 개수로 감점을 계산

int compare\_resultfile(char \*file1, char \*file2);

//학생 프로그램의 결과물과 정답 프로그램의 결과물을 서로 비교하는 함수

//두 프로그램으로 생성된 파일을 비교하여 같은지 다른지 판단하여 반환

void do\_iOption(char (\*ids)[FILELEN]);

//사용자가 i옵션을 입력했을 때, 입력받은 학생의 틀린 문제를 출력하는 함수

//학번을 입력받아 그 학번의 학생의 0점 문제를 출력함

int is\_exist(char (\*src)[FILELEN], char \*target);

// 학번을 바탕으로 존재하는 학생인지 판단하는 함수, 디렉터리 내에 학번 폴더 존재 여부를 리턴

int is\_thread(char \*qname);

//문제 번호를 바탕으로 –lpthread 옵션을 주고 컴파일해야하는 문제인지 확인하는 함수

//문제 번호를 보고 컴파일 옵션을 결정함

void redirection(char \*command, int newfd, int oldfd);

//파일 디스크립터를 다른 곳으로 옮겼다가 복구시키는 함수

// old 디스크립터를 new 디스크립터로 옮겨서 command를 system에 실행한 후, oldfd를 복구해줌

int get\_file\_type(char \*filename);

//파일이 텍스트 파일인지 소스코드인지 구분하는 함수

//파일 이름을 바탕으로 파일의 형태를 반환함

void rmdirs(const char \*path);

//디렉터리를 삭제하는 함수

void to\_lower\_case(char \*c);

// 모든 영문자를 소문자로 교체하는 함수

void set\_scoreTable(char \*ansDir);

//문제 배점 파일을 생성하는 함수

void read\_scoreTable(char \*path);

//문제 배점 파일을 읽어와 배열로 저장하는 함수

void make\_scoreTable(char \*ansDir);

//문제 배점 파일에 문제번호를 불러오는 함수

void write\_scoreTable(char \*filename);

//문제 배점 파일에 문제의 배점을 입력하고 저장하는 함수

void set\_idTable(char \*stuDir);

//학생들의 점수가 기록될 파일에 학생들의 학번을 입력하는 함수

int get\_create\_type();

//초기에 문제 배점 파일이 없는 경우 사용자로부터 문제의 배점을 입력받기 위한 함수

void sort\_idTable(int size);

//학생들의 학번을 기준으로 학생 득점 파일에 학번을 작성하기 전 배열의 학번을 정렬하는 함수

void sort\_scoreTable(int size);

//문제번호 순서대로 문제 배점 배열을 정렬하는 함수

void get\_qname\_number(char \*qname, int \*num1, int \*num2);

//배열에서 문제 번호를 불러오는 함수

void do\_mOption(void);

//m옵션을 실행하는 함수

**<blank.c>**

void compare\_tree(node \*root1, node \*root2, int \*result);

//두개의 트리를 서로 비교하는 함수

node \*make\_tree(node \*root, char (\*tokens)[MINLEN], int \*idx, int parentheses);

//토큰을 바탕으로 트리를 생성하는 함수, 토큰을 바탕으로 생성된 루트 반환

node \*change\_sibling(node \*parent);

//트리의 형제 노드(같은 레벨의 노드)를 서로 교환하는 함수, 자식노드를 교환하고 부모 노드를 반환함

node \*create\_node(char \*name, int parentheses);

// 노드를 새로 생성하는 함수, 새

int get\_precedence(char \*op);

//연산자의 우선순위를 결정하는 함수, 계산한 연산자의 우선순위를 반환함

int is\_operator(char \*op);

// 주어진 문자가 연산자인지 확인하는 함수, 연산자인지 아닌지를 반환함

void print(node \*cur);

//트리를 순환하면 출력하는 함수

node \*get\_operator(node \*cur);

//트리 탐색 중 연산자를 구분하는 함수, 노드의 데이터가 연산자 여부를 판단함

node \*get\_root(node \*cur);

//트리 탐색 중 root노드를 탐색하기 위한 함수, 루트 노드를 찾아서 반환

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

// 두 노드 중 더 높은 우선순위의 노드를 계산하는 함수, 더 높은 우선순위의 노드를 반환함

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

// 가장 우선순위가 높은 노드를 구분하는 함수, 가장 우선 순위가 높은 노드를 반환

node \*insert\_node(node \*old, node \*new);

// 트리에 새로 노드를 삽입하기 위한 함수, 새 노드의 부모 노드를 반환

node \*get\_last\_child(node \*cur);

//노드의 가장 마지막 자식 노드를 탐색하는 함수, 형제 노드의 가장 마지막 노드를 반환함

void free\_node(node \*cur);

//트리를 위해 지정한 메모리를 해제해주는 함수

int get\_sibling\_cnt(node \*cur);

//형제 노드의 총 개수를 계산하는 함수, 형제 노드의 개수를 반환함

int make\_tokens(char \*str, char tokens[TOKEN\_CNT][MINLEN]);

// 학생의 정답과 정답지의 정답을 토큰으로 분해할 때 사용하는 함수, 토큰으로 분해의 설공 여부를 리턴함

int is\_typeStatement(char \*str);

//명령어를 구분하는 함수, 명령어 여부를 리턴

int find\_typeSpecifier(char tokens[TOKEN\_CNT][MINLEN]);

//문자의 type을 결정하는 인자를 추출하는 함수, 문자의 타입을 리턴

int find\_typeSpecifier2(char tokens[TOKEN\_CNT][MINLEN]);

//변수의 type을 결정하는 인자를 추출해내는 함수, 변수의 타입을 리턴

int is\_character(char c);

// 문자인지 확인하는 함수, 문자 여부를 리턴함

int all\_star(char \*str);

//문자열이 모두 \*문자인지 확인하는 함수, true 또는 false 리턴함

int all\_character(char \*str);

//문자열이 모두 문자로 이루어져 있는지 확인하는 함수, true 또는 false 리턴함

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

//분해한 토큰을 다시 문자열로 합치는 함수

void clear\_tokens(char tokens[TOKEN\_CNT][MINLEN]);

//모든 토큰을 널문자로 초기화하는 함수

int get\_token\_cnt(char tokens[TOKEN\_CNT][MINLEN]);

// NULL문자가 아닌 토큰의 개수를 세는 함수, 토큰의 개수를 반환함

char \*rtrim(char \*\_str);

//문자열의 뒤쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수

char \*ltrim(char \*\_str);

//문자열의 앞쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수

void remove\_space(char \*str);

//문자열의 공백을 제거하는 함수

int check\_brackets(char \*str);

//여는 괄호와 닫는 괄호의 개수가 일치하는지 확인하는 함수, 일치 여부를 리턴함

char\* remove\_extraspace(char \*str);

//문자열에 공백이 1개 이상 연속될 때, 1개만 남기고 공백을 제거하는 함수, 공백을 제거한 문자열 리턴

**<main.c>**

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

//프로그램의 총 실행시간을 확인하는 함수, 저장해 둔 시작 시간과 종료 시간의 차이를 계산함

4. 구현

1) void do\_mOption(void)

- m옵션이 입력되었을 때, 이를 실행하기 위한 함수

- 사용자로부터 배점을 수정할 문제 번호, 새 배점을 입력 받아 저장하고, 새로운 배점을 적용하여 채점한다.

- 존재하지 않는 문제 번호를 사용자가 입력하였을 경우, 에러 메시지 출력 후 재입력을 받는다.

- 이후 바뀐 배점을 배점을 저장하는 파일에 저장한다.

**void** do\_mOption(**void**) {

**int** I, ft = 0;

**char** tmp[30];

**int** cnt;

**double** ns;

**while**(1) {

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

scanf("%s", tmp);

**if**(!strcmp(tmp, "no"))

**break**;

*//no라고 입력한 경우 문제 배점 수정 종료*

**for**(i = 0; i < QNUM; i++){

cnt = strncmp(tmp, score\_table[i].qname, strlen(tmp));

**if**(!cnt) {

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

printf("New score : ");

scanf("%lf", &ns);

score\_table[i].score = ns;

ft = 1;

}

} *//문제의 새 배점을 입력받아 출력함*

if(!ft) printf(“Wrong question number!!\n”);

*// 존재하지 않는 문제일 경우, 에러메시지 출력*

} *// break조건이 나올떄 까지 반복하여 입력받음*

write\_scoreTable("score\_table.csv"); *//바뀐 배점으로 score\_table을 새로 작성*

**return**;

} *//m옵션이 주어졌을 경우 문제의 배점을 수정하는 함수*

2) int check\_option (int argc, char \*argv[])

- 입력된 옵션이 무엇인지 확인하여, 옵션에 맞는 함수를 호출하기 위한 함수

- 프로그램에서 지원하는 m, e, t, i, h 각각의 옵션에 대해 지정된 명령을 실행한다.

- 프로그램에서 지정하지 않은 옵션이 입력되었을 경우, 프로그램을 종료하게 된다.

**int** check\_option(**int** argc, **char** \*argv[])

{ *//입력된 옵션을 체크 하는 함수*

**int** i, j;

**int** c;

**while**((c = getopt(argc, argv, "me:ti:")) != -1)

{ *//지정된 옵션이 입력되었는지 확인함*

**switch**(c){

**case** 'm': *//m옵션시 옵션 스위치를 켬*

mOption = true;

**break**;

**case** 'e': *//e옵션이 입렫외었을 경우 지정된 이름의 error 디렉터리를 생성함*

eOption = true;

strcpy(errorDir, optarg);

**if**(access(errorDir, F\_OK) < 0)

mkdir(errorDir, 0755);

**else**{

rmdirs(errorDir);

mkdir(errorDir, 0755);

}

**break**;

**case** 't': *//t옵션의 스위치를 켜고 어느 문제를 thread를 이용하여 채점할지 기억함*

tOption = true;

i = optind;

j = 0;

**while**(i < argc && argv[i][0] != '-'){

**if**(j >= ARGNUM)

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

**else**

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

i++;

j++;

}

**break**;

**case** 'i': *//i옵션 인 경우 어느 학생의 틀린 문제를 출력할지 저장함*

iOption = true;

i = optind - 1;

j = 0;

**while**(i < argc && argv[i][0] != '-'){

**if**(j >= ARGNUM)

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

**else**

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

i++;

j++;

}

**break**;

**case** '?': *//그 외의 지정되지 않은 옵션이 들어올 경우 판단불가능을 반환함*

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

**return** false;

}

} *//모든 옵션을 판단한 후 while문 종료*

**return** true;

}

3) void do\_iOption(char (\*ids)[FILELEN])

- i 옵션이 입력되었을 때, 옵션의 명령을 수행하는 함수

- 채점 결과가 있는 경우, 입력한 학생의 틀린 문제의 목록을 출력한다.

- 최대 5명의 학생의 틀린 문제 목록을 한번에 출력할 수 있다.

**void** do\_iOption(**char** (\*ids)[FILELEN])

{ *//학생의 틀린 문제 번호를 출력하는 함수*

**FILE** \*fp;

**char** tmp[BUFLEN];

**char** ques\_num[BUFLEN];

**char** ques\_num\_backup[BUFLEN];

**int** i = 0;

**char** \*p, \*saved, \*qn, \*nextptr;

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

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

**return**;

} *//score.csv를 오픈함, 실패시 에러출력*

fscanf(fp, "%s**\n**", ques\_num);

strcpy(ques\_num\_backup, ques\_num);

*//문제번호 줄 읽어옴*

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

{

p = strtok(tmp, ",");

**if**(!is\_exist(ids, p))

**continue**;

*//학생의 학번이 올바른지 판단함*

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

strcpy(ques\_num, ques\_num\_backup);

qn = strtok\_r(ques\_num, ",", &nextptr);

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

**if**(!strcmp(p, "0"))

printf("%s ", qn);

qn = strtok\_r(NULL, ",", &nextptr);

} *//학생의 점수가 0인 문제의 반호를 문제의 끝까지 출력함*

printf("**\n**");

}

fclose(fp);

}

4) void redirection(char \* command, int new, int old)

- 학생 또는 정답의 프로그램 파일 실행시, 프로그램 파일에서 표준 출력으로 출력하는 오류메시지 또는 결과 값을 사용자의 표준 출력으로 출력 하는 것이 아닌, 별도의 파일에 출력하도록 파일 디스크립터를 교체해주는 함수

- 표준 출력과 표준 에러를 임의의 파일 디스크립터로 변경 한다

**void** redirection(**char** \*command, **int** new, **int** old)

{ *//파일 디스크립터를 덮어썼다가 다시 복구 시켜주는 함수*

**int** saved, saved2;

saved = dup(old);

saved2 = dup(STDERR);

dup2(new, old);

dup2(new, STDERR);

system(command);

dup2(saved, old);

close(saved);

close(saved2);

} *// 학생 및 정답 프로그램 파일에서 오류나 출력값이 나올때 화면이 아닌 임의의 파일로 출력하는데 사용*

5) double compile\_program(char \*id, char \*fimename)

- compile\_program 함수는 학생과 정답 디렉터리에 있는 프로그램 파일을 컴파일 하는 함수이다.

- 함수에서 파일을 불러오는 경로에 변경이 필요하여, 파일을 불러오는 과정에 있는 코드를 수정하였다.

double compile\_program(char \*id, char \*filename)

{ *//프로그램 문제를 컴파일 하는 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));

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.')));

isthread = is\_thread(qname);

*//-t 옵션으로 지정된 문제인지 판단*

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

sprintf(tmp\_e, "%s/%s.exe", ansDir, qname);

**if**(tOption && isthread)

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f);

**else**

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f);

sprintf(tmp\_e, "%s/%s/%s\_error.txt", ansDir, qname, qname);

fd = creat(tmp\_e, 0666);

*//e옵션이 적용되어있는 경우, 오류메시지를 지정한 디렉터리에 텍스트 파일로 저장함*

redirection(command, fd, STDERR);

size = lseek(fd, 0, SEEK\_END);

close(fd);

unlink(tmp\_e);

**if**(size > 0)

**return** false;

sprintf(tmp\_f, "%s/%s/%s", stuDir, id, filename);

sprintf(tmp\_e, "%s/%s/%s.stdexe", stuDir, id, qname);

**if**(tOption && isthread)

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f);

**else**

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f);

*// t옵션의 유무에 따라 컴파일을 다르게 적용함*

sprintf(tmp\_f, "%s/%s/%s\_error.txt", stuDir, id, qname);

fd = creat(tmp\_f, 0666);

*//에러 파일은 다른 곳로 출력함*

redirection(command, fd, STDERR); *// STDERR의 fd를 다른 파일로 옮김*

size = lseek(fd, 0, SEEK\_END); *//파일 끝으로 가서 파일의 사이즈를 판단함*

close(fd);

**if**(size > 0){

**if**(eOption)

{ *//e옵션이 있을 경우 오류메시지를 사용자가 지정한 폴더로 내보냄*

sprintf(tmp\_e, "%s/%s", errorDir, id);

**if**(access(tmp\_e, F\_OK) < 0)

mkdir(tmp\_e, 0755);

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

rename(tmp\_f, tmp\_e);

result = check\_error\_warning(tmp\_e);

}

**else**{ *//e옵션이 없을 경우 에러나 경고의 갯수만을 판단함*

result = check\_error\_warning(tmp\_f);

unlink(tmp\_f);

}

**return** result;

}

unlink(tmp\_f);

**return** true;

}

6) int execute\_program(char \*id, char \*filename)

- execute\_program() 함수는 학생과 정답 디렉터리에 있는 프로그램 문제를 실행하는 함수이다.

- 함수에서 파일의 위치를 불러오는 과정에서 수정이 필요하여, 파일을 불러오는 코드를 수정하였다.

int execute\_program(char \*id, char \*filename)

{ *//학생들의 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;

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

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.')));

sprintf(ans\_fname, "%s/%s.stdout", ansDir, qname);

fd = creat(ans\_fname, 0666);

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

redirection(tmp, fd, STDOUT); *//프로그램 문제 정답 프로그램의 표준 출력을 임시 파일로 돌림*

close(fd);

sprintf(std\_fname, "%s/%s/%s.stdout", stuDir, id, qname);

fd = creat(std\_fname, 0666);

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

start = time(NULL);

redirection(tmp, fd, STDOUT); *//학생들의 프로그램 문제의 표준 출력을 임시 파일로 돌림*

sprintf(tmp, "%s.stdexe", qname);

**while**((pid = inBackground(tmp)) > 0){ *//백그라운드에서 돌고 있는 학생 프로그램의 프로세스 ID를 가져옴*

end = time(NULL);

**if**(difftime(end, start) > OVER){ *//5초이상 소요되었을 경우 프로세스를 KILL하고 0점 부여*

kill(pid, SIGKILL);

close(fd);

**return** false;

}

}

close(fd);

**return** compare\_resultfile(std\_fname, ans\_fname);

}

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

- ssu\_score() 함수는 ssu\_score.c의 메인 함수와 같은 역할을 하는 함수이다.

- main.c 에서의 호출을 통해 전체 프로그램 및 채점을 진행하게 하는 함수이다.

- 사용자가 표준 입력으로 입력한 인자들을 바탕으로, 각종 옵션과 디렉터리를 파악한 후, 프로그램을 구동한다.

- 각 옵션에 대해 적용되는 위치, 또는 채점을 위해 필요한 함수를 호출하고, 디렉터리를 파악하여 각 함수에게 넘겨주는 역할을 한다.

- 옵션이 여러 개가 입력 될 경우에도, 우선순위를 구분하여 옵션을 적용하여 준다.

- 옵션을 모두 적용한 후, 문제의 배점을 저장할 배열과, 학생의 학번을 저장하는 배열을 설정한 후, score\_students() 의 호출을 통해 학생들의 답안을 채점한다.

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

{

**char** saved\_path[BUFLEN];

**int** i, j, log;

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

**if**(!strcmp(argv[i], "-h")){

print\_usage();

**return**;

}

} *// h옵션이 있을 경우 h옵션을 출력하고 프로그램 종료*

memset(saved\_path, 0, BUFLEN); *//saved\_path배열 초기화*

**if**(argc >= 3 && (log = strcmp(argv[1], "-i")) != 0){

strcpy(stuDir, argv[1]);

strcpy(ansDir, argv[2]);

} *//옵션 i가 아닐경우 분리*

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

exit(1);

*//입력된 옵션을 체크함*

j = 0;

**if**(!eOption && !tOption && iOption && !log) {

do\_iOption(cIDs);

**return**;

} *//i옵션만 활성화되어있고, 사용자가 문제 디렉터리를 입력하지 않았을 경우*

*//i옵션과 관련된 함수 실행*

getcwd(saved\_path, BUFLEN);

*//현재 작업디렉터리의 경로를 얻어옴*

**if**(chdir(stuDir) < 0){

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

**return**;

} *//현재 작업디렉터리 변경*

getcwd(stuDir, BUFLEN);

*//학생 디렉터리의 정보 저장*

chdir(saved\_path); *//정답 디렉터리 확인*

**if**(chdir(ansDir) < 0){

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

**return**;

} *//정답디렉터리의 존재 유무 확인*

getcwd(ansDir, BUFLEN);

*//정답디렉터리 불러옴*

chdir(saved\_path);

*//정답디렉터리 정보저장*

set\_scoreTable(ansDir); *//문제의 점수를 저장하는 csv파일을 생성*

set\_idTable(stuDir); *//학생의 득점을 저장하기 위한 csv 파일 생성*

**if**(mOption) {

do\_mOption();

} *//m옵션을 입력하였을 경우, m옵션 함수 실행*

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

score\_students(); *//학생들의 답안 채점*

**if**(iOption)

do\_iOption(cIDs); *//다른 조건들을 실행하고 i옵션이 있을 경우 실행함*

**return**;

}

8) 여러가지 옵션을 구현하기 위해서, 입력된 옵션을 저장하고 있어야 한다.

- 옵션의 실행을 위해서, 각 옵션의 적용 여부를 결정할 변수를 선언하여, 토글 스위치와 같은 효과를 통해 옵션의 작용 사항을 한눈에 알기 쉽게 프로그래밍하였다.

- 각 변수들은 전역 변수로 선언하여, 다른 한수에서 그 옵션의 적용 여부를 쉽게 확인 할 수 있도록 설계하였다.

- 프로그램에서 사용할 수 있는 옵션인, e, t, I, h, m 옵션에 대해 각각의 스위치를 설정하였다.

**int** eOption = false;

**int** tOption = false;

**int** iOption = false;

**int** hOption = false;

**int** mOption = false;

9) ssu\_score.c 를 구성하는 함수 및 부가기능 설명

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

*//main함수의 호출로 불러져 전체 채점을 시작하게 하는 함수,*

**int** check\_option(**int** argc, **char** \*argv[]);

*//사용자로부터 인자로 받은 옵션을 확인하여 옵션에 알맞은 함수를 실행시키는 함수*

**void** print\_usage();

*//-h 옵션 사용법 출력에 사용하는 함수*

**void** score\_students();

*//학생들의 득점을 파일에 저장하고 총점을 계산하는 함수*

**double** score\_student(**int** fd, **char** \*id);

*//실질적으로 학생들의 점수를 계산하는 함수*

**void** write\_first\_row(**int** fd);

*//학생 배점 파일 첫줄에 문제 번호를 작성하는 함수*

**char** \*get\_answer(**int** fd, **char** \*result);

*//학생 파일 또는 정답 파일에서 정답 텍스트만을 추출하는 함수*

**int** score\_blank(**char** \*id, **char** \*filename);

*//빈칸문제를 채점하는 함수*

**double** score\_program(**char** \*id, **char** \*filename);

*//프로그램 문제를 채점하는 함수*

**double** compile\_program(**char** \*id, **char** \*filename);

*//학생 및 정답의 프로그램 문제를 컴파일하는 함수*

**int** execute\_program(**char** \*id, **char** \*filname);

*// 컴파일된 프로그램 문제를 직접 실행시키는 함수*

**pid\_t** inBackground(**char** \*name);

*//백그라운드에서 실행중인 학생 또는 정답 프로그램의 프로세스 ID를 읽어옴*

**double** check\_error\_warning(**char** \*filename);

*// 학생 프로그램의 에러 또는 경고의 개수를 계산하는 함수*

**int** compare\_resultfile(**char** \*file1, **char** \*file2);

*//학생 프로그램의 결과물과 정답 프로그램의 결과물을 서로 비교하는 함수*

**void** do\_iOption(**char** (\*ids)[FILELEN]);

*//사용자가 i옵션을 입력했을 때, 입력받은 학생의 틀린 문제를 출력하는 함수*

**int** is\_exist(**char** (\*src)[FILELEN], **char** \*target);

*// 학번을 바탕으로 존재하는 학생인지 판단하는 함수*

**int** is\_thread(**char** \*qname);

*//문제 번호를 바탕으로 –lpthread 옵션을 주고 컴파일해야하는 문제인지 확인하는 함수*

**void** redirection(**char** \*command, **int** newfd, **int** oldfd);

*//파일 디스크립터를 다른 곳으로 옮겼다가 복구시키는 함수*

**int** get\_file\_type(**char** \*filename);

*//파일이 텍스트 파일인지 소스코드인지 구분하는 함수*

**void** rmdirs(**const** **char** \*path);

*//디렉터리를 삭제하는 함수*

**void** to\_lower\_case(**char** \*c);

*// 모든 영문자를 소문자로 교체하는 함수*

**void** set\_scoreTable(**char** \*ansDir);

*//문제 배점 파일을 생성하는 함수*

**void** read\_scoreTable(**char** \*path);

*//문제 배점 파일을 읽어와 배열로 저장하는 함수*

**void** make\_scoreTable(**char** \*ansDir);

*//문제 배점 파일에 문제번호를 불러오는 함수*

**void** write\_scoreTable(**char** \*filename);

*//문제 배점 파일에 문제의 배점을 입력하고 저장하는 함수*

**void** set\_idTable(**char** \*stuDir);

*//학생들의 점수가 기록될 파일에 학생들의 학번을 입력하는 함수*

**int** get\_create\_type();

*//초기에 문제 배점 파일이 없는 경우 사용자로부터 문제의 배점을 입력받기 위한 함수*

**void** sort\_idTable(**int** size);

*//학생들의 학번을 기준으로 학생 득점 파일에 학번을 작성하기 전 배열의 학번을 정렬하는 함수*

**void** sort\_scoreTable(**int** size);

*//문제번호 순서대로 문제 배점 배열을 정렬하는 함수*

**void** get\_qname\_number(**char** \*qname, **int** \*num1, **int** \*num2);

*//배열에서 문제 번호를 불러오는 함수*

**void** do\_mOption(**void**);

*//m옵션을 실행하는 함수*

**struct** ssu\_scoreTable{

**char** qname[FILELEN];

**double** score;

}; *//문제 번호와 그 문제의 배점을 저장하기 위한 구조체*

10) blank.c를 구성하는 함수 및 기능 설명

**typedef** **struct** node{

**int** parentheses;

**char** \*name;

**struct** node \*parent;

**struct** node \*child\_head;

**struct** node \*prev;

**struct** node \*next;

}node; *//트리를 구성하기 위한 노드 구조체*

**typedef** **struct** operator\_precedence{

**char** \*operator;

**int** precedence;

}operator\_precedence; *//연산자의 우선순위를 저장하는 구조체*

**void** compare\_tree(node \*root1, node \*root2, **int** \*result);

*//두개의 트리를 서로 비교하는 함수*

node \*make\_tree(node \*root, **char** (\*tokens)[MINLEN], **int** \*idx, **int** parentheses);

*//토큰을 바탕으로 트리를 생성하는 함수*

node \*change\_sibling(node \*parent);

*//트리의 형제 노드(같은 레벨의 노드)를 서로 교환하는 함수*

node \*create\_node(**char** \*name, **int** parentheses);

*// 노드를 새로 생성하는 함수*

**int** get\_precedence(**char** \*op);

*//연산자의 우선순위를 결정하는 함수*

**int** is\_operator(**char** \*op);

*// 주어진 문자가 연산자인지 확인하는 함수*

**void** print(node \*cur);

*//트리를 순환하면 출력하는 함수*

node \*get\_operator(node \*cur);

*//트리 탐색 중 연산자를 구분하는 함수*

node \*get\_root(node \*cur);

*//트리 탐색 중 root노드를 탐색하기 위한 함수*

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

*// 두 노드 중 더 높은 우선순위의 노드를 계산하는 함수*

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

*// 가장 우선순위가 높은 노드를 구분하는 함수*

node \*insert\_node(node \*old, node \*new);

*// 트리에 새로 노드를 삽입하기 위한 함수*

node \*get\_last\_child(node \*cur);

*//노드의 가장 마지막 자식 노드를 탐색하는 함수*

**void** free\_node(node \*cur);

*//트리를 위해 지정한 메모리를 해제해주는 함수*

**int** get\_sibling\_cnt(node \*cur);

*//형제 노드의 총 개수를 계산하는 함수*

**int** make\_tokens(**char** \*str, **char** tokens[TOKEN\_CNT][MINLEN]);

*// 학생의 정답과 정답지의 정답을 토큰으로 분해할 때 사용하는 함수*

**int** is\_typeStatement(**char** \*str);

*//명령어를 구분하는 함수*

**int** find\_typeSpecifier(**char** tokens[TOKEN\_CNT][MINLEN]);

*//문자의 type을 결정하는 인자를 추출하는 함수*

**int** find\_typeSpecifier2(**char** tokens[TOKEN\_CNT][MINLEN]);

*//변수의 type을 결정하는 인자를 추출해내는 함수*

**int** is\_character(**char** c);

*// 문자인지 확인하는 함수*

**int** all\_star(**char** \*str);

*//문자열이 모두 \*문자인지 확인하는 함수*

**int** all\_character(**char** \*str);

*//문자열이 모두 문자로 이루어져 있는지 확인하는 함수*

**int** reset\_tokens(**int** start, **char** tokens[TOKEN\_CNT][MINLEN]);

*//분해한 토큰을 다시 문자열로 합치는 함수*

**void** clear\_tokens(**char** tokens[TOKEN\_CNT][MINLEN]);

*//모든 토큰을 널문자로 초기화하는 함수*

**int** get\_token\_cnt(**char** tokens[TOKEN\_CNT][MINLEN]);

*// NULL문자가 아닌 토큰의 개수를 세는 함수*

**char** \*rtrim(**char** \*\_str);

*//문자열의 뒤쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수*

**char** \*ltrim(**char** \*\_str);

*//문자열의 앞쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수*

**void** remove\_space(**char** \*str);

*//공백을 제거하는 함수*

**int** check\_brackets(**char** \*str);

*//여는 괄호와 닫는 괄호의 개수가 일치하는지 확인하는 함수*

**char**\* remove\_extraspace(**char** \*str);

*//문자열에서 1개 이상 중복된 공백을 1개만 남기고 삭제하는 함수*

11) main.c를 구성하는 함수 및 부가 기능 설명

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

*//프로그램의 메인 함수, ssu\_score()를 호출하고, 소요시간 계산을 위한 인자를 저장함*

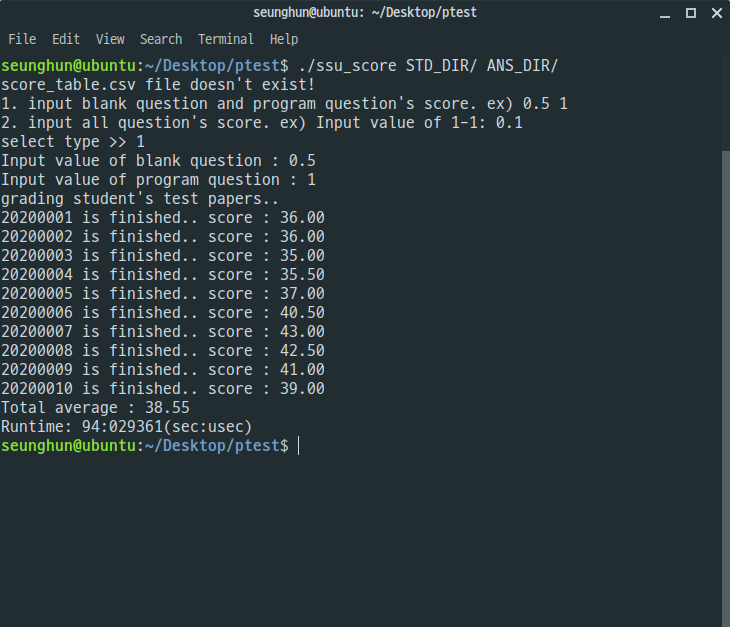
**void** ssu\_runtime(**struct** timeval \*begin\_t, **struct** timeval \*end\_t);

*//프로그램 총 소요시간 계산 후 출력하는 함수*

5. 테스트 및 결과

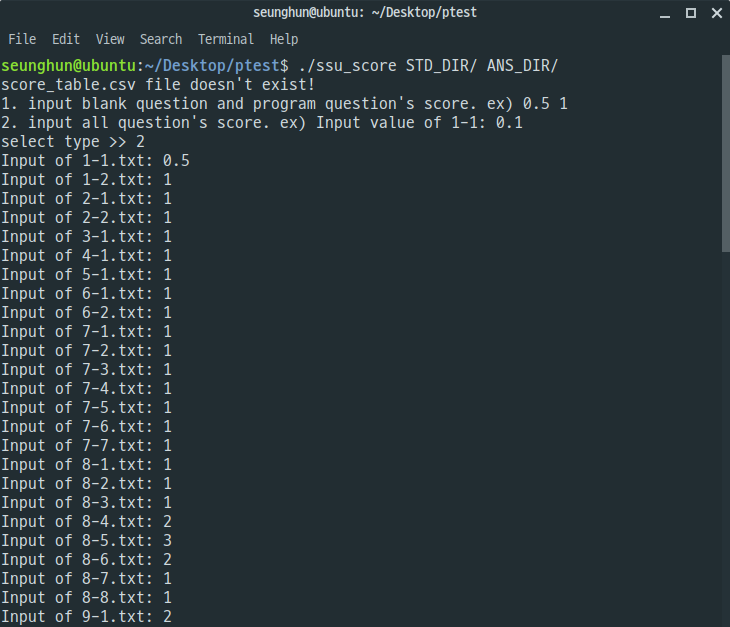
1) 옵션 없이 실행하는 경우

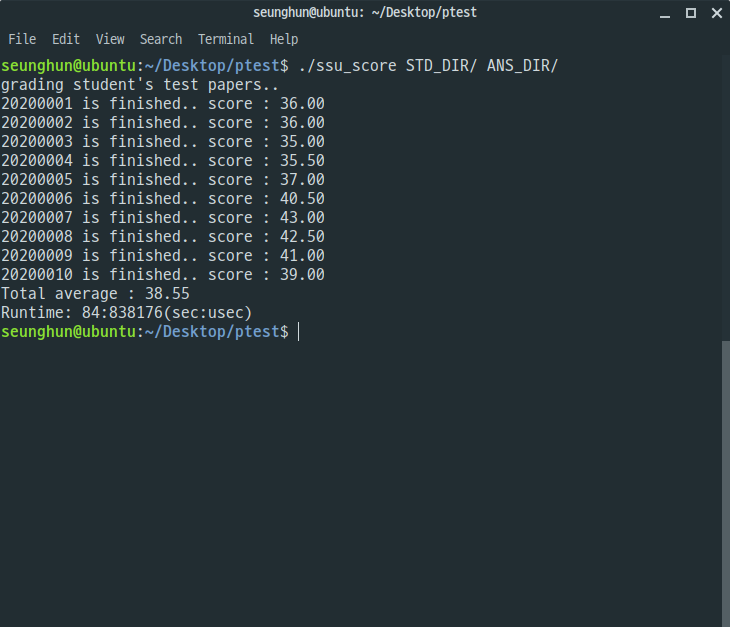
- score\_table.csv가 없는 프로그램 가장 초기에 실행하였을 경우



- score\_table.csv 의 문제의 배점을 저장하는 파일이 없는 경우, 프로그램 시작 시에 사용자로부터 문제의 배점을 입력받는다.

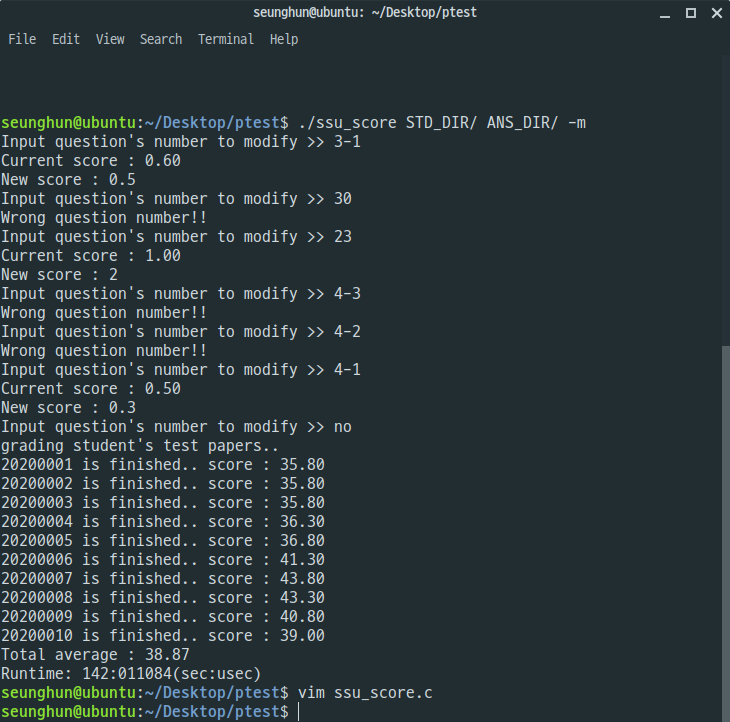
- 1번을 선택할 경우, 위와 같이 빈칸 문제와 프로그램 문제의 배점을 한번에 설정할 수 있고, 2번을 선택할 경우 아래의 경우처럼 각각의 문제에 대해 입력을 진행할 수 있다.





- 만약, 이미 score\_table.csv의 문제 배점 파일이 존재할 경우, 바로 채점을 시작한다.

2) m옵션을 실행하는 경우



- m옵션을 사용할 경우, 사용자가 원하는 문제의 배점을 수정할 수 있다.

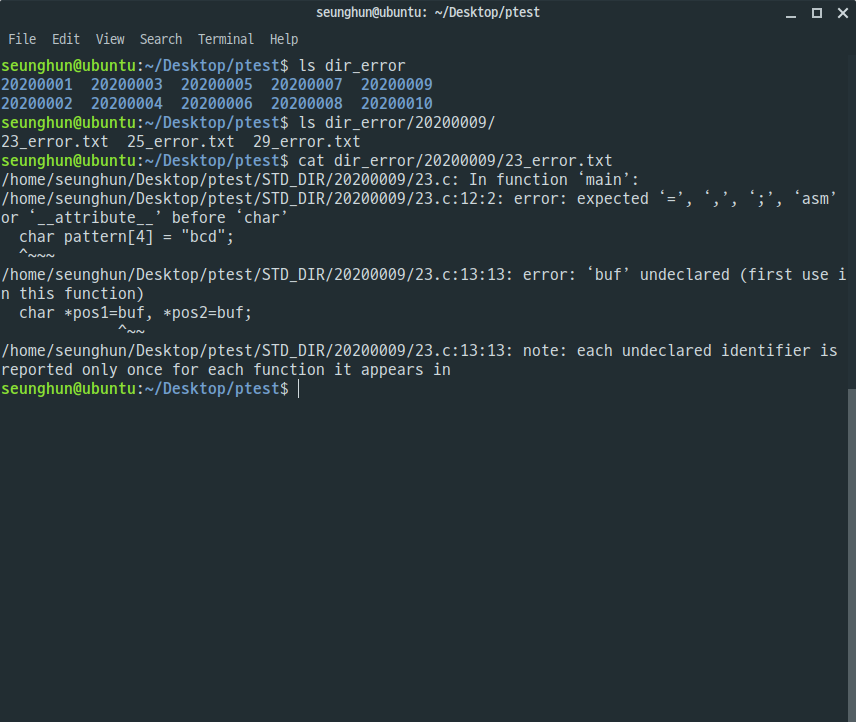
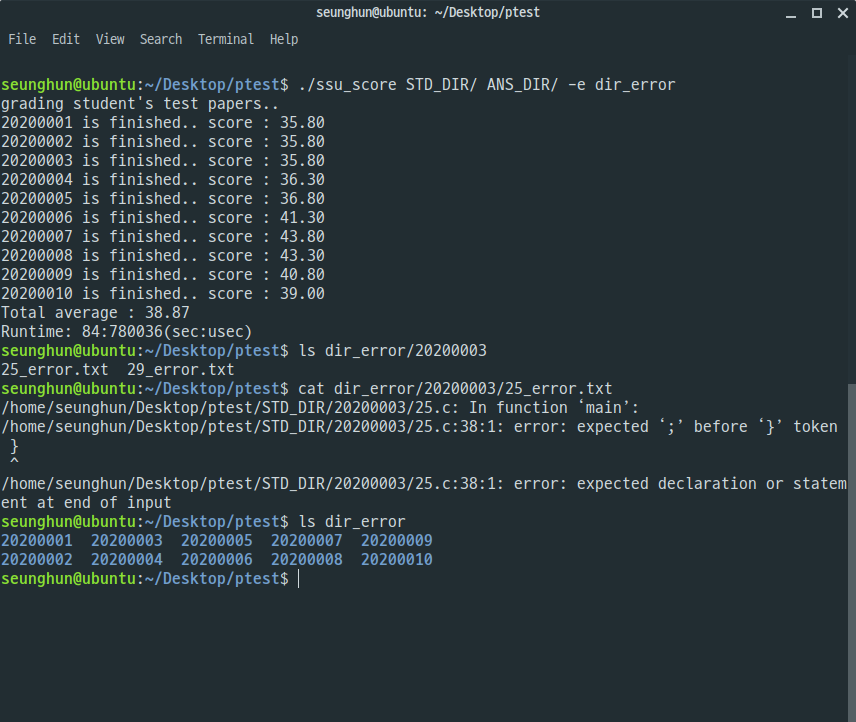
- 채점 시작 전 사용자에게 배점을 수정할 문제 번호를 입력받고, 그 문제의 현재 배점을 출력하고 새 배점을 입력받는다.

- 없는 문제 번호를 입력하였을 경우, 오류 메시지를 출력하고, 재입력을 대기한다.

- 사용자가 “no”를 입력하기 전까지 반복하여 문제의 배점을 수정한다.

- 사용자가 “no”를 입력하고 배점 수정을 마치면 채점을 시작한다.

3) e 옵션을 실행한 경우



- e 옵션을 실행시에 에러파일을 저장할 디렉터리의 이름도 함께 입력하여야 한다.

- e 옵션이 적용되어있을 때, 디렉터리이름/학번/문제번호\_error.txt로 틀린 문제의 에러 메시지가 저장된다.

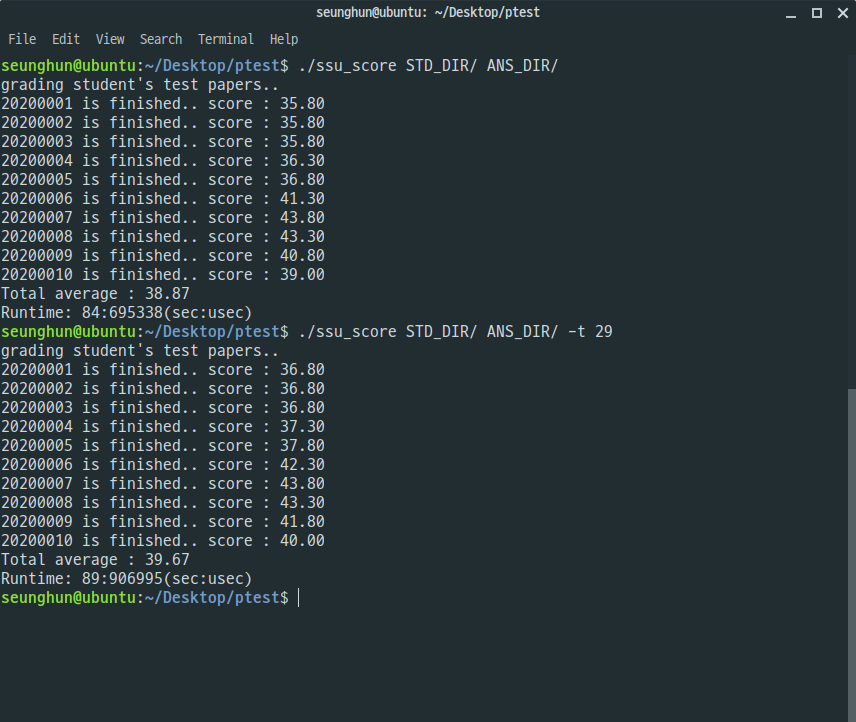
- 실제로 파일에 접근해 보면 에러 메시지가 저장되어있는 것을 확인할 수 있다.

- 채점을 진행한 모든 학생에 대해 틀린문제에 대한 폴더와 파일을 생성한다.

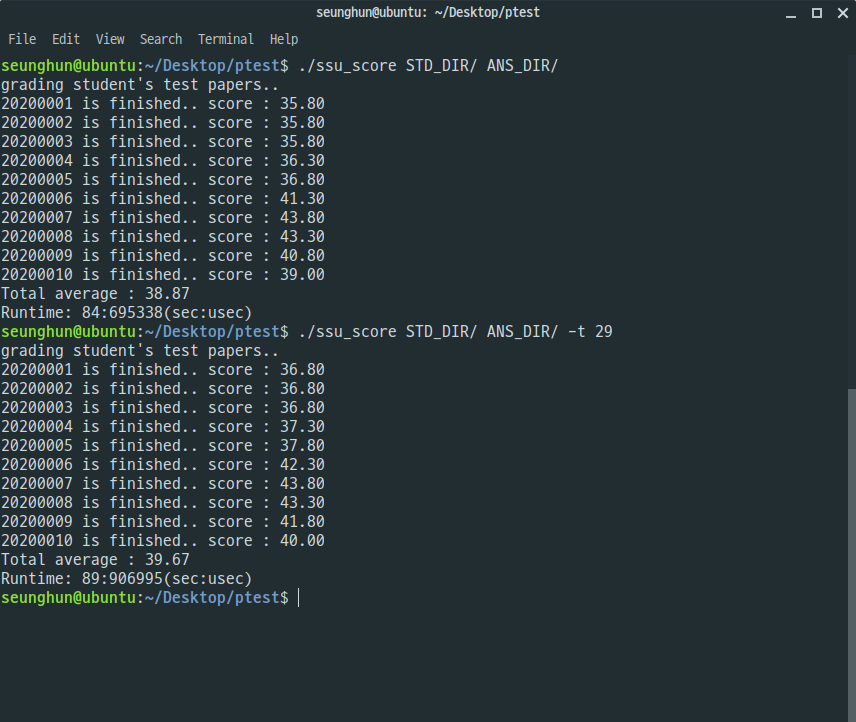
4) t 옵션을 실행한 경우

- t 옵션은 문제를 컴파일할 때, -lpthread 옵션을 포함하여 컴파일하게 하는 함수이다.

- 아래의 경우는 t 옵션을 적용하지 않은 경우이다.

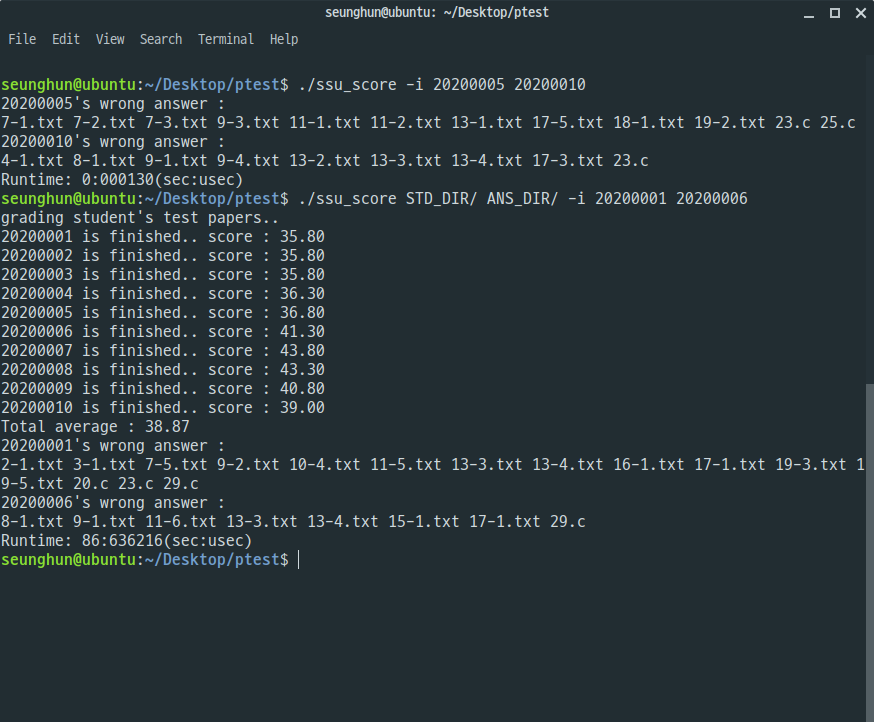


- 아래의 경우는 29번 문제에 t 옵션을 적용한 경우이다.

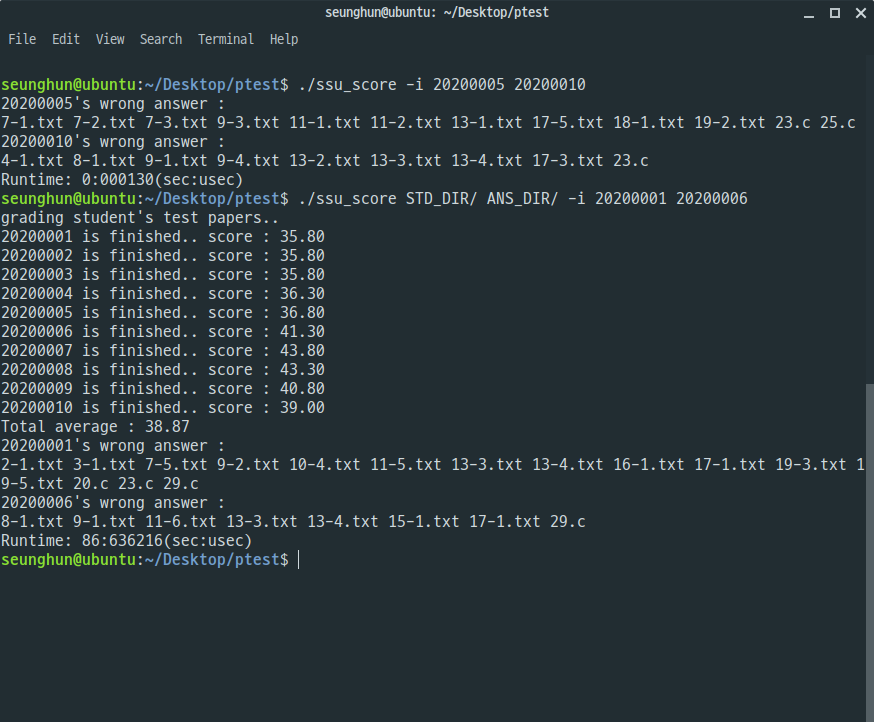


- 테스트를 실행할 당시, 29번 문제에는 1점의 배점이 부여되어 있었다. 29번이 정답처리된 학생은 1점이 상승한 것을 확인할 수 있다.

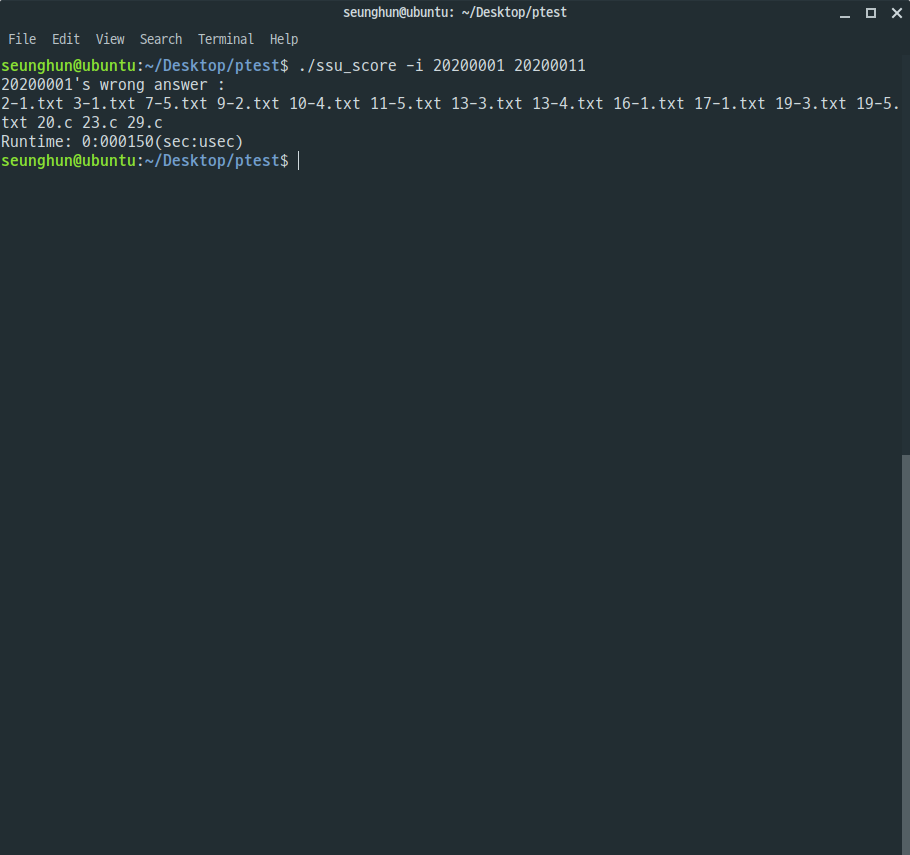
5) i 옵션을 실행한 경우



- 학생디렉터리, 정답디렉터리없이 단독 실행 한 경우, 바로 학생들의 틀린 문제를 출력하고 종료한다.

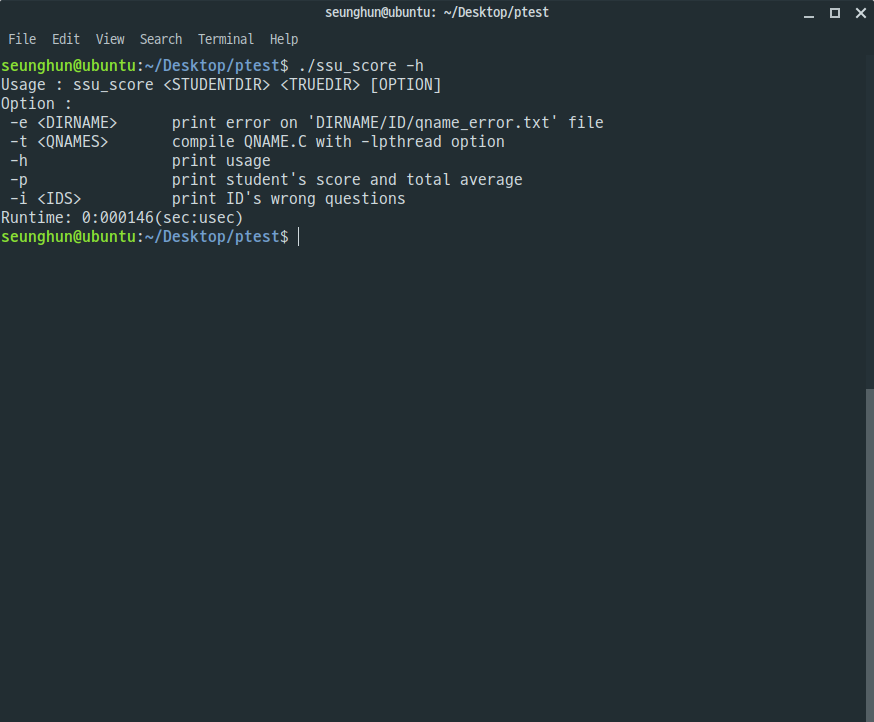


- 학생디렉터리, 정답디렉터리를 입력하였을 경우, 채점을 진행한 후에, 틀린 문제 파일을 출력한다.

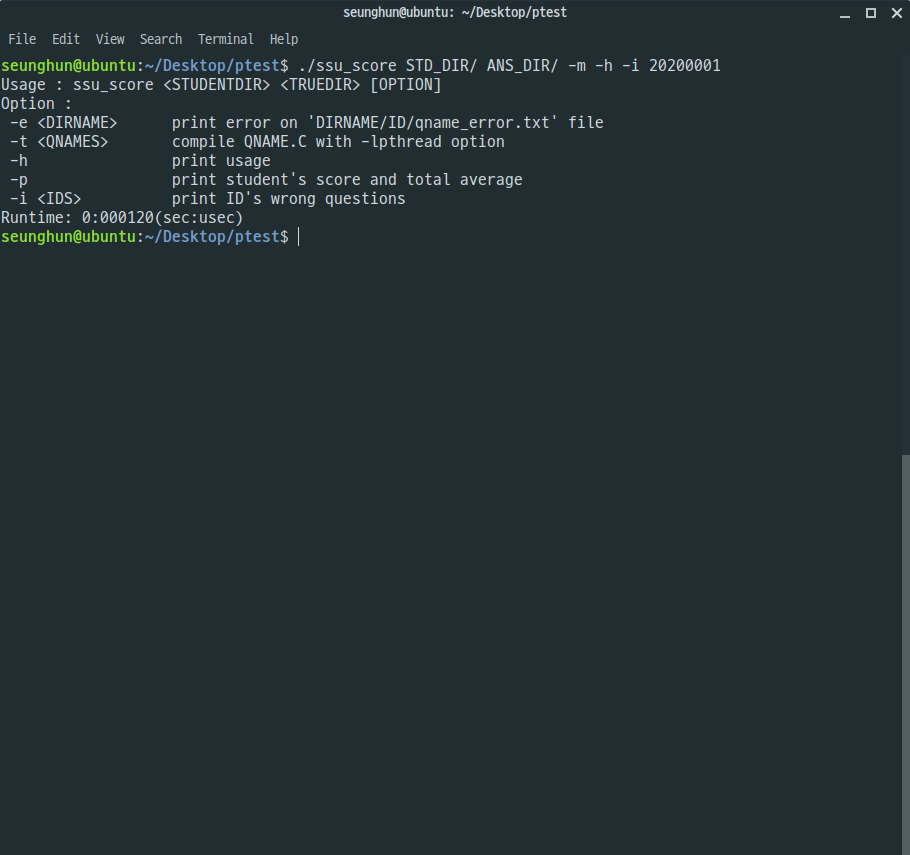


- 없는 학생이 인자로 입력되었을 경우, 그 입력된 인자는 무시하고 프로그램을 진행한다.

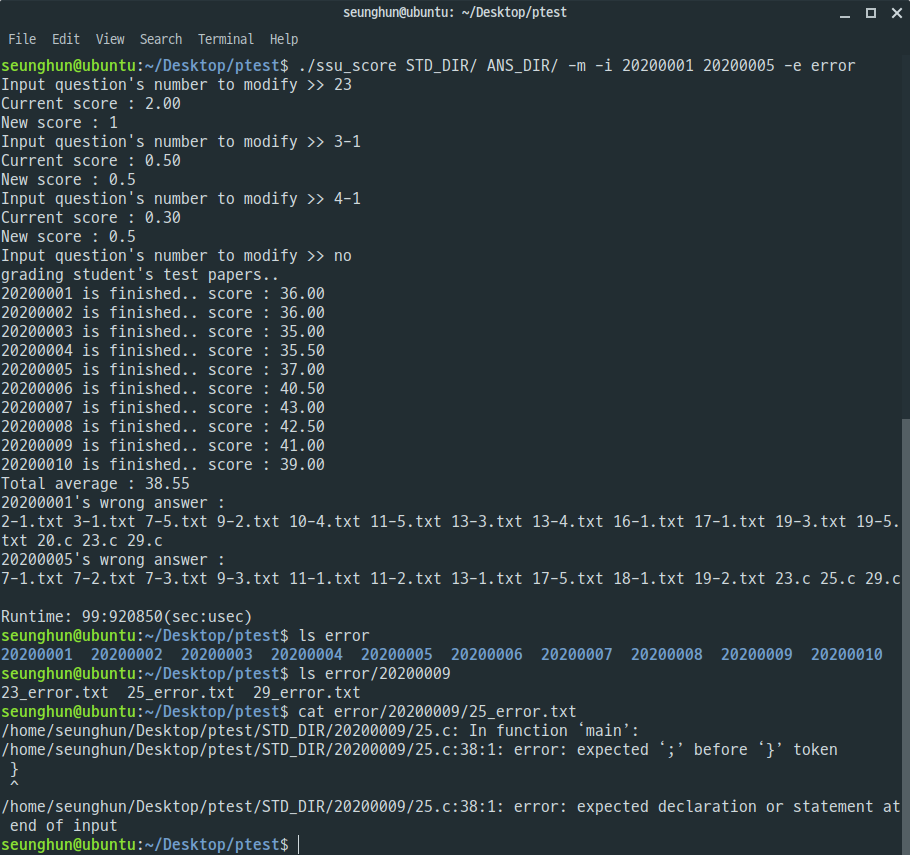
6) h 옵션을 실행한 경우



- h 옵션을 실행한 경우, 프로그램의 사용법을 출력하고 프로그램을 종료한다.

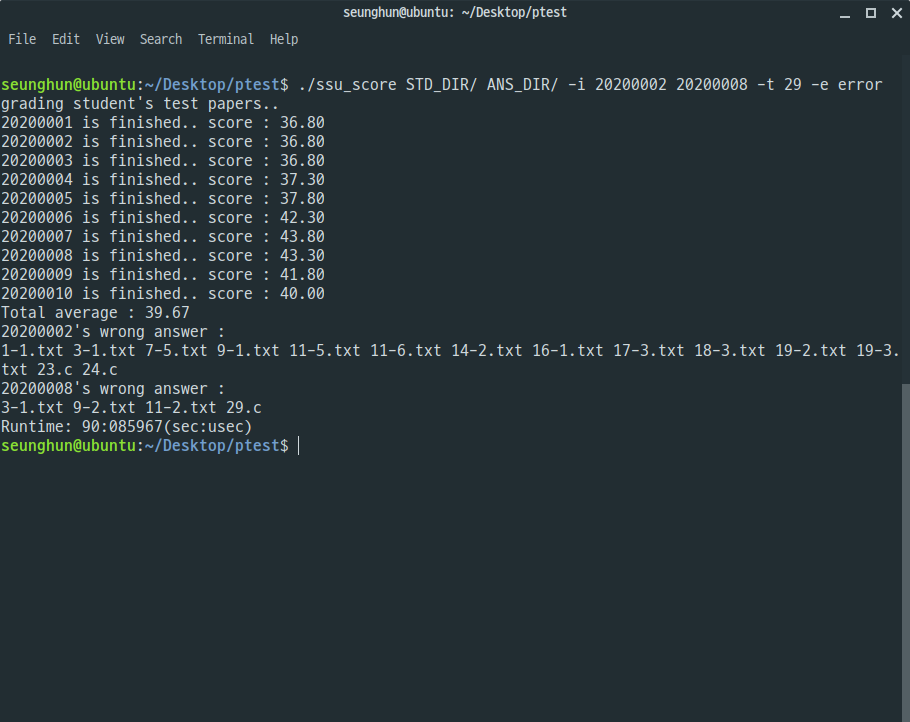


- 다른 옵션과 같이 h 옵션이 사용될 경우, 다른 옵션은 무시하고 사용법을 출력하고 프로그램을 종료한다.

7) 여러 옵션을 동시에 사용한 경우

- 위의 실행 화면은 i 옵션과 m 옵션, e 옵션을 동시에 진행한 테스트이다.

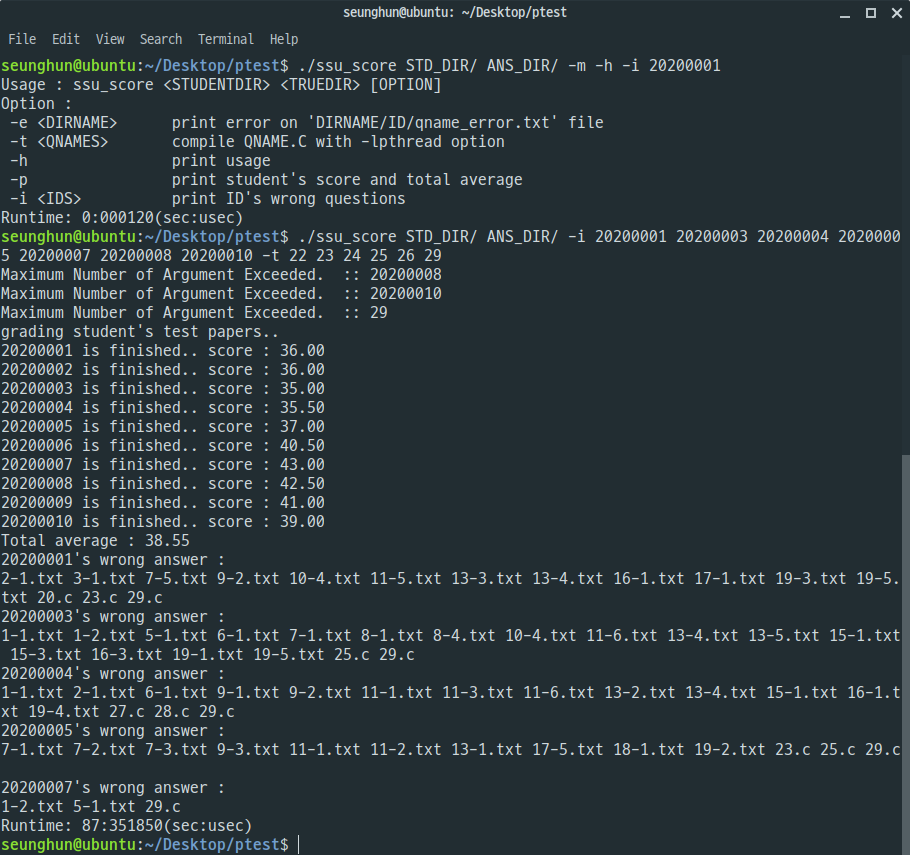
- 프로그램 실행 전 m 옵션을 통해 문제의 번호를 수정하고, 채점을 진행한다. 이후 i옵션을 통해 지정된 학생의 틀린 문제 번호를 출력하고, 디렉터리를 확인해보면 e 옵션을 통해 오류 메시지가 저장된 폴더가 생성되었음을 알 수 있다.



- i 옵션과 t 옵션과 e 옵션을 동시에 적용한 테스트이다.

- 문제 없이 작동하고, 옵션에 따라 출력된 사항을 확인할 수 있다.

8) 인자가 5개 이상 입력될 경우



- 인자가 5개 이상 입력되었을 경우, 추가로 입력되어 초과된 인자가 무엇인지 프로그램 처음에 출력하여주고, 채점을 수행한다. 이후 5개의 인자에 대해서 옵션을 적용하여 출력한다.

9) gettimeofday() 함수의 사용

- 모든 프로그램 수행 캡쳐화면에서, 프로그램이 완료되기 직전, 프로그램의 실행시간이 출력되는 것을 확인할 수 있다.

6. 소스코드

- 헤더파일을 포함한 전체 프로그램의 소스코드이다.

**<main.c>**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/time.h>

#include "ssu\_score.h"

#define SECOND\_TO\_MICRO 1000000 // 1초는 백만마이크로초

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); //시작시간 저장

ssu\_score(argc, argv);

gettimeofday(&end\_t, NULL); //끝시간 저장

ssu\_runtime(&begin\_t, &end\_t); //시간계산, 끝시간 - 처음 시간

exit(0); //프로그램 종료

}

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

{

end\_t->tv\_sec -= begin\_t->tv\_sec; //끝난 시간에서 처음시간을 뺌

if(end\_t->tv\_usec < begin\_t->tv\_usec){

end\_t->tv\_sec--;

end\_t->tv\_usec += SECOND\_TO\_MICRO;

} //나중의 us값이 작을경우 1초를 뺴서 us값이 더해줌

end\_t->tv\_usec -= begin\_t->tv\_usec;

printf("Runtime: %ld:%06ld(sec:usec)\n", end\_t->tv\_sec, end\_t->tv\_usec);

} //프로그램 총 소요시간 계산 후 출력하는 함수

**<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

#endif

#ifndef WARNING

#define WARNING -0.1

#endif

#ifndef ERROR

#define ERROR 0

#endif

#define FILELEN 64

#define BUFLEN 1024

#define SNUM 100

#define QNUM 100

#define ARGNUM 5

struct ssu\_scoreTable{

char qname[FILELEN];

double score;

}; //문제 번호와 그 문제의 배점을 저장하기 위한 구조체

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

//main함수의 호출로 불러져 전체 채점을 시작하게 하는 함수,

int check\_option(int argc, char \*argv[]);

//사용자로부터 인자로 받은 옵션을 확인하여 옵션에 알맞은 함수를 실행시키는 함수

void print\_usage();

//-h 옵션 사용법 출력에 사용하는 함수

void score\_students();

//학생들의 득점을 파일에 저장하고 총점을 계산하는 함수

double score\_student(int fd, char \*id);

//실질적으로 학생들의 점수를 계산하는 함수

void write\_first\_row(int fd);

//학생 배점 파일 첫줄에 문제 번호를 작성하는 함수

char \*get\_answer(int fd, char \*result);

//학생 파일 또는 정답 파일에서 정답 텍스트만을 추출하는 함수

int score\_blank(char \*id, char \*filename);

//빈칸문제를 채점하는 함수

double score\_program(char \*id, char \*filename);

//프로그램 문제를 채점하는 함수

double compile\_program(char \*id, char \*filename);

//학생 및 정답의 프로그램 문제를 컴파일하는 함수

int execute\_program(char \*id, char \*filname);

// 컴파일된 프로그램 문제를 직접 실행시키는 함수

pid\_t inBackground(char \*name);

//백그라운드에서 실행중인 학생 또는 정답 프로그램의 프로세스 ID를 읽어옴

double check\_error\_warning(char \*filename);

// 학생 프로그램의 에러 또는 경고의 개수를 계산하는 함수

int compare\_resultfile(char \*file1, char \*file2);

//학생 프로그램의 결과물과 정답 프로그램의 결과물을 서로 비교하는 함수

void do\_iOption(char (\*ids)[FILELEN]);

//사용자가 i옵션을 입력했을 때, 입력받은 학생의 틀린 문제를 출력하는 함수

int is\_exist(char (\*src)[FILELEN], char \*target);

// 학번을 바탕으로 존재하는 학생인지 판단하는 함수

int is\_thread(char \*qname);

//문제 번호를 바탕으로 –lpthread 옵션을 주고 컴파일해야하는 문제인지 확인하는 함수

void redirection(char \*command, int newfd, int oldfd);

//파일 디스크립터를 다른 곳으로 옮겼다가 복구시키는 함수

int get\_file\_type(char \*filename);

//파일이 텍스트 파일인지 소스코드인지 구분하는 함수

void rmdirs(const char \*path);

//디렉터리를 삭제하는 함수

void to\_lower\_case(char \*c);

// 모든 영문자를 소문자로 교체하는 함수

void set\_scoreTable(char \*ansDir);

//문제 배점 파일을 생성하는 함수

void read\_scoreTable(char \*path);

//문제 배점 파일을 읽어와 배열로 저장하는 함수

void make\_scoreTable(char \*ansDir);

//문제 배점 파일에 문제번호를 불러오는 함수

void write\_scoreTable(char \*filename);

//문제 배점 파일에 문제의 배점을 입력하고 저장하는 함수

void set\_idTable(char \*stuDir);

//학생들의 점수가 기록될 파일에 학생들의 학번을 입력하는 함수

int get\_create\_type();

//초기에 문제 배점 파일이 없는 경우 사용자로부터 문제의 배점을 입력받기 위한 함수

void sort\_idTable(int size);

//학생들의 학번을 기준으로 학생 득점 파일에 학번을 작성하기 전 배열의 학번을 정렬하는 함수

void sort\_scoreTable(int size);

//문제번호 순서대로 문제 배점 배열을 정렬하는 함수

void get\_qname\_number(char \*qname, int \*num1, int \*num2);

//배열에서 문제 번호를 불러오는 함수

void do\_mOption(void);

//m옵션을 실행하는 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]; //학생정답폴더의 디렉터리를 저장

char ansDir[BUFLEN]; //정답폴더의 디렉터리를 저장

char errorDir[BUFLEN]; //e옵션이 적용 되었을 때, 에러를 저장할 디렉터리 이름을 저장

char threadFiles[ARGNUM][FILELEN]; //t옵션 적용 시, -lpthread 옵션을 통해 컴파일 할 문제 번호를 저장함

char cIDs[ARGNUM][FILELEN]; //옵션의 조건으로 입력되는 데이터를 저장하기 위한 배열

int eOption = false;

int tOption = false;

int iOption = false;

int hOption = false;

int mOption = false;

//각 옵션들을 위한 스위치

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

{

char saved\_path[BUFLEN];

int i, j, log;

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

if(!strcmp(argv[i], "-h")){

print\_usage();

return;

}

} // h옵션이 있을 경우 h옵션을 출력하고 프로그램 종료

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

if(argc >= 3 && (log = strcmp(argv[1], "-i")) != 0){

strcpy(stuDir, argv[1]);

strcpy(ansDir, argv[2]);

} //옵션 i가 아닐경우 분리

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

exit(1);

//입력된 옵션을 체크함

j = 0;

if(!eOption && !tOption && iOption && !log) {

do\_iOption(cIDs);

return;

} //i옵션만 활성화되어있고, 사용자가 문제 디렉터리를 입력하지 않았을 경우

//i옵션과 관련된 함수 실행

getcwd(saved\_path, BUFLEN);

//현재 작업디렉터리의 경로를 얻어옴

if(chdir(stuDir) < 0){

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

return;

} //현재 작업디렉터리 변경

getcwd(stuDir, BUFLEN);

//학생 디렉터리의 정보 저장

chdir(saved\_path); //정답 디렉터리 확인

if(chdir(ansDir) < 0){

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

return;

} //정답디렉터리의 존재 유무 확인

getcwd(ansDir, BUFLEN);

//정답디렉터리 불러옴

chdir(saved\_path);

//정답디렉터리 정보저장

set\_scoreTable(ansDir); //문제의 점수를 저장하는 csv파일을 생성

set\_idTable(stuDir); //학생의 득점을 저장하기 위한 csv 파일 생성

if(mOption) {

do\_mOption();

} //m옵션을 입력하였을 경우, m옵션 함수 실행

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

score\_students(); //학생들의 답안 채점

if(iOption)

do\_iOption(cIDs); //다른 조건들을 실행하고 i옵션이 있을 경우 실행함

return;

}

void do\_mOption(void) {

int i, ft = 0;

char tmp[30];

int cnt;

double ns;

while(1) {

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

scanf("%s", tmp);

if(!strcmp(tmp, "no"))

break;

//no라고 입력한 경우 문제 배점 수정 종료

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

cnt = strncmp(tmp, score\_table[i].qname, strlen(tmp));

if(!cnt) {

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

printf("New score : ");

scanf("%lf", &ns);

score\_table[i].score = ns;

ft = 1;

}

} //문제의 새 배점을 입력받아 출력함

if(!ft) //사용자가 없는 문제 번호를 입력할 경우, 에러 출력 후 새 입력을 대기함

printf("Wrong question number!!\n");

} // break조건이 나올떄 까지 반복하여 입력받음

write\_scoreTable("score\_table.csv"); //바뀐 배점으로 score\_table을 새로 작성

return;

} //m옵션이 주어졌을 경우 문제의 배점을 수정하는 함수

int check\_option(int argc, char \*argv[])

{ //입력된 옵션을 체크 하는 함수

int i, j;

int c;

while((c = getopt(argc, argv, "me:ti:")) != -1)

{ //지정된 옵션이 입력되었는지 확인함

switch(c){

case 'm': //m옵션시 옵션 스위치를 켬

mOption = true;

break;

case 'e': //e옵션이 입렫외었을 경우 지정된 이름의 error 디렉터리를 생성함

eOption = true;

strcpy(errorDir, optarg);

if(access(errorDir, F\_OK) < 0)

mkdir(errorDir, 0755);

else{

rmdirs(errorDir);

mkdir(errorDir, 0755);

}

break;

case 't': //t옵션의 스위치를 켜고 어느 문제를 thread를 이용하여 채점할지 기억함

tOption = true;

i = optind;

j = 0;

while(i < argc && argv[i][0] != '-'){

if(j >= ARGNUM)

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

else

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

i++;

j++;

}

break;

case 'i': //i옵션 인 경우 어느 학생의 틀린 문제를 출력할지 저장함

iOption = true;

i = optind - 1;

j = 0;

while(i < argc && argv[i][0] != '-'){

if(j >= ARGNUM)

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

else

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

i++;

j++;

}

break;

case '?': //그 외의 지정되지 않은 옵션이 들어올 경우 판단불가능을 반환함

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

return false;

}

} //모든 옵션을 판단한 후 while문 종료

return true;

}

void do\_iOption(char (\*ids)[FILELEN])

{ //학생의 틀린 문제 번호를 출력하는 함수 -i 옵션을 위함

FILE \*fp;

char tmp[BUFLEN];

char ques\_num[BUFLEN];

char ques\_num\_backup[BUFLEN];

int i = 0;

char \*p, \*saved, \*qn, \*nextptr;

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

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

return;

} //score.csv를 오픈함, 실패시 에러출력

fscanf(fp, "%s\n", ques\_num);

strcpy(ques\_num\_backup, ques\_num);

//문제번호 줄 읽어옴

//printf("%s\n", ques\_num);

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

{

p = strtok(tmp, ",");

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

continue;

//학생의 학번이 올바른지 판단함

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

strcpy(ques\_num, ques\_num\_backup);

qn = strtok\_r(ques\_num, ",", &nextptr);

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

if(!strcmp(p, "0"))

printf("%s ", qn);

qn = strtok\_r(NULL, ",", &nextptr);

} //학생의 점수가 0인 문제의 반호를 문제의 끝까지 출력함

printf("\n");

}

fclose(fp);

}

int is\_exist(char (\*src)[FILELEN], char \*target)

{ //입력된 학생의 학번이 실제로 목록에 존재하는지 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;

} //파일의 첫 줄부터 돌면서 학생이 존재하는지 확인함

return false; //확인되지 않을 경우 false 반환

}

void set\_scoreTable(char \*ansDir)

{ //문제의 배점을 저장하는 함수

char filename[FILELEN];

sprintf(filename, "./%s", "score\_table.csv"); //I changed

//현재 디렉터리에 문제 배점 저장하는 파일 생성

if(access(filename, F\_OK) == 0)

read\_scoreTable(filename); //배점 파일을 읽어옴

else{

make\_scoreTable(ansDir); //배점 파일을 만듦

write\_scoreTable(filename); //배점 파일을 저장함

}

}

void read\_scoreTable(char \*path)

{ //배점 파일을 읽어오는 함수

FILE \*fp;

char qname[FILELEN];

char score[BUFLEN];

int idx = 0;

if((fp = fopen(path, "r")) == NULL){

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

return ;

} // 배점 파일 열기

while(fscanf(fp, "%[^,],%s\n", qname, score) != EOF){

strcpy(score\_table[idx].qname, qname);

score\_table[idx++].score = atof(score);

} //문제 배점 파일을 한 줄 씩 읽어 문제 번호와 점수를 score\_table 배열에 저장함

fclose(fp);

}

void make\_scoreTable(char \*ansDir)

{ //score\_table.csv 초기에 배점을 사용자에게 입력받는 함수

int type, num;

double score, bscore, pscore;

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

DIR \*dp, \*c\_dp;

char tmp[BUFLEN];

int idx = 0;

int i;

num = get\_create\_type();

//사용자의 메뉴 입력을 받아옴

if(num == 1)

{

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

scanf("%lf", &bscore);

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

scanf("%lf", &pscore);

} //빈칸 문제와 프로그램 눈제의 점수를 설정함

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

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

return;

} //dp는 정답 폴더의 디렉터리 포인터

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

{

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

continue;

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

if(get\_file\_type(dirp->d\_name) != -1)

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

}//정답 폴더에 있는 문제 번호를 불러옴, 어떠한 문제 번호들을 분석함

closedir(dp);

sort\_scoreTable(idx); //문제번호 순으로 문제 정렬

for(i = 0; i < idx; i++)

{

type = get\_file\_type(score\_table[i].qname);

//문제 번호를 보고 문제가 텍스트 문제인지 c프로그램 파일인지 구분함

if(num == 1)

{ //사용자가 1번을 선택하였을 경우, 모든 빈칸문제는 bscore로, 모든 프로그램 문제는 pscore로 설정

if(type == TEXTFILE)

score = bscore;

else if(type == CFILE)

score = pscore;

}

else if(num == 2)

{ //사용자가 2번을 선택한 경우, 모든 문제에 대해 입력받음

printf("Input of %s: ", score\_table[i].qname);

scanf("%lf", &score);

}

//입력받은 점수를 score\_table배열에 저장

score\_table[i].score = score;

}

}

void write\_scoreTable(char \*filename)

{ //배열에 저장해 두었던 문제별 배점데이터를 csv 파일에 쓰는 함수

int fd;

char tmp[BUFLEN];

int i;

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

if((fd = creat(filename, 0666)) < 0){

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

return;

} //score\_table 파일을 생성함

for(i = 0; i < num; i++)

{

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

break;

sprintf(tmp, "%s,%.2f\n", score\_table[i].qname, score\_table[i].score);

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

} //파일에 문제 번호와 점수를 저장함

close(fd);

}

void set\_idTable(char \*stuDir)

{ //학생 디렉터리를 읽고 학번을 추출해서 저장하는 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);

} //학생들의 정답이 있는 폴더을 오픈함

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

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

continue;

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

stat(tmp, &statbuf);

if(S\_ISDIR(statbuf.st\_mode))

strcpy(id\_table[num++], dirp->d\_name);

else

continue;

} //폴더안을 돌면서 학생의 학번을 수집하여 배열에 저장함

sort\_idTable(num); // 학번순으로 정렬

}

void sort\_idTable(int size)

{ //불러와 저장한 학번을 정렬하는 함수

int i, j;

char tmp[10];

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)

{ //문제번호 순서대로 문제배점 배열을 정렬하는 함수

int i, j;

struct ssu\_scoreTable tmp;

int num1\_1, num1\_2;

int num2\_1, num2\_2;

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);

get\_qname\_number(score\_table[j+1].qname, &num2\_1, &num2\_2);

//비교를 위한 두개의 문제번호를 불러옴

if((num1\_1 > num2\_1) || ((num1\_1 == num2\_1) && (num1\_2 > num2\_2))){

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]));

} // 문제번호순대로 배열의 원소를 교환함

}

}//버블 정렬을 이용하여 문제번호 순으로 정렬

}

void get\_qname\_number(char \*qname, int \*num1, int \*num2)

{ //베열에서 문제번호를 불러오는 함수

char \*p;

char dup[FILELEN];

strncpy(dup, qname, strlen(qname));

\*num1 = atoi(strtok(dup, "-."));

//문제 번호를 - 앞까지 끊어서 저장

p = strtok(NULL, "-.");

if(p == NULL)

\*num2 = 0;

else

\*num2 = atoi(p);

} // - 이후로도 문제번호가 있을경우 별도로 .txt 또는 .c 앞까지 읽어서 저장

int get\_create\_type()

{ //초기에 문제배점 테이블이 없을때 사용자로부터 입력을 받기 위한 메뉴 함수

int num;

while(1)

{ //1번 모든 빈칸 문제 프로그램 문제를 한번에 설정함, 2번 각각의 문제에 대해 설정함

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

printf("1. input blank question and program question's score. ex) 0.5 1\n");

printf("2. input all question's score. ex) Input value of 1-1: 0.1\n");

printf("select type >> ");

scanf("%d", &num);

if(num != 1 && num != 2)

printf("not correct number!\n"); //1 또는 2가 들어올때 까지 반복

else

break;

}

return num; //사용자의 입력 반환

}

void score\_students()

{ //학생들의 득점을 저장하는 함수

double score = 0;

int num;

int fd;

char tmp[BUFLEN];

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

if((fd = creat("score.csv", 0666)) < 0){

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

return;

} //학생들의 점수를 저장할 파일을 생성함

write\_first\_row(fd); //score.csv에 문제번호 입력

for(num = 0; num < size; num++)

{

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

break;

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

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

//처음에 학생의 학번 입력

score += score\_student(fd, id\_table[num]);

} //학생의 득점을 순차적으로 입력함 , score 변수에 평균 계산을 위해 학생의 총 득점을 더해둠

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

//학생들의 총 평균 출력

close(fd); //학생들의 득점 파일 close

}

double score\_student(int fd, char \*id)

{ //학생들의 점수를 실질적으로 계산하는 함수

int type;

double result;

double score = 0;

int i;

char tmp[BUFLEN];

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

//문제 갯수만큼 for문을 돌리기 위해 문제 갯수 계산

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

{

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

break;

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

//학생들의 정답 위치로 이동

if(access(tmp, F\_OK) < 0)

result = false;

else

{

if((type = get\_file\_type(score\_table[i].qname)) < 0)

continue;

//텍스트파일 또는 프로그램 파일이 아닐 경우 무시

if(type == TEXTFILE)

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

else if(type == CFILE)

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

} //빈칸문제일 경우 score\_blank에서 점수 계산

//프로그램 문제일 경우 score\_program에서 점수 계산

if(result == false)

write(fd, "0,", 2); //틀렸을 경우 0점 입력

else{

if(result == true){

score += score\_table[i].score;

sprintf(tmp, "%.2f,", score\_table[i].score);

}//맞았을 경우 배열에 점수 저장, 학생의 총점 계산을 위해 score에 저장

else if(result < 0){

score = score + score\_table[i].score + result;

sprintf(tmp, "%.2f,", score\_table[i].score + result);

} //result가 0보다 작을 경우 감점도 점수에 더해줌

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

} //계산된 점수를 씀

}

printf("%s is finished.. score : %.2f\n", id, score);

//점수 계산이 끝난 후 학생별 점수 출력

sprintf(tmp, "%.2f\n", score);

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

//학생의 총점을 파일에 쓰고 반환함

return score;

}

void write\_first\_row(int fd)

{ // 학생 배점 파일 첫줄에 문제 번호를 작성하는 함수

int i;

char tmp[BUFLEN];

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

write(fd, ",", 1);

//가장 왼쪽 위의 칸은 빈칸

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

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

break;

//문제의 끝에 도달하면 읽기 종료

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

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

}

write(fd, "sum\n", 4); // 문제 번호를 입력하고 마지막에 sum 입력칸을 만들어줌

}

char \*get\_answer(int fd, char \*result)

{ //답안에서 정답을 뽑아오는 함수

char c;

int idx = 0;

memset(result, 0, BUFLEN);

while(read(fd, &c, 1) > 0)

{

if(c == ':')

break;

//정답을 콜론으로 구분하여 하나씩 추출함

result[idx++] = c;

}

if(result[strlen(result) - 1] == '\n')

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

//개행 문자를 널문자로 교체하여 문장의 끝 표시

return result;

}

int score\_blank(char \*id, char \*filename)

{ //빈칸 문제 채점 함수

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];

int fd\_std, fd\_ans;

int result = true;

int has\_semicolon = false;

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

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.')));

sprintf(tmp, "%s/%s/%s", stuDir, id, filename);

fd\_std = open(tmp, O\_RDONLY);

strcpy(s\_answer, get\_answer(fd\_std, s\_answer));

//학생답을 불러옴

if(!strcmp(s\_answer, "")){

close(fd\_std);

return false;

} //학생 답이 없으면 0점 처리

if(!check\_brackets(s\_answer)){

close(fd\_std);

return false;

} //괄호 갯수가 맞지 않으면 0점 처리

strcpy(s\_answer, ltrim(rtrim(s\_answer)));

//왼쪽 오른쪽 공백 지우기

if(s\_answer[strlen(s\_answer) - 1] == ';'){

has\_semicolon = true; //세미콜론 유무 확인

s\_answer[strlen(s\_answer) - 1] = '\0'; //세미콜론 삭제

}

if(!make\_tokens(s\_answer, tokens)){

close(fd\_std);

return false;

} //토큰 만들기

idx = 0;

std\_root = make\_tree(std\_root, tokens, &idx, 0);

//학생 정답을 이용해서 트리를 생성함

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

fd\_ans = open(tmp, O\_RDONLY); //정답파일 불러오기

while(1)

{

ans\_root = NULL; //정답이 여러개일수 있으니 NULL

result = true;

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

memset(tokens[idx], 0, sizeof(tokens[idx]));

//토큰 초기화

strcpy(a\_answer, get\_answer(fd\_ans, a\_answer));

if(!strcmp(a\_answer, ""))

break;

//정답 파일이 비었을 경우 break

strcpy(a\_answer, ltrim(rtrim(a\_answer)));

//정답의 양쪽 공백 제거

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; //정답 파일에 대한 토큰 생성

idx = 0;

ans\_root = make\_tree(ans\_root, tokens, &idx, 0);

//정답파일의 토큰을 이용해서 트리 생성

compare\_tree(std\_root, ans\_root, &result);

// 학생의 정답 트리와 정답지의 트리를 서로 비교한다

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;

//트리가 일치 할 경우 정답처리

}

}

close(fd\_std);

close(fd\_ans);

if(std\_root != NULL)

free\_node(std\_root);

if(ans\_root != NULL)

free\_node(ans\_root);

return false;

} //트리가 같지 않을 경우 오답 처리

double score\_program(char \*id, char \*filename)

{ //프로그램 문제를 채점하는 함수

double compile;

int result;

compile = compile\_program(id, filename);

//컴파일 한 결과를 받아와서 판단함

if(compile == ERROR || compile == false)

return false;

//컴파일이 안되거나 에러가 나올 경우 오답처리

result = execute\_program(id, filename);

//컴파일이 된 프로그램 문제에 대해 실행 여부 확인

if(!result)

return false;

//결과가 다를 경우 오답 처리

if(compile < 0)

return compile;

//오류 발생시 감점 적용

return true; //맞았을 경우 정답 처리

}

int is\_thread(char \*qname)

{ //t옵션을 적용해야 하는 문제인지 배열에서 검색하는 함수

int i;

int size = sizeof(threadFiles) / sizeof(threadFiles[0]);

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

if(!strcmp(threadFiles[i], qname))

return true;

}

return false;

} //t 옵션으로 지정된 문제 번호에 컴파일시 -lpthread 옵션 추가

double compile\_program(char \*id, char \*filename)

{ //프로그램 문제를 컴파일 하는 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));

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.')));

isthread = is\_thread(qname);

//-t 옵션으로 지정된 문제인지 판단

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

sprintf(tmp\_e, "%s/%s.exe", ansDir, qname);

if(tOption && isthread)

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f);

else

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f);

//t옵션이 적용된 문제의 경우 -lpthread 옵션 추가

sprintf(tmp\_e, "%s/%s/%s\_error.txt", ansDir, qname, qname);

fd = creat(tmp\_e, 0666);

//e옵션이 적용되어있는 경우, 오류메시지를 지정한 디렉터리에 텍스트 파일로 저장함

redirection(command, fd, STDERR);

size = lseek(fd, 0, SEEK\_END);

close(fd);

unlink(tmp\_e);

if(size > 0)

return false;

sprintf(tmp\_f, "%s/%s/%s", stuDir, id, filename);

sprintf(tmp\_e, "%s/%s/%s.stdexe", stuDir, id, qname);

if(tOption && isthread)

sprintf(command, "gcc -o %s %s -lpthread", tmp\_e, tmp\_f);

else

sprintf(command, "gcc -o %s %s", tmp\_e, tmp\_f);

// t옵션의 유무에 따라 컴파일을 다르게 적용함

sprintf(tmp\_f, "%s/%s/%s\_error.txt", stuDir, id, qname);

fd = creat(tmp\_f, 0666);

//에러 파일은 다른 곳로 출력함

redirection(command, fd, STDERR); // STDERR의 fd를 다른 파일로 옮김

size = lseek(fd, 0, SEEK\_END); //파일 끝으로 가서 파일의 사이즈를 판단함

close(fd);

if(size > 0){

if(eOption)

{ //e옵션이 있을 경우 오류메시지를 사용자가 지정한 폴더로 내보냄

sprintf(tmp\_e, "%s/%s", errorDir, id);

if(access(tmp\_e, F\_OK) < 0)

mkdir(tmp\_e, 0755);

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

rename(tmp\_f, tmp\_e);

result = check\_error\_warning(tmp\_e);

}

else{ //e옵션이 없을 경우 에러나 경고의 갯수만을 판단함

result = check\_error\_warning(tmp\_f);

unlink(tmp\_f);

}

return result;

}

unlink(tmp\_f);

return true;

}

double check\_error\_warning(char \*filename)

{ //학생들의 프로그램에서 에러 또는 경고의 갯수를 계산하는 함수

FILE \*fp;

char tmp[BUFLEN];

double warning = 0;

if((fp = fopen(filename, "r")) == NULL){

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

return false;

}

while(fscanf(fp, "%s", tmp) > 0){

if(!strcmp(tmp, "error:"))

return ERROR;

else if(!strcmp(tmp, "warning:"))

warning += WARNING;

}

//warning의 갯수만큼 -0.1점함, error 발생시 0점

return warning;

}

int execute\_program(char \*id, char \*filename)

{ //학생들의 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;

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

memcpy(qname, filename, strlen(filename) - strlen(strrchr(filename, '.')));

sprintf(ans\_fname, "%s/%s.stdout", ansDir, qname);

fd = creat(ans\_fname, 0666);

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

redirection(tmp, fd, STDOUT); //프로그램 문제 정답 프로그램의 표준 출력을 임시 파일로 돌림

close(fd);

sprintf(std\_fname, "%s/%s/%s.stdout", stuDir, id, qname);

fd = creat(std\_fname, 0666);

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

start = time(NULL);

redirection(tmp, fd, STDOUT); //학생들의 프로그램 문제의 표준 출력을 임시 파일로 돌림

sprintf(tmp, "%s.stdexe", qname);

while((pid = inBackground(tmp)) > 0){ //백그라운드에서 돌고 있는 학생 프로그램의 프로세스 ID를 가져옴

end = time(NULL);

if(difftime(end, start) > OVER){ //5초이상 소요되었을 경우 프로세스를 KILL하고 0점 부여

kill(pid, SIGKILL);

close(fd);

return false;

}

}

close(fd);

//마지막으로 결과 파일을 비교하여 점수를 부여함

return compare\_resultfile(std\_fname, ans\_fname);

}

pid\_t inBackground(char \*name)

{//백그라운드에서 실행중인 학생 또는 정답 프로그램의 프로세스 ID를 반환함

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);

//백그라운드에서 돌아가는 프로그램의 pid를 파일 하나에 저장해둠

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

redirection(command, fd, STDOUT); //프로그램의 표준 출력을 다른 파일로 변경함

lseek(fd, 0, SEEK\_SET);

read(fd, tmp, sizeof(tmp));

if(!strcmp(tmp, "")){

unlink("background.txt");

close(fd);

return 0;

}

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

close(fd);

unlink("background.txt"); //파일 삭제

return pid;

}

int compare\_resultfile(char \*file1, char \*file2)

{ //결과물 파일 두개를 서로 비교함

int fd1, fd2;

char c1, c2;

int len1, len2;

fd1 = open(file1, O\_RDONLY);

fd2 = open(file2, O\_RDONLY);

while(1)

{

while((len1 = read(fd1, &c1, 1)) > 0){

if(c1 == ' ')

continue;

else

break;

}

while((len2 = read(fd2, &c2, 1)) > 0){

if(c2 == ' ')

continue;

else

break;

} //각 파일에서 한글자씩 읽어옴

if(len1 == 0 && len2 == 0)

break;

//파일이 끝났을 경우 종료

to\_lower\_case(&c1);

to\_lower\_case(&c2);

//각 글자를 소문자로 변환한다

if(c1 != c2){

close(fd1);

close(fd2);

return false;

} //글자가 서로 다를 경우 false를 리턴

}

close(fd1);

close(fd2);

return true;

} //글자가 다른 경우가 없을 경우 파일을 닫고 true를 리턴

void redirection(char \*command, int new, int old)

{ //파일 디스크립터를 덮어썼다가 다시 복구 시켜주는 함수

int saved, saved2;

saved = dup(old);

saved2 = dup(STDERR);

dup2(new, old);

dup2(new, STDERR);

system(command);

dup2(saved, old);

close(saved);

close(saved2);

} // 학생 및 정답 프로그램 파일에서 오류나 출력값이 나올때 화면이 아닌 임의의 파일로 출력하는데 사용

int get\_file\_type(char \*filename)

{

char \*extension = strrchr(filename, '.');

if(!strcmp(extension, ".txt"))

return TEXTFILE;

else if (!strcmp(extension, ".c"))

return CFILE;

else

return -1;

} //파일의 확장자를 통해 텍스트파일과 C파일을 구분함

void rmdirs(const char \*path)

{ //입력된 디렉터리를 삭제하는데에 사용하는 함수

struct dirent \*dirp;

struct stat statbuf;

DIR \*dp;

char tmp[1024];

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

return;

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

{

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

continue;

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

if(lstat(tmp, &statbuf) == -1)

continue;

if(S\_ISDIR(statbuf.st\_mode))

rmdirs(tmp);

else

unlink(tmp);

} //디렉터리를 읽어서 내부에 디렉터리가 존재할 경우 디렉터리를 삭제함

closedir(dp);

rmdir(path);

}

void to\_lower\_case(char \*c)

{ //모두 소문자로 전환함

if(\*c >= 'A' && \*c <= 'Z')

\*c = \*c + 32;

}

void print\_usage() //-h 옵션일때 사용 방법 출력

{

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

printf("Option : \n");

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

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

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

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

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

#define DATATYPE\_SIZE 35

#define MINLEN 64

#define TOKEN\_CNT 50

typedef struct node{

int parentheses;

char \*name;

struct node \*parent;

struct node \*child\_head;

struct node \*prev;

struct node \*next;

}node; //트리를 구성하기 위한 노드 구조체

typedef struct operator\_precedence{

char \*operator;

int precedence;

}operator\_precedence; //연산자의 우선순위를 저장하는 구조체

void compare\_tree(node \*root1, node \*root2, int \*result);

//두개의 트리를 서로 비교하는 함수

node \*make\_tree(node \*root, char (\*tokens)[MINLEN], int \*idx, int parentheses);

//토큰을 바탕으로 트리를 생성하는 함수

node \*change\_sibling(node \*parent);

//트리의 형제 노드(같은 레벨의 노드)를 서로 교환하는 함수

node \*create\_node(char \*name, int parentheses);

// 노드를 새로 생성하는 함수

int get\_precedence(char \*op);

//연산자의 우선순위를 결정하는 함수

int is\_operator(char \*op);

// 주어진 문자가 연산자인지 확인하는 함수

void print(node \*cur);

//트리를 순환하면 출력하는 함수

node \*get\_operator(node \*cur);

//트리 탐색 중 연산자를 구분하는 함수

node \*get\_root(node \*cur);

//트리 탐색 중 root노드를 탐색하기 위한 함수

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

// 두 노드 중 더 높은 우선순위의 노드를 계산하는 함수

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

// 가장 우선순위가 높은 노드를 구분하는 함수

node \*insert\_node(node \*old, node \*new);

// 트리에 새로 노드를 삽입하기 위한 함수

node \*get\_last\_child(node \*cur);

//노드의 가장 마지막 자식 노드를 탐색하는 함수

void free\_node(node \*cur);

//트리를 위해 지정한 메모리를 해제해주는 함수

int get\_sibling\_cnt(node \*cur);

//형제 노드의 총 개수를 계산하는 함수

int make\_tokens(char \*str, char tokens[TOKEN\_CNT][MINLEN]);

// 학생의 정답과 정답지의 정답을 토큰으로 분해할 때 사용하는 함수

int is\_typeStatement(char \*str);

//명령어를 구분하는 함수

int find\_typeSpecifier(char tokens[TOKEN\_CNT][MINLEN]);

//문자의 type을 결정하는 인자를 추출하는 함수

int find\_typeSpecifier2(char tokens[TOKEN\_CNT][MINLEN]);

//변수의 type을 결정하는 인자를 추출해내는 함수

int is\_character(char c);

// 문자인지 확인하는 함수

int all\_star(char \*str);

//문자열이 모두 \*문자인지 확인하는 함수

int all\_character(char \*str);

//문자열이 모두 문자로 이루어져 있는지 확인하는 함수

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

//분해한 토큰을 다시 문자열로 합치는 함수

void clear\_tokens(char tokens[TOKEN\_CNT][MINLEN]);

//모든 토큰을 널문자로 초기화하는 함수

int get\_token\_cnt(char tokens[TOKEN\_CNT][MINLEN]);

// NULL문자가 아닌 토큰의 개수를 세는 함수

char \*rtrim(char \*\_str);

//문자열의 뒤쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수

char \*ltrim(char \*\_str);

//문자열의 앞쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수

void remove\_space(char \*str);

//문자열의 공백을 제거하는 함수

int check\_brackets(char \*str);

//여는 괄호와 닫는 괄호의 개수가 일치하는지 확인하는 함수

char\* remove\_extraspace(char \*str);

//문자열에 공백이 1개 이상 연속될 때, 1개만 남기고 공백을 제거하는 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"};

//연산자 우선순위 저장

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}

};

void compare\_tree(node \*root1, node \*root2, int \*result)

{ // 정답과 학생의 답으로 만든 트리를 서로 비교하는 함수

node \*tmp;

int cnt1, cnt2;

if(root1 == NULL || root2 == NULL){

\*result = false;

return;

} //트리가 없으면 끝

if(!strcmp(root1->name, "<") || !strcmp(root1->name, ">") || !strcmp(root1->name, "<=") || !strcmp(root1->name, ">=")){

if(strcmp(root1->name, root2->name) != 0){

if(!strncmp(root2->name, "<", 1))

strncpy(root2->name, ">", 1);

else if(!strncmp(root2->name, ">", 1))

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

else if(!strncmp(root2->name, "<=", 2))

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

else if(!strncmp(root2->name, ">=", 2))

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

root2 = change\_sibling(root2);

}

} //첫 노드가 부등호일 경우 두 트리의 부등호 뱡향을 통일 시키기 위해 자식 노드를 교환함

if(strcmp(root1->name, root2->name) != 0){

\*result = false;

return;

} //트리 시작이 다르면 false 리턴

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

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

\*result = false;

return;

} //자식 노드의 시작이 달라도 false 리턴

else if(root1->child\_head != NULL){

if(get\_sibling\_cnt(root1->child\_head) != get\_sibling\_cnt(root2->child\_head)){

\*result = false;

return;

} //자식 노드의 갯수가 다르면 false 리턴

if(!strcmp(root1->name, "==") || !strcmp(root1->name, "!="))

{// 트리의 시작이 비교 연산자일 경우

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

//루트 노드의 자식 노드를 비교함

if(\*result == false)

{

\*result = true;

root2 = change\_sibling(root2);

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

} //false가 나왔을 경우 자식 노드의 순서를 바꿔서 다시 확인함

}

else if(!strcmp(root1->name, "+") || !strcmp(root1->name, "\*")

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

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

{//트리의 시작이 + \* | || & && 일 경우

if(get\_sibling\_cnt(root1->child\_head) != get\_sibling\_cnt(root2->child\_head)){

\*result = false;

return;

} //자식 노드 갯수가 다르면 false

tmp = root2->child\_head;

while(tmp->prev != NULL)

tmp = tmp->prev;

//마지막 자식노드로 이동

while(tmp != NULL)

{ // 학생 트리와 정답 트리의 같은 깊이의 자식들을 서로 비교함

compare\_tree(root1->child\_head, tmp, result);

if(\*result == true)

break;

else{

if(tmp->next != NULL)

\*result = true;

tmp = tmp->next;

}

}

}

else{ //루트 다음 단계의 노드들끼리의 비교를 시작함

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

}

}

if(root1->next != NULL){

//최상위 노드의 형제 노드가 있을 경우

if(get\_sibling\_cnt(root1) != get\_sibling\_cnt(root2)){

\*result = false;

return;

} //형제 노드의 갯수를 비교하고 갯수가 다를 경우 false 리턴

if(\*result == true)

{

tmp = get\_operator(root1);

//최상위 노드의 연산자를 가져옴

if(!strcmp(tmp->name, "+") || !strcmp(tmp->name, "\*")

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

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

{ //최상위 노드의 연산자가 위의 종류중에 있을경우

tmp = root2;

while(tmp->prev != NULL)

tmp = tmp->prev;

//형제 노드의 마지막으로 이동

while(tmp != NULL)

{

compare\_tree(root1->next, tmp, result);

if(\*result == true)

break;

else{

if(tmp->next != NULL)

\*result = true;

tmp = tmp->next;

}

} // 그 노드의 형제 노드들을 비교함

}

else //또는 그 다음 단계의 노드를 비교하기 시작함

compare\_tree(root1->next, root2->next, result);

}

}

}

int make\_tokens(char \*str, char tokens[TOKEN\_CNT][MINLEN])

{ //학생의 답과 빈칸의 정답을 토큰으로 분해하는 함수

char \*start, \*end;

char tmp[BUFLEN];

char str2[BUFLEN];

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

int row = 0;

int i;

int isPointer;

int lcount, rcount;

int p\_str;

clear\_tokens(tokens);

//시작전 토큰 NULL로 초기화

start = str;

if(is\_typeStatement(str) == 0)

return false;

//명령어는 빼버림

while(1)

{

if((end = strpbrk(start, op)) == NULL)

break;

//더이상 연산자가 없는 경우 종료

if(start == end){

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

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

return false;

//문자열의 시작이 ++, --, ++++, ----일 경우 false 반환

if(is\_character(\*ltrim(start + 2))){ //왼쪽 공백을 지운것이 문자일 경우

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

return false;

//

end = strpbrk(start + 2, op); //연산자가 나오는 앞까지 end로 저장

if(end == NULL) //문자열에 연산자가 없는경우

end = &str[strlen(str)];

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++;

}

}

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

if(strstr(tokens[row - 1], "++") != NULL || strstr(tokens[row - 1], "--") != NULL)

return false;

//row열에 있는 토큰이 문자일 경우

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

strncpy(tmp, start, 2);

strcat(tokens[row - 1], tmp);// 토큰으로 만들어서 저장

start += 2;

row--;

}

else{

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

strncpy(tmp, start, 2);

strcat(tokens[row], tmp);

start += 2;

}

}

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)){

//문자열이 위의 연산자와 같은경우

strncpy(tokens[row], start, 2);

start += 2;

}//토큰으로 저장하고 2칸 전진

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

{//멤버 포인터 연산자일 경우

end = strpbrk(start + 2, op);

//연산자가 나올때까지 끊어서 end에 저장함

if(end == NULL)

end = &str[strlen(str)];

while(start < end){

if(\*start != ' ')

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

start++;

}

row--;

}

else if(\*end == '&')

{ //'&'연산자 일때

if(row == 0 || (strpbrk(tokens[row - 1], op) != NULL)){

end = strpbrk(start + 1, op);

if(end == NULL)

end = &str[strlen(str)];

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

start++;

//start를 토큰으로 저장

while(start < end){

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

return false;

else if(\*start != ' ')

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

start++;

}//직전에 만든 토큰이 &가 아닐때 새로 토큰을 만들어 저장

}

else{

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

start += 1;

}

}

else if(\*end == '\*')

{ //end가 \*를 가리킬 떄

isPointer=0;

if(row > 0)

{

for(i = 0; i < DATATYPE\_SIZE; i++) { //DS는 35

if(strstr(tokens[row - 1], datatype[i]) != NULL){

strcat(tokens[row - 1], "\*");

start += 1;

isPointer = 1;

break;

}//datatype과 같지 않으면 새 토큰으로 \* 추가

}

if(isPointer == 1)

continue;

if(\*(start+1) !=0)

end = start + 1;

if(row>1 && !strcmp(tokens[row - 2], "\*") && (all\_star(tokens[row - 1]) == 1)){

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

row--;

}//토큰에 start의 길이만큼 복사해 넣음

else if(is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1]) == 1){

strncat(tokens[row], start, end - start);

}//전 토큰의 마지막이 문자일때, 새 토큰으로 하나 생성

else if(strpbrk(tokens[row - 1], op) != NULL){

strncat(tokens[row] , start, end - start);

}// 전 토큰에 연산자 전까지 NULL이 아니라면 새 토큰 생성

else

strncat(tokens[row], start, end - start);

//새 토큰 생성

start += (end - start);

} //문자열 길이만큼 진행

else if(row == 0)

{ //토큰이 하나도 없을 때

if((end = strpbrk(start + 1, op)) == NULL){

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

start += 1;

} // 문자열에 연산자 전으로 아무것도 없을 때, 한바이트 만큼 토큰 생성

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++;

}//공백이 아닐 경우 문자 하나만큼 토큰으로 만듬

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

row--;

//토큰이 \*문자일 경우 row 하나 감소

}

}

}

else if(\*end == '(')

{ //토큰의 시작이 '('일경우

lcount = 0;

rcount = 0;

if(row>0 && (strcmp(tokens[row - 1],"&") == 0 || strcmp(tokens[row - 1], "\*") == 0)){

while(\*(end + lcount + 1) == '(')

lcount++;

start += lcount;

end = strpbrk(start + 1, ")");

//닫는 괄호 나올때까지 end에 저장

if(end == NULL)

return false; //닫는 괄호 없으면 문제 있는거

else{

while(\*(end + rcount +1) == ')')

rcount++;

end += rcount;

if(lcount != rcount)

return false;

//여는 괄호와 닫는 괄호의 갯수가 달라도 문제 있는 것

if( (row > 1 && !is\_character(tokens[row - 2][strlen(tokens[row - 2]) - 1])) || row == 1){

strncat(tokens[row - 1], start + 1, end - start - rcount - 1);

row--;

start = end + 1;

}//1번 줄이거나 row가 1보다 클때, 전 토큰의 마지막이 문자일떄

else{

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

start += 1;

}//새 토큰 생성

}

}

else{

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

start += 1;

}

}

else if(\*end == '\"')

{//문자열이 큰 따옴표일 때

end = strpbrk(start + 1, "\"");

//다시 큰따옴표가 나올때까지 end에 저장

if(end == NULL)

return false;

//큰따옴표가 없으면 문제 있는 것

else{

strncat(tokens[row], start, end - start + 1);

start = end + 1;

} //따옴표를 제외한 부분을 새 토큰으로 생성

}

else{ //그 외의 문자열일때

if(row > 0 && !strcmp(tokens[row - 1], "++"))

return false;

//전의 토큰이 ++ 이였으면 false

if(row > 0 && !strcmp(tokens[row - 1], "--"))

return false;

//전의 토큰이 --였으면 false

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

start += 1;

//start에서 토큰으로 한 문자를 불러옴

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

//현재 토큰이 -, +, --, ++일경우

if(row == 0)

row--;

else if(!is\_character(tokens[row - 1][strlen(tokens[row - 1]) - 1])){

if(strstr(tokens[row - 1], "++") == NULL && strstr(tokens[row - 1], "--") == NULL)

row--;

}//토큰이 ++ 또는 --가 아닌 경우 전의 토큰으로 올라감

}

}

}

else{ //start == end가 아닐떄

if(all\_star(tokens[row - 1]) && row > 1 && !is\_character(tokens[row - 2][strlen(tokens[row - 2]) - 1]))

row--;

if(all\_star(tokens[row - 1]) && row == 1)

row--;

//row가 1이고 첫번째 토큰이 \* 문자일때

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

if(i > 0 && \*(start + i) == '.'){

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

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

i++;

}

else if(start[i] == ' '){

while(start[i] == ' ')

i++;

break;

} //start에서 공백을 제외한 부분을 토큰으로 저장함

else

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

}

if(start[0] == ' '){

start += i;

continue;

}

start += i;

}

strcpy(tokens[row], ltrim(rtrim(tokens[row])));

//토큰의 좌우에서 공백까지 제거함

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] == '.' ) ){

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

{//여는 괄호가 있을떄

if(strcmp(tokens[row - 1], "struct") != 0 && strcmp(tokens[row - 1],"unsigned") != 0)

return false;

}//괄호 안에 struct나 unsigned가 있으면 false리턴

else if(row == 1 && is\_character(tokens[row][strlen(tokens[row]) - 1])) {

if(strcmp(tokens[0], "extern") != 0 && strcmp(tokens[0], "unsigned") != 0 && is\_typeStatement(tokens[0]) != 2)

return false;

}//토큰의 마지막이 문자일때 가장 처음의 토큰이 extern 또는 unsigned 일 경우 false 리턴

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

if(strcmp(tokens[row - 2], "unsigned") != 0 && strcmp(tokens[row - 2], "extern") != 0)

return false;

}// 전 토큰이 명령어 일 때, 그 전의 토큰이 unsigned 또는 extern일때, false 리턴

}

if((row == 0 && !strcmp(tokens[row], "gcc")) ){

clear\_tokens(tokens);

strcpy(tokens[0], str);

return 1;

} //토큰이 gcc 일 경우 토큰을 초기화하고 str을 첫번째 토큰으로 복사함

row++;

}

if(all\_star(tokens[row - 1]) && row > 1 && !is\_character(tokens[row - 2][strlen(tokens[row - 2]) - 1]))

row--;

if(all\_star(tokens[row - 1]) && row == 1)

row--;

//start의 길이 만큼 start를 분석함

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

{

if(start[i] == ' ')

{

while(start[i] == ' ')

i++;

if(start[0]==' ') {

start += i;

i = 0;

}//처음 공백 제거

else

row++;

i--;

}

else

{

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

if( start[i] == '.' && i<strlen(start)){

while(start[i + 1] == ' ' && i < strlen(start))

i++;

//공백이 아닐때까지 start 포인터를 이동시켜서 한 문자를 토큰으로 가져옴

}

}

strcpy(tokens[row], ltrim(rtrim(tokens[row])));

//토큰의 좌우에서 공백까지 제거함

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

strcat(tokens[row - 1], tokens[row]);

memset(tokens[row], 0, sizeof(tokens[row]));

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] == '.') ){

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

{//여는 괄호가 있을떄

if(strcmp(tokens[row-1], "struct") != 0 && strcmp(tokens[row-1], "unsigned") != 0)

return false;

}//괄호 안에 struct나 unsigned가 있으면 false리턴

else if(row == 1 && is\_character(tokens[row][strlen(tokens[row]) - 1])) {

if(strcmp(tokens[0], "extern") != 0 && strcmp(tokens[0], "unsigned") != 0 && is\_typeStatement(tokens[0]) != 2)

return false;

}//토큰의 마지막이 문자일때 가장 처음의 토큰이 extern 또는 unsigned 일 경우 false 리턴

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

if(strcmp(tokens[row - 2], "unsigned") != 0 && strcmp(tokens[row - 2], "extern") != 0)

return false;

}// 전 토큰이 명령어 일 때, 그 전의 토큰이 unsigned 또는 extern일때, false 리턴

}

}

if(row > 0)

{ //기존에 민들어진 토큰이 있을 때

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

clear\_tokens(tokens);

strcpy(tokens[0], remove\_extraspace(str));

}//토큰이 위와 같은 경우, str에서 빈칸을 지운 문자열을 토큰으로 만듬

}

//문자열이 명령어이거나, extern이 첫번째 토큰일 떄

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

for(i = 1; i < TOKEN\_CNT; i++){

if(strcmp(tokens[i],"") == 0)

break;

if(i != TOKEN\_CNT -1 )

strcat(tokens[0], " ");

strcat(tokens[0], tokens[i]);

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

}//0번 토큰 뒤에 i번 토큰을 붙이고 i번 토큰 초기화

}

while((p\_str = find\_typeSpecifier(tokens)) != -1){

if(!reset\_tokens(p\_str, tokens))

return false;

} // 남은 문자열에 토큰으로 만들게 있으면 false리턴

while((p\_str = find\_typeSpecifier2(tokens)) != -1){

if(!reset\_tokens(p\_str, tokens))

return false;

} // 남은 문자열에 토큰으로 만들게 있으면 false리턴

return true;

}

node \*make\_tree(node \*root, char (\*tokens)[MINLEN], int \*idx, int parentheses)

{ //토큰을 바탕으로 트리를 생성하는 함수

node \*cur = root;

node \*new;

node \*saved\_operator;

node \*operator;

int fstart;

int i;

while(1)

{

if(strcmp(tokens[\*idx], "") == 0)

break;

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

return get\_root(cur);

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

return get\_root(cur);

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

{ //여는 괄호부터 닫는 괄호가 나올때 까지 내부로 트리를 만들어감

if(\*idx > 0 && !is\_operator(tokens[\*idx - 1]) && strcmp(tokens[\*idx - 1], ",") != 0){

fstart = true;

//토큰이 연산자가 아니고 콤마가 아닐때

while(1)

{

\*idx += 1;

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

break;

//닫는 괄호가 나오면 종료

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

//재귀호출로 자식노드들의 트리 생성

if(new != NULL){

if(fstart == true){

cur->child\_head = new;

new->parent = cur;

//자식 노드 생성 후 부모 노드와 연결

fstart = false;

}

else{

cur->next = new;

new->prev = cur;

}

cur = new;

} //자식노드의 트리가 있을떄 형제 노드들을 생성함

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

break; //닫는 괄호가 나오면 종료

}

}

else{ //토큰이 연산자이거나 콤마일때

\*idx += 1;

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

if(cur == NULL)

cur = new;

else if(!strcmp(new->name, cur->name)){

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

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

{ //and or 등의 연산자일떄

cur = get\_last\_child(cur);

if(new->child\_head != NULL){

new = new->child\_head;

//새로운 자식노드를 시작함

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

new->parent = NULL;

new->prev = cur;

cur->next = new;

}

}

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

{ //더하기 또는 곱하기일때

i = 0;

while(1)

{

if(!strcmp(tokens[\*idx + i], ""))

break;

//토큰이 공백이면 break

if(is\_operator(tokens[\*idx + i]) && strcmp(tokens[\*idx + i], ")") != 0)

break;

//토큰이 닫는 괄호거나 연산자일 경우 break

i++;

}

if(get\_precedence(tokens[\*idx + i]) < get\_precedence(new->name))

{ //연산자의 우선순위를 고려하여 자식노드와 부모노드를 결정함

cur = get\_last\_child(cur);

cur->next = new;

new->prev = cur;

cur = new;

}

else

{

cur = get\_last\_child(cur);

if(new->child\_head != NULL){

new = new->child\_head;

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

new->parent = NULL;

new->prev = cur;

cur->next = new;

} //마지막 자식노드로 추가함

}

}

else{ //더하기나 곱하기가 아닐때

cur = get\_last\_child(cur);

cur->next = new;

new->prev = cur;

cur = new;

}

}

else

{ //새 노드와 현재 노드가 같은 문자일때

cur = get\_last\_child(cur);

cur->next = new;

new->prev = cur;

cur = new;

}

}

}

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

{ //토큰이 연산자일때

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

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

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

{//토큰이 위와 같은 연산자일때

if(is\_operator(cur->name) == true && !strcmp(cur->name, tokens[\*idx]))

operator = cur;

//현재 노드 저장

else

{

new = create\_node(tokens[\*idx], parentheses);

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

//둘의 가장 상위 노드를 불러와 저장

if(operator->parent == NULL && operator->prev == NULL){

//연산자의 부모와 형제가 모두 없을떄

if(get\_precedence(operator->name) < get\_precedence(new->name)){

cur = insert\_node(operator, new);

} //두 연산자의 우선순위를 비교하여 새 노드 삽입

else if(get\_precedence(operator->name) > get\_precedence(new->name))

{

if(operator->child\_head != NULL){

operator = get\_last\_child(operator);

cur = insert\_node(operator, new);

}

}//연산자 우선순위를 비교하여 자식노드의 마지막에 새 노드 생성

else

{

operator = cur;

while(1)

{

if(is\_operator(operator->name) == true && !strcmp(operator->name, tokens[\*idx]))

break;

//operator가 토큰과 같아지면 탈출

if(operator->prev != NULL)

operator = operator->prev;

else

break;

}//operator의 마지막 형제 노드까지 이동하고 루프 탈출

if(strcmp(operator->name, tokens[\*idx]) != 0)

operator = operator->parent;

//토큰과 operator가 같지 않다면 부모노드로 이동

if(operator != NULL){

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

cur = operator;

}//연산자가 null이 아니고 토큰과 같을떄 현재 노드를 operator로 덮어씀

}

}

else

cur = insert\_node(operator, new);

}//연산자의 부모 현제노드가 없는 경우가 아닐 경우 새 노드 생성 후 트리에 추가함

}

else

{//토큰이 위와 같은 연산자가 아닐때

new = create\_node(tokens[\*idx], parentheses);

if(cur == NULL)

cur = new;

else

{

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

//현재 노드중 최상위 노드를 불러옴

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

cur = insert\_node(operator, new);

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

//operator의 부모 형제 노드 모두 없을때

if(get\_precedence(operator->name) > get\_precedence(new->name))

{ //연산자 순위 비교

if(operator->child\_head != NULL){

operator = get\_last\_child(operator);

cur = insert\_node(operator, new);

}//operator의 자식노드가 있는 경우 자식노드의 가장 마지막에 추가함

}

else

cur = insert\_node(operator, new);

}//operator의 prev에 새노드 추가

else

cur = insert\_node(operator, new);

} //operator의 prev에 새노드 추가

}

}

else //토큰이 여는 괄호가 아니고 연산자도 아닐때

{

new = create\_node(tokens[\*idx], parentheses);

//새 노드 생성

if(cur == NULL)

cur = new;

//현재 노드가 없을 경우 현재노드가 됨

else if(cur->child\_head == NULL){

cur->child\_head = new;

new->parent = cur;

cur = new;

} //현재노드의 자식만 없을 경우 자식 노드가 됨

else{

//그 외의 경우 마지막 자식노드의 prev 노드가 됨

cur = get\_last\_child(cur);

cur->next = new; //왼쪽 노드

new->prev = cur; //오른쪽 노드

cur = new;

}

}

\*idx += 1;

}//다음 토큰으로 이동

return get\_root(cur);

}//루트를 리턴함

node \*change\_sibling(node \*parent)

{ //자식 노드의 순서를 바꾸는 함수

node \*tmp;

tmp = parent->child\_head;

//원래의 child\_head 저장

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

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

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

//원래의 child\_head의 next를 새 child\_head로 지정

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;

//새 child\_head의 next를 원래의 child\_head로 지정

return parent;

}

node \*create\_node(char \*name, int parentheses)

{ //트리의 노드를 새로 생성하는 함수

node \*new;

new = (node \*)malloc(sizeof(node));

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

strcpy(new->name, name);

new->parentheses = parentheses;

new->parent = NULL;

new->child\_head = NULL;

new->prev = NULL;

new->next = NULL;

return new;

}

int get\_precedence(char \*op)

{ //지정한 연산자 우선순위를 검색함

int i;

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

if(!strcmp(operators[i].operator, op))

return operators[i].precedence;

}

return false;

} //찾은 연산자 우선순위를 반환함

int is\_operator(char \*op)

{ //입력된 문자가 연산자인지 확인하는 함수

int i;

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

{

if(operators[i].operator == NULL)

break;

if(!strcmp(operators[i].operator, op)){

return true;

}

}

return false;

}

void print(node \*cur)

{ //트리를 하나씩 출력하는 함수

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

print(cur->child\_head);

printf("\n");

}

if(cur->next != NULL){

print(cur->next);

printf("\t");

}

printf("%s", cur->name);

}

node \*get\_operator(node \*cur)

{ // 트리 탐색중 연산자를 불러오기 위한 함수

if(cur == NULL)

return cur;

if(cur->prev != NULL)

while(cur->prev != NULL)

cur = cur->prev;

return cur->parent;

} //child\_head의 부모노드는 연산자이므로 이를 반환함

node \*get\_root(node \*cur)

{ // 트리 탐색중 root 노드를 확인하기 위한 함수

if(cur == NULL)

return cur;

while(cur->prev != NULL)

cur = cur->prev;

if(cur->parent != NULL)

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

return cur;

} //루트 노드에 닿을때까지 반복한다

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

{ //상위에 있는 노드를 탐색하기 위한 함수

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

if(get\_precedence(cur->name) < get\_precedence(new->name))

return cur;

if(cur->prev != NULL){

while(cur->prev != NULL){

cur = cur->prev;

return get\_high\_precedence\_node(cur, new);

}

//가장 상단에 도착할 떄까지 재귀 순환

if(cur->parent != NULL)

return get\_high\_precedence\_node(cur->parent, new);

}

if(cur->parent == NULL)

return cur;

}

node \*get\_most\_high\_precedence\_node(node \*cur, node \*new)

{ //가장 상위에 있는 노드를 탐색하기 위한 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)

operator = get\_high\_precedence\_node(saved\_operator->prev, new);

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

operator = get\_high\_precedence\_node(saved\_operator->parent, new);

saved\_operator = operator;

} //get\_high\_precedence\_node() 함수를 재귀적으로 호출하여 최상위의 노드를 탐색함

return saved\_operator;

}

node \*insert\_node(node \*old, node \*new)

{ // 트리에 새 노드를 추가하는 함수

if(old->prev != NULL){

new->prev = old->prev;

old->prev->next = new;

old->prev = NULL;

}

new->child\_head = old;

old->parent = new;

return new;

}

node \*get\_last\_child(node \*cur)

{ //노드의 가장 마지막 자식 노드를 탐색하는 함수

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

cur = cur->child\_head;

while(cur->next != NULL)

cur = cur->next;

return cur;

}

int get\_sibling\_cnt(node \*cur)

{ //자식노드의 총 갯수를 탐색하는 함수

int i = 0;

while(cur->prev != NULL)

cur = cur->prev;

while(cur->next != NULL){

cur = cur->next;

i++;

}

return i;

} //형제 노드의 가장 마지막에서부터 head로 오면서 갯수를 셈

void free\_node(node \*cur)

{ //트리를 만들면서 지정한 메모리를 free 해주는 함수

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

free\_node(cur->child\_head);

if(cur->next != NULL)

free\_node(cur->next);

if(cur != NULL){

cur->prev = NULL;

cur->next = NULL;

cur->parent = NULL;

cur->child\_head = NULL;

free(cur);

}

} //트리 전체를 순환하며 남는 것이 없게 메모리를 해제해줌

int is\_character(char c)

{ //문자인지 확인하는 함수

return (c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z');

}

int is\_typeStatement(char \*str)

{ //명령어를 구분해 내는 함수

char \*start;

char str2[BUFLEN] = {0};

char tmp[BUFLEN] = {0};

char tmp2[BUFLEN] = {0};

int i;

start = str;

strncpy(str2,str,strlen(str));

remove\_space(str2);

while(start[0] == ' ')

start += 1;

//gcc 명령어가 포함되어있을 경우

if(strstr(str2, "gcc") != NULL)

{

strncpy(tmp2, start, strlen("gcc"));

if(strcmp(tmp2,"gcc") != 0)

return 0;

else

return 2;

}//start에서 tmp2로 3개만큼 문자열 복사

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

{

if(strstr(str2,datatype[i]) != NULL)

{

strncpy(tmp, str2, strlen(datatype[i]));

strncpy(tmp2, start, strlen(datatype[i]));

//문자열을 tmp, tmp2로 지정한 문자열의 길이만큼 복사

if(strcmp(tmp, datatype[i]) == 0)

if(strcmp(tmp, tmp2) != 0)

return 0;

else

return 2;

} //복사한 문자열들을 비교 판단함

}

return 1;

}

int find\_typeSpecifier(char tokens[TOKEN\_CNT][MINLEN])

{ //문자의 type을 결정하는 인자를 추출하는 함수

int i, j;

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

{

for(j = 0; j < DATATYPE\_SIZE; j++)

{

if(strstr(tokens[i], datatype[j]) != NULL && i > 0)

{

if(!strcmp(tokens[i - 1], "(") && !strcmp(tokens[i + 1], ")")

&& (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])))

return i;

}

}

}

return -1;

}

int find\_typeSpecifier2(char tokens[TOKEN\_CNT][MINLEN])

{ //변수의 type을 결정하는 인자를 추출해내는 함수

int i, j;

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

{

for(j = 0; j < DATATYPE\_SIZE; j++)

{

if(!strcmp(tokens[i], "struct") && (i+1) <= TOKEN\_CNT && is\_character(tokens[i + 1][strlen(tokens[i + 1]) - 1]))

return i;

}

}

return -1;

}

int all\_star(char \*str)

{ // 문자열이 모두 \*인지 확인하는 함수

int i;

int length= strlen(str);

if(length == 0)

return 0;

for(i = 0; i < length; i++)

if(str[i] != '\*')

return 0;

return 1;

}

int all\_character(char \*str)

{ //문자열이 모두 문자인지 확인하는 힘수

int i;

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

if(is\_character(str[i]))

return 1;

return 0;

}

int reset\_tokens(int start, char tokens[TOKEN\_CNT][MINLEN])

{ //분해한 토큰을 다시 하나의 문자열로 합치는 함수

int i;

int j = start - 1;

int lcount = 0, rcount = 0;

int sub\_lcount = 0, sub\_rcount = 0;

if(start > -1){

if(!strcmp(tokens[start], "struct")) {

strcat(tokens[start], " ");

strcat(tokens[start], tokens[start+1]);

for(i = start + 1; i < TOKEN\_CNT - 1; i++){

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) {

strcat(tokens[start], " ");

strcat(tokens[start], tokens[start + 1]);

strcat(tokens[start], tokens[start + 2]);

for(i = start + 1; i < TOKEN\_CNT - 1; i++){

strcpy(tokens[i], tokens[i + 1]);

memset(tokens[i + 1], 0, sizeof(tokens[0]));

}

}

j = start + 1;

while(!strcmp(tokens[j], ")")){

rcount ++;

if(j==TOKEN\_CNT)

break;

j++;

}

j = start - 1;

while(!strcmp(tokens[j], "(")){

lcount ++;

if(j == 0)

break;

j--;

}

if( (j!=0 && is\_character(tokens[j][strlen(tokens[j])-1]) ) || j==0)

lcount = rcount;

if(lcount != rcount )

return false;

if( (start - lcount) >0 && !strcmp(tokens[start - lcount - 1], "sizeof")){

return true;

}

else if((!strcmp(tokens[start], "unsigned") || !strcmp(tokens[start], "struct")) && strcmp(tokens[start+1], ")")) {

strcat(tokens[start - lcount], tokens[start]);

strcat(tokens[start - lcount], tokens[start + 1]);

strcpy(tokens[start - lcount + 1], tokens[start + rcount]);

for(int i = start - lcount + 1; i < TOKEN\_CNT - lcount -rcount; i++) {

strcpy(tokens[i], tokens[i + lcount + rcount]);

memset(tokens[i + lcount + rcount], 0, sizeof(tokens[0]));

}

}

else{

if(tokens[start + 2][0] == '('){

j = start + 2;

while(!strcmp(tokens[j], "(")){

sub\_lcount++;

j++;

}

if(!strcmp(tokens[j + 1],")")){

j = j + 1;

while(!strcmp(tokens[j], ")")){

sub\_rcount++;

j++;

}

}//괄호의 갯수를 맞추기 위해 여는 괄호와 닫는 괄호의 갯수를 계산함

else

return false;

if(sub\_lcount != sub\_rcount)

return false;

strcpy(tokens[start + 2], tokens[start + 2 + sub\_lcount]);

for(int i = start + 3; i<TOKEN\_CNT; i++)

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

}

strcat(tokens[start - lcount], tokens[start]);

strcat(tokens[start - lcount], tokens[start + 1]);

strcat(tokens[start - lcount], tokens[start + rcount + 1]);

for(int i = start - lcount + 1; i < TOKEN\_CNT - lcount -rcount -1; i++) {

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])

{

int i;

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

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

} //토큰들을 모두 널문자로 초기화 하는 함수

char \*rtrim(char \*\_str)

{// 문자열의 뒤쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수

char tmp[BUFLEN];

char \*end;

strcpy(tmp, \_str);

end = tmp + strlen(tmp) - 1;

while(end != \_str && isspace(\*end))

--end;

\*(end + 1) = '\0';

\_str = tmp;

return \_str;

}

char \*ltrim(char \*\_str)

{// 문자열의 앞쪽부터 공백이 나올때까지 자르고 나머지를 리턴하는 함수

char \*start = \_str;

while(\*start != '\0' && isspace(\*start))

++start;

\_str = start;

return \_str;

}

char\* remove\_extraspace(char \*str)

{ //문자열에 공백이 1개 이상 연속될 때, 1개만 남기고 공백을 지우는 함수

int i;

char \*str2 = (char\*)malloc(sizeof(char) \* BUFLEN);

char \*start, \*end;

char temp[BUFLEN] = "";

int position;

if(strstr(str,"include<")!=NULL){

start = str;

end = strpbrk(str, "<");

position = end - start;

strncat(temp, str, position);

strncat(temp, " ", 1);

strncat(temp, str + position, strlen(str) - position + 1);

str = temp;

} //문자열의 시작이 include 일 경우 뒤에 오는 헤더파일을 구분하기 위해 문자열을 자름

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

{

if(str[i] ==' ')

{

if(i == 0 && str[0] ==' ')

while(str[i + 1] == ' ')

i++;

else{

if(i > 0 && str[i - 1] != ' ')

str2[strlen(str2)] = str[i];

while(str[i + 1] == ' ')

i++;

}

} //문자열 중간에 있는 공백을 제거함

else

str2[strlen(str2)] = str[i];

}

return str2;

}

void remove\_space(char \*str)

{ //문자열의 공백을 제거하는 함수

char\* i = str;

char\* j = str;

while(\*j != 0)

{

\*i = \*j++;

if(\*i != ' ')

i++;

}//공백문자를 뛰어넘는 방식으로 공백을 없앰

\*i = 0;

}

int check\_brackets(char \*str)

{ //여는괄호 닫는 괄호 갯수가 일치하는지 확인하는 함수

char \*start = str;

int lcount = 0, rcount = 0;

while(1){

if((start = strpbrk(start, "()")) != NULL){

if(\*(start) == '(')

lcount++;

else

rcount++;

start += 1;

}

else

break;

}

if(lcount != rcount)

return 0;

else

return 1;

}

int get\_token\_cnt(char tokens[TOKEN\_CNT][MINLEN])

{ //토큰의 갯수를 세는 함수

int i;

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

if(!strcmp(tokens[i], ""))

break;

return i;

}