**컴퓨터학부 20182647 이형규**

**1. 개요**

- ssu\_score 실행 프로그램은 <ANS\_DIR>에 존재하는 정답 파일과 <STD\_DIR>에 존재하는 각 학생들의 답안파일을 비교하여 자동으로 채점해주는 프로그램이다.

- 빈칸문제의 경우 연산자의 의해 다수의 정답이 존재하는 경우 모두 정답처리 하는 기능이 있다.

- 프로그램문제의 경우 컴파일 에러 발생시 0점 , warning시 -0.5점을 진행한다.

- -m 옵션, -i 옵션, -h 옵션, -e 옵션, -t 옵션을 이용하여 특정 문제에 대해 lpthread와 같은 옵션을 설정하고, 각 학생 별로 어떤 문제가 틀렸고 어떤 에러나 Warning을 냈는지 알 수 있는 기능들이 존해하는 프로그램이다.

- m 옵션 : 특정 문제의 배점을 수정 할 수 있다.

- i <IDS> 옵션 : 최대 5개까지 학번을 인자로 주면 각 학생 별로 틀린 문제의 번호를 출력해준다

- 학생정답파일 디렉토리와 정답파일 디렉토리가 없어도 score.csv파일이 존재하면 사용 가능하다.

- h 옵션 : 사용법을 출력한다

- t 옵션 : 프로그램 컴파일시 lpthread옵션을 적용하여 채점한다

- e <DIR> 옵션 : DIR에 각 학생별로 컴파일 에러나 Warning정보를 출력한다.

**2. 분석**

**(1) 수정되거나 추가된 부분 ( 코드 부분은 <5. 소스코드 및 주석> 에 추가했습니다. )**

**- void do\_mOption(char \*path);**

- 인자로 score\_table.csv파일이 저장된 디렉토리 경로를 인자로 받는다.

- m옵션이 특정 문제의 점수 배점을 수정하고 이를 score\_table.csv에 반영하여 채점한다.

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

- 옵션들의 기호와 기능이 변경됨에 따라 ethmi 기호로 수정하여 각 옵션 별 flag 설정및 데이터 저장

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

- 인자로 학번을 받아 cIDs배열에 저장하여 cIDs를 인자로 입력받는다.

- score.csv파일이 존재하면 score.csv파일에서 학번을 찾아내 0점 문제의 문제의 이름을출력해준다.

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

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

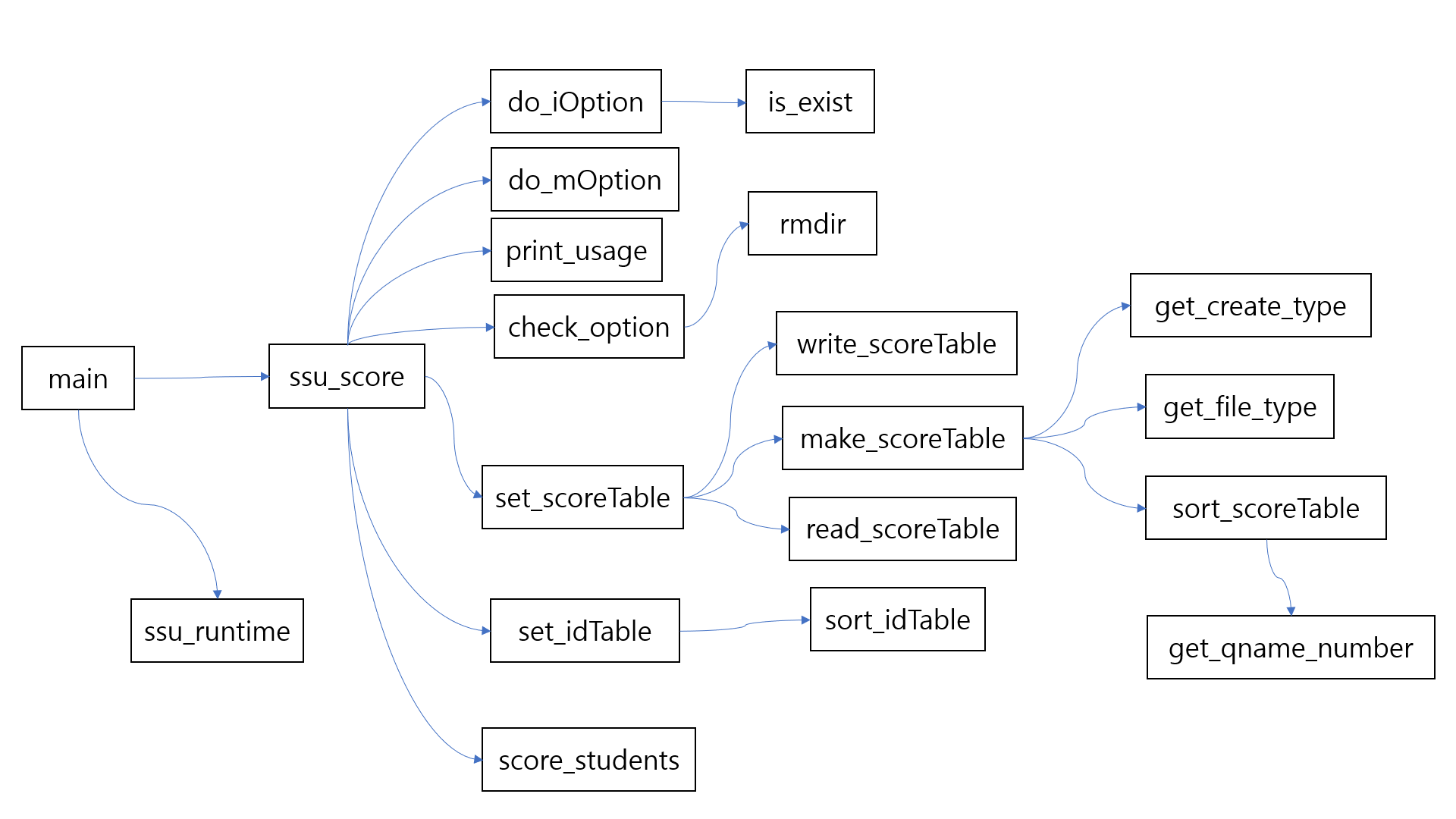
**- int score\_blank(char \*id, char \*filename);**

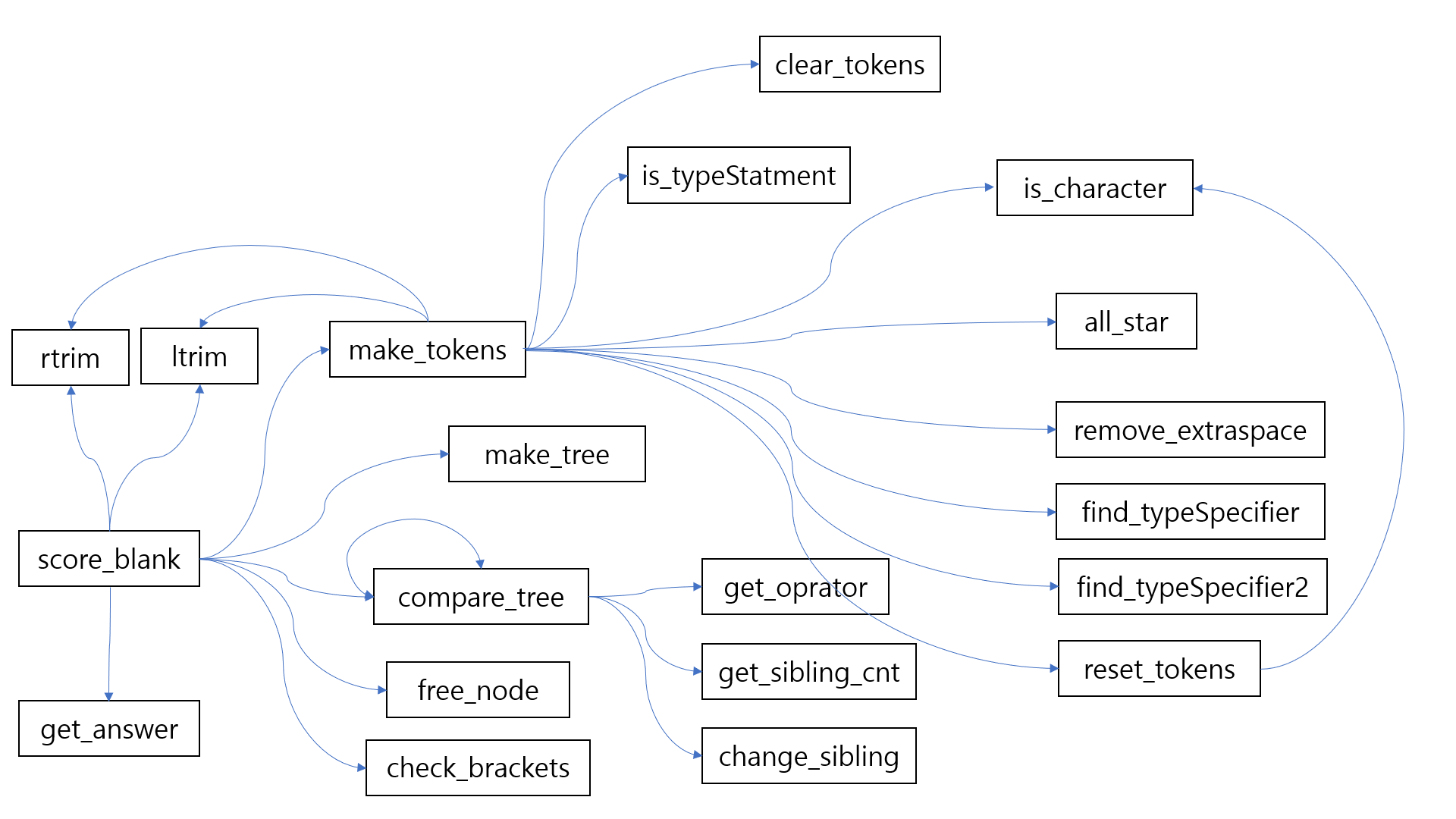
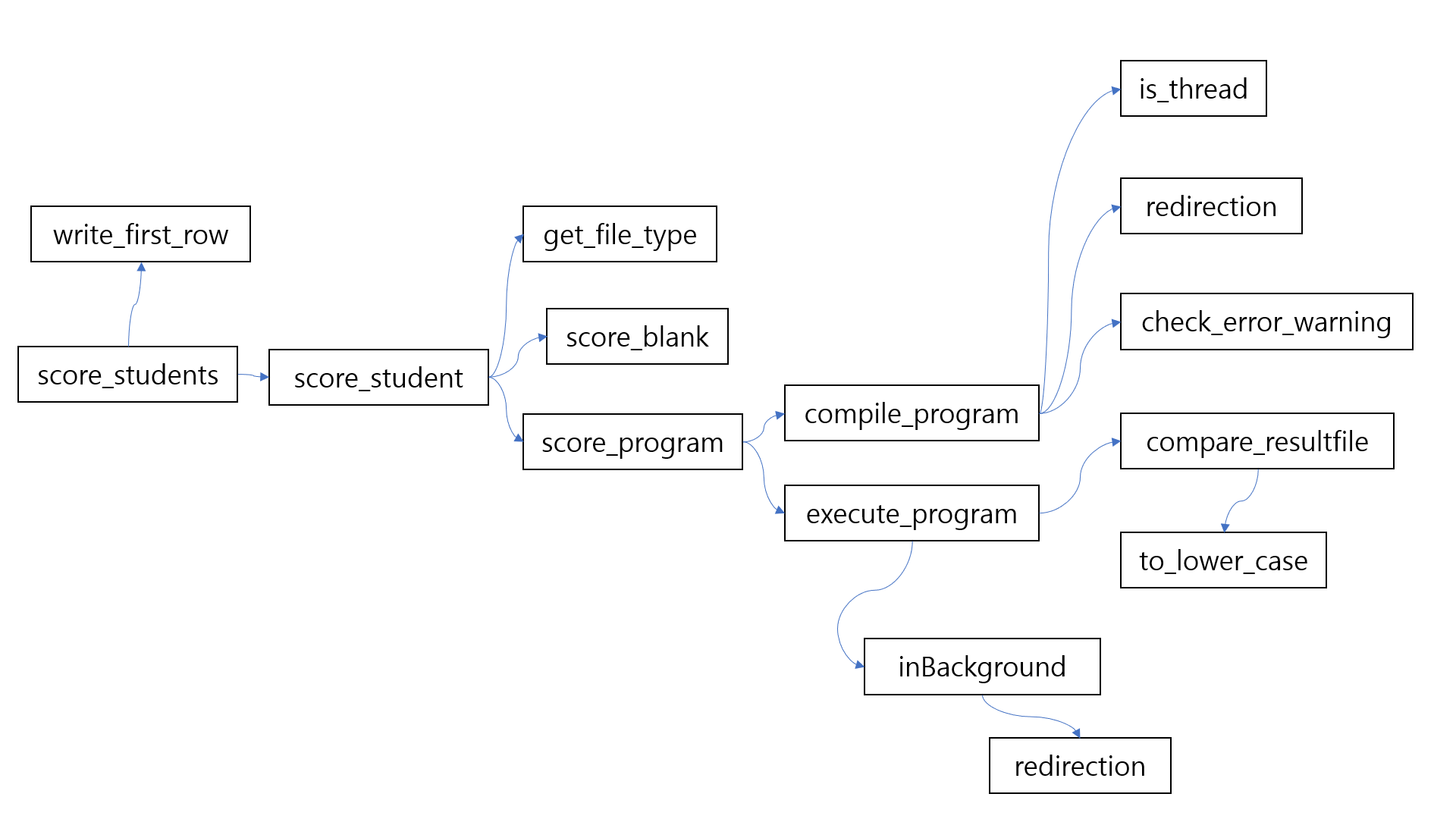
- 위 3개의 함수 모두 정답파일의 경로를 문제번호로 서브 디렉토리가 없는 경로로 바꾸어 프로그램 컴파일, 프로그램 실행, 빈칸 채점을 진행하였음.

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

- 이전의 옵션의 플래그가 true일 경우에만 점수를 출력하던 방식을 항상 출력하도록 변경

**(2) 함수별 흐름도**





**3. 설계 및 구현**

**- 상세 설명**

**- ssu\_score.c**

- 옵션들을 체크하여 플래그를 설정해준다.

- m옵션이 걸려있으면 채점 전에 배점을 수정한다.

- 점수 배점 테이블(score\_table.csv)의 제작, 정렬, 읽어오기 작업 수행

- 학번 테이블 제작, 정렬 수행

- 빈칸문제의 경우 blank.c에서 처리한다

- 토큰화 -> 트리화 시킨 뒤에 트리를 서로 비교한다.

- 프로그램문제의 경우 컴파일 과정과 실행 과정으로 나뉜다.

- 컴파일 과정에서 정답파일들과 학생파일을 컴파일 시킨 뒤 에러시 0점 warning시 개수 당 감점.

- 실행 과정에서 옵션이 걸린 문제는 lpthread를 추가해주고 기본적으로 백그라 운드로 실행 시킨다음 실행 결과를 서로 비교하여 채점한다.

**- blank.c**

- 넘겨 받은 문자열을 연산자 별로 토큰화 시킨다.

- 토큰화 시킨 배열을 이용하여 자료형, 변수명 등등 특정 상황들을 거른다음 make\_tree함수를 이용해 트리화를 시킨다. 트리는 기본적으로 연산자 우선순위 가 높은 연산자가 밑으로 간다.

**- main.c**

- ssu\_score 실행과 수행시간을 측정한다

**- 함수 프로토타입**

**<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

//트리를 구성할 노드 구조체

typedef struct node{

int parentheses;

char \*name;

struct node \*parent;

struct node \*child\_head;

struct node \*prev;

struct node \*next;

}node;

//연산자 우선순위 저장

typedef struct operator\_precedence{

char \*operator;

int precedence;

}operator\_precedence;

//트리 비교 함수

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

//토큰들을 이용해 트리 제작함수

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

//순서가 바뀔 수 있는 연산자에 대한 자식 교한 함수

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

//새로운 노드를 제작하는 함수

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

//인자로 받은 문자의 우선순위를 리턴시켜주는 함수

int get\_precedence(char \*op);

//연산자인지 확인하는 함수

int is\_operator(char \*op);

void print(node \*cur);

//트리에서 연산자를 찾는함수

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

//루트를 반환하는 함수

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

//우선순위가 높은 노드를 리턴

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

//가장 최상위 우선순위 노드를 리턴

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

//새로운 노드를 삽입시키는 함수

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

//가장 마지막 자식을 찾는 함수

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

//메모리해제함수

void free\_node(node \*cur);

//자식의 갯수를 리턴시키는 함수

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

//입력받은 str을 여러규칙에 따라 토큰화 시킨다

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

//gcc인지 자료형인지 확인

int is\_typeStatement(char\* str);

//타입지정자 찾기

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

//타입지정자 찾기 2

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

//문자인지 확인하는 함수

int is\_character(char c);

//\*로만 이루어져 있는지 확인하는 함수

int all\_star(char \*str);

//다 문자인지 확인

int all\_character(char \*str);

//struct와 unsigned 등등 처리화 나머지 토큰들 초기화 시킴

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

//토큰이 저장된 배열 초기화

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

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

//오른쪽의 공백을 제거시키는 함수

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

//왼쪽의 공백을 제거시키는 함수

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

//문자열 사이 공백 제거

void remove\_space(char \*str);

//괄호의 갯수가 맞는지 세주는 함수

int check\_brackets(char \*str);

//추가적인 공백들을 삭제시키는 함수

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

#endif

**<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

//문제별 배점을 저장할 구조체

struct ssu\_scoreTable{

char qname[FILELEN];

double score;

};

//채점프로그램 시작 함수

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

//옵션들 체크

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

//사용법 출력

void print\_usage();

//학생들 채점하기

void score\_students();

//학생 개인별 채점

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

//score.csv파일의 첫번째 줄 작성

void write\_first\_row(int fd);

//파일로부터 정답을 불러옴

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

//빈칸문제 채점함수

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

//프로그램문제 채점함수

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

//프로그램문제 파일 컴파일 함수

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

//프로그램문제 파일 실행 함수

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

//프로그램문제 파일 백그라운드로 실행시키는함수

pid\_t inBackground(char \*name);

//컴파일에 에러나 warning이 있는지 확인

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

//결과파일 비교

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

//-m옵션 수행

void do\_mOption(char \*path);

//-i옵션 수행

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

//-i옵션으로 받은 학번이 존재하는지 확인

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

//스레드파일인지 확인

int is\_thread(char \*qname);

//출력 재지정 및 명령어 실행

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

//파일이 txt인지 c인이 확인

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

//디렉토리삭제함수

void rmdirs(const char \*path);

//대문자를 소문자로 변경

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

//점수배점 테이블 세팅

void set\_scoreTable(char \*ansDir);

//점수배점 테이블이 존재하면 읽어드림

void read\_scoreTable(char \*path);

//점수배점 테이블을 만드는 함수

void make\_scoreTable(char \*ansDir);

//score\_table.csv파일에 데이터 작성

void write\_scoreTable(char \*filename);

//학번 저장테이블 세팅

void set\_idTable(char \*stuDir);

//score\_table.csv를 어떤 타입으로 만들건지 물어보는 함수

int get\_create\_type();

//학번 테이블 정렬

void sort\_idTable(int size);

//score\_table 정렬

void sort\_scoreTable(int size);

//문제 번호 추출함수

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

#endif

**<main.c>**

#define SECOND\_TO\_MICRO 1000000

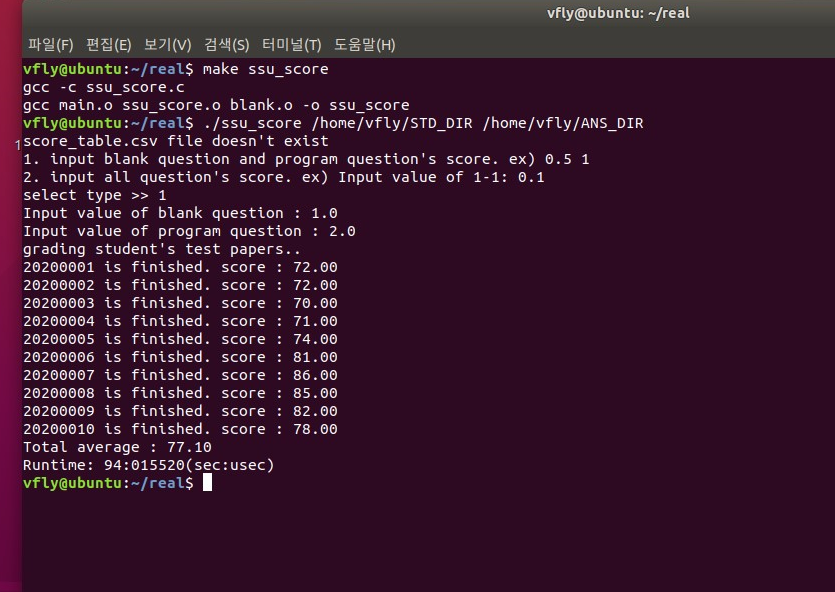
//프로그램 수행 시간 측정

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

**4. 테스트 및 결과**



**- 옵션 없이 실행 했을 때**

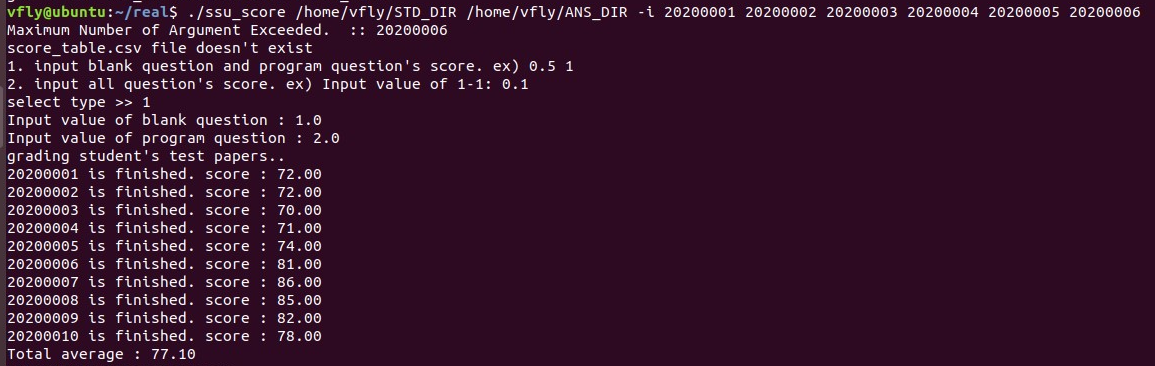
****

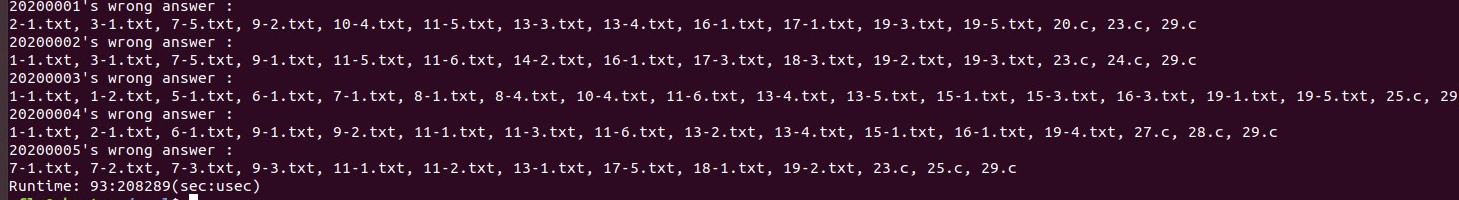
**- -i 옵션 적용 후 실행 시**

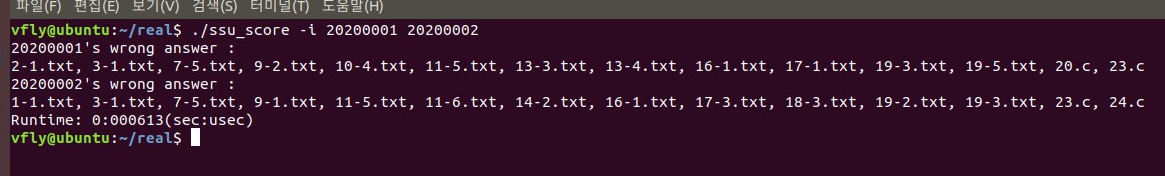
- 실행 후 인자로 넘겨받은 학번의 틀린 문제 출력 및 인자 개수 최대 5개

- 각 학번 별로 틀린 문제들을 출력

- <ANS\_DIR> <STD\_DIR>이 없어도 score.csv파일이 존재하면 틀린 문제 정보 출력

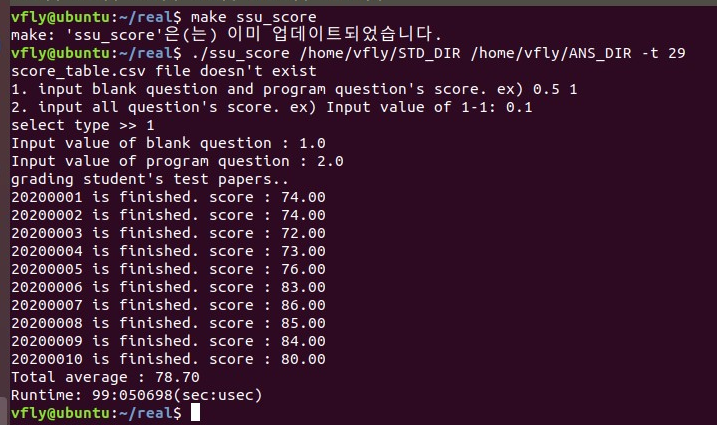
****

****

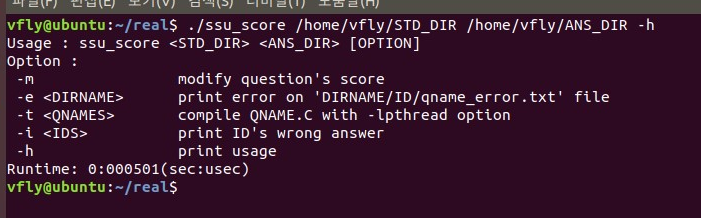
****

**- -t 옵션 적용 후 실행**

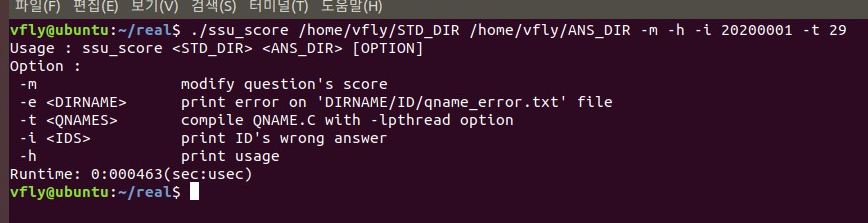
**-** 그냥 실행했을 때와는 점수가 다른 것을 확인 가능

****

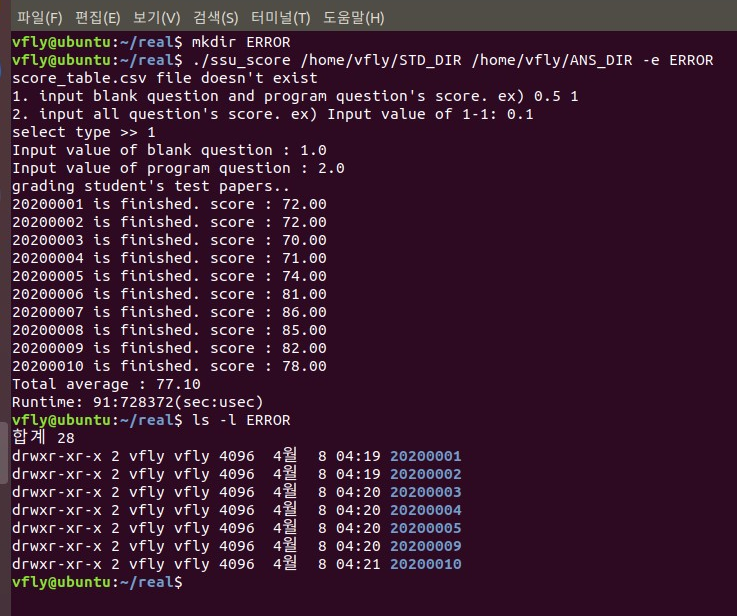
**- -h 옵션 적용 후 실행 ( 옵션 별 사용법을 출력 해준다 )**

****

**- 여러 개의 옵션 중 -h 있을 때 -h만 실행하고 종료**

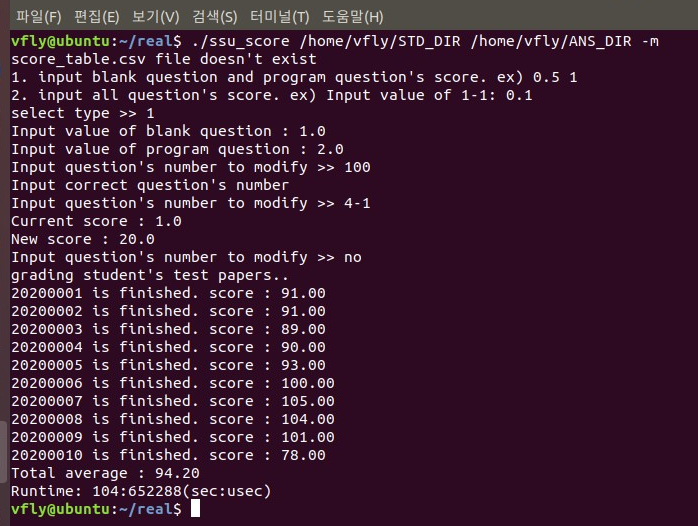
****

**- e 옵션 실행 후 디렉토리 확인**

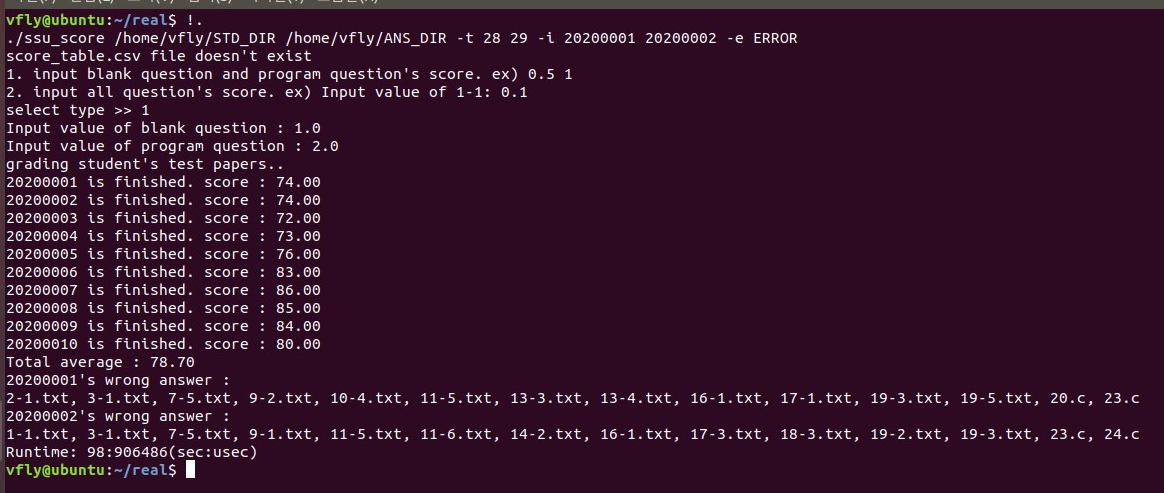
****

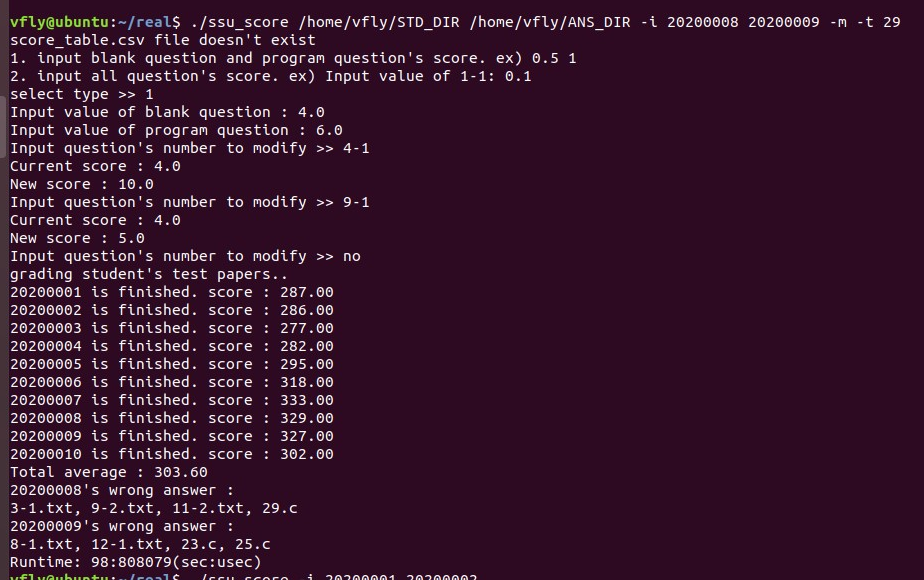
**- -m 옵션 적용 후 실행**

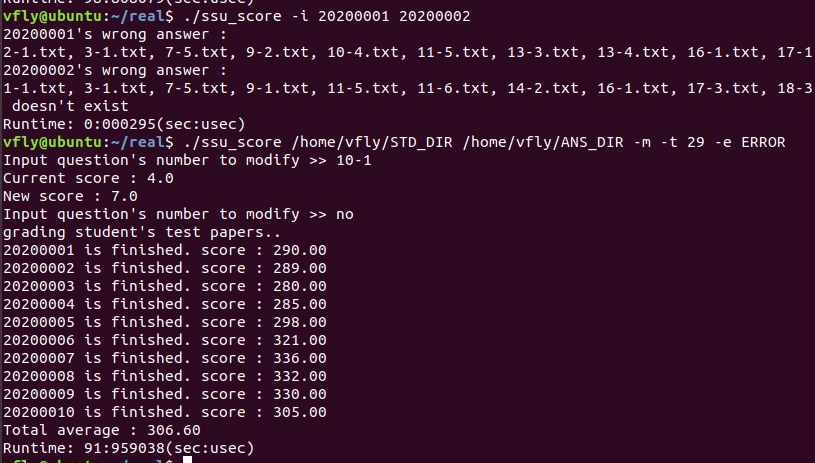
- 없는 문제 번호를 입력했을 경우 재입력 문구가 뜨도록 설정

****

**- 다수의 옵션 적용 시**

****

****

****

**5. 소스코드 및 주석**

**<main.c>**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/time.h>

#include "ssu\_score.h"

#define SECOND\_TO\_MICRO 1000000

//프로그램 수행 시간 측정

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

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

{

struct timeval begin\_t, end\_t;

//시작시간 체크

gettimeofday(&begin\_t, NULL);

ssu\_score(argc, argv);

//끝나는 시간 체크

gettimeofday(&end\_t, NULL);

//실행 시간 계산 및 출력

ssu\_runtime(&begin\_t, &end\_t);

exit(0);

}

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

{

//end\_t에다가 끝난시각 - 시작한 시각을 저장

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

//끝난 시각의 마이크로초가 더 작을 경우 초를 하나 줄이고 마이크로초 계산하여 반영

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

end\_t->tv\_sec--;

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

}

//마이크로초도 끝난 시각 - 시작시각

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

//걸리시간 초랑 마이크로초까지 출력

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

}

**<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];

//score table 선언

struct ssu\_scoreTable score\_table[QNUM];

//학생들 id 저장하는 배열

char id\_table[SNUM][10];

//학생 디렉토리 경로 저장 배열

char stuDir[BUFLEN];

//정답 디렉토리 경로 저장 배열

char ansDir[BUFLEN];

//e 옵션의 경우 에러디렉토리

char errorDir[BUFLEN];

//스레드파일의 정보를 저장하는 배열

char threadFiles[ARGNUM][FILELEN];

//틀린문제 정보 출력할 학번들 저장하는 배열

char cIDs[ARGNUM][FILELEN];

//옵션 여부 flag

int iOption = false;

int eOption = false;

int mOption = false;

int tOption = false;

//디렉토리없이 i옵션만 있는지 확인

int first\_is\_i = false;

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

{

//기존 경로 저장하기 위한 버퍼

char saved\_path[BUFLEN];

int i;

//맨처음이 i옵션인지 확인

if(!strcmp(argv[1], "-i"))

first\_is\_i = true;

//-h옵션이 존재하는지 확인

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

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

print\_usage();

return;

}

}

//saved\_path 초기화

memset(saved\_path, 0, BUFLEN);

//argv[1] 가 -i옵션이 아닐경우 학생디렉토리와 정답디렉토리를 저장

//-i옵션이면 score.csv파일만 읽어서 출력할 예정

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

strcpy(stuDir, argv[1]);

strcpy(ansDir, argv[2]);

}

//옵션들의 flag설정

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

exit(1);

//학생,정답 디렉토리없이 i옵션만 있을때 score.csv참고해서 정보 출력

if(first\_is\_i && iOption)

do\_iOption(cIDs);

//현재 작업디렉토리의 경로를 임시 버퍼에 저장

getcwd(saved\_path, BUFLEN);

//학생 디렉토리가 존재하지않을 경우 에러처리 존재하면 이동

if(chdir(stuDir) < 0){

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

return;

}

//학생디렉토리의 경로를 저장

getcwd(stuDir, BUFLEN);

//원래 위치로 다시 이동

chdir(saved\_path);

//정답 디렉토리가 존재하지 않을 경우 에러처리 존재하면 이동

if(chdir(ansDir) < 0){

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

return;

}

//정답 디렉토리의 경로를 저장한다

getcwd(ansDir, BUFLEN);

//원래 위치로 다시 돌아온다

chdir(saved\_path);

//scoreTable 세팅해준다

set\_scoreTable(ansDir);

//채점 시작전에 m옵션 수행

if(mOption) do\_mOption(ansDir);

//학번테이블 세팅

set\_idTable(stuDir);

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

//학생들 채점시작

score\_students();

//채점이 끝난 후 score.csv파일을 이용해 i옵션 수행

if(iOption)

do\_iOption(cIDs);

return;

}

//옵션들 점검

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

{

int i, j;

int c;

//e의 경우 추가인자가 발생

//t와 i도 발생하기 하지만 case에서 따로 처리해줌

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

{

switch(c){

//e 옵션의 경우 errorDir에 다음 인자를 저장함(에러디렉토리)

case 'e':

//e옵션 플레그 세우기

eOption = true;

strcpy(errorDir, optarg);

//파일이 존재하지않으면 새로 만듬

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

mkdir(errorDir, 0755);

//존재하면 지우고 만듬

else{

rmdirs(errorDir);

mkdir(errorDir, 0755);

}

break;

//t옵션일때

case 't':

tOption = true;

//다음 인자의 인덱스를 가리킴

i = optind;

j = 0;

//argc보다 작고 옵션을 나타내는 -가 아닐때까지

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

//최대인자 갯수를 넘어가면 적용되지않는 문제 출력

if(j >= ARGNUM)

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

//lpthread를 적용한 문제들 threadFiles에 저장

else

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

i++;

j++;

}

break;

//i옵션의 경우

case 'i':

//flag를 세워주고

iOption = true;

//i가 다음인자의인덱스를 저장

i = optind;

j = 0;

//argc보다 작고 옵션을 나타내는 -가 아닐때까지

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

//인자 최대갯수 5개를 넘어가면 넘어간거 출력

if(j >= ARGNUM)

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

//아니면 cIDS에 저장

else

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

i++;

j++;

}

break;

//m옵션의경우 flag만 세워줌

case 'm':

mOption = true;

break;

//지정한 옵션이 아닐경우 알수없다고 띄움

case '?':

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

return false;

}

}

return true;

}

//m옵션 수행

void do\_mOption(char \*path)

{

int fd;

char tmp[BUFLEN];

char buf[BUFLEN];

char buf2[BUFLEN];

char file\_path[BUFLEN];

int i;

int find=false;

double new\_score;

//score\_table.csv의 파일 경로를 저장

memset(file\_path, 0, BUFLEN);

sprintf(file\_path, "%s/%s", path, "score\_table.csv");

//문제별 배점이 저장된 score\_table.csv파일 RDONLY로 염

if ((fd = open(file\_path, O\_RDWR)) < 0)

{

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

return;

}

while(1)

{

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

scanf("%s",tmp);

//no를 입력받으면 break;

if(!strcmp(tmp, "no") || !strcmp(tmp, "NO") || !strcmp(tmp, "No")) break;

find = false;

//score\_table[i]에서 입력받은 문제를 search

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

{

//읽을게 없으면 끝

if(!strcmp(score\_table[i].qname, ""))

break;

//확장자표시 .을 가리키고

char \*p = strchr(score\_table[i].qname, '.');

//txt나 c파일 일경우에만 문제번호를 비교해서 같으면 find 플래그를 true로 설정

if(!strcmp(p, ".txt") || !strcmp(p, ".c"))

{

memset(buf, 0, BUFLEN);

strncpy(buf, score\_table[i].qname, strlen(score\_table[i].qname) - strlen(p));

if(!strcmp(buf,tmp))

{

find = true; break;

}

}

}

//존재하지 않는 문제번호면 다시 입력하게 유도

if(!find)

{

printf("Input correct question's number\n");

continue;

}

//존재하는 문제라면

else

{

//현재 배점을 보여주고

printf("Current score : %.1f\n", score\_table[i].score);

//새로운 배점을 저장해줌

printf("New score : "); scanf("%lf",&new\_score);

score\_table[i].score = new\_score;

//새롭게 업데이트된 정보들을 다시 써줌

lseek(fd, 0, SEEK\_SET);

//score\_Table 크기 계산

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

//파일에 변경된 내용 처음부터 다시쓰기

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

{

//읽을게 없으면 스탑

if (!strcmp(score\_table[i].qname, ""))

break;

memset(buf2, 0, BUFLEN);

sprintf(buf2, "%s,%.2f\n", score\_table[i].qname, score\_table[i].score);

write(fd, buf2, strlen(buf2));

}

}

}

}

//i옵션 실행

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

{

FILE \*fp;

char tmp[BUFLEN];

int i = 0;

char \*p, \*saved;

//score.csv에 접근이 불가능하면 리턴 1

if(access("score.csv", F\_OK) < 0)

return 1;

//score.csv파일 열기를 시도

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

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

return 0;

}

//score.csv파일 맨 첫줄을 읽어드림

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

//첫번째 문제이름 저장

p = strtok(tmp, ",");

strcpy(score\_table[0].qname, p);

//','을 토큰으로 하여 문제이름들 score\_table[i]에 하나씩 저장함

i=1;

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

{

//sum일경우 스탑

if(!strcmp(p, "sum")) break;

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

i++;

}

int index=i;

//score.csv파일에서 읽을게 없을때 까지

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

{

//','을토큰으로 학번의 시작을 가리킴

p = strtok(tmp, ",");

//score.csv에서 읽은 학번이 cIDs에 존재하는 확인

//없으면 continue

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

continue;

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

i=0;

//','을 토큰으로 점수를 하나씩읽음

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

{

//점수가 0점일 경우 문제이름 출력

if(atof(p) == 0.0)

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

i++;

}

//맨마지막에 ,생기는거 없애줌

printf("\b\b \n");

}

fclose(fp);

return 0;

}

//target이 cIDs배열에 존재하는지 확인

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

{

int i = 0;

//cIDS에 target이 존재하는지 확인

while(1)

{

//최대갯수 5개보다 넘어가면

if(i >= ARGNUM)

return false;

//더 이상 읽을게 없으면 false

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

return false;

//같으면 true 리턴

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

return true;

}

//없으면 false 리턴

return false;

}

//score\_table 세팅하는 함수

void set\_scoreTable(char \*ansDir)

{

char filename[FILELEN];

//filename에다가 "정답디렉토리 경로/score\_table.csv" 문자열을 저장한다

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

//파일이 이미 존재하면 scoreTable 읽기

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

read\_scoreTable(filename);

//파일이 존재하지 않으면 scoreTable을 만들고 쓴다.

else{

make\_scoreTable(ansDir);

write\_scoreTable(filename);

}

}

//score\_table이 존재하면 읽는 함수

void read\_scoreTable(char \*path)

{

FILE \*fp;

//확장자를 제외한 이름 만 저장하는 배열

char qname[FILELEN];

//점수 배점을 저장

char score[BUFLEN];

int idx = 0;

//"r" 모드로 넘겨받은 path에 있는 파일을 열기 실패하면 에러처리

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

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

return ;

}

//파일이 끝까지 읽는데 ','전까지 qname에 저장 나머지는 score에 저장

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

//score\_table.qname에 하나씩 문제이름 저장

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

//sore\_table.score에 하니씩 문제이름 저장

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

}

//열었던 파일 닫아줌

fclose(fp);

}

//score\_table 제작

void make\_scoreTable(char \*ansDir)

{

int type, num;

double score, bscore, pscore;

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

DIR \*dp, \*c\_dp;

char tmp[BUFLEN];

int idx = 0;

int i;

//어떤 형식으로 score\_table을 만들껀지 결정

//1이면 한번에 형식 지정, 2면 문제하나씩 지정

num = get\_create\_type();

//1이면 빈칸문제와 프로그램 문제 점수 입력

if(num == 1)

{

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

scanf("%lf", &bscore);

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

scanf("%lf", &pscore);

}

//정답 디렉토리를 열기를 시도, 실패하면 에러처리

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

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

return;

}

//dp를 통해 디렉토리에서 파일들을 하나씩 읽어들임

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

{

//. 이나 .. 일 경우 무시한

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

continue;

//tmp 배열에다가 "정답 디렉토리 경로/읽은 파일 이름"을 저장

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

//score\_table.qname에 문제이름을 추가함

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

}

closedir(dp);

//score\_table을 정렬해준다

sort\_scoreTable(idx);

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

{

//파일의 타입을 읽어옴

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

//한번에 결정하는 경우 였을경우

if(num == 1)

{

//.txt 파일과 .c 파일 각각에 점수 배점

if(type == TEXTFILE)

score = bscore;

else if(type == CFILE)

score = pscore;

}

//일일이 점수를 배점하는 경우

else if(num == 2)

{

//입력받아서 배점

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

scanf("%lf", &score);

}

//scoreTable.score부분에 점수배점 등록

score\_table[i].score = score;

}

}

//score\_table 작성

void write\_scoreTable(char \*filename)

{

int fd;

char tmp[BUFLEN];

int i;

//score\_table의 크기 계산

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

//filname의 이름으로 파일을 생성 실패시 에러처리

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

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

return;

}

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

{

//0점을 만나면 break

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

break;

//문제이름과 점수 배점을 합쳐서 tmp에 저장

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

//fd를 통해 tmp 배열을 작성

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

}

close(fd);

}

//학번테이블 세팅

void set\_idTable(char \*stuDir)

{

struct stat statbuf;

struct dirent \*dirp;

DIR \*dp;

char tmp[BUFLEN];

int num = 0;

//학생 디렉토리를 열기를 시도 실패시 에러처리

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

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

exit(1);

}

//학생디렉토리를 열었을때

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

//'.', '..'은 무시

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

continue;

//학번 디렉토리의 경로를 저장

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

//디렉토리 정보 추출

stat(tmp, &statbuf);

//디렉토리이면 id\_table의 학번추가

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

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

//그렇지 않으면 continue

else

continue;

}

//학번 테이블 오름차순 정렬

sort\_idTable(num);

}

//학번 테이블 정렬

void sort\_idTable(int size)

{

int i, j;

char tmp[10];

//버블 정렬을 이용

//학번을 오름차순으로 정렬

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);

}

}

}

}

//score\_table 정렬시켜줌

void sort\_scoreTable(int size)

{

int i, j;

struct ssu\_scoreTable tmp;

int num1\_1, num1\_2;

int num2\_1, num2\_2;

//버블정렬을 이용

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

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

//비교할 두개의 이름에서 번호를 추출함

//1-2.txt의 경우 num1\_1에 1 num1\_2에 2가 들어감

get\_qname\_number(score\_table[j].qname, &num1\_1, &num1\_2);

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

//앞번호가 크면 뒤로, 앞번호가 같으면 뒷번호를비교해서 뒷번호가 큰걸 뒤로 보냄

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

//서로 교환

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]));

}

}

}

}

//문제해서 번호를 추출

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

{

char \*p;

char dup[FILELEN];

//dup에다가 문제이름을 복사해줌

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

//num1에다가 "-."전까지 끝어서 저장해줌

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

//p에다가 "-."로 다시한번 토큰화

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

//NULL이면 0

if(p == NULL)

\*num2 = 0;

//존재하면 정수로 변환후 저장

else

\*num2 = atoi(p);

}

//어떤 형태로 score\_table을 만들지 판별

int get\_create\_type()

{

int num;

//1이나 2를 입력받을때까지 반복

while(1)

{

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

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

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

printf("select type >> ");

scanf("%d", &num);

//1이나 2가 아니면 다시

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

printf("not correct number!\n");

else

break;

}

//1이나 2를 리턴

return num;

}

//학생들 채점함수

void score\_students()

{

double score = 0;

int num;

int fd;

char tmp[BUFLEN];

//학번테이블의 크기 계산

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

//score.csv파일을 생성하고 에러처리

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

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

return;

}

//score.csv의 첫줄을 작성하는 함수

write\_first\_row(fd);

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

{

//학번테이블을 읽었을 때 빈거면 break

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

break;

//그렇지 않다면 학번, 을 fd를 통해 작성

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

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

//학생의 점수를 구한다

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

}

//전체평균 출력

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

close(fd);

}

//학생 개인 채점 함수

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

{

int type;

double result;

double score = 0;

int i;

char tmp[BUFLEN];

//score\_table의 크기를 계산

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

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

{

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

break;

//경로 "학생디렉토리/학번/문제이름"을 tmp 에 저장

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

//파일에 접근 가능한지 판별 없으면 틀린거임

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

result = false;

//접근 가능할때

else

{

//파일 타입을 가져온다, .txt나 .c이외의 것을 경우 continue

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

continue;

//.txt파일이면 빈칸문제 채점

if(type == TEXTFILE)

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

//.c파일이면 프로그램 채점

else if(type == CFILE)

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

}

//결과가 false일때 ( 파일이 없거나, 0점이거나 )

if(result == false)

write(fd, "0,", 2);

//결과가 뭐든지 간에 존재할때

else{

if(result == true){

//다 맞았으면 score에 추가시켜줌

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

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

}

else if(result < 0){

//warning이 있을 경우 점수를 감점

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

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

}

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

}

}

//학번과 채점 결과 출력

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

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

//결과파일에 점수 합계 작성

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

return score;

}

//score.csv의 첫줄

void write\_first\_row(int fd)

{

int i;

//임시버퍼

char tmp[BUFLEN];

//scoretable의 크기 구하기

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

//맨처음은 공백

write(fd, ",", 1);

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

//점수 배점이 없는걸 만나면 break

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

break;

//문제이름을 tmp에 저장한후

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

//fd를 통해 파일에 작성

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

}

//마지막에sum 출력

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

}

//파일에서 :단위로 문자열 읽어줌

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

{

char c;

int idx = 0;

//버퍼 초기화

memset(result, 0, BUFLEN);

//한글자 씩 읽는데 :를 기준으로 종료

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

{

if(c == ':')

break;

//한글자씩저장

result[idx++] = c;

}

//끝에다가 개행이면 널문자로 대체

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

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

return result;

}

//빈칸문제 채점

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

{

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;

//qname 초기화

//qname에 확장자를 제외한 이름만 저장

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

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

//tmp의 "학생디렉토리/학번/문제" 저장

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

//문제파일 열기

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

//파일에서 :단위로 적은답을 가져와서 s\_answer에 저장

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

//적은 답이 없으면 return false

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

close(fd\_std);

return false;

}

//괄호의 갯수가 맞지 않으면 return false

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

close(fd\_std);

return false;

}

//가져온 답의 앞뒤 공백을 없애준다

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

//맨뒤가 세미콜론이면 세미콜론이 있다는 flag를 세워주고 널문자로 대체

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

has\_semicolon = true;

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

}

//답을 토큰화 시켰는데 문제가 있으면 틀림

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

close(fd\_std);

return false;

}

idx = 0;

//토큰화 시킨걸로 트리를 제작함

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

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

//정답디렉토리/문제 저장함

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

//문제파일을 열어줌

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

while(1)

{

ans\_root = NULL;

result = true;

//토큰이 들어있던 배열 초기화 후 재사용

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

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

//정답파일에서 : 단위로 답을 가져와 버퍼에 저장

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

//공백이면 break

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

break;

//답의 앞뒤 공백을 제거한다

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

//학생답에 세미콜론 없었는데 정답에는 있으면 continue

if(has\_semicolon == false){

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

continue;

}

//학생답에 세미콜론이 있었는데

else if(has\_semicolon == true)

{

//정답에도 있으면 continue

if(a\_answer[strlen(a\_answer) - 1] != ';')

continue;

//그렇지 않으면 맨 마지막에 널문자로 대체

else

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

}

//정답을토큰화 시킨다

if(!make\_tokens(a\_answer, tokens))

continue;

idx = 0;

//토큰화 시킨걸로 트리를 제작

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

//트리 비교

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

//결과가 true면 트리들 다 free 시켜줌

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;

}

}

//트리들 다 free시켜줌

close(fd\_std);

close(fd\_ans);

if(std\_root != NULL)

free\_node(std\_root);

if(ans\_root != NULL)

free\_node(ans\_root);

//틀림

return false;

}

//프로그램 문제 채점

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

{

double compile;

int result;

//학번과 파일이름을 넘겨주면서 컴파일을 시켜봄

compile = compile\_program(id, filename);

//ERROR이거나 false가뜨면 실패

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

return false;

//학번과 파일이름을 넘겨주면서 실행시킴

result = execute\_program(id, filename);

//결과가 다르거나 실행에 문제가 있을시 return false

if(!result)

return false;

//컴파일했을떄 error는 없고 warning이 있을 때 그 점수 리턴

if(compile < 0)

return compile;

//아무 문제없으면 true 리턴

return true;

}

//스레드파일인지 확인

int is\_thread(char \*qname)

{

int i;

//threadFiles 배열의 크기를 구함

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

//같은게 있으면 return true

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

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

return true;

}

//없으면 false

return false;

}

//프로그램 컴파일시킴

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

{

int fd;

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

char command[BUFLEN];

char qname[FILELEN];

int isthread;

off\_t size;

double result;

//qname버퍼 초기화

//qname에다가 확장자를 제외한 이름만 저장

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

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

//쓰레드파일인지 확인

isthread = is\_thread(qname);

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

//"정답디렉토리/문제이름"을 경로 저장

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

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

//실행파일의 경로를 저장

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);

//컴파일 에러나 warning이 떳을때 문구들을 저장할 파일의 경로를 저장

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

//컴파일 에러확인용 파일 생성

fd = creat(tmp\_e, 0666);

//STDERR를 fd로 재설정하고 명령어 실행

redirection(command, fd, STDERR);

//파일포인터를 맨끝으로 보냄

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

close(fd);

//에러파일 삭제

unlink(tmp\_e);

//파일의 크기가 0보다 크면 return false

if(size > 0)

return false;

//"학생디렉토리/학번/문제이름"을 저장

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

//실행파일 경로를 배열에 저장

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);

//에러가 났을때 출력할 에러파일 경로 저장

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

//에러출력할 파일 생성

fd = creat(tmp\_f, 0666);

//STDERR를 fd로 재지정하고 명령어 실행

redirection(command, fd, STDERR);

//파일포인터를 파일의 끝으로 이동

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

close(fd);

//에러나 warning이 있었을 경우

if(size > 0){

//e 옵션이 존재하는경우

if(eOption)

{

//에러디렉토리/학번 저장

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

//만약 그 디렉토리가 존재하지 않으면 디렉토리생성

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

mkdir(tmp\_e, 0755);

//에러를 출력할 파일 경로저장

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

//디렉토리에서 파일로 tmp\_f 변경

rename(tmp\_f, tmp\_e);

//error나 warning의 갯수를 셈

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

}

//e 옵션 없을경우

else{

//error나 warning의 갯수를 세고

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

//error출력했던 파일 unlink

unlink(tmp\_f);

}

//결과 값 리턴

return result;

}

//error나 warning출력했던 파일 unlink후 결과 리턴

unlink(tmp\_f);

return true;

}

//컴파일 결과중 error 와 warning 체크

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

{

FILE \*fp;

char tmp[BUFLEN];

double warning = 0;

//파일이름을 넘겨 받아서 열기를 시도 열기 실패시 에러처리

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

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

return false;

}

//파일의 내용중 error:가 있으면 return ERROR warning:이 있으면 갯수를 셈

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

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

return ERROR;

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

warning += WARNING;

}

//에러는 안났고 warning만 있는경우 점수 합산해서 넘겨줌

return warning;

}

//프로그램 실행

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

{

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

char tmp[BUFLEN];

char qname[FILELEN];

time\_t start, end;

pid\_t pid;

int fd;

//qname 초기화

//qname에다가 확장를 뺸 이름만 저장

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

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

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

//결과 저장 파일 경로를 ans\_fname에 저장

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

//결과를 저장할 파일 생성

fd = creat(ans\_fname, 0666);

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

//실행 파일 경로를 tmp에 저장

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

//STDOUT과 STDERR를 fd로 변경시킨후 복구

int out = dup(STDOUT);

int err = dup(STDERR);

//STDOUT,STDERR를 fd로 바꿈

dup2(fd, STDOUT); dup2(fd, STDERR);

system(tmp);

//원래대로 복구

dup2(out, STDOUT); dup2(err, STDERR);

close(out); close(err);

//redirection(tmp, fd, STDOUT);

close(fd);

//학생파일의 결과를 저장할 파일의 경로를 std\_fname에 저장

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

//결과를 저장할 파일 생성

fd = creat(std\_fname, 0666);

//실행파일의 경로를 tmp에 저장하는데 백그라운드로서 실행하기 위해 &를 붙혀줌

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

//시작시간 측정

start = time(NULL);

//STDOUT과 STDERR를 fd를 바꿔주고 복구

out = dup(STDOUT);

err = dup(STDERR);

//STDOUT STDERR를 fd로 변경

dup2(fd, STDOUT); dup2(fd, STDERR);

system(tmp);

//원래대로 복구

dup2(out, STDOUT); dup2(err, STDERR);

close(out); close(err);

//redirection(tmp, fd, STDOUT);

//실행파일의 이름을 tmp에 저장

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

//백그라운드에 존재할경우 pid를 받음

while((pid = inBackground(tmp)) > 0){

end = time(NULL);

//시간을 측정해서 5초보다 길어지면 강제롤 kill신호를 보내서 죽임

if(difftime(end, start) > OVER){

kill(pid, SIGKILL);

close(fd);

//5초이상 걸렸기 떄문에 틀림

return false;

}

}

close(fd);

//결과를 비교

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

}

//백그라운드 실행

pid\_t inBackground(char \*name)

{

pid\_t pid;

char command[64];

char tmp[64];

int fd;

off\_t size;

//임시 버퍼 초기화

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

//현재 실행중인 목록을 저장할 파일 생성

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

//ps -> 현재 실행 중인 프로세스를 출력을 표준입력으로 하여 grep %s를 실행하는 명령어를 구성

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

//명령어를 실행

redirection(command, fd, STDOUT);

//파일 포인터를 처음으로 보내고

lseek(fd, 0, SEEK\_SET);

//tmp만큼 읽는다

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

//비어있으면 "background.txt"파일 unlink

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

unlink("background.txt");

close(fd);

return 0;

}

//뭔가 있으면 " "을 토큰으로 pid번호를 저장

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

close(fd);

//그 후 unlink

unlink("background.txt");

//pid를 넘겨줌

return pid;

}

//결과 파일을 비교한다

//프로그램 문제의 경우

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

{

int fd1, fd2;

char c1, c2;

int len1, len2;

//파일들의 경로를 받고 읽기모드로 열어준다

fd1 = open(file1, O\_RDONLY);

fd2 = open(file2, O\_RDONLY);

while(1)

{

//fd1에서 문자하나를 읽는다

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

//공백이면 continue

if(c1 == ' ')

continue;

//그렇지 않으면 스탑

else

break;

}

//fd2에서 문자하나를 읽는다

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

//공백은 무시

if(c2 == ' ')

continue;

//그렇지 않으면 스탑

else

break;

}

//둘다 읽은 문자가 널 문자면 스탑

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

break;

//가져온 문자 2개를 둘다 소문자로 변경

to\_lower\_case(&c1);

to\_lower\_case(&c2);

//서로다르면 틀린거임

if(c1 != c2){

close(fd1);

close(fd2);

return false;

}

}

//다 같았으면 정답

close(fd1);

close(fd2);

return true;

}

//파일디스크립터 변경

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

{

//기존 파일디스크립터를 저장하는 변수

int saved;

//old의 파일디스크립터를 saved에 저장

saved = dup(old);

//old를 new로 복사

dup2(new, old);

//명령어 실행

system(command);

//old에다가 saved를 저장

//결국 기존껄로 돌아옴

dup2(saved, old);

close(saved);

}

//파일을 타입을 읽어드림

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

{

//filname 에서 '.'을 찾아서 위치 리턴받음

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

//빈칸문제인경우

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

return TEXTFILE;

//프로그램문제인경우

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

return CFILE;

//예외

else

return -1;

}

//디렉토리 삭제 함수

void rmdirs(const char \*path)

{

struct dirent \*dirp;

struct stat statbuf;

DIR \*dp;

char tmp[BUFLEN];

//열기 실패시 리턴

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

return;

//디렉토리에서 하나씩 읽음

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

{

//. , .. 은 무시한다

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

continue;

//"경로/파일이름" 을 tmp에 저장

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

//stat을 통해 정보를 얻어 올 수 없으면 무시함

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

continue;

//만약 그 파일이 디렉토리라면 그 디렉토리안에 있는 것들을 삭제하기 위해 rmdirs 실행

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

rmdirs(tmp);

//디렉토리가 아니면 그냥 unlink

else

unlink(tmp);

}

closedir(dp);

//마지막으로 이 디렉토리도 삭제

rmdir(path);

}

//영어소문자 일경우 대문자로 변경

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

{

//영어 소문자 'a'가 97이기 때문이 'A' = 65 에서 32를 더하면 소문자가 됨

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

\*c = \*c + 32;

}

//프로그램사용방법 출력

void print\_usage()

{

printf("Usage : ssu\_score <STD\_DIR> <ANS\_DIR> [OPTION]\n");

printf("Option : \n");

//m 옵션은 특정 문제의 배점 수정

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

//e 옵션은 디렉토리 이름을 건네주면 error정보를 학번별로 저장해줌

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

//-t 옵션은 문제에 lpthread기능 추가

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

//특정 학번의 틀린 문제 정보 출력

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

//사용법 출력

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

}

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

**<blank.c>**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <ctype.h>

#include "blank.h"

//데이터타입들을 미리 저장해둔 배열

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" };

//우선순위 저장 값이 작을수록 높은 우선순위

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}

};

//트리 비교함수

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

{

node\* tmp;

int cnt1, cnt2;

//둘 중하나가 null일경우 return에다가 false 저장 후 종료

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

\*result = false;

return;

}

// 비교연산자의 경우

//한쪽을 변경해서 자식들의 순서를 바꿔줌

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);

//자식 순서를 바꿔줌

root2 = change\_sibling(root2);

}

}

//현재 노드의 이름이 다르면 return false

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

\*result = false;

return;

}

//한쪽은 자식이 있는데 한쪽은 자식이없으면 false

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

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

\*result = false;

return;

}

//root1의 자식이 존재하면

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

//root1과 root2의 자식갯수를 세서 다르면 false

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

\*result = false;

return;

}

//같은지 연산하는 연산자의 경우

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

{

//자식으로 이동후 트리비교함수 다시 호출

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

//자식트리를 비교했는데 false일 경우

if (\*result == false)

{

//결과를 true로 다시 설정 후

\*result = true;

//root2는 자식의 순서를 변경

root2 = change\_sibling(root2);

//다시 비교

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

}

}

//조건식의 연산자 일경우

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

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

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

{

//자식의 갯수가 다르면 false

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

\*result = false;

return;

}

//tmp의 roo2의 자식노드를 저장하고

tmp = root2->child\_head;

//이전이 null이 아닌동안 이전으로 이동

while (tmp->prev != NULL)

tmp = tmp->prev;

//root1과 root2를 따라가는 임시 tmp를 이용해 자식들을 비교한다

while (tmp != NULL)

{

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

//자식이 같으면 break;

if (\*result == true)

break;

//다르면 tmp를 next로 이동시킨후 다시 비교

else {

if (tmp->next != NULL)

\*result = true;

tmp = tmp->next;

}

}

}

//다른 연산자들의 경우 자식으로가서 비교함수 다시 호출

else {

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

}

}

//root1의 다음이 존재한다면

if (root1->next != NULL) {

//자식의 갯수가 같은지 확인하고 다르면 false

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

\*result = false;

return;

}

//자식 비교해서 다른게 없었다면

if (\*result == true)

{

tmp = get\_operator(root1);

//순서가 바뀌어도 상관없는 연산자의 경우

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

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

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

{

//임시로 tmp에 저장 후

tmp = root2;

//tmp에 이전이 없을 때까지 tmp을 이전으로 이동시킴

while (tmp->prev != NULL)

tmp = tmp->prev;

//tmp가 뭔가를 가리키고 있다면

while (tmp != NULL)

{

//트리 비교

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

//결과가 똑같다면 break

if (\*result == true)

break;

//결과가 다르다면

else {

//tmp에 next가 존재하면 결과를 true로 설정

if (tmp->next != NULL)

\*result = true;

//tmp를 다음으로 이동시킴

tmp = tmp->next;

}

}

}

//순서가 중요한 연산자일 경우 트리 둘다 next로 이동 후 비교

else

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

}

}

}

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

{

char\* start, \* end;

char tmp[BUFLEN];

char str2[BUFLEN];

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

int row = 0;

int i;

int isPointer;

int lcount, rcount;

int p\_str;

//tokens배열 초기화

clear\_tokens(tokens);

start = str;

//자료형이나 gcc확인후 에러 뜨면 false 리턴

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

return false;

while (1)

{

//op중 하나라도 걸리는 문자가 있으면 거기가 end 없으면 종료

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

break;

//start와 end가 가리키는게 op중 하나라면

if (start == end) {

//증감연산자의 경우

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

//4개씩 있는 경우 return false

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

return false;

//증감연산자를 건너뛰고 다음이 문자라면 전위연산자

if (is\_character(\*ltrim(start + 2))) {

//앞에 토큰이 하나존재하고 이전이 문자라면 에러

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

return false;

//증감연산자 건너뛰고 다음 op에 걸리는 것을 end가 가리키게함

end = strpbrk(start + 2, op);

//없으면 문자열 맨끝으로 이동시킴

if (end == NULL)

end = &str[strlen(str)];

//그 사이에 있는 데이터들 토큰에 하나씩추가

while (start < end) {

//변수명 사이에 공백이 존재하면 에러

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

return false;

//공백이 아니면 추가

else if (\*start != ' ')

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

start++;

}

}

//앞에 토큰이 하나 존재하고 row-1토큰의 마지막 문자가 character라면 ++은 후위 연산자

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

//row-1토큰이 ++이면 에러

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

return false;

//이전토큰에다가 증감연산자 붙여줌

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

strncpy(tmp, start, 2);

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

start += 2;

row--;

}

//나머지 경우 토큰에 추가시켜줌

else {

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

strncpy(tmp, start, 2);

strcat(tokens[row], tmp);

start += 2;

}

}

//비교연산자, 대입연산자를 축소시킨 것들인 경우 그냥 토큰배열에 추가

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)) {

//토큰배열에 2개 복사

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

//포인터 2칸 이동

start += 2;

}

//화살표 연산자의 경우

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

{

//화살표연산자를 건너뛰고 그 다음 op에 걸리는 무언가를 찾음

end = strpbrk(start + 2, op);

//그 다음이 없으면 끝 부분을 이 함수에서 인자로 받은 문자열의 끝으로 이동시킴

if (end == NULL)

end = &str[strlen(str)];

//공백을 제외한 문자열들을 이전토큰에 넣어줌 [row-1]에 ->와 ->다음 문자열들을 추가

while (start < end) {

if (\*start != ' ')

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

start++;

}

//토큰 한줄 뒤로

row--;

}

//주소추출 연산자, 비트연산일경우

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

{

//첫번째 토큰이거나 이전 토큰이 연산자일 경우 - 주소연산자

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

//주소연산자를 제외하고 다음 op에 걸리는 연산자를 가리킴

end = strpbrk(start + 1, op);

//없으면 문자열 맨뒤를 가리키게 함

if (end == NULL)

end = &str[strlen(str)];

//먼저 주소연산자를 토큰에 넣어줌

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

start++;

//그 사이의 내용들을

while (start < end) {

//주소연산자와 변수명사이에 공백이 아니라 문자열간에 공백이 있으면 에러

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

return false;

//공백이 아니면 토큰에 추가시켜줌

else if (\*start != ' ')

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

start++;

}

}

//아니면 비트연산자일경우 그냥 토큰에 추가시킴

else {

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

start += 1;

}

}

//포인터거나 곱셈의 연산자 '\*'의 경우

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

{

//포인터인지 확인용 flag

isPointer = 0;

//앞에 최소 토큰이 1개 있을때

if (row > 0)

{

//앞 토큰이 자료형인지 확인

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

//자료형일경우 \*을 앞토큰에 \*을 붙혀줌

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

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

start += 1;

//포인터 플래그 활성화

isPointer = 1;

break;

}

}

//만약 포인터라면 다음 토큰작업으로 이동

if (isPointer == 1)

continue;

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

end = start + 1;

//앞에 토큰이 2개 존재하고 row-2토큰이 \*이고 row-1이 다 '\*'토큰이면

//row-1에다가 \*추가해주고 row값 유지

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

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

row--;

}

//row-1토큰이 문자들로만 이루어진 경우 새로운 토큰 \*을 추가

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

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

}

//row-1토큰이 연산자로 이루어진 경우 포인터를 나타내는 \*토큰 새로 추가

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

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

}

//나머지의 경우들은 그냥 \*을 하나로 토큰화 시킴

else

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

start += (end - start);

}

//맨 처음 토큰일 경우

else if (row == 0)

{

//start 위치를 하나 증가시킨상태에서 op에 걸리는 다음 연산자를 찾아서 end가 가리키게함

//end가 null일 경우 start가 가리키는 '\*' 토큰화

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

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

start += 1;

}

//걸리는 연산자가 있을 경우

else {

//start와 end사이 문자들을 토큰화 시켜줌

while (start < end) {

//문자간 공백이 있을 경우 return false

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

return false;

//공백이 아니면 토큰에 넣어줌

else if (\*start != ' ')

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

start++;

}

//만약 집어넣은 문자들이 다 \*이면 row값 유지

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

row--;

}

}

}

//여는 괄호일경우

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

{

lcount = 0;

rcount = 0;

//이전이 포인터,주소관련 연산자 였을경우

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

//마지막 여는 괄호를 만날때까지 이동시키고 갯수를 셈

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

lcount++;

start += lcount;

//그 다음 닫는 괄호를 찾아 )이동

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

//없으면 false

if (end == NULL)

return false;

else {

//마지막 닫는 괄호를 만날때까지 이동시키고 갯수를 셈

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

rcount++;

end += rcount;

//만약 여는 괄호랑 닫는괄호 갯수가 맞이 않으면 false

if (lcount != rcount)

return false;

//앞에 토큰이 2개존재하고 2번째전 토큰의 마지막 문자가 character가 아닌경우 와 row==1일때

//&,\*일경우

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

//앞토큰에다가 괄호를 제외한 정보를 추가시켜줌

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

row--;

start = end + 1;

}

//나머지면 그냥 토큰에 추가

else {

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

start += 1;

}

}

}

//포인터관련 연산자가 아니면 그냥 토큰에 추가

else {

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

start += 1;

}

}

//큰 따움표의 경우

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

{

//처음 찾은 따움표를 건너뛰고 닫는 따움표를 찾아서 end가 가리키게 함

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

//없으면 return false -> 틀린거임

if (end == NULL)

return false;

//찾았으면

else {

//따움표와 따움표사이의 내용을 토큰에 추가

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

start = end + 1;

}

}

//이외의 연산자의 경우

else {

//앞 토큰이 ++이면 에러

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

return false;

//앞 토큰이 --이면 에러

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

return false;

//일단 연산자 추가후

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

start += 1;

//조건문에 연산자일 경우

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

//맨처음인경우

//부호나 전위증감연산자일경우

if (row == 0)

row--;

//이전 토큰이character가 아니라면

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

//이전토큰에서 ++나 --을 찾아서없으면 row를 증가시키지 않고 그대로 둠

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

row--;

}

}

}

}

//start와 end 사이에는 문자열이 존재 ex)변수명

else {

//앞 토큰이 다 '\*'이 포인터를 나타내는 것이면 row 증가x

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

row--;

//row ==1 이고 앞이 '\*'를 나타내면 row 증가 x

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

row--;

//start와 end사이에 문자들을 토큰에 넣어줌

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

//.으로 연결되는 문자열도 하나의 토큰에 집어넣어줌

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

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

//공백은무시하고 진행

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

i++;

}

//공백은 무시하고 진행

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

while (start[i] == ' ')

i++;

break;

}

//일반적인 경우면 토큰에 하나 넣어줌

else

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

}

//공백을 만나면 start와 end사이에 문자열 길이만큼 start 이동시킴

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

start += i;

continue;

}

//start와 end사이에 문자열 길이만큼 start 이동시킴

start += i;

}

//토큰의 왼쪽 오른쪽 공백을 지워버림

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

//(앞에 1개토큰이 존재하고 현재토큰의 끝이 문자이고 (자료형이거나 row-1의끝이 문자거나 '.'))일때

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] == '.')) {

//앞에 토큰이 2개있고 row-2 가 여는 괄호인데

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

{

//row-1이 struct나 unsigned가 아니면 false

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

return false;

}

//row == 1이고 현재토큰의 맨마지막이 문자일때

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

//extern,unsigned나 자료형이 없으면 false

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

return false;

}

//앞에 토큰이 2개있고 row-1토큰이 타입명일때

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

//row-2에 unsigned, extern이 없으면 false

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

return false;

}

}

//첫번째 토큰이고 그 토큰이 gcc인 경우 문자열 전체를 토큰에 넣어버림

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

//토큰을 담았던 tokens 리셋시키고

clear\_tokens(tokens);

//문자열 토큰의 맨 처음에 넣고

strcpy(tokens[0], str);

//return 1

return 1;

}

//한줄 증가

row++;

}

//맨 마지막에 남은 토큰 처리

//앞 토큰이 다 '\*'이 포인터를 나타내는 것이면 row 증가x

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

row--;

//row ==1 이고 앞이 '\*'를 나타내면 row 증가 x

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

row--;

//남은 문자 토큰처리

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

{

//공백은 무시하고 start증가

if (start[i] == ' ')

{

while (start[i] == ' ')

i++;

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

start += i;

i = 0;

}

else

row++;

i--;

}

//공백이 아니라면

else

{

//문자 하나를 토큰에 집어넣고

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

//만약 '.'이고 아직 다 넣지 않았다면

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

//'.'다음 공백들을 무시함

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

i++;

}

}

//토큰 앞뒤 공백을 제거시킴

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

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--;

}

//(앞에 1개토큰이 존재하고 현재토큰의 끝이 문자이고 (자료형이거나 row-1의끝이 문자거나 '.'))일때

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] == '.')) {

//앞에 토큰이 2개있고 row-2 가 여는 괄호인데

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

{

//row-1이 struct나 unsigned가 아니면 false

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

return false;

}

//row == 1이고 현재토큰의 맨마지막이 문자일때

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

//extern,unsigned나 자료형이 없으면 false

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

return false;

}

//앞에 토큰이 2개있고 row-1토큰이 타입명일때

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

//row-2에 unsigned, extern이 없으면 false

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

return false;

}

}

}

//앞에 토큰이 하나 있는데

if (row > 0)

{

//#include계열이나 strcut계열이면

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

//토큰들 다 초기화시키고

clear\_tokens(tokens);

//첫번째 토큰에다가 공백을 지운 문자열을 넣어줌

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

}

}

//첫번째 토큰이 자료형이거나 extern일 경우

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

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

//읽을 토큰이 없으면 break;

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

break;

//i가 토큰배열의 가장 마지막이 아니라면

if (i != TOKEN\_CNT - 1)

//extern 다음에 공백 하나 추가

strcat(tokens[0], " ");

//첫번째 토큰에다가 i번째 토큰을 연결 시켜줌 ( extern 변수명 )

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

//i번째 토큰을 0으로 초기화 시켜줌

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

}

}

//자료형과 특정조건문을 만족하는 토큰을 찾으면 인덱스를 리턴받고 토큰리셋 진행

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

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

return false;

}

//struct 관련 토큰이 있는 index를 찾아서 p\_str에 넣고 토큰리셋 진행

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

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

return false;

}

return true;

}

//트리제작

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

{

node\* cur = root;

node\* new;

node\* saved\_operator;

node\* operator;

int fstart;

int i;

while (1)

{

//토큰이 비어있으면 끝

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

break;

//닫는 괄호를 만나면 루트를 리턴

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

return get\_root(cur);

//','를 만나도 루트를 리턴

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

return get\_root(cur);

//여는 괄호일 경우

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

{

//이전에 토큰이 1개 존재하고 이전 토큰이 연산자가 아니고 ','가 아니면

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

//괄호 시작을 알리는 flag

fstart = true;

while (1)

{

//괄호는 트리에 안넣고 인덱스 하나 증가

\*idx += 1;

//현재 토큰이 닫는 괄호라면 스탑

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

break;

//닫는괄호가 아니라면 새로운 트리를 생성하게 하고 괄호갯수를 하나 증겨서 만듬

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

//트리를 생성했다면

if (new != NULL) {

//여는괄호 flag가 세워져있다면 cur이 가리키는 노드의 자식으로 연결

if (fstart == true) {

cur->child\_head = new;

new->parent = cur;

fstart = false;

}

//그렇지 않으면 next로 연결시켜줌

else {

cur->next = new;

new->prev = cur;

}

cur = new;

}

//닫는 괄호라면 break 다시한번 체크

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

break;

}

}

else {

\*idx += 1;

//괄호갯수인자를 하나 추가해서 추가적인 트리를 제작

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

//트리가 없는 상태이면 cur = new

if (cur == NULL)

cur = new;

//cur의 이름과 new의 이름이 같으면

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

//순서를 바꿔도 상관이 없는 연산자들의 경우

//조건문의 연산자의 경우

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

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

{

//현재노드의 마지막 자식을 가져온다

cur = get\_last\_child(cur);

//새로 만든 트리의 자식이 존재하면

//연산자 말고 변수들만 cur 다음에 연결시켜준다

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

new = new->child\_head;

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

new->parent = NULL;

new->prev = cur;

cur->next = new;

}

}

//+나 \*도 순서가 바뀌어도 상관없음

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

{

i = 0;

while (1)

{

//공백이면 break;

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

break;

//토큰이 연산자고 닫는괄호면 braek;

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

break;

i++;

}

//토큰의 우선순위 새로만든트리의 우선순위를 비교해서 토큰이 더 위면

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

{

//마지막 자식 다음에 연결시켜준다

cur = get\_last\_child(cur);

cur->next = new;

new->prev = cur;

cur = new;

}

//새로만든 트리가 더 높거나 같으면

else

{

//마지막 자식을 가리키고

cur = get\_last\_child(cur);

//새로만든 트리의 자식이 존재하면

//cur next에다가 new를 연결시켜쥼

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

new = new->child\_head;

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

new->parent = NULL;

new->prev = cur;

cur->next = new;

}

}

}

//순서가 중요한 연산자의 경우

else {

//cur노드에서 가장마지막자식다음에 연결시켜줌

cur = get\_last\_child(cur);

cur->next = new;

new->prev = cur;

cur = new;

}

}

//이름이 다를 경우

else

{

//가장 마지막 자식 다음에 연결시켜준다

cur = get\_last\_child(cur);

cur->next = new;

new->prev = cur;

cur = new;

}

}

}

//연산자일 경우

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

{

//순서가 바뀌어도 상관없는 연산자의 경우

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

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

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

{

//cur이 가리키고 있는게 위에 조건식에 나온 연산자와 똑같다면

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

operator = cur;

//다를 경우

else

{

//새로운 노드를 만들고

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

//새로운 노드를 기준으로 현재부터시작해서 최상위 우선순위 노드를 찾아서 operator에 저장

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

//만약 operator가 트리중 루트 노드일경우

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

//우선순위를 비교해서 operator가 더 높으면

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

//operator를 밑으로 new노드를 추가함

cur = insert\_node(operator, new);

}

//new노드가 더 높을경우

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

{

//operator의 자식이 존재하면 가장 마지막 자식을 밑으로하여 new를 추가시킴

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

operator = get\_last\_child(operator);

cur = insert\_node(operator, new);

}

}

//같을 경우

else

{

operator = cur;

while (1)

{

//operator랑 토큰이 같은 연산자일 경우 break

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

break;

//operator가 가리키는 것중 이전이 존재하면 이전으로 계속 이동시킴

if (operator->prev != NULL)

operator = operator->prev;

else

break;

}

//이전으로 계속 이동시킨 다음 토큰과 비교해서 다르면 operator의 부모를 operator에 저장

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

operator = operator->parent;

//operator가 뭔가를 가리키고 있는 상태라면 토큰과 다시비교해줌

if (operator != NULL) {

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

cur = operator;

}

}

}

//그 외의 노드일 경우

else

//operator를 밑으로 new노드를 추가함

cur = insert\_node(operator, new);

}

}

//그 외의 연산자의 경우

else

{

//새로운 노드를 생성하고

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

//cur이 null이면 cur = new

if (cur == NULL)

cur = new;

else

{

//new노드와 비교하여 가장높은 우선순위를 가지는 노드를 리턴받음

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

//operator 노드가 괄호 갯수가 더 많으면 new밑에 operator노드를 연결시켜줌

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

cur = insert\_node(operator, new);

//operator가 최상단 노드일경우

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

//operator와 new노드의 우선순위를 비교해서 새로운 노드가 우선순위가 더 높은경우

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

{

//operator의 자식노드가 존재하면 가장 마지막 자식에 new를 연결

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

operator = get\_last\_child(operator);

cur = insert\_node(operator, new);

}

}

//new가 더 낮거나 같을경우 new밑에 operator를 연결시켜줌

else

cur = insert\_node(operator, new);

}

//나머지의 경우 그냥 new밑에 operator를 연결

else

cur = insert\_node(operator, new);

}

}

}

//변수명 같은 문자열의 경우

else

{

//새로운 노드를 생성하고

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

if (cur == NULL)

cur = new;

//자식이 존재하지않으면 자식으로서 연결

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

cur->child\_head = new;

new->parent = cur;

cur = new;

}

//자식 노드가 존재하면

else {

//가장 마지막 자식을 가져와서

cur = get\_last\_child(cur);

//마지막자식에 next로 연결

cur->next = new;

new->prev = cur;

cur = new;

}

}

//인덱스 증가

\*idx += 1;

}

//그 다음 루트를 리턴시켜줌

return get\_root(cur);

}

//자식을 교환해주는 함수

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

{

node\* tmp;

tmp = parent->child\_head;

//단순히 부모노드를 인자로 건네받으면

//자식의 순서를 변경해주는 코드

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

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

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

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

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

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

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

return parent;

}

//노드를 생성하는 함수(이름과 괄호의 갯수)

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

{

node\* new;

//노드를 새로이 동적할당하고

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

//이름을 저장할 포인터에도 동적할당 시켜줌

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

//새로 할당한 배열에 이름을 복사해서 넣어줌

strcpy(new->name, name);

//괄호갯수 넣어주고 나머지는 null로 초기화

new->parentheses = parentheses;

new->parent = NULL;

new->child\_head = NULL;

new->prev = NULL;

new->next = NULL;

return new;

}

//operators배열에서 우선순위를 가져옴

int get\_precedence(char\* op)

{

int i;

//같은 연산자가 operators의 존재하면 우선순위를 리턴

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

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

return operators[i].precedence;

}

//없으면 0

return false;

}

//연산자인지 확인

int is\_operator(char\* op)

{

int i;

//operators 배열에서 연산자인지 확인함

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

{

//연산자배열에서 null이면 break;

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

break;

//연산자라면 return true;

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

return true;

}

}

//없다면 return false

return false;

}

//끝에서부터 역으로 문자 출력

void print(node\* cur)

{

//cur노드의 자식이 있으면 자식 노드로 내려감

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

print(cur->child\_head);

printf("\n");

}

//cur 다음에 next가 있으면 next로 이동

if (cur->next != NULL) {

print(cur->next);

printf("\t");

}

//그 노드의 저장된 문자 출력

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

}

//연산자를 가져온다

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

{

//현재노드가 null이면 그냥 cur을 리턴

if (cur == NULL)

return cur;

//현재 노드에서 이전이 존재하면 이전이 없을때까지 이전으로 이동시킴

if (cur->prev != NULL)

while (cur->prev != NULL)

cur = cur->prev;

//이전으로 다 이동시킨 다음 현재노드의 부모를 리턴

return cur->parent;

}

//현재노드에서 root를 리턴시켜주는 함수

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

{

if (cur == NULL)

return cur;

//이전이 존재하면 이전으로 계속이동

while (cur->prev != NULL)

cur = cur->prev;

//부모노드가 존재하면 부모노드로 이동후 다시 get\_root실행

if (cur->parent != NULL)

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

//그 다음 cur이 가리키는 노드 리턴

return cur;

}

//우선순위가 높은 노드를 찾아서 리턴 받음

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

{

//현재 노드가 연산자라면

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

//연산자의 우선순위를 비교함 현재노드의 우선순위가 더 높으면 cur을 리턴

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

return cur;

//연산자가 아니라면

//이전이 존재할경우 이전으로 이동시키고

if (cur->prev != NULL) {

while (cur->prev != NULL) {

cur = cur->prev;

//한 칸 이전으로 이동하고 거기서 다시 함수 호출

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

}

//부모가 존재하면 부모로 이동 후 다시 함수 호출

if (cur->parent != NULL)

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

}

//부모노드가 없을경우 cur return

if (cur->parent == NULL)

return cur;

}

//가장 높은 우선순위 노드를 찾음

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

{

//현재 노드와 새로운 노드 중 우선순위가 높은 노드를 가져옴

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

node\* saved\_operator = operator;

while (1)

{

//부모 노드가 없으면 saved\_operator가 최상단 노드라는 소리

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

break;

//이전이 존재하면 이전으로 이동해서 우선순위 비교후 높은걸 리턴받음

if (saved\_operator->prev != NULL)

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

//부모가 존해하면 부모로 이동해서 우선순위 비교후 높은걸 리턴받음

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

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

saved\_operator = operator;

}

//가장높은 우선순위의 노드를 리턴

return saved\_operator;

}

//new노드 밑에 old로 넣어주고 이전노드도 new를 가리키게 함

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

{

//old노드의 이전이 존재하면

//새로운노드의 이전이 old의 이전노드가되고

//그 이전노드가 가리키는 다음은 new가 됨

//그리고 old의 이전은 null

if (old->prev != NULL) {

new->prev = old->prev;

old->prev->next = new;

old->prev = NULL;

}

//새로운 노드의 자식은 기존 노드가 됨

new->child\_head = old;

//기존노드의 부모는 새로운 노드

old->parent = new;

return new;

}

//현재노드에서 가장 마지막 자식을 리턴시켜줌

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

{

//자식이 존재하면 일단 자식노드로 이동

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

cur = cur->child\_head;

//그 다음 next가 없을때까지 계속이동

while (cur->next != NULL)

cur = cur->next;

//그럼 마지막 자식을 리턴

return cur;

}

//형제들의 갯수를 셈

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

{

int i = 0;

//일단 이전이 없을때까지 이전으로 계속 이동시킨뒤

while (cur->prev != NULL)

cur = cur->prev;

//next로 이동하면서 갯수를 세준다

while (cur->next != NULL) {

cur = cur->next;

i++;

}

//갯수를 리턴

return i;

}

//노드들을 해방?해제?시켜준다

void free\_node(node\* cur)

{

//자식이 존재하면 자식으로 내려가서 함수 다시 진행

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

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

//다음 노드가 존재하면 다음 노드로 이동해서 다음 노드의 next나 자식들을 free시켜줌

if (cur->next != NULL)

free\_node(cur->next);

//현재 노드가 존재하면 값들을 다 null로 바꿔줌

if (cur != NULL) {

cur->prev = NULL;

cur->next = NULL;

cur->parent = NULL;

cur->child\_head = NULL;

//메모리 해제

free(cur);

}

}

//문자인지 확인 0~9, a~z, A~Z

int is\_character(char c)

{

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

}

//gcc인지 데이터타입인지 미리 확인

int is\_typeStatement(char\* str)

{

char\* start;

char str2[BUFLEN] = { 0 };

char tmp[BUFLEN] = { 0 };

char tmp2[BUFLEN] = { 0 };

int i;

start = str;

//str2에다가 인자로 들어온 문자열 복사

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

//str2의 사이사이 공백들 삭제

remove\_space(str2);

//앞에있는 공백들 무시

while (start[0] == ' ')

start += 1;

//문자열에 gcc가 있을경우

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

{

//tmp2에 복사후

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

//복사가 잘되었으면 return 2, 안되었으면 0

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

return 0;

else

return 2;

}

//데이터타입을 확인한다

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

{

//데이터타입이 걸리는 것이 있다면

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

{

//tmp와 tmp2에 데이터타입의 길이 만큼 복사해주고

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

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

//복사가 정상적으로되었으면 return2 안되었으면 return 0

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

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

return 0;

else

return 2;

}

}

//아무것도 안걸리면 1

return 1;

}

//타입지정자 찾기

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

{

int i, j;

//토큰 갯수 만큼

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

{

//각각 토큰마다 데이터타입갯수 만큼 확인

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

{

//i는0보다 크고 토큰이랑 데이터타입이 맞아떨어지는것이 있다면

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

{

//밑에 조건일경우 토큰위치를 리턴

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;

}

}

}

//아무것도 맞는게 없으면 -1 리턴

return -1;

}

//타입지정자 찾기2

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

{

int i, j;

//토큰 갯수만큼확인

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

{

//데이터타입크기만큼확인

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

{

//struct관련 지정자 찾기

//현재 토큰이 struct이고 다음 토큰이 변수명이면 return i

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

return i;

}

}

return -1;

}

//다 '\*'문자인지 확인

int all\_star(char\* str)

{

int i;

//문자열의 길이 연산

int length = strlen(str);

//길이가 0이면 return 0

if (length == 0)

return 0;

//문자열줄 '\*'이 아닌것이 있다면 return 0;

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

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

return 0;

//다 '\*'이면 return 1

return 1;

}

//다 문자인지 확인

int all\_character(char\* str)

{

int i;

//문자열 길이만큼 str[i]가 문자인지 확인

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

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

return 1;

return 0;

}

//struct와 unsigned 등등 처리화 나머지 토큰들 초기화 시킴

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

{

int i;

int j = start - 1;

int lcount = 0, rcount = 0;

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

//시작위치가 -1이 아닐 경우

if (start > -1) {

//struct일 경우

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

//struct 다음에 공백을 추가 시킨뒤

strcat(tokens[start], " ");

//다음 토큰은 변수명일테니 다음토큰을 현재 토큰에 연결 시켜줌

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

//그 이후 모든 토큰들은 모두 초기화 시킴

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

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

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

}

}

//unsigned일 경우 && 다음 토큰이 닫는 괄호가 아닐 경우

else if (!strcmp(tokens[start], "unsigned") && strcmp(tokens[start + 1], ")") != 0) {

//unsigned 다음에 공백 추가해주고

strcat(tokens[start], " ");

//자료형 뒤에 추가

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

//변수명 뒤에 추가

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

//나머지 토큰 모드 초기화 시킴

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

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

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

}

}

//start를 1증가 시킨 상태에서 닫는괄호랑 토큰이 같은 동안 계속 증가시키면서 닫는 괄호 갯수를 세줌

j = start + 1;

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

rcount++;

if (j == TOKEN\_CNT)

break;

j++;

}

//start를 하나 줄인 상태에서 뒤로 이동하면서 여는 괄호의 갯수를 세줌

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;

//괄호 갯수가 다를 경우 false

if (lcount != rcount)

return false;

//마지막 여는 괄호 다름 이전이 sizeof일 경우 true 리턴 시켜줌

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

return true;

}

//현재 토큰이 unsigned나 struct이고 다음 토큰이 닫는 괄호가 아니라면

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

//마지막 여는 괄호가 존재하는 토큰에다가 현재 토큰, 다음 토큰 뒤에다가 추가시켜줌

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

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

//마지막 닫는 괄호도 추가시켜줌

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

//이후 나머지 토큰들 초기화 시켜줌

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 {

//다다음 토큰의 시작문자가 (일 경우

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

//start를 2칸이동한 지점 부터

j = start + 2;

//닫는 괄호와 여는 괄호를 각각 세준뒤

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

sub\_lcount++;

j++;

}

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

j = j + 1;

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

sub\_rcount++;

j++;

}

}

else

return false;

//닫는 괄호과 여는괄호 갯수가 서로 다르면 return false

if (sub\_lcount != sub\_rcount)

return false;

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

//나머지 토큰들 전부 초기화 시켜줌

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

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

}

//마지막 여는 괄호가 존재하는 토큰에다가 현재 토큰, 다음 토큰 뒤에다가 추가시켜줌

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

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

//마지막 닫는 괄호도 추가시켜줌

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

//이후 나머지 토큰들 초기화 시켜줌

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;

}

//토큰 배열 초기화

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

{

int i;

//토큰배열 각각 초기화

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

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

}

//오른쪽 공백을 없애줌

char\* rtrim(char\* \_str)

{

char tmp[BUFLEN];

char\* end;

//tmp에다가 원래 문자열 복사

strcpy(tmp, \_str);

//끝을 tmp시작에다가 길이만큼 더하고 - 1

//맨끝을 가리킴

end = tmp + strlen(tmp) - 1;

//시작지점이 아니고 가리키는 문자가 공백인동안 end--

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

--end;

//맨끝에다가 널문자를 넣어줌

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

//공백을 제거한 문자열을 저장

\_str = tmp;

return \_str;

}

//왼쪽 공백들을 없애줌

char\* ltrim(char\* \_str)

{

char\* start = \_str;

//널 문자가 아니고 공백인 동안

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

//하나씩 증가

++start;

//처음으로 공백이 아닌게 나오면 그 문자를 가리키게 설정

\_str = start;

return \_str;

}

//추가적인 공간들 삭제 시키는 함수

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

{

int i;

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

char\* start, \* end;

char temp[BUFLEN] = "";

int position;

//인자에서 #include<가 존재하면

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

start = str;

//end를 <를 가리키게 만듬

end = strpbrk(str, "<");

//position은

position = end - start;

//tmp에다가 str에 position 수 만큼 뒤에 붙혀주고

strncat(temp, str, position);

//공백 추가

strncat(temp, " ", 1);

//그 다음 문자들 추가

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

str = temp;

}

//str의 길이만큼 진행한다

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

{

//문자가 공백일 경우

if (str[i] == ' ')

{

//시작지점이 공백이라면

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

//공백이 아닐때까지 i증가

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

i++;

else {

//이전 문자가 공백이 아니면 str2의 뒤에다가 문자 추가

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

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

//공백은 무시하고 진행

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

i++;

}

}

//아니라면 str2의 맨뒤에다가 문자 추가

else

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

}

return str2;

}

//입력받은 문자열에서 공백들을 없애줌

void remove\_space(char\* str)

{

char\* i = str;

char\* j = str;

//j가 가리키는게 0이 아닐때까지

//결론적으로 공백을 만나면 포인터를 하나씩 움직이면서 뒤에 문자들을 앞으로 땡겨줌

while (\*j != 0)

{

\*i = \*j++;

//공백을 만나면 i를 하나 증가시킴

if (\*i != ' ')

i++;

}

//맨 마지막에 널 문자

\*i = 0;

}

//괄호 갯수를 체크

int check\_brackets(char\* str)

{

//입력받은 문자열을 가르키는 문자열 포인터

char\* start = str;

int lcount = 0, rcount = 0;

while (1) {

//"()"중 하나라도 만날때까지 start를 이동하고 처음만나는 ()중 하나의 위치를 리턴

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

//지금 가리키는게 '('면 왼쪽 갯수 하나 증가

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

lcount++;

//')'라면 오른쪽 갯수 하나증가

else

rcount++;

//포인터 하나 이동

start += 1;

}

//없다면 break

else

break;

}

//왼쪽괄호 오른쪽괄호 갯수 가 다르면 return 0

if (lcount != rcount)

return 0;

//같으면 return 1

else

return 1;

}

//토큰의 갯수를 리턴시켜줌

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

{

int i;

//토큰이 빈거 일때 스탑함

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

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

break;

//공백을 만났을때의 i값 리턴(토큰갯수)

return i;

}

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

**<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

//문제별 배점을 저장할 구조체

struct ssu\_scoreTable{

char qname[FILELEN];

double score;

};

//채점프로그램 시작 함수

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

//옵션들 체크

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

//사용법 출력

void print\_usage();

//학생들 채점하기

void score\_students();

//학생 개인별 채점

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

//score.csv파일의 첫번째 줄 작성

void write\_first\_row(int fd);

//파일로부터 정답을 불러옴

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

//빈칸문제 채점함수

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

//프로그램문제 채점함수

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

//프로그램문제 파일 컴파일 함수

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

//프로그램문제 파일 실행 함수

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

//프로그램문제 파일 백그라운드로 실행시키는함수

pid\_t inBackground(char \*name);

//컴파일에 에러나 warning이 있는지 확인

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

//결과파일 비교

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

//-m옵션 수행

void do\_mOption(char \*path);

//-i옵션 수행

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

//-i옵션으로 받은 학번이 존재하는지 확인

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

//스레드파일인지 확인

int is\_thread(char \*qname);

//출력 재지정 및 명령어 실행

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

//파일이 txt인지 c인이 확인

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

//디렉토리삭제함수

void rmdirs(const char \*path);

//대문자를 소문자로 변경

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

//점수배점 테이블 세팅

void set\_scoreTable(char \*ansDir);

//점수배점 테이블이 존재하면 읽어드림

void read\_scoreTable(char \*path);

//점수배점 테이블을 만드는 함수

void make\_scoreTable(char \*ansDir);

//score\_table.csv파일에 데이터 작성

void write\_scoreTable(char \*filename);

//학번 저장테이블 세팅

void set\_idTable(char \*stuDir);

//score\_table.csv를 어떤 타입으로 만들건지 물어보는 함수

int get\_create\_type();

//학번 테이블 정렬

void sort\_idTable(int size);

//score\_table 정렬

void sort\_scoreTable(int size);

//문제 번호 추출함수

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

#endif

**<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

//트리를 구성할 노드 구조체

typedef struct node{

int parentheses;

char \*name;

struct node \*parent;

struct node \*child\_head;

struct node \*prev;

struct node \*next;

}node;

//연산자 우선순위 저장

typedef struct operator\_precedence{

char \*operator;

int precedence;

}operator\_precedence;

//트리 비교 함수

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

//토큰들을 이용해 트리 제작함수

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

//순서가 바뀔 수 있는 연산자에 대한 자식 교한 함수

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

//새로운 노드를 제작하는 함수

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

//인자로 받은 문자의 우선순위를 리턴시켜주는 함수

int get\_precedence(char \*op);

//연산자인지 확인하는 함수

int is\_operator(char \*op);

void print(node \*cur);

//트리에서 연산자를 찾는함수

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

//루트를 반환하는 함수

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

//우선순위가 높은 노드를 리턴

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

//가장 최상위 우선순위 노드를 리턴

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

//새로운 노드를 삽입시키는 함수

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

//가장 마지막 자식을 찾는 함수

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

//메모리해제함수

void free\_node(node \*cur);

//자식의 갯수를 리턴시키는 함수

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

//입력받은 str을 여러규칙에 따라 토큰화 시킨다

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

//gcc인지 자료형인지 확인

int is\_typeStatement(char\* str);

//타입지정자 찾기

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

//타입지정자 찾기 2

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

//문자인지 확인하는 함수

int is\_character(char c);

//\*로만 이루어져 있는지 확인하는 함수

int all\_star(char \*str);

//다 문자인지 확인

int all\_character(char \*str);

//struct와 unsigned 등등 처리화 나머지 토큰들 초기화 시킴

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

//토큰이 저장된 배열 초기화

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

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

//오른쪽의 공백을 제거시키는 함수

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

//왼쪽의 공백을 제거시키는 함수

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

//문자열 사이 공백 제거

void remove\_space(char \*str);

//괄호의 갯수가 맞는지 세주는 함수

int check\_brackets(char \*str);

//추가적인 공백들을 삭제시키는 함수

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

#endif