컴퓨터학부, 20190511, 배준형

# 과제 개요

|  |
| --- |
| ssu\_score 프로그램은 학생들의 답안을 채점하는 프로그램이다. 단순히 1대1 대응으로 한 가지 방식으로 채점하는 것이 아닌 마치 컴파일러의 lexical analysizer (lex) 가 lexeme 을 분리하고 Syntax Analyzer가 토큰 기준으로 Parse Tree 를 만드는 것처럼 구성되어있다는 것이 이번 프로그램의 핵심이다. 해당 채점방식을 기준으로 여러가지 옵션을 지원하고 있다. 지원하는 옵션은 다음과 같다.  -h : 프로그램이 어떻게 실행되는지 help() 메시지를 띄워준다.  -n <입력 경로> : 해당 입력 경로로 학생 점수를 계산한 score .csv 파일을 만든다.  -m : 문제 배점 기준을 바꾼다. no, 를 입력 시 문제배점 수정을 종료한다.  -p : 실시간으로 학생이 틀린 문제를 출력한다. (최대 5명, 혹은 전체 출력)  -c : 실시간으로 학생 점수를 출력한다. (최대 5명, 혹은 전체 출력)  --> 해당 옵션은 p 옵션과 함께 사용가능하다. (c옵션 선 출력 후 p 옵션출력 가능)  -s : 학생 학번 기준, 혹은 총 점수 기준으로 오름차순, 내림차순 정렬을 하여 csv 파일에 저장하는 옵션  -t : 프로그램 문제의 대하여 멀티쓰레딩 컴파일 옵션을 지원하는 옵션  -e <입력경로> : 해당 입력경로를 기준으로 에러 디렉토리를 만들어 각 학생 디렉토리에 컴파일에러 및 워닝을 텍스트파일로 출력하는 함수이다.  채점을 지원하는 문제는, 빈칸문제와 프로그램 문제로 빈칸 채우기 문제는 답안 텍스트파일과 비교하여 :을 기준으로 다수의 정답이 존재할 수 있으며 이 역시 parse 트리를 이용하여 채점이 진행된다.  프로그램 문제는 5초 이상이 걸리면 0점처리하도록 설계되어있음. |

# 구현기능

## 실질적 구현 내용

|  |
| --- |
| <부수적인 기능>   1. 인자들이 받는 절대경로를 상대경로로 바꾸는 함수 realpathS2를 만들어 상대경로를 받을 수 있게함  * 만약 상대경로에 ../../ 를 남발하여 Root 밑으로 넘어가는 경로에 대해서는 Wrong path를 에러메시지를 출력하고 프로그램 종료  1. 연결리스트를 사용하여 상대경로를 절대경로로 바꾸었음 2. 학번 연결리스트(StdList, Snode)와 학번노드 (Snode)내부에 문제들을 관리하는 연결리스트 Sclist 존재   <p옵션>   1. [STUDENTS...] 인자로 입력받은 학생이 0명이면 전체 학생에 대하여 틀린문제 출력/ 2. 입력받은 학생이 5명이 넘어갈 경우 초과한 학생들을 출력하고 출력에 반영하지 않음. 3. 입력받은 학생이 없을 시 프로그램 종료 4. 모든 학생에 대하여 채점할 때마다 틀린문제를 StdList가 연결리스트로 관리하며, 출력할 때 iIDs 전역변수 배열에 존재하는 학생인지 확인 후 함수 **print\_wrongL** 에서 틀린문제를 양식에 맞게 출력 5. p c 옵션이 동시에 들어왔을 때 p 와 c 옵션에 개별적인 학번 리스트가 들어왔을 때 프로그램이 종료되도록 예외처리 (p,c 옵션이 동시에 왔을 때 둘 중 하나만 옵션을 받을 수 있음.)   <s옵션>   1. 기본적으로 정렬방식은 버블 정렬로 정렬이 진행이됨 (연결리스트 정렬은 버블정렬이 탁월하다판단) 2. 정렬을 관리하는 연산함수인 **sort\_manager**가 4개의 정렬함수 **sort\_a(d)secnedL/S** 를 조작하여 정렬하도록 설정 3. 마지막에 **write\_sort** 함수를 호출하여 정렬된 연결리스트를 파일에 출력함   <n옵션>   1. n 옵션 사용 시 상대경로/절대경로로 들어온 파일이 csv파일이면 해당 경로에 정답 레코드 .csv 파일이 생성되도록 구현 2. realpathS 함수를 이용하여 상대경로를 절대경로로 변환 3. 들어온 경로의 마지막 파일이 .csv 확장자를 가지지 않거나 확장자가 없는 경우 csv 에러를 출력하고 프로그램 종료 4. 프로그램 마지막에 변경된 경로를 출력   <c옵션>   1. 프로그램 마지막에 출력된 학생의 평균을 출력 2. [STUDENTS...] 인자로 입력받은 학생이 0명이면 전체 학생에 대하여 틀린문제 출력/ 3. 입력받은 학생이 5명이 넘어갈 경우 초과한 학생들을 출력하고 출력에 반영하지 않음. 4. 입력받은 학생이 없을 시 프로그램 종료   <m옵션>   1. <ANS\_DIR>/score\_table.csv 파일이 존재하지 않을 시 에러메시지 출력 후 프로그램 종료 2. 수정할 문제를 입력받지 않을 시 Wrong 메시지 출력 후 루프, no 입력 시 입력종료 3. 수정된 전역변수 score\_table을 score\_table.csv 파일에 출력하여 덮어씌움 (수정된 배점 적용)   <e옵션>   1. 프로그램 실행 중 발생하는 에러/워닝을 새로운 경로로 error.txt 출력하는 옵션구현 2. <DIRNAME> 에 입력받은 경로를 **realpathS2**로 절대경로로 변환함 3. 같은 에러디렉토리에 파일을 덮어씌우는 경우 내용을 삭제하고 다시 에러파일 생성 4. realpathS2 에서 경로가 잘못된경우 (Root 범위를 벗어나는경우) 5. 에러파일 생성 중 에러발생 시 프로그램 종료   <t옵션>   1. 입력받은 문제번호가 현재 score\_table.csv 에 존재하지 않는 경우 해당 에러 문제를 출력하고 프로그램 종료 2. t 옵션을 받은 문제는 -lpthread 옵션을 사용해서 구현   <h옵션>   1. h 옵션과 다른 옵션이 들어오는 경우 에러처리 2. <STD\_DIR> <ANS\_DIR> 없이 h 옵션이 오는 경우 허용 help 메시지 출력 3. <STD\_DIR> <ANS\_DIR> 과 -h 단독으로 오는 경우 허용 help 메시지 출력   [기타]  **blank.h 내부에서 remove\_extraspace 함수 내부에서 malloc 후 초기화를 안시켜주어 쓰레기값이 들어가 배점이 이상하게 나오는 문제를 해결하기 위해 memset(str2, ‘\0’, BUFLEN); 사용해서 초기화를 시켜줌으로써 안정성을 높임**  **감점된 점수가 추합이 0점을 넘어 음수의 점수가 나온다면 0점처리** |

score 프로토타입 (실질적인 ssu\_score 헤더)

|  |
| --- |
| void ssu\_score(int argc, char \*argv[]);  int check\_option(int argc, char \*argv[]);  void print\_usage();   // -h 옵션  /\*\*   \* score.csv 제작함수 (csv 파일 생성 함수)   \* 전체 score.csv에 전체학생 점수를 추가해서 저장하는듯함.   \* id\_table 인덱스 개수만큼 파일에 데이터입력   \*  \*/  void score\_students();  /\*\*   \* 학생 답 채점함수   \* fd := csv 파일디스크립터, id := 학번 (학생학번)   \* score\_table 배열 안의 점수만큼 채점을 한다.   \* 학생 한 명에대해서 점수총점을 구해서 csv file에 추가.   \*  \*/  double score\_student(int fd, char \*id, Snode\* std\_node);  /\*\*   \*  $(PWD)/score.csv 제목 데이터 (score\_table 을 이용하여 테이블 개수를 구하고)   \*  첫 열은 빈칸처리 (,) : 학번이 들어갈꺼니까   \*  두 번째 열 ~ n-1 번 째 열까지 score\_table 의 qname을 출력한다.   \*  n번째(마지막열) 에는 총합 sum을 만든다.   \*  \*/  void write\_first\_row(int fd);  /\*\*   \* fd 디스크립터 파일의 답(내용) 을 가져와서 result 출력.  \*/  char \*get\_answer(int fd, char \*result);  /\*\*   \* 빈 칸 문제 채점  \*/  int score\_blank(char \*id, char \*filename);  /\*\*   \* compile\_program() 호출, execute\_program() 호출을 통해 프로그램 워닝 개수, 컴파일 성공여부를 바탕으로 점수추출  \*/  double score\_program(char \*id, char \*filename);  double compile\_program(char \*id, char \*filename);           // 프로그램 컴파일  int execute\_program(char \*id, char \*filname);               // 프로그램문제 실행, 성공시 1, 실패시 0  /\*\*   \* background.txt (임시파일) 을 만들어서   \* 프로그램 문제의 실행파일을 system(ps | grep 실행파일.stdexe) 를 받아와서   \*  indirect() 호출을 통해 해당 프로그램을 실행시키고, (해당 프로그램 표준출력--> background.txt 파일로 전환)   \*  background.txt 를 처음부터 읽는다. (lseek(fd,0,SEEK\_SET)   \*  background.txt 파일이 비어져 있는 경우 0 return   \*  프로세스가 생성되어 파일이 채워져 있는 경우 해당 파일에는 processID가 있을 것이고 해당 pid return   \*  \*/  pid\_t inBackground(char \*name);  /\*\*   \*  컴파일 결과를 출력해놓은 filename을 읽어서   \*      컴파일오류시 0 리턴, 워닝 당 0.1 점씩 감점   \*   \* filename은 컴파일 결과를 넣어둔 파일이고   \* 해당 파일에서 Error 존재 시 Error(0) return   \* 해당 파일에서 Warning 존재 시 warning 만큼 0.1 점씩 점수를깎아서 return   \*  \*/  double check\_error\_warning(char \*filename);  /\*\*   \* 실행결과가 같은지 비교  \*/  int compare\_resultfile(char \*file1, char \*file2);  void do\_iOption(char (\*ids)[FILELEN]);  void do\_mOption();  /\*\*   \*  학번 (target) 이 src에 존재하면 True, 없으면 False  \*/  int is\_exist(char (\*src)[FILELEN], char \*target);  /\*\*   \* 스레드 문제개수  \*/  int is\_thread(char \*qname);  /\*\*   \* new를 old 디스크립터로 출력(system) 하고 다시 복귀, old:표준입출력   \*  즉 command 를 newfd --> oldfd 로 파일시스템을 이용  \*/  void redirection(char \*command, int newfd, int oldfd);  /\*\*   \* 파일 타입 가져오기   \* .txt파일이면 : TEXTFILE(3)   \* .c 파일이면  : CFILE(4)   \* 실패하면     : -1  \*/  int get\_file\_type(char \*filename);                           /// 파일 타입 가져오기  /\*\* path 경로 내부 디렉토리 all clear\*/  void rmdirs(const char \*path);  /\*\* 대문자-->소문자\*/  void to\_lower\_case(char \*c);  void set\_scoreTable(char \*ansDir);  /\*\*   \* ssu\_scoreTable 구조체 : {문제, 점수} 를 정의한 타입   \* csvfile이 존재하면   \* 고급 테크닉   \*      %[,] : 쉼표포함해서 쉼표앞까지 읽어라   \*      %[^,] : 쉽표 앞까지 읽어라.   \*      쉼표까지읽어서, 문제번호(qname), 점수 (score)를 받아와서 문제테이블 score\_table[] 에 저장  \*/  void read\_scoreTable(char \*path);  /\*\*   \* score\_table.csv 파일 생성   \*      ansDir 내부 파일들을 모두 순회하면서 score\_table 구조체 배열에 문제이름 할당.  \*/  void make\_scoreTable(char \*ansDir);  /\*\*   \*   \* score\_table 구조체로부터 filename으로 점수표 생성가능.  \*/  void write\_scoreTable(char \*filename);  /\*\*   \* 학번 테이블 생성   \* STUDIR 안의 학번을 모두 순회해서 학번디렉토리가 존재할 시   \* id\_table에 학번 저장.  \*/  void set\_idTable(char \*stuDir);  /\*\*   \* 문제 점수 어떻게 넣을건지 선택받게 하기 1,2  \*/  int get\_create\_type();  /\*\*   \*   학번 테이블을 오름차순으로 정렬, (오름차순으로 정렬했으니까, 반대로 내림치차순이면 이걸 건들이면된다.)   \*   정렬알고리즘은 단순 NN 정렬   \*/  void sort\_idTable(int size);  /\*\*   \* 1-2 > 1-3 > 2-1 순으로 문제번호 정렬식으로 score\_table를 정렬함.  \*/  void sort\_scoreTable(int size);  /\*\*   \*   \* qname 문자열을 받아서 문제가 1-3.txt 라면 num1(1), num2(3) 이 들어감,   \*       만약, 1.txt 라면 num1(1), num2(0) 이다.   \* strtok(dup, "-.") 를 넣어서 dup을 - 와 . 단위로 토큰을 끊어냄   \* 해당 방식으로 1-3.txt 를 1, 3, txt 로 분리시킬 수 있음.  \*/  void get\_qname\_number(char \*qname, int \*num1, int \*num2);  [추가 구현함수]  //sclist 초기화함수  Sclist\* new\_sclists (char\* qname, double curS);  //문제 배점 붙은 Sclist 초기화함수  Sclist\* new\_sclistss (char\* qname, double curS, double s);  //id 로 Snode 초기화함수  Snode\* new\_stdnode (char\* id);  //stdlist 초기화함수  StdList\* new\_stdlist();  // list 에 해당 id로 Snode 생성, return으로 append한 구조체 반환  Snode\* append\_list (StdList\* list, char\* id);  void print\_list (StdList\* list);  // Sclist 문제 요소 출력 (맞은문제, 문제배점)  void print\_score (Snode\* node);  // Snode 에 문제 구조체 Sclist append.  int append\_score(Snode\* node, char\* qname, double cur\_score, double score);  // qname 과 일치하는 score 원래의 배점을 score\_table 로부터 찾기  double find\_score (char\* qname);  // 버블 정렬에 필요한 a <-> b 링크드리스트 구조 변경  int swap\_list (StdList\* list, Snode\* p, Snode\* a, Snode\* b);  void sort\_descentS(StdList\* list); //list 의 sums(총함) 내림차순으로 정렬(버블sort)  void sort\_aescendS(StdList\* list); //list 의 sums(총함) 오름차순으로 정렬(버블sort)  void sort\_descentI(StdList\* list); //list 의 id (학번) 내림차순으로 정렬(버블sort)  void sort\_aescendI(StdList\* list); //list 의 id (학번) 오름차순으로 정렬(버블sort)  void sort\_manager(StdList\* list, int type, int updown); //sorting 관리 구조체  void print\_wrongL (Snode\* node); // node 안의 틀린문제들 요소 출력  int write\_sort (StdList\* list, int fd);  // 정렬된 리스트를 출력  char\*\* path\_arr(char\* str); // str을 / 기준으로 토큰배열화  int realpathS2 (char \*str); // 링크드리스트기반 경로찾기 함수 |

## blank.h 프로토타입 (빈칸 문제를 매기기 위한 헤더)

|  |
| --- |
| [**프로토타입 및 함수 구현기능]**  typedef struct node{      int parentheses;      char \*name;      struct node \*parent;      struct node \*child\_head;      struct node \*prev;      struct node \*next;  }node;    typedef struct operator\_precedence{      char \*operator;      int precedence;  }operator\_precedence;    // root1, root2 parse tree 비교 함수  void compare\_tree(node \*root1,  node \*root2, int \*result);  // 해당 token 을 바탕으로 파스 트리 생성 (parentheses 괄호개수)  node \*make\_tree(node \*root, char (\*tokens)[MINLEN], int \*idx, int parentheses);  //자식노드의 형제노드들을 swaping하는 함수 (next <-> prev 변경)  node \*change\_sibling(node \*parent);  // name: 학생 답안 정보, parenthesese : 괄호 개수 --> node 생성자  node \*create\_node(char \*name, int parentheses);  // 연산자의 우선순위를 알려주는 함수  int get\_precedence(char \*op);  // 연산자가 위의 정의된 연산자 중에 존재하는지 체크 있으면 true, 없으면 false  int is\_operator(char \*op);  // 트리 출력함수  void print(node \*cur);  // cur 노드의 형제노드가 있으면 형 노드 반환, 없으면 parent 반환;  node \*get\_operator(node \*cur);  // 트리의 루트 노드 반환  node \*get\_root(node \*cur);  // cur의 형/부모 노드들 중 우선순위가 new보다 하나라도 높은 연산자 노드 return  node \*get\_high\_precedence\_node(node \*cur, node \*new);  // cur 노드 중 가장 부모노드쪽에 있으면서 new 노드모다 연산자 우선순위가 높은 노드 return  node \*get\_most\_high\_precedence\_node(node \*cur, node \*new);  // new를 old 의 부모노드로 연결  node \*insert\_node(node \*old, node \*new);  // 노드의 막내의 가장 오른편 노드를 return  node \*get\_last\_child(node \*cur);  // cur 노드 기준으로 내려가면서 모두 free 하는 함수  void free\_node(node \*cur);  // 자기 자신을 미포함한 형제노드 개수 반환.  int get\_sibling\_cnt(node \*cur);  //str 을 토큰화 (pl시간에 배운 lex 함수와 유사)  int make\_tokens(char \*str, char tokens[TOKEN\_CNT][MINLEN]);  // 타입 종류에 대해 알려주는 함수. (0:타입이 존재하는 문자열, 1: 아무것도 아닌 것(괄호가 캐스팅용), 2:해당타입으로 시작하는 문장)  int is\_typeStatement(char \*str);  // 타입을 순회하며 (타입) 이후 연산자가 오거나 문자가 온다면 (?캐스팅이라면) 해당 토큰 번호 반환 (없으면 -1)  int find\_typeSpecifier(char tokens[TOKEN\_CNT][MINLEN]);  // struct 토큰을 사용하면서 다음토큰의 마지막값이 문자나정수라면 --> 구조체 토큰이라면 return i; (없으면 -1)  int find\_typeSpecifier2(char tokens[TOKEN\_CNT][MINLEN]);  // 영문자나, 정수이면 true return  int is\_character(char c);  // 현재 str이 모두 \* 일 경우 true 아니면 false  int all\_star(char \*str);  // str 에서 영문자나 숫자가 하나라도 존재하면 rturn 1;  int all\_character(char \*str);  // token에 캐스팅이나, struct 가 있는 경우 문법확인 (괄호 짝 갯수) 및 토큰 정렬.  int reset\_tokens(int start, char tokens[TOKEN\_CNT][MINLEN]);  // 토큰을 모두 비우는 (초기화) 하는 함수  void clear\_tokens(char tokens[TOKEN\_CNT][MINLEN]);  // 총 토큰 개수 return  int get\_token\_cnt(char tokens[TOKEN\_CNT][MINLEN]);  // \_str 오른쪽 공백 제거  char \*rtrim(char \*\_str);  // \_str 왼쪽 공백 제거  char \*ltrim(char \*\_str);  // str 공백 제거  void remove\_space(char \*str);  // ( ) 개수 짝이 맞으면 1, 틀리면 0  int check\_brackets(char \*str);  // 잔여 공백 제거  char\* remove\_extraspace(char \*str);    // 자료형을 정리해둔 배열 datatype  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"};  // 24 개의 연산자 우선순위 (번호가 낮을수록 우선순위가 높음)  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}  }; |

# 상세설계

## score 상세설계

|  |
| --- |
| **[추가로 구현한 내용들]**   1. **링크드 리스트기반으로 상대경로를 절대경로로 바꾸는 함수 realpathS2, path\_arr 함수**  * 설계과제1번에서 상대경로를 절대경로로 바꾸는 알고리즘에서 착안하여 제작하였음. 우선 pathNode * char\*\* path\_arr(char\* str) 를 이용하여 str 경로에서 / 를 기준으로 문자열을 분리하여 배열로 만들어 return 해주는 함수이다. 예를들어, str = “/home/junhyeong/go2/ANS/score.csv” 라고 하면 ‘/’ 를 기준으로 문자열을 분리하여 Arr = {“home”, “junhyeong”, “go2”, “ANS”, “score.csv”} 를 만들어 return해주는 함수이다. * 구조체(pathList)를 이용하여 path, prev, next (자기참조구조체) 리스트 프레임을 만든다. * realpathS2 함수에서 기본적으로str 에서 ~/ 로 시작하거나 / 로 시작하지 않는 경우 getenv(“HOME”) 를 이용하거나 getcwd() 를 이용하여 절대경로 주소지의 **actual path(현재 자기경로 이후의경로**) 부분과 합쳐주는 1차적인 작업을 한다. * 그리고 **path\_arr(str)** 함수를 불러 / 단위로 lexeme 을 구분한다. * 이후 구조체 pathList 를 이용하여 이중 연결리스트를 만드는데 아래 4가지 법칙을 따른다. * 1. 처음에 더미 노드를 만든다. * 2. 만약 . 이 온다면 연결리스트를 연결하지 않고 넘어간다. * 3. 만약 .. 이 온다면 연결리스트에서 pop 을 하드시 prev 부분으로 넘어가서 next 를 NULL 을 한다. * 4. 2,3번 경우가 아니라면 그냥 연결한다. * 이렇게 연결된 연결리스트를 바탕으로 ‘/’ 토큰을 번갈아가며 넣어주면서 절대경로를 완성한다. * 해당 경로를 인자에 strcpy() 를 통해 복사해준다.  1. **p옵션과 s옵션에서 사용될 연결리스트 함수와 구조체**   **[구조체]**    typedef struct sclist {      char qname[FILELEN];        // 문제번호      double cur\_score;              // 현 점수      double score;                  // 원래배점      struct sclist\* file\_next;        // 다음에 연결된 문제  }Sclist;    typedef struct stdnode {      char id\_name[10];           //학번      double sums;                //총합      int file\_cnt;              //연결된 문제개수      struct stdnode\* next;              // 다음 학번      Sclist\* list\_head;        // head -> .. -> .. -> tail ->NULL 형태로 연결      Sclist\* list\_tail;  }Snode;    typedef struct slist {      int id\_cnt;      Snode\* head;            //header -> ... -> ... -> tail      Snode\* tail;  }StdList;   * **stdList, Snode, Sclist 구조체 가 존재하며** * **stdList : Snode 리스트를 연결리스트 형태로 head, tail 을 보관하여 관리해주는 리스트** * **Snode : 학번, 총합(Sums), 연결된 문제의 개수 (file\_cnt), 그리고 다음에 연결된 Snode를 가리키는 포인터(next), 해당 학번에 연결된 문제 연결리스트(Sclist 연결리스트) (list\_head, list\_tail) 이 존재한다.** * **Sclist : Snode가 가지고 있는 학생 별 틀린문제 (p옵션), 전체문제(s 옵션) 을 관리하는 연결리스트로 문제번호(qname), 해당 문제의 학생이 받은 점수 (cur\_score), 해당 문제 원 배점 (score), 그리고 해당 문제Sclist의 다음 Sclist를 가리킬 포인터 (file\_next) 를 가지고 있다.** * 요약하면, 학번의 연결리스트인 Snode 안에 Sclist 연결리스트가 존재하며 Snode가 개별적으로 Sclist 리스트를 가지고 있다. 그리고 이 Snode를 총괄적으로 관리하는 리스트가 StdList이다.   **[함수]**   * **StdList 를 사용하기 위한 추가적인 연산들을 지원하는 메소드 역할을 하는 함수들이다.**   (초기화함수)  Sclist\* **new\_sclists (char\* qname, double curS);**  Sclist\* **new\_sclistss (char\* qname, double curS, double s);**  >> 위는 qname에 해당하는 문제의 배점을 score\_table 전역변수를 이용하여 자동으로 찾아서 Sclist 노드를 생성해주는 생성자이고,  >> 아래는 qname에 해당하는 문제의 배점을 s 파라미터를 이용하여 직접 Sclist의 모든 요소를 초기화하는 생성자이다.  Snode\* **new\_stdnode (char\* id);**  >> Sclist 노드의 학번만 id로 초기화해주고 나머지는 다 0혹은 NULL 시켜주는 생성자이다.  StdList\* **new\_stdlist();**  >> Snode를 연결하는 리스트 Stdlist의 생성자이다.  (지원하는 연산)  **연산1. 출력**  **void print\_list** (StdList\* list);  >> StdList 를 출력하는 함수  **void print\_score** (Snode\* node);  >> Snode 를 출력하는 함수  void **print\_wrongL** (Snode\* node);  >> Snode의 연결된 Sclist를 p옵션 출력 양식에 맞게 **틀린문제번호(해당 원배점)** 을 출력하는 함수.  int **write\_sort** (StdList\* list, int fd);  >> 정렬된 리스트를 file descriptor **(fd)** 에 출력하여 score.csv 를 제작하는 함수이다.  >> **내부적으로 write\_first\_row(fd);** 를 호출하여 출력할 csv 파일의 프레임을 만든다.  >> 그리고 Snode를 순회하며 Snode가 관리하는 Sclist 연결리스트의 요소(해당 학번의 채점결과) 를 csv 파일에 (, 를 붙여주면서) 차례대로 출력한다.  **연산2. 연결**  Snode\* **append\_list** **(StdList\* list, char\* id);**  >> StdList 리스트에 해당 학번 id 으로 new\_stdnode 생성자를 호출하여 StdList에 연결한 후 **해당 노드를 return 한다. (후에 p,s 옵션 사용 시 return 받은 노드의 요소들을 새롭게 추가할 수 있도록 하기위함: 선연결 후조작)**  int **append\_score (Snode\* node, char\* qname, double cur\_score, double score);**  >> Snode node 에 문제번호(qname), 해당 학생의 문제 채점 점수 (cur\_score), 문제 원점수 (score) 를 초기화해주는 함수이다. **일반적으로 append\_list 해서 출력받은 Snode 를 해당 append\_score** 을 이용해서 **초기화한다**.  **연산3. 정렬**  int swap\_list (StdList\* list, Snode\* p, Snode\* a, Snode\* b);  **>> (버블)정렬에 사용될 핵심 함수로써, a,b 노드의 위치를 연결리스트의 위치를 바꾸는 함수이다.**  >> 정렬구조는 항상 p->a->b->next 형태로 되어있다.  >> a가 가장 앞 노드인 경우 StdList->head 를 b로 설정하고 a를 b의 위치로 두는 정렬을 한다.  >> 그 외의 경우 a->next = b->next; p->next = b; b->next = a; 를 하여 연결리스트를 swap을 해준다.  void **sort\_descentS(StdList\* list);**  >> StdList를 **총점** (sums) 를 기준으로 **내림**차순으로 **버블정렬**하는 함수이다.  void **sort\_aescendS(StdList\* list);**  >> StdList를 **총점** (sums) 를 기준으로 **오름**차순으로 **버블정렬**하는 함수이다.  void **sort\_descentI(StdList\* list);**  >> StdList를 **학번** (id) 를 기준으로 **내림**차순으로 **버블정렬**하는 함수이다.  void **sort\_aescendI(StdList\* list);**  >> StdList를 **학번** (id) 를 기준으로 **오름**차순으로 **버블정렬**하는 함수이다.  **void sort\_manager(StdList\* list, int type, int updown);**  >> StdList 를 type ( 0:학번, 1:총점수(sums)) 과 updown (-1:내림차순,1:오름차순) 으로 정렬을 관리하는 함수이다.  >> 해당 함수 **내부**에서 **aescendl, descendl, aescendS, descendS** 를 type에 맞게 호출한다.  **n 옵션**   * **기본적으로 do\_nOption 함수를 만들어 전역변수 문자열csvDir 를 수정해주는 함수이다.** * **즉, Coupling (함수 side effect) 가 증가할 수 있으니 주의를 해야한다.** * n옵션을 수행하기 위해 **realpathS2** (상대경로를 절대경로 바꿔주는함수) 와 **csv**\_**check** (파일 확장자가 csv파일인지 확인하는 함수), **mkdirs** (경로의 앞부분부터 확인해서 마지막 경로 전까지(부모디렉토리) 까지 재귀적으로 디렉토리 확인 및 생성 (디렉토리가 아니면 삭제 후 디렉토리로 생성)) 을 이용한다. * 그리고 해당 생설될 name이 절대경로로 바꾸는데 Root 범위를 벗어나는 등의 잘못된 경로라면 에러 메시지출력 후 프로그램을 종료한다. * **이후 새로운 경로를 csvDir 전역변수에 strcpy 해준다.**   **c 옵션**   * score\_score(기본 실행 함수) 에서 c 옵션으로 받은 학생이 없는 학생이라면 에러처리를 한다. * 마지막에 인자에 따라 학번 인자가 하나도 없을 시 전 학번에 대해서 score을 실시간으로 알려준다. * 프로그램 종료 전 출력된 학생들의 평균을 알려준다. * 초과하는 학생이 5명이 넘는 경우 check\_option 함수(옵션 분류함수)에서 getopt함수로 처리할 때 Exceed 를 출력하며 해당 학생들은 채점에 반영하지 않았다.   **p 옵션**   * c 옵션과 마찬가지로 **check\_option** 함수(옵션 분류함수)에서 **getopt**함수로 처리할 때 Exceed 를 출력하며 해당 학생들은 채점에 반영하지 않았다. * **score\_score**(기본 실행 함수) 에서 c 옵션으로 받은 학생이 없는 학생이라면 에러처리를 한다. * **score\_students** 함수에서 StdList를 생성한다. 그리고 new\_Snode 생성자를 호출하여 연결한 다음, **score\_student**함수에 해당 Snode를 넘겨준다. * **score\_student** 함수에서 받은 Snode 에 대하여 틀린 문제가 있을 경우 **append\_score** 함수를 호출한다. * 그리고 마지막에 iIDs 전역변수에 학번이 존재하거나, 아니면 IIDs\_cnt (인자 개수) 가 0 인경우 **print\_wrongL** 를 호출한다. * c옵션과 같이 오는 경우 c옵션을 먼저출력하고 p 옵션의 **print\_wrontL** 을 출력한다.   **s 옵션**   * c 옵션과 마찬가지로 check\_option 함수(옵션 분류함수)에서 인자로 오는 <CATEGORY> 에 stdid 나 score 아닌 문자열이 오거나 <1|-1> 에 다른 숫자가 오는 경우 예외처리하였다. * **check\_option** 함수에서 옵션을 받을 때 <CATEGORY>를 sortType[0] 전역변수에 저장, <1 | -1> 을 sortType[1] 에 저장해놓는다. * score\_students 함수에서 StdList를 생성한다. 그리고 new\_Snode 생성자를 호출하여 **학번만 초기화해서** 연결한 다음, score\_student함수에 해당 Snode를 넘겨준다. * score\_student 함수에서 받은 Snode 에 전 문제에 대하여 **append\_score** 함수를 호출한다. (Sclist 연결) * **score**\_**student** 함수의 마지막에 **sort\_manager** 함수를 호출하여 링크드리스트를 정렬 규칙에 맞게 정렬시킨다. **+ write\_sort** 를 호출하여 해당 정렬된 리스트를 순차적으로 csv 파일에 출력한다.   **[기본으로 구현된 내용들]**  m 옵션   * **score\_score** 함수에서<ANS\_DIR>/score\_table.csv 파일이 존재하지 않을 시 에러처리 * 수정할 문제번호를 입력받아 score\_table 전역변수에 존재 시 새로운 배점으로 score\_table 배점 수정 후 no를 입력받을 때 까지 while loop * 만약 해당 문제가 존재하지 않을 시 wrong number 출력 * 마지막에 **write\_scoreTable**를 호출하여 수정된 전역변수 배열 score\_table을 바탕으로 csv파일 재출력   t 옵션   * **score\_score** 함수에서 입력받은 문제가 존재하지 않을 시 is not existed 출력 후 프로그램 종료(에러처리) * t 옵션 받을 시 **score\_table함수 내부에서 호출하는 compile\_program (프로그램 문제 채점 함수)**에서 lpthread 옵션을 추가하여 compile 실행   e 옵션   * 입력받은 에러디렉토리를realpathS2 를 호출하여 상대경로를 절대경로로 변환함. * e 옵션이 없을 때 **redirection** 함수를 통해 <STD\_DIR> 에 저장된 error.txt를 에러디렉토리안으로 **rename**함수를 사용하여 이동. * **compile\_program** 에서 원래 저장되던 error.txt 를 * score\_stuednts 함수의 마지막에 error saved.. 출력   h 옵션   * **score\_score** 에서 다른 옵션과 함께 사용되면 에러처리, 홀로쓰이면 **print\_usage** 를 호출하여 help를 출력한다.   [student\_score 함수 (각각의 학생에대한 채점함수) 는 흐름도를 이용해서 설명하겠다.]  <각 함수별 흐름도>    [ssu\_score() 함수]    [ssu\_students 함수]    [score\_blank() 함수]    [ssu\_program 함수] |

## blank 상세설계

|  |
| --- |
| blank.h 의 가장 큰 함수는 크게 3가지로, make\_tokens, make\_tree, compare\_tree 로 구성되어있다.  이 3 함수의 가장 큰 특징으로는, 다음과 같다.  make\_tokens : 마치 컴파일러가 lexeme이 들어오면 토큰을 분리해서 parse tree를 만드는 것 처럼 make\_tokens 함수 내부에서 blank.h 의 함수들을 사용하여 datatype 에 정의된 타입 및 변수명/함수명으로 토큰을 분리한다. (Keyword/Identifier 분리). 예를들어, open (fname,O\_WRONLY | O\_CREAT | O\_TRUNC, 0644) 라고 토큰을 만든다면 토큰 배열은 다음과 같이 생성할 것이다. tokens = [“open”, “(“, “fname”, “,” , “O\_WRONLY”, “|”, “O\_CREAT”, “|”, “O\_TRUNC”, “,” , “0644”, “)” ] 이렇게 만들어진 토큰 배열은 make\_tree로 들어가 parse 트리가 생성된다.  make\_tree : 해당 토큰을 바탕으로 함수 내부에서 node 를 이용하여 트리를 만든다. node트리는 일반적으로 같은 우선순위에 있는 토큰들은 next, prev 노드로 연결이 되고있고, 우선순위가 있거나 내부 괄호가 있으면 child, parent 관계로 노드가 연결된다. 또한 토큰배열에 있는 ‘,’ 와 ‘(‘ ‘)’ 는 트리에 node 트리에 들어가지 않는다. 이들은 동일(헝제)관계, 자식부모관계를 구조화하는데 사용될 키워드로 사용될 예정이다.  예시를 들자면 위의 open(fname,O\_WRONLY | O\_CREAT | O\_TRUNC, 0644) 를 node를 이용하여 트리를 만든다면 , open 이 루트노드가 될 것이고, 자식노드로 fname -> | -> 0644 가 올 것이다. 자식 노드 중 | 는 또한 자식노드로 O\_WRONLY -> O\_CREAT -> O\_TRUNC 를 갖게되는 트리형태를 가지게 될 것이다. 이렇게 마치 BNF 를 파스트리를 만드는 형태로 트리가 만들어지게 된다.  compare\_tree  make\_tokens를 거쳐 만들어진 노드들을 바탕으로 트리가 연결된다. 해당 트리는 자식노드와 자식 수가 일치하는지 트리형태가 일치하는지 확인한다. 만약 답이 일치하지 않는다면 정답 노드에 다른 답이 : 으로 분리되어있는지 확인한 후 정답을 비교하는 함수이다. 재귀적으로 호출하였는데 트리가 일치한다면 true 를 일치하지 않는다면 false 를 리턴한다.    **[compare\_tree() 흐름도]**    **[Make\_tokens 흐름도]**    **[make\_tree 흐름도]** |

# 실행결과

## ssu\_score 구현

1. 점수 테이블 파일(score\_table.csv) 생성 및 score.csv 파일 생성  
   분석 : 파일 생성 후 바로 실행되는데, <ANS\_DIR>결과파일에 ssu\_test.txt 와 같은 파일이 존재하지 않은 채 바로 실행되어 open error가 발생하였다. 차후에 score\_table.csv 를 만든 후 수동적으로 ssu\_test.txt 파일을 디렉토리에 추가해주면 open error가 발생하지 않는다.

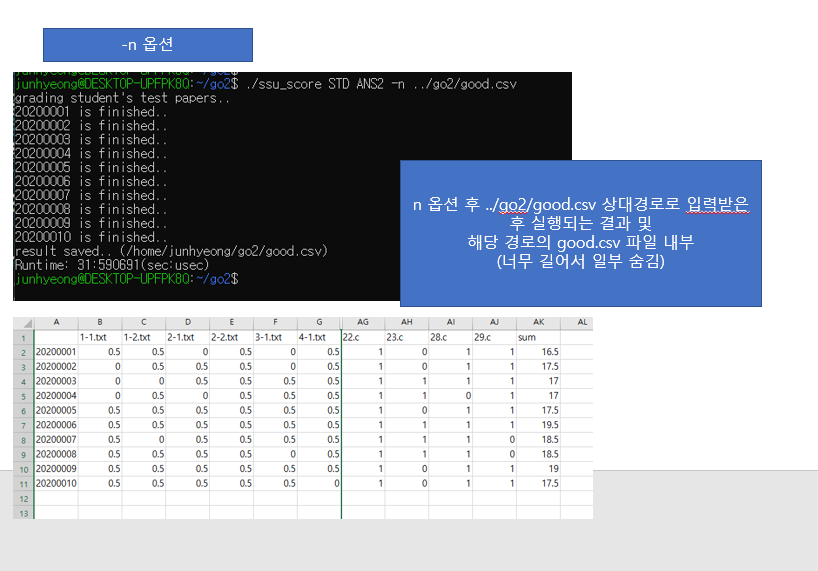
텍스트이(가) 표시된 사진

자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명

## n 옵션

* n 옵션에서 상대경로로 받은 <CSVFILENAME> 에서 csv 파일이 아니면 기본적으로 에러처리가 되도록 하였고, 상대경로 제작 시 /../ 를 남용하여 root 보다 범위가 벗어나는 경우 에러처리를 하였다.
* 실행 결과 마지막에 resulted saved.. (절대경로화) 하여 출력하도록 하였다.

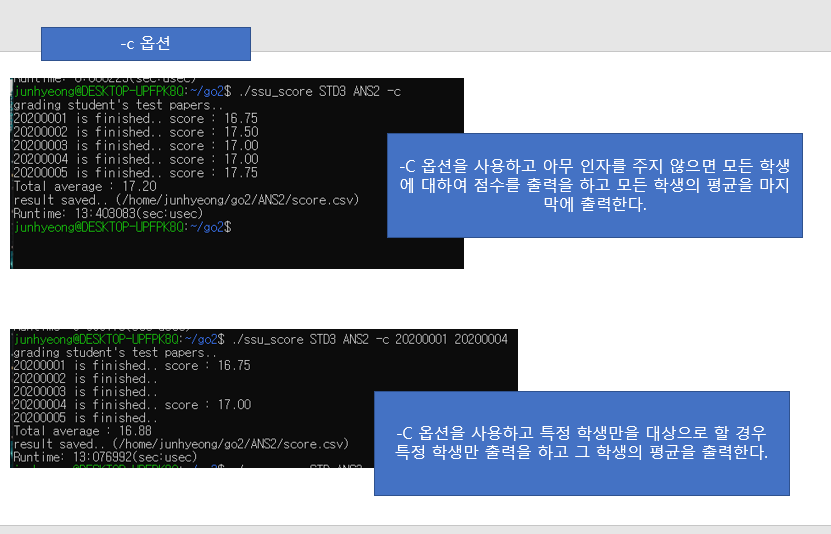


## m 옵션

* 점수 테이블 내의 특정 번호 배점 수정을 할 수 있고, no를 입력하면 배점 수정을 종료한다.



## c 옵션



텍스트이(가) 표시된 사진

자동 생성된 설명

## p 옵션

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

## t 옵션

* 현재 Ubuntu 22.04 gcc 컴파일러는 lpthread 옵션을 사용하지 않아도 자동으로 동적링킹하여 <pthread.h> 파일들을 컴파일 해주는 듯 하다. 그래서 실제로 -t 옵션을 사용 전/후 가 동일한 채점결과를 보인다는 분석을 하였다.
* 텍스트이(가) 표시된 사진





  자동 생성된 설명
* 텍스트이(가) 표시된 사진

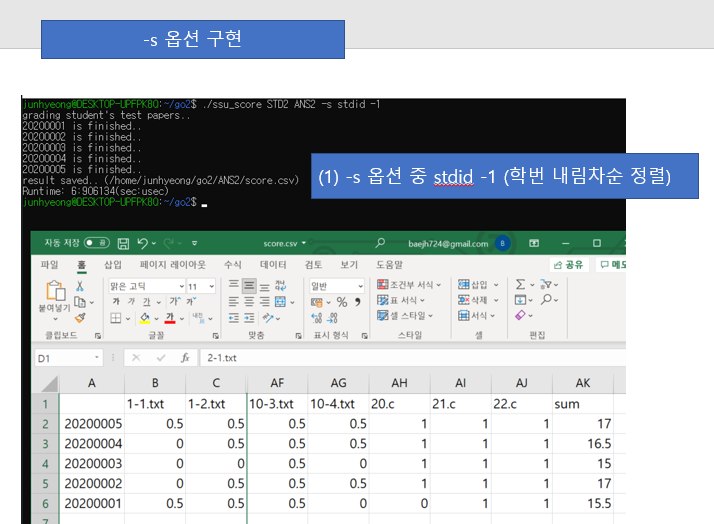
  자동 생성된 설명

## s 옵션

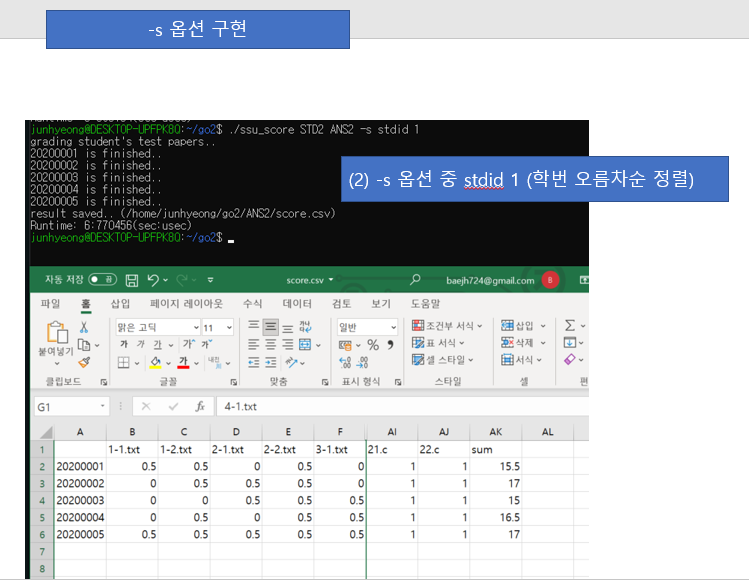
* 해당옵션은 p 옵션이랑 같이사용하지 않도록 설계되어있다. (같이 사용시 에러메시지 출력 후 종료)

: p 옵션에서 링크드리스트는 틀린문제만을 관리하도록 설계되어있다. 반대로 s 옵션은 채점한 모든 문제에 대하여 연결리스트로 관리하도록 설계해두었다. 그 과정에서 **관리리스트 충돌**이 발생한다는 문제점이 존재하였다. 물론 Snode\* 내부변수 중 현점수 변수와 원래배점 점수를 비교하도록 하는 내부 메커니즘이 있어 연산에는 문제가 없지만, 리스트 관리차원에서는 문제가 생기므로 s와 p 옵션을 동시에 설정하지 못하도록 설정하였다.

1. stdid -1 옵션 : 학번 오름차순



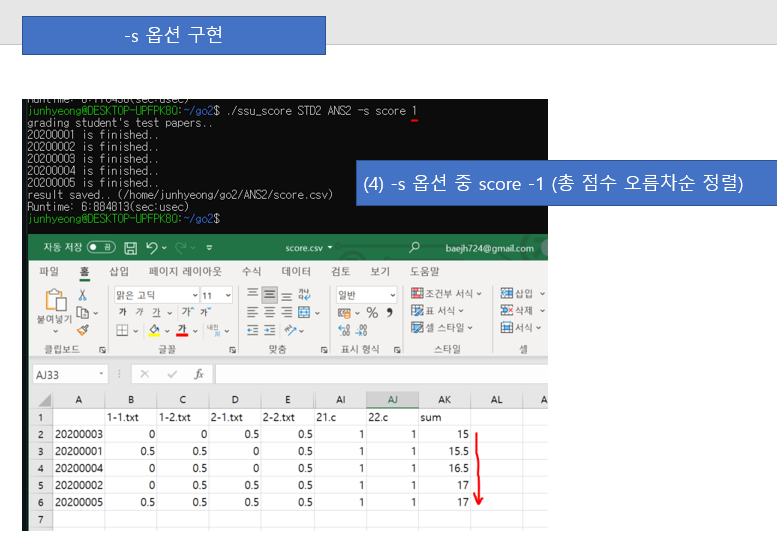
1. stdid 1 옵션 : 학번 내림차순



1. score -1 옵션 : 총 점수 오름차순



1. score 1 옵션 : 총 점수 내림차순



1. S옵션 예외처리 (잘못된 인자에 대한 예외처리)

텍스트이(가) 표시된 사진

자동 생성된 설명

## e 옵션

텍스트이(가) 표시된 사진

자동 생성된 설명

## h 옵션

텍스트이(가) 표시된 사진

자동 생성된 설명

# ++ 과제 설계도

텍스트, 편지이(가) 표시된 사진

자동 생성된 설명

