Report

C프로그래밍(CSE2035-03) 프로젝트 2

학교 생활 관리



제출일 : 2017 . 12 . 14

과목명 : C프로그래밍(CSE2035-03)

담당교수 : 김세준

전공 : 컴퓨터공학 심화

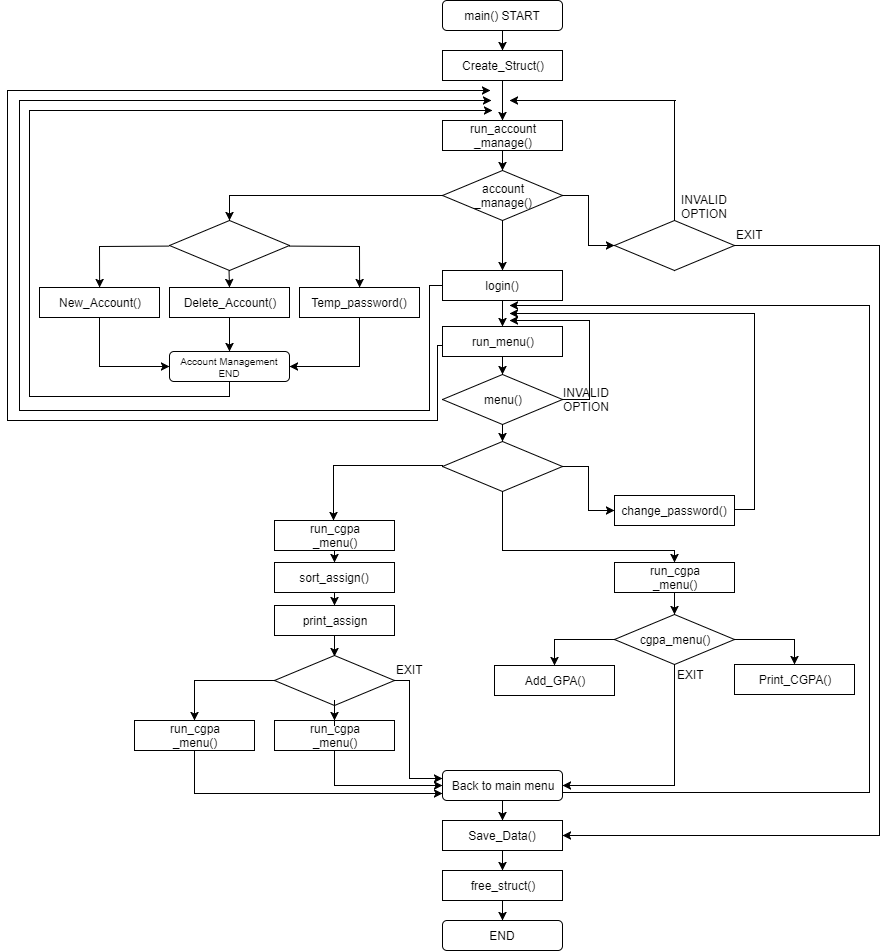
학번 : 20171678

이름 : 이찬희

**1. 목적**

C 프로그래밍 수업에서 배운 파일 입출력과 구조체를 이용하여 과제, 학점 관리 프로그램을 만들어 본다.

**2. 프로그램 구성도 및 흐름도**



**3. 함수 사용**

3.1 main

|  |  |
| --- | --- |
| 기능요약 | Create\_Struct를 호출하여 구조체를 생성한다. While문을 사용하여 프로그램의 주 기능을 실행, 예외들을 처리하고 Save\_Data를 호출하여 구조체의 데이터들을 파일에 저장한다. |

3.2 run\_account\_manage

|  |  |
| --- | --- |
|  | void run\_account\_manage (); |
| 기능요약 | Account\_Manage를 호출하여 command를 받아온 후 switch문을 사용하여 받은 command에 해당하는 기능을 실행한다. |

3.3 run\_menu

|  |  |
| --- | --- |
|  | Int run\_menu (STUDENT \*login\_st); |
| 기능요약 | Menu를 호출하여 command를 받아온 후 switch 문을 사용하여 받은 command에 해당하는 기능을 실행한다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.4 run\_cgpa\_menu

|  |  |
| --- | --- |
|  | Int run\_cgpa\_menu (STUDENT \*login\_st); |
| 기능요약 | Cgpa\_menu를 호출하여 command를 받아온 후 switch 문을 사용하여 받은 command에 해당하는 기능을 실행한다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.5 run\_assign\_menu

|  |  |
| --- | --- |
|  | Int run\_assign\_menu (STUDENT \*login\_st); |
| 기능요약 | Sort\_Assign을 호출하여 assign를 날짜순으로 정렬하고 Print\_Assign을 호출하여 이를 출력한다. 이후wgetch()를 이용하여 command를 받아온 후 switch 문을 사용하여 받은 command에 해당하는 기능을 실행한다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.6 Save\_Data

|  |  |
| --- | --- |
|  | Void Save\_Data (); |
| 기능요약 | 3중 for문과 fprintf()를 사용하여 구조체의 데이터를 파일에 저장한다. |

3.7 search\_year

|  |  |
| --- | --- |
|  | Int search\_year (char year[]); |
| 기능요약 | For 문과 strcmp를 사용하여 year과 같은 문자열을 저장하고 있는 ST\_YEAR의 인덱스를 찾는다. |
| 파라미터 | Year는 찾고자 하는 연도를 저장하는 문자열이다. |

3.8 search\_num

|  |  |
| --- | --- |
|  | STUDENT\* search\_num (char number[], int year\_num) |
| 기능요약 | for문과 strcmp를 사용하여 number과 같은 문자열을 저장하고 있는 STUDENT\*를 찾는다. |
| 파라미터 | Number는 찾고자 하는 학번을 저장하는 문자열이고, year\_num은 해당 학번이 위치하는 ST\_YEAR의 인덱스이다. |

3.9 Create\_Struct

|  |  |
| --- | --- |
|  | Void Create\_Struct (); |
| 기능요약 | Data.txt를 이용하여 구조체를 생성한다. |
| 설명 | Data.txt를 읽기 전용으로 열고 TOP을 동적으로 할당한 후, TOP->Year\_Size를 0으로 초기화 해준다. Read\_data를 호출하여 파일에서 데이터를 읽고 구조체를 만든 후 파일을 닫는다. |

3.10 read\_data

|  |  |
| --- | --- |
|  | Void read\_data (FILE \*fpoint); |
| 기능요약 | 파일스트림에서 데이터를 읽어 구조체를 생성한다. |
| 파라미터 | Fpoint는 데이터를 읽어오는 파일스트림을 저장하고 있는 변수이다. |
| 설명 | Year, number, password, Assign\_size, CGPA\_size를 파일스트림에서 읽어오고, create\_student를 호출하여 STUENT를 생성, 데이터를 저장한다. 이 때 읽을 수 없으면 while문을 빠져나온다. 이후 search\_year와 search\_num을 차례로 호출하여 생성한 STUDENT\*를 찾고, 여기에 assign\_size와 cgpa\_size만큼 정보를 읽어와 각각 create\_assign, create\_cgpa를 호출하여 생성해주고, 이전의 list와 연결해준다. |

3.11 create\_student

|  |  |
| --- | --- |
|  | void create\_student  (char year[5], char number[5], char password[16], int a\_size, int c\_size); |
| 기능요약 | 전달받은 파라미터를 이용하여 STUDENT\*를 생성하고 데이터를 저장한다. |
| 파라미터 | number, password, a\_size, c\_size는 각각 만들어지는 STUDENT에 저장할 number, password, Assign\_Size, CGPA\_Size를 저장하는 변수이고, year은 해당 STUDENT\*가 속하는 ST\_YEAR의 인덱스를 검색하기 위한 문자열이다. |
| 설명 | 만약 TOP->Year\_Size가 0 이라면 year를 이용하여 ST\_YEAR를 검색할 필요 없이 STUDENT\*를 할당하고 ST\_YEAR[0].ST\_NUM->link에 연결한다. 그렇지 않다면 우선 search\_year를 이용하여 ST\_YEAR의 인덱스, 즉 year\_num을 찾고 존재하는 경우 마지막 STUDENT의 link와, 존재하지 않는 경우 ST\_YEAR을 한 사이즈 크게 재할당하여 헤드의 link와 연결해주고 데이터를 저장한다. 이후 ASSIGN\* 과 CGPA\*를 하나씩 할당하여 헤드로 Child\_A, Child\_C에 연결해준다. |

3.12 create\_assign

|  |  |
| --- | --- |
|  | void create\_assign  (ASSIGN \*new, char name[100], char describe[100], char professor[100], int month, int date); |
| 기능요약 | 전달받은 데이터를 new안에 저장한다. |
| 파라미터 | New는 생성된 ASSIGN\*를, 나머지 파라미터는 이에 저장될 값들을 저장하는 변수이다. |

3.13 create\_cgpa

|  |  |
| --- | --- |
|  | void create\_cgpa (CGPA \*new, int semester, float score); |
| 기능요약 | 전달받은 데이터를 new안에 저장한다. |
| 파라미터 | New는 생성된 CGPA\*를, 나머지 파라미터는 이에 저장될 값들을 저장하는 변수이다. |

3.14 Account\_Manage

|  |  |
| --- | --- |
|  | char Account\_Manage(); |
| 기능요약 | Function reads a user’s choice and passes. |

3.15 menu

|  |  |
| --- | --- |
|  | char menu(); |
| 기능요약 | Function reads a user’s choice and passes. |

3.16 cgpa\_menu

|  |  |
| --- | --- |
|  | char cgpa\_menu(); |
| 기능요약 | Function reads a user’s choice and passes. |

3.17 run\_assign\_menu

|  |  |
| --- | --- |
|  | int run\_assign\_menu (STUDENT \*login\_st); |
| 기능요약 | Sort\_Assign을 호출하여 assign들을 정렬하고Print\_assign을 호출하여 이를 출력한다. 이후 메뉴를 출력하고 유저의 결정을 읽어 그에 맞는 기능을 실행하거나 값을 리턴한다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.18 Print\_Assign

|  |  |
| --- | --- |
|  | void Print\_Assign(STUDENT \*login\_st); |
| 기능요약 | Assign의 D-day를 계산하고, Assign들을 출력한다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |
| 설명 | 과제가 마무리되어야 할 날짜를 저장할 struct 스 d\_day를 생성하여 여기에 과제의 due-date를 넣고 mktime을 호출하여 반환된 t\_time의 값을 d에 저장한다. 이와 t(현재 시간의 time\_t)의 차를 이용하여 d-day를 계산하고 출력한다. |

3.19 Add\_GPA

|  |  |
| --- | --- |
|  | void Add\_GPA (STUDENT \*login\_st); |
| 기능요약 | 새로 만들 GPA에 대한 정보를 입력 받고 semester가 올바르지 못하거나 GPA가 범위를 벗어날 경우 예외로 처리한다. Semester의 값을 판단하여 새로운 GPA를 만들어야 할 경우 만들고, 수정해야 할 경우 Cor\_GPA를 호출해 수정한다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.20 Cor\_GPA

|  |  |
| --- | --- |
|  | void Cor\_GPA(STUDENT \*login\_st, char semester, float gpa); |
| 기능요약 | Semester를 통해 수저할 CGPA를 찾고 그 score를 gpa로 바꾼다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. Semester는 수정 될 GPA의 학기를, gpa는 수정할 성적값이 저장된 변수이다. |

3.21 Print\_CGPA

|  |  |
| --- | --- |
|  | void Print\_CGPA(STUDENT \*login\_st); |
| 기능요약 | Prints out the statistics with numbers. Calculates CGPA by GPAs’ average. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.22 Sort\_Assign

|  |  |
| --- | --- |
|  | void Sort\_Assign (STUDENT \*login\_st); |
| 기능요약 | Insertion sort를 사용하여 assign들을 날짜기준으로 정렬한다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.23 Add\_Assign

|  |  |
| --- | --- |
|  | void Add\_Assign(STUDENT \*login\_st); |
| 기능요약 | 추가할 assignment의 이름, 설명, 교수님, 날짜를 입력받은 후 create\_assign을 호출하여 추가한다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.24 Delete\_Assign

|  |  |
| --- | --- |
|  | void Delete\_Assign(STUDENT \*login\_st); |
| 기능요약 | 삭제할 assignment의 이름을 입력받고, 만약 존재하지 않는다면 에러메시지를 출력한다. 존재한다면 해당 list의 앞과 뒤 list를 연결하고 해당 list를 free 해준다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.25 New\_Account

|  |  |
| --- | --- |
|  | int New\_Account(); |
| 기능요약 | 새로 만들 계정의 학번과 비밀번호를 입력 받고, 만약 이미 존재하는 학번이라면 NEW\_ACCOUNT\_FAIL을 반환하여 에러처리를, 그렇지 않다면 create\_student를 호출하여 새 계정을 만든 후 NEW\_ACCOUNT\_SUCCESS를 반환한다. |

3.26 Delete\_Account

|  |  |
| --- | --- |
|  | int Delete\_Account(); |
| 기능요약 | 삭제할 계정의 학번과 비밀번호를 입력 받고, 만약 존재하지 않는 학번이라면 DELETE\_ACCOUNT\_FAIL\_INVALID를, 비밀번호가 틀렸다면 DELETE\_ACCOUNT\_FAIL\_PASSWORD를 반환하여 에러처리를 해준다. 존재하는 학번이라면 앞, 뒤 list를 연결해주고 free\_student를 호출하여 free 해준다. |

3.27 free\_struct

|  |  |
| --- | --- |
|  | void free\_struct (); |
| 기능요약 | For문에서 free\_student\_year를 호출하여TOP->ST\_YEAR[i].ST\_NUM->link를 free해주고 TOP->ST\_YEAR[i].ST\_NUM또한 free 해준다. For문이 종료되면 TOP->ST\_YEAR를 free하고 TOP을 free해준다. |

3.28 free\_year

|  |  |
| --- | --- |
|  | void free\_year (STUDENT\* sptr); |
| 기능요약 | Sptr이 list의 마지막 node일 때 까지 free\_year(sptr->link)를 재귀적으로 호출하여 STUDENT\*를 free 해준다. |
| 파라미터 | Free해 줘야 하는 STUDENT\*를 저장한다. |

3.29 free\_student

|  |  |
| --- | --- |
|  | void free\_student (STUDENT \*account); |
| 기능요약 | Free\_assign()과 free\_cgpa()를 호출하여 account의 ASSIGN과 CGPA들을 free해준 후 account를 free해준다. |
| 파라미터 | Free 해 줘야 하는 STUDENT\*를 저장한다. |

3.30 free\_assign

|  |  |
| --- | --- |
|  | void free\_assign (ASSIGN \*aptr); |
| 기능요약 | Aptr이 list의 마지막 node일 때 까지 free\_assign(aptr->link)를 재귀적으로 호출하여 ASSIGN\*를 free 해준다. |
| 파라미터 | Free해 줘야 하는 ASSIGN\*를 저장한다. |

3.31 free\_cgpa

|  |  |
| --- | --- |
|  | void free\_cgpa (CGPA \*cptr); |
| 기능요약 | cptr이 list의 마지막 node일 때 까지 free\_cgpa(cptr->link)를 재귀적으로 호출하여 CGPA\*를 free 해준다. |
| 파라미터 | Free해 줘야 하는 CGPA\*를 저장한다. |

3.32 Change\_Password

|  |  |
| --- | --- |
|  | int Change\_Password(STUDENT \*login\_st); |
| 기능요약 | 바꾸고자 하는 비밀번호를 두 번 입력받고 그 둘이 같으면 비밀번호를 바꾼다. |
| 파라미터 | Login\_st는 현재 로그인한 계정의 STUDENT 포인터값을 나타낸다. |

3.33 login

|  |  |
| --- | --- |
|  | int login(); |
| 기능요약 | 학번과 비밀번호를 입력 받아 로그인을 한다. 만약 없는 계정이거나 비밀번호가 틀렸다면 LOGIN\_FAILED를 반환한다. |

3.34 temp\_password

|  |  |
| --- | --- |
|  | void Temp\_Password(); |
| 기능요약 | 학번과 가장 최근 학기의 GPA를 입력받아 일치한다면 임시 비밀번호를 발급한다. 이 때 임시 비밀번호는 8자리 문자열로, 각각의 문자는rand()를 이용하여 33~126사이의 난수를 생성하여 만든다. |

**4. 예외 처리**

- 로그인을 할 때 존재하지 않는 계정이거나 비밀번호가 틀렸을 시 Login Failed!를 출력한다.

- 새 계정을 만들고자 할 때 이미 존재하는 계정이면 Failed to create new accont. (Existing account)를 출력한다.

- 계정을 삭제하고자 할 때 존재하지 않는 계정이면 Failed to delete account. (NON-existing account)를 출력한다.

- 계정을 삭제하고자 할 때 비밀번호를 틀렸다면 Failed to delete account. (Wrong password)를 출력한다.

- 모든 메뉴에서 유효하지 않은 값을 입력하면 INVAILD\_CHOICE를 반환하고 Please input valid choice. 를 출력한다.

- 비밀번호를 바꾸고자 할 때 두 입력값이 같지 않으면 Failed to change password. (Incorrect confirm password)를 출력한다.

- GPA를 추가할 때 semester 나 gpa의 값이 올바른 범위를 벗어나면 You entered wrong number!를 출력한다.

- Assignment를 추가할 때 존재하지 않으면 You don't have assignment named '%s'!를 출력한다.

- 임시 비밀번호를 발급받을 때 존재하지 않는 학번을 입력받으면 Failed to get temporary password. (Wrong student number)를 출력한다.

- 임시 비밀번호를 발급받을 때 학번에 GPA정보가 존재하지 않으면 Failed to get temporary password. (No CGPA Information)를 출력한다.

- 임시 비밀번호를 발급받을 때 입력한 마지막 학기의 GPA가 저장되어있는 정보와 다르면 Failed to get temporary password. (Wrong GPA)를 출력한다.

**5. 추가 구현**

- linked list를 이용하여 프로그램을 제작하였다. STUDENT, ASSIGN, CGPA에 link를 추가하여 linked list로 사용하였다.

- Assignment를 만들고 불러올 때 fgets()와 getstr()를 사용하여 공백문자를 포함한 문자열도 받을 수 있다.

**6. 코드**

#include "management.h"

int main(){

int exit=0;

int login\_flag=0;

STUDENT \*login\_st;

initscr();

Create\_Struct();

while(!exit){

switch(run\_account\_manage(login\_st)){

case LOGIN\_FAILED :

clear();

printw("Login Failed!\n\n");

continue;

case NEW\_ACCOUNT\_SUCCESS :

clear();

printw("New account Succesfully made!\n\n");

continue;

case NEW\_ACCOUNT\_FAIL :

clear();

printw("Failed to create new accont. (Existing account)\n\n");

continue;

case DELETE\_ACCOUNT\_SUCCESS :

clear();

printw("Succesfully deleted account!\n\n");

continue;

case DELETE\_ACCOUNT\_FAIL\_INVALID :

clear();

printw("Failed to delete account. (NON-existing account)\n\n");

continue;

case DELETE\_ACCOUNT\_FAIL\_PASSWORD :

clear();

printw("Failed to delete account. (Wrong password)\n\n");

continue;

case EXIT :

login\_flag = 0;

exit = 1;

break;

case INVAILD\_CHOICE :

clear();

printw("Please input valid choice.\n\n");

continue;

case LOGIN\_SCCUESS :

login\_flag = 1;

break;

}

if(login\_flag == 1){

char number[5] = {Curr\_Num[4], Curr\_Num[5], Curr\_Num[6], Curr\_Num[7], '\0'};

login\_st = search\_num(number, Login\_Year);

}

clear();

while(login\_flag){

switch(run\_menu(login\_st)){

case INVAILD\_CHOICE :

clear();

printw("Please input valid choice.\n\n");

continue;

case LOGOUT :

clear();

printw("Successfully loged out!\n\n");

login\_flag = 0;

continue;

case CHANGE\_PASSWORD\_FAIL :

clear();

printw("Failed to change password. (Incorrect confirm password)\n\n");

continue;

case CHANGE\_PASSWORD\_SUCCESS :

clear();

printw("Successfully changed password!\n\n");

continue;

}

clear();

}

clear();

}

endwin();

Save\_Data();

system("clear");

free\_struct();

return 0;

}

int run\_account\_manage () {

char command = Account\_Manage();

switch(command) {

case MENU\_LOGIN :

return login();

case MENU\_NEW :

return New\_Account();

case MENU\_DEL :

return Delete\_Account();

case MENU\_TEMP :

Temp\_Password();

break;

case MENU\_QUIT :

return EXIT;

default :

return INVAILD\_CHOICE;

}

return 0;

}

int run\_menu( STUDENT \*login\_st ) {

char command = menu();

int flag = 1;

switch(command) {

case MENU\_ASSIGN :

clear();

flag = 1;

while(flag){

switch(run\_assign\_menu(login\_st)){

case MAIN\_MENU :

flag = 0;

clear();

continue;

case INVAILD\_CHOICE :

clear();

printw("Please input valid choice.\n\n");

continue;

}

clear();

}

return MAIN\_MENU;

case MENU\_CGPA :

clear();

flag = 1;

while(flag){

switch(run\_cgpa\_menu(login\_st)){

case MAIN\_MENU :

flag = 0;

clear();

continue;

case INVAILD\_CHOICE :

clear();

printw("Please input valid choice.\n\n");

continue;

}

clear();

}

return MAIN\_MENU;

case MENU\_CHANGE :

return Change\_Password(login\_st);

case MENU\_LOGOUT :

return LOGOUT;

default :

return INVAILD\_CHOICE;

}

return 0;

}

int run\_cgpa\_menu ( STUDENT \*login\_st ) {

char command = cgpa\_menu();

switch(command) {

case CGPA\_ADD :

Add\_GPA(login\_st);

break;

case CGPA\_VIEW :

Print\_CGPA(login\_st);

break;

case CGPA\_QUIT :

return MAIN\_MENU;

default :

return INVAILD\_CHOICE;

}

return 0;

}

void Save\_Data(){

FILE\* fpoint;

fpoint = fopen("data.txt", "w");

for(int i = 0; i < TOP->Year\_Size; i++){

STUDENT \*current\_st = TOP->ST\_YEAR[i].ST\_NUM;

for(int j = 0; j< TOP->ST\_YEAR[i].Num\_Size; j++){

current\_st = current\_st->link;

fprintf(fpoint, "%s%s / %s / %d / %d\n",

TOP->ST\_YEAR[i].year,

current\_st->number,

current\_st->password,

current\_st->Assign\_Size,

current\_st->CGPA\_Size);

ASSIGN \*current\_as = current\_st->Child\_A;

for(int k = 0; k < current\_st->Assign\_Size; k++){

current\_as = current\_as->link;

fprintf(fpoint, "%s\n", current\_as->name);

fprintf(fpoint, "%s\n", current\_as->describe);

fprintf(fpoint, "%s\n", current\_as->professor);

fprintf(fpoint, "%d %d\n", current\_as->date[0], current\_as->date[1]);

}

CGPA \*current\_cg = current\_st->Child\_C;

for(int k = 0; k < current\_st->CGPA\_Size; k++){

current\_cg = current\_cg->link;

fprintf(fpoint, "%d %f\n", current\_cg->semester, current\_cg->score);

}

}

}

fclose(fpoint);

}

int search\_year (char year[]) {

int year\_num;

for(year\_num = 0; year\_num < TOP->Year\_Size; year\_num++){

if(strcmp(TOP->ST\_YEAR[year\_num].year, year) == 0){

break;

}

}

return year\_num;

}

STUDENT\* search\_num (char number[], int year\_num) {

STUDENT \*sptr;

sptr = TOP->ST\_YEAR[year\_num].ST\_NUM;

for(int i = 0; i < TOP->ST\_YEAR[year\_num].Num\_Size+1; i++){

if(strcmp(sptr->number, number) == 0){

break;

}

sptr = sptr->link;

}

return sptr;

}

void Create\_Struct(){

FILE\* fpoint;

fpoint = fopen("data.txt","r");

TOP = malloc(sizeof(TREE\_HEAD));

TOP->Year\_Size = 0;

read\_data(fpoint);

fclose(fpoint);

}

void read\_data (FILE \* fpoint) {

int a\_size, c\_size, month, date, year\_num, semester;

float score;

char password[16], year[5], number[5], name[100], describe[100], professor[100];

while(1){

if( fscanf(fpoint, "%4s%4s / %s / %d / %d\n",

year, number, password, &a\_size, &c\_size) != 5 ){

break;

}

else{

create\_student(year, number, password, a\_size, c\_size);

}

year\_num = search\_year(year);

STUDENT \*current\_st = search\_num(number, year\_num);

if(a\_size != 0){

ASSIGN \*current\_as = current\_st->Child\_A;

for(int i = 0; i < a\_size; i++){

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

current\_as->link = new;

fgets(name, 100, fpoint);

fgets(describe, 100, fpoint);

fgets(professor, 100, fpoint);

fscanf(fpoint, "%d %d\n", &month, &date);

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

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

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

create\_assign(new, name, describe, professor, month, date);

current\_as = current\_as->link;

}

}

if(c\_size != 0){

CGPA \*current\_cg = current\_st->Child\_C;

for(int i = 0; i < c\_size; i++){

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

current\_cg->link = new;

fscanf(fpoint, "%d %f\n", &semester, &score);

create\_cgpa(new, semester, score);

current\_cg = current\_cg->link;

}

}

}

}

void create\_student (char year[5], char number[5], char password[16], int a\_size, int c\_size) {

int year\_num;

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

if(TOP->Year\_Size == 0){

YEAR \*year\_head = malloc(sizeof(YEAR));

TOP->ST\_YEAR = year\_head;

TOP->Year\_Size = 1;

strcpy(TOP->ST\_YEAR[0].year,year);

STUDENT \*head = malloc(sizeof(STUDENT));

head->link = new;

TOP->ST\_YEAR[0].ST\_NUM = head;

TOP->ST\_YEAR[0].Num\_Size = 1;

strcpy(new->number, number);

strcpy(new->password, password);

new->Assign\_Size = a\_size;

new->CGPA\_Size = c\_size;

}

else{

year\_num = search\_year(year);

if(year\_num == TOP->Year\_Size){

TOP->Year\_Size++;

TOP->ST\_YEAR = realloc(TOP->ST\_YEAR, TOP->Year\_Size \* sizeof(YEAR));

strcpy(TOP->ST\_YEAR[year\_num].year,year);

STUDENT \*head = malloc(sizeof(STUDENT));

TOP->ST\_YEAR[year\_num].ST\_NUM = head;

TOP->ST\_YEAR[year\_num].ST\_NUM->link = new;

TOP->ST\_YEAR[year\_num].Num\_Size = 1;

strcpy(new->number, number);

strcpy(new->password, password);

new->Assign\_Size = a\_size;

new->CGPA\_Size = c\_size;

}

else{

STUDENT \*sptr = TOP->ST\_YEAR[year\_num].ST\_NUM;

for(int j = 0; j < TOP->ST\_YEAR[year\_num].Num\_Size; j++){

sptr = sptr->link;

}

sptr->link = new;

TOP->ST\_YEAR[year\_num].Num\_Size++;

strcpy(new->number, number);

strcpy(new->password, password);

new->Assign\_Size = a\_size;

new->CGPA\_Size = c\_size;

}

}

ASSIGN \*head\_as = malloc(sizeof(ASSIGN));

new->Child\_A = head\_as;

head\_as->link = NULL;

CGPA \*head\_cg = malloc(sizeof(CGPA));

new->Child\_C = head\_cg;

head\_cg->link = NULL;

new->link = NULL;

//DONE (for now : 1208)

}

void create\_assign

(ASSIGN \*new, char name[100], char describe[100], char professor[100], int month, int date) {

strcpy(new->name, name);

strcpy(new->describe, describe);

strcpy(new->professor, professor);

new->date[0] = month;

new->date[1] = date;

new->link = NULL;

}

void create\_cgpa (CGPA \*new, int semester, float score) {

new->semester = semester;

new->score = score;

new->link = NULL;

}

char Account\_Manage(){

printw("1. Login\n");

printw("2. Create new Account\n");

printw("3. Delete Account\n");

printw("4. Get Temporary Password\n");

printw("5. Exit\n");

noecho();

return wgetch(stdscr);

}

char menu(){

printw("1. Assignment Management\n");

printw("2. CGPA Management\n");

printw("3. Change Password\n");

printw("4. Logout\n");

noecho();

return wgetch(stdscr);

}

int run\_assign\_menu (STUDENT \*login\_st){

char command;

printw("<Assignment Management for %s>\n\n",Curr\_Num);

Sort\_Assign(login\_st);

Print\_Assign(login\_st);

printw("1. New Assignment\n");

printw("2. Delete Assignment\n");

printw("3. Return to main menu\n");

noecho();

command = wgetch(stdscr);

switch(command){

case '1':

clear();

Add\_Assign(login\_st);

break;

case '2':

clear();

Delete\_Assign(login\_st);

break;

case '3':

return MAIN\_MENU;

default :

return INVAILD\_CHOICE;

}

clear();

}

void Print\_Assign(STUDENT \*login\_st){

int differ;

time\_t t, d;

struct tm d\_day, \*today;

ASSIGN \*aptr = login\_st->Child\_A;

t = time(NULL);

today = localtime(&t);

for(int i = 0; i < login\_st->Assign\_Size; i++){

aptr = aptr->link;

printw("<%d> Name\t\t : %s\n", i + 1, aptr->name);

printw(" Describe\t\t : %s\n", aptr->describe);

printw(" Professor\t\t : %s\n", aptr->professor);

printw(" Due : %d/%d\n", aptr->date[0], aptr->date[1]);

d\_day.tm\_year = today->tm\_year;

d\_day.tm\_mon = aptr->date[0] - 1;

d\_day.tm\_mday = aptr->date[1];

d\_day.tm\_hour = today->tm\_hour;

d\_day.tm\_min = today->tm\_min;

d\_day.tm\_sec = today->tm\_sec;

d\_day.tm\_isdst = 0;

d = mktime(&d\_day);

differ = (int)(t - d);

if(differ > 0){

printw(" D-day over!!!\n\n");

}

else if(differ == 0){

printw(" D-day\t\t : TODAY!\n\n ");

}

else if(differ < 0){

printw(" D-day\t\t : %d\n\n", (int)(differ / 86400));

}

}

}

char cgpa\_menu() {

printw("1. Add new GPA or Change existing GPA\n");

printw("2. View all\n");

printw("3. Exit\n");

noecho();

return wgetch(stdscr);

}

void Add\_GPA (STUDENT \*login\_st) {

char semester;

float gpa;

int gpa\_size;

clear();

echo();

printw("Input the semester of GPA : ");

scanw("%c", &semester);

gpa\_size = login\_st->CGPA\_Size;

if(semester - '0' > (gpa\_size + 1) || semester - '0' < 1) {

printw("You entered wrong number!\n");

getch();

return ;

}

printw("Input the GPA : ");

scanw("%f", &gpa);

if(gpa < 0 || gpa > 4.31) {

printw("You entered wrong number!\n");

getch();

return ;

}

if(semester - '0' <= gpa\_size) {

Cor\_GPA(login\_st, semester, gpa);

return ;

}

login\_st->CGPA\_Size++;

gpa\_size++;

CGPA \*cptr = login\_st->Child\_C;

for(int i = 0; i < gpa\_size - 1; i++){

cptr = cptr->link;

}

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

cptr->link = new;

new->semester = semester - '0';

new->score = gpa;

new->link = NULL;

printw("GPA has added successfully!\n");

getch();

return ;

}

void Cor\_GPA(STUDENT \*login\_st, char semester, float gpa) {

CGPA \*cptr = login\_st->Child\_C;

for(int i = 0; i < semester - '0'; i++){

cptr = cptr->link;

}

cptr->score = gpa;

printw("GPA has changed successfully!\n");

getch();

return ;

}

void Print\_CGPA(STUDENT \*login\_st) {

float sum = 0;

printw("%s\n", login\_st->number);

CGPA \*cptr = login\_st->Child\_C;

clear();

printw("<CGPA Management for %s>\n", Curr\_Num);

printw("------------------------------\n");

printw(" Semester | GPA | CGPA | \n");

for(int i = 0 ; i < login\_st->CGPA\_Size ; i++) {

cptr = cptr->link;

sum += cptr->score;

printw("------------------------------\n");

printw(" %d | %.2f | %.2f |\n", cptr->semester, cptr->score, sum/(i + 1));

}

printw("------------------------------\n\n");

getch();

}

void Sort\_Assign (STUDENT \*login\_st) {

ASSIGN \*check, \*check\_bf, \*bef, \*af;

for(int i = 1; i < login\_st->Assign\_Size; i++){

check = login\_st->Child\_A->link;

check\_bf = login\_st->Child\_A;

for(int j = 0; j < i; j++){

check = check->link;

check\_bf = check\_bf->link;

}

for(int j = 0; j < i; j++){

bef = login\_st->Child\_A->link;

for(int k = 0; k < j; k++){

bef = bef->link;

}

if(j == 0){

if(100\*check->date[0] + check->date[1] < 100\*bef->date[0] + bef->date[1]){

login\_st->Child\_A->link = check;

check\_bf->link = check->link;

check->link = bef;

j = i;

break;

}

}

else ;

if(j == (i - 1)) ;

else{

if(100\*check->date[0] + check->date[1]

>= 100\*bef->date[0] + bef->date[1]

&& 100\*check->date[0] + check->date[1]

< 100\*bef->link->date[0] + bef->link->date[1]){

af = bef->link;

bef->link = check;

check\_bf->link = check->link;

check->link = af;

j = i;

break;

}

}

}

}

}

void Add\_Assign(STUDENT \*login\_st) {

char name[100], describe[100], professor[100];

int month, date;

echo();

printw("Enter the name of new assignment :\n");

printw("");

getstr(name);

printw("Enter the description of new assignment :\n");

getstr(describe);

printw("Enter the professor of new assignment :\n");

getstr(professor);

printw("Enter the due month of new assignment : ");

scanw("%d\n", &month);

printw("Enter the due date of new assignment : ");

scanw("%d\n", &date);

ASSIGN \*aptr = login\_st->Child\_A;

for(int i = 0; i < login\_st->Assign\_Size; i++){

aptr = aptr->link;

}

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

aptr->link = new;

create\_assign(new, name, describe, professor, month, date);

login\_st->Assign\_Size++;

printw("\nNew Assignment is successfully added!\n");

getch();

}

void Delete\_Assign(STUDENT \*login\_st) {

char name[100];

echo();

printw("Input the name of assignment you want to delete :\n");

getstr(name);

ASSIGN \*aptr = login\_st->Child\_A->link;

ASSIGN \*before = login\_st->Child\_A;

for(int i = 0; i < login\_st->Assign\_Size; i++){

if(strcmp(aptr->name, name) == 0){

break;

}

aptr = aptr->link;

before = before->link;

}

if(aptr == NULL){

printw("\nYou don't have assignment named '%s'!\n", name);

getch();

return ;

}

else if(strcmp(aptr->name, name) != 0){

printw("\nYou don't have assignment named '%s'!\n", name);

getch();

return ;

}

else{

before->link = aptr->link;

free(aptr);

login\_st->Assign\_Size--;

}

}

int New\_Account() {

char year[5], number[5], password[16];

clear();

printw("Student Number : ");

echo();

scanw("%4s%4s\n", year, number);

printw("Password : ");

noecho();

scanw("%s",password);

if(search\_year(year) != TOP->Year\_Size){

if(search\_num(number, search\_year(year)) != NULL){

return NEW\_ACCOUNT\_FAIL;

}

}

create\_student(year, number, password, 0, 0);

return NEW\_ACCOUNT\_SUCCESS;

}

int Delete\_Account() {

int account\_year;

STUDENT \*account;

char year[5], number[5], password[16];

clear();

printw("Student Number : ");

echo();

scanw("%4s%4s\n", year, number);

printw("Password : ");

noecho();

scanw("%s",password);

account\_year = search\_year(year);

account = search\_num(number, account\_year);

if(account == NULL){

return DELETE\_ACCOUNT\_FAIL\_INVALID;

}

if(strcmp(account->password, password) != 0){

return DELETE\_ACCOUNT\_FAIL\_PASSWORD;

}

STUDENT \*sptr = TOP->ST\_YEAR[account\_year].ST\_NUM;

for(int i = 0; i < TOP->ST\_YEAR[account\_year].Num\_Size; i++){

if( sptr->link == account ){

break;

}

sptr = sptr->link;

}

sptr->link = account->link;

free\_student(account);

TOP->ST\_YEAR[account\_year].Num\_Size--;

return DELETE\_ACCOUNT\_SUCCESS;

}

void free\_struct () {

for(int i = 0; i < TOP->Year\_Size; i++){

if(TOP->ST\_YEAR[i].Num\_Size != 0 ){

free\_year(TOP->ST\_YEAR[i].ST\_NUM->link);

}

free(TOP->ST\_YEAR[i].ST\_NUM);

}

if(TOP->Year\_Size != 0){

free(TOP->ST\_YEAR);

}

free(TOP);

}

void free\_year (STUDENT\* sptr) {

if(sptr->link != NULL){

free\_year(sptr->link);

}

free\_student(sptr);

free(sptr);

}

void free\_student (STUDENT \*account) {

ASSIGN \*head\_as = account->Child\_A;

CGPA \*head\_cg = account->Child\_C;

free\_assign(head\_as);

free\_cgpa(head\_cg);

}

void free\_assign (ASSIGN \*aptr){

if(aptr->link != NULL){

free\_assign(aptr->link);

}

free(aptr);

}

void free\_cgpa (CGPA \*cptr){

if(cptr->link != NULL){

free\_cgpa(cptr->link);

}

free(cptr);

}

int Change\_Password(STUDENT \*login\_st) {

char password[16], confirm\_password[16];

clear();

printw("Enter new password : ");

noecho();

scanw("%s",password);

printw("Confirm new password : ");

noecho();

scanw("%s",confirm\_password);

if(strcmp(password, confirm\_password) == 0){

strcpy(login\_st->password, password);

return CHANGE\_PASSWORD\_SUCCESS;

}

else{

return CHANGE\_PASSWORD\_FAIL;

}

}

int login() {

STUDENT \*login\_st;

char year[5] = {};

char number[5] = {};

char password[16] = {};

clear();

printw("Student Number : ");

echo();

scanw("%4s%4s\n", year, number);

printw("Password : ");

noecho();

scanw("%s",password);

Login\_Year = search\_year(year);

if(Login\_Year == TOP->Year\_Size){

return LOGIN\_FAILED;

}

login\_st = search\_num(number, Login\_Year);

if(login\_st == NULL){

return LOGIN\_FAILED;

}

if(strcmp(login\_st->password, password) == 0){

strcpy(Curr\_Num, year);

strcat(Curr\_Num, number);

return LOGIN\_SCCUESS;

}

return LOGIN\_FAILED;

}

void Temp\_Password(){

char year[5], number[5];

char temppassword[8];

float temp;

int current\_yr;

STUDENT \*current\_st;

clear();

echo();

srand(time(NULL));

printw("Student Number : ");

scanw("%4s%4s",year, number);

current\_yr = search\_year(year);

current\_st = search\_num(number, current\_yr);

if(current\_st == NULL){

printw("Failed to get temporary password. (Wrong student number)\n");

getch();

return ;

}

if(current\_st->Child\_C->link == NULL){

printw("Failed to get temporary password. (No CGPA Information)\n");

getch();

return ;

}

printw("Input latest semester's GPA : ");

scanw("%f", &temp);

CGPA \*cptr = current\_st->Child\_C;

for(int i = 0; i < current\_st->CGPA\_Size; i++){

cptr = cptr->link;

}

if(cptr->score != temp){

printw("Failed to get temporary password. (Wrong GPA)\n",current\_st->password);

getch();

return ;

}

for(int i = 0; i < 8; i++){

temppassword[i] = (rand()%93 + 33);

}

strcpy(current\_st->password,temppassword);

printw("Temporary Password is <%s>\n",current\_st->password);

getch();

return ;

}