C프로그래밍및실습

대학생활 도우미 프로그램

진척 보고서 #2

제출일자: 2023-12-10
제출자명: 최지원
제출자학번: 224571

1. 프로젝트 목표

1) 배경 및 필요성

학교에 막 입학한 학생이나 복학한 학생들은 학교 생활에 적응하기 힘들 수도 있다. 수업 장소는 어딘지 정확한 수업 시간은 언제인지 쉬는 시간은 몇 분인지 정확하게 파악하기 힘든 경우가 많다. 또 학점을 관리하고 해야 할 일이 많은 경우 기억하지 못하는 경우도 있다. 이 모든 걸 관리해주는 프로그램이 있으면 학교 생활에 더 도움이 될 것이라고 생각한다.

2) 프로젝트 목표

수업 장소와 시간, 교수님을 알려주고 목표학점을 입력하면 그에 맞게 학점을 분석해주고 투두리스트를 만들어 해야할 일의 마감기한을 관리할 수 있는 학교생활 도우미 프로그램

3) 차별점

기존에 있던 일반적인 대학생 앱과 같은 서비스와는 개인의 학교생활 관리에 더 중심적인 프로그램임. 더 직관적이게 원하는 항목을 선택 가능하게 하여 사용자가 더 쉽게 프로그램을 이용할 수 있음. 해야할 일 마감기한과 학점 분석과 같은 기능을 추가하여 기존에 앱과는 차별화된 기능이 있음.

2. 기능 계획

1) 전체 틀 구성

프로그램의 전체 흐름과 틀을 구성

2) 시간표 정리

- 현재 시간표를 정리하는 기능

(1) 사용자가 수업 시간, 장소, 교수님을 입력함

(2) 사용자 입력 후 현재 시간표를 정리해서 보여준다

- 입력이 끝난 후 수업 시간, 수업 장소, 교수님 정보를 확인 가능함

(3) 시간표의 수업 수정, 삭제, 추가 기능

3) 성적 분석

- 목표 성적과 현재 각 과목의 성적을 입력하면 목표성적까지의 성적을 분석

(1) 목표 성적 입력

(2) 사용자의 과목별 성적 입력

(3) 사용자의 성적 평균 분석

(4) 목표 성적까지 얼마나 성적을 올려야 하는지 알려줌

4) 투두리스트 관리&디데이 기능

- 해야할 일의 마감 디데이와 해야할 일 목록을 확인하는 기능

(1) 할 일 추가, 수정, 삭제 기능

(2) 현재 컴퓨터 시간을 기반으로 디데이 계산

- 현재 컴퓨터 시간과 입력한 시간정보를 바탕으로 디데이를 계산함

(3) 현재 해야 일 과제 목록 나열

- 디데이가 끝난 할 일들은 자동으로 삭제됨

3. 진척사항

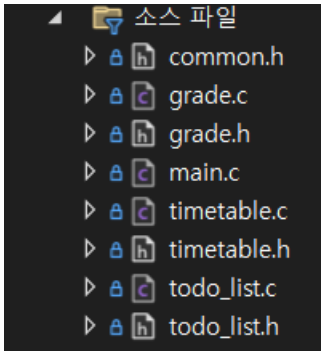
1) 기능 구현

(1) 헤더파일과 소스코드 분리

- 설명: 조금 더 가독성 있고 코드의 유지보수를 편리하게 하기 위해서 기능별로 헤더파일을 나눠 분리하였습니다. 시간표 관리, 성적 관리, 해야할 일 관리 총 3가지로 나누었습니다. common.h 에는 공통으로 사용하는 구조체, 상수의 정보들이 담겨있습니다.

- 적용된 배운 내용: 헤더파일

- 스크린샷

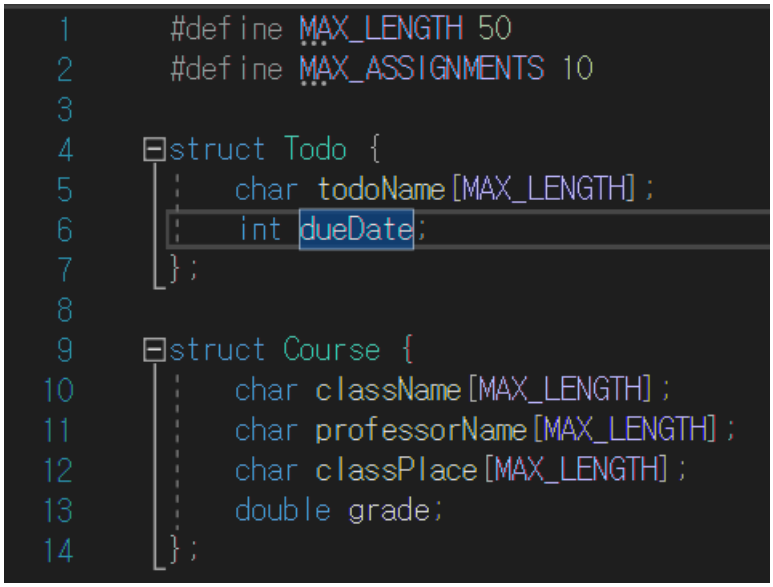


(2) common.h

- 설명: 기존에 main문에 다 들어있던 구조체와 상수 정보들을 "common.h" 파일에 모아두어 조금 더 관리가 쉽게 만들었습니다.

- 적용된 배운 내용: 구조체

- 코드 스크린샷



(3). 모든 기능 함수화

1) 시간표 관리 프로그램 함수화

- 입출력

  - 입력: 수업 과목명, 교수명, 수업 장소와 총 듣는 수업의 개수
  - 출력: 수업 시간표

- 설명: 사용자의 수업 시간표를 입력하고 입력한 시간표를 보여주는 기능

- 적용된 배운 내용: if문, 포인터, 동적 배열, 함수

- 코드 스크린샷

```
//시간표 관리 함수
void manageTimetable(struct Course** timetable, int* hakjum) {

    // 시간표 출력
    displayTimetable(*timetable, *hakjum);

    // 시간표 수정
    printf(_Format: "₩n1. 수업 추가₩n2. 수업 삭제₩n3. 이전으로₩n");
    printf(_Format: "원하는 번호를 입력하세요.₩n");

    int option = 0;
    scanf_s(_Format: "%d", &option);

    if (option == 1) {
        // 수업 추가
        *hakjum=addCourse(*timetable, *hakjum);
        // 수업 추가 후 시간표 출력
        displayTimetable(*timetable, *hakjum);
    }
    else if (option == 2) {
        // 수업 삭제
        *hakjum=deleteCourse(*timetable, *hakjum);

        //수업 삭제 후 시간표 출력
        displayTimetable(*timetable, *hakjum);
    }
    else if (option == 3) {
        // 다시 메인메뉴 보이게 함
    }
    else {
        printf(_Format: "잘못된 번호입니다.₩n");
    }
}
```

2) 수업 추가 함수화

- 입출력

  - 입력: 수업 추가
  - 출력: 수업이 추가 됐다는 메시지

- 설명: 처음에 시간표를 입력하고 그 이후에 수업을 추가할 수 있는 기능

- 적용된 배운 내용: 배열, if문, 함수

- 코드 스크린샷

```
// 수업 추가 함수
int addCourse(struct Course* timetable, int hakjum) {
    printf("추가할 수업의 수업명, 교수명, 장소명을 차례대로 입력하세요. (띄어쓰기로 구분)₩n");
    scanf_s("%s %s %s", timetable[hakjum].className, MAX_LENGTH, timetable[hakjum].professorName, MAX_LENGTH, timetable[hakjum].classPlace, MAX_LENGTH);

    hakjum++;
    printf("수업이 추가되었습니다.₩n");

    return hakjum;
}
```

3) 수업 삭제 함수화

- 입출력

  - 입력: 수업 삭제
  - 출력: 수업이 삭제됐다는 메시지

- 설명: 처음에 시간표를 입력하고 그 이후에 수업을 삭제할 수 있는 기능

- 적용된 배운 내용: 반복문, 배열, if문, 함수

- 코드 스크린샷

```
// 수업 삭제 함수
int deleteCourse(struct Course* timetable, int hakjum) {
    if (hakjum > 0) {
        printf("삭제할 수업의 번호를 입력하세요 (1부터 시작)₩n");
        int courseNumber = 0;
        scanf_s("%d", &courseNumber);

        if (courseNumber >= 1 && courseNumber <= hakjum) {
            for (int i = courseNumber - 1; i < hakjum - 1; i++) {
                timetable[i] = timetable[i + 1];
            }
            hakjum--;
            printf("수업이 삭제되었습니다.₩n");
        }
        else {
            printf("잘못된 번호입니다.₩n");
        }
    }
    else {
        printf("현재 수업이 없어 삭제할 수 없습니다.₩n");
    }

    return hakjum;
}
```

4) 성적 관리 프로그램 함수화

- 입출력

- 입력: 성적 관리 프로그램에서 원하는 항목 번호
- 출력: 사용자가 입력한 항목 번호에 따른 기능

- 설명: 성적관리 프로그램의 전체적인 틀, 목표 성적을 기존에 입력 했다면 목표 성적이 처음에 뜨도록 함. 사용자는 성적관리 프로그램에서 원하는 항목을 선택할 수 있고 볼 수 있음.

- 적용된 배운 내용: if문, 함수, while문, 포인터

- 코드 스크린샷

```
//성적 관리 프로그램 함수
void manageGrades(struct Course* timetable, int hakjum, double* goalGrade,int check) {
    int setGrade = (*goalGrade == 0.0) ? 0 : 1;
    int option = 0;


    while (1) {
        printf(_Format:"₩n₩n성적관리 프로그램₩n--------------------₩n");
        printf(_Format:"1. 목표성적 입력 및 수정₩n2. 각 수업 별 성적 입력 및 수정₩n3. 성적분석₩n4. 이전으로₩n");
        scanf_s(_Format:"%d", &option);

        switch (option) {
        //목표 설정
        case 1:
            setGoalGrade(goalGrade);
            setGrade = 1;
            break;
        //과목별 성적 추가 및 수정
        case 2:
            if (setGrade) {
                if (!check) {
                    setSubjectGrades(timetable, hakjum);
                    check = 1;
                }
                else {
                    modifySubjectGrade(timetable, hakjum);
                }
                displayGradeTimetable(timetable, hakjum);
```

```
            }
            displayGradeTimetable(timetable, hakjum);
        }
        else {
            printf("목표 성적을 먼저 입력하세요.₩n");
        }
        break;
    //성적 분석
    case 3:
        if (setGrade) {
            analyzeGrade(timetable, hakjum, *goalGrade);
        }
        else {
            printf("목표 성적을 먼저 입력하세요.₩n");
        }
        break;

    case 4:
        return;

    default:
        printf("잘못된 번호입니다.₩n");
        break;
    }
  }
}
```

5) 목표 성적 입력 및 수정 함수화

- 입출력

  - 입력: 사용자의 목표 성적
  - 출력: 사용자가 입력한 목표 성적

- 사용자가 목표 성적을 입력하고 목표 성적을 입력한 후에는 수정할 수 있는 기능

- 적용된 배운 내용: if문, 포인터, 함수

-코드스크린샷

```
//목표 성적 입력 및 수정
void setGoalGrade(double* goalGrade) {
    if (*goalGrade == 0.0) {
        printf("목표 성적을 입력하세요: ");
        scanf_s("%lf", goalGrade);
        printf("당신의 목표 성적은 %.2lf 입니다.", *goalGrade);
    }
    else {
        printf("수정할 목표 성적을 입력하세요.₩n");
        scanf_s("%lf", goalGrade);
        printf("수정한 당신의 목표 성적은 %.2lf입니다.", *goalGrade);
    }
}
```

6) 과목 별 성적 입력 함수화

- 입출력

  - 입력: 사용자의 목표 성적
  - 출력: 사용자가 입력한 목표 성적

- 사용자가 수강하고 있는 수업 별 성적을 입력할 수 있는 기능.

- 적용된 배운 내용: if문, 포인터, 함수, 반복문, 배열

- 코드 스크린샷

```
//과목별 성적 입력
void setSubjectGrades(struct Course* timetable, int hakjum) {
    if (hakjum > 0) {
        for (int i = 0; i < hakjum; i++) {
            printf(_Format: "%s 과목의 성적을 입력하세요: ₩n", timetable[i].className);
            scanf_s(_Format: "%lf", &timetable[i].grade);
        }
    }
    else {
        printf(_Format: "현재 수업이 없어 성적을 입력할 수 없습니다.₩n");
    }
}
```

7) 과목 별 성적 수정 함수화

- 입출력

  - 입력: 수정할 성적
  - 출력: 수정이 됐다는 메시지

- 설명: 처음 과목 별 성적을 입력한 후에 성적을 수정할 수 있는 함수

- 적용된 배운 내용: 반복문, 배열, if문, 함수

- 코드 스크린샷

```c
//과목 별 성적 수정
void modifySubjectGrade(struct Course* timetable, int hakjum) {
    if (hakjum > 0) {
        int subNum = 0;
        printf(_Format:"성적을 수정할 과목의 번호를 입력하세요.₩n");
        for (int i = 0; i < hakjum; i++) {
            printf(_Format:"%d. %s₩n", i + 1, timetable[i].className);
        }
        scanf_s(_Format:"%d", &subNum);

        if (subNum >= 1 && subNum <= hakjum) {
            printf(_Format:"%s 과목의 성적을 수정하세요.₩n", timetable[subNum - 1].className);
            scanf_s(_Format:"%lf", &timetable[subNum - 1].grade);
            printf(_Format:"성적이 수정 되었습니다.₩n");
        }
        else {
            printf(_Format:"잘못된 번호입니다.₩n");
        }
    }
    else {
        printf(_Format:"현재 수업이 없어 성적을 수정할 수 없습니다.₩n");
    }
}
```

8) 성적 분석 함수화

- 입출력

  - 입력: 사용자가 기존에 입력했던 목표 성적과 과목 별 성적
  - 출력: 목표 성적과 사용자의 성적 비교

- 목표 성적과 현재 사용자의 성적을 비교하여 알려준다.

- 적용된 배운 내용: if문, 포인터, 배열

```c
//성적 분석
void analyzeGrade(struct Course* timetable, int hakjum, double goalGrade) {
    printf(_Format:"목표 성적: %.2lf₩n", goalGrade);

    double currentGrade = 0.0;
    double sum = 0.0;

    for (int i = 0; i < hakjum; i++) {
        sum += timetable[i].grade;
    }

    currentGrade = sum / hakjum;

    if (sum != 0.0) {
        printf(_Format:"현재 당신의 성적: %.2lf₩n", currentGrade);

        if (goalGrade > currentGrade) {
            printf(_Format:"목표 성적까지 %.2lf만큼 남았습니다.₩n", goalGrade - currentGrade);
            printf(_Format:"조금 더 열심히 공부하세요!₩n");
        }
        else if (goalGrade < currentGrade) {
            printf(_Format:"목표 성적보다 %.2lf만큼 높습니다.₩n", currentGrade - goalGrade);
            printf(_Format:"멋집니다!₩n");
        }
        else {
            printf(_Format:"목표성적과 현재 성적이 같습니다.₩n");
        }
    }
    else {
        printf(_Format:"현재 성적을 입력해주세요.₩n");
    }
```

(4) 투두 리스트 틀 구성

- 입출력

  - 입력: 해야할 일
  - 출력: 해야할 일 목록

- 설명: 해야 할 일을 추가/삭제할 수 있는 함수.

- 적용된 배운 내용: do-while문, switch문, 함수

- 코드 스크린샷

```c
void manageTodoList(struct Todo** todoList) {

    int index = 0;

    printf(_Format: "₩n투두 리스트 입니다.₩n");
    printf(_Format: "--------------------------------₩n");
    printf(_Format: "해야할 일을 추가/삭제 하세요.₩n");

    int menu=0;

    do {
        printf(_Format: "₩n--------------------------------₩n");
        printf(_Format: "1. 해야할 일 추가하기₩n");
        printf(_Format: "2. 해야할 일 삭제하기₩n");
        printf(_Format: "3. 이전으로₩n");
        scanf_s(_Format: "%d", &menu);


        switch (menu) {
        case 1:
            // 해야할 일 추가


        case 2:
            // 해야할 일 삭제

        }

        displayTodo();
    } while (menu != 3);


}
```

2) 테스트 결과

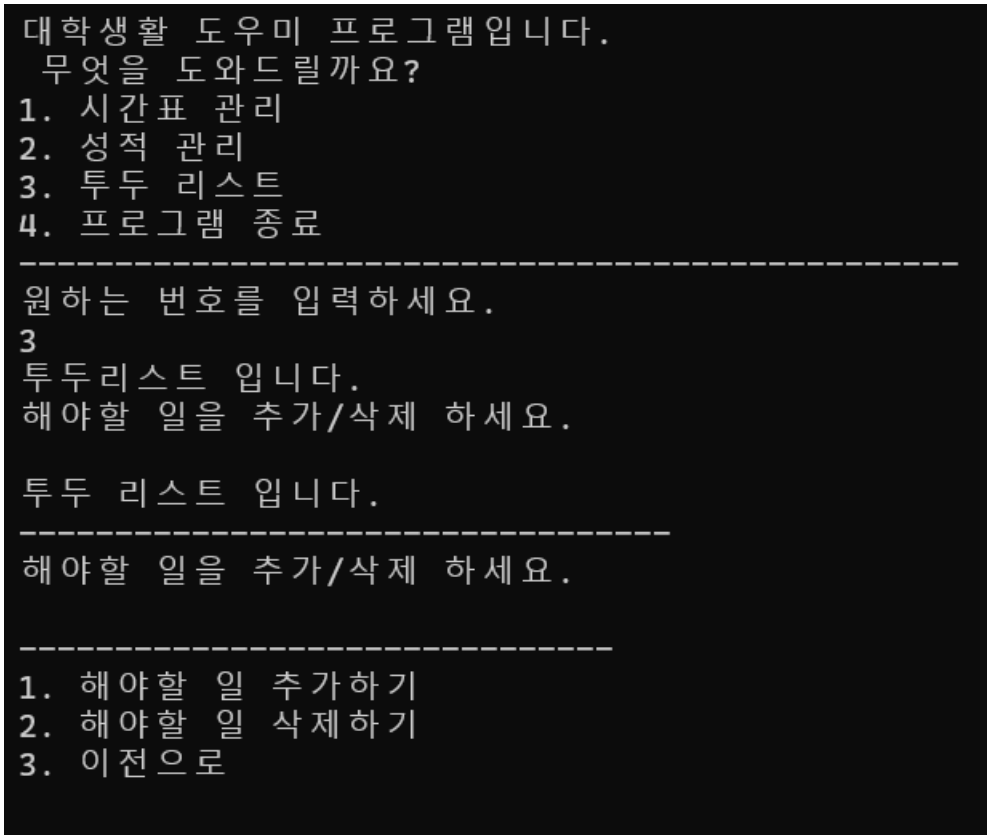
(1)~(3). 헤더파일 변경과 함수화

- 헤더파일로 코드 분리, 기능 별 함수화로 테스트 결과는 기존과 같음

(4). 투두리스트

- 투두리스트 메뉴로 들어가지고 해야할 일을 추가/삭제 할수 있다.

- 테스트 결과 스크린샷



4. 계획 대비 변경 사항

1) 과제 관리 -> 투두리스트

- 이전: 기존에는 수업 과목 별 과제를 입력할 수 있게 하여 조금 더 사용자의 편의를 주었음. 수업이 삭제되면 그 수업안에 있는 과제들도 다 삭제가 될 수 있도록 구조체로 연결 할 의도였음.

- 이후: 과제관리에서 투두리스트로 기능 변경.

- 사유: 과제 관리 시 과제 구조체 배열을 동적 할당하는데 이 경우 구조체의 크기가 처음부터 정해지는게 아니고 0부터 시작해서 과제가 계속 늘어나면 하나씩 늘어나는 구조인데 코드를 짜도 계속 과제명을 입력하는 부분만 가면 프로그램이 몇번 깜박이다가 종료되는 현상이 발생함. 또한 구조체 멤버변수인 과제의 수를 처음 0으로 초기화하는데 어려움을 겪음. 어쩔 수 없이 수업 별 과제 입력이 아닌 그냥 할 일을 입력할 수 있는

기능으로 변경함. 디데이 기능은 여전히 유지.

5. 프로젝트 일정

업무		11/3	11/10	11/17	11/26	12/1	12/8	12/15
제안서 작성		진행완료						
기능1			진행완료					
시간표 정리	시간표 입력			진행 완료				
	수업 추가			진행완료				
	수업 삭제			진행완료				
성적 관리	목표 성적				진행 완료			
	과목별 성적				진행 완료			
	성적 분석				진행 완료			
투두 리스트	할일 추가 /삭제					---------->		
	할일 수정						---------->	
	할일 목록						---------->	
	디데이 추가						---------->	