C++프로그래밍및실습

공유형 일정 관리 프로그램 개발

진척 보고서 #1

제출일자: 2023-11-24
제출자명: 김준석
제출자학번: 182671

1. 프로젝트 목표

1) 배경 및 필요성

SNS가 우리의 삶에서 차지하는 비중은 점차 높아지고 있다. 고객들은 자신의 일상을 주변인들에게 공유하고자 하는 욕구가 강하다. 이는 일정에 관해서도 예외가 아니다. 고객들은 자신의 일정을 남들에게 공유할 수 있는 프로그램을 원하고 있으며, 기존의 개인 일정 기록 뿐인 단순한 프로그램은 고객의 만족도를 저하시키고 이는 매출과 수익의 감소로 이어질 수 있다.

2) 프로젝트 목표

고객들이 그룹을 형성해 멤버들 간 일정을 공유하고, 상대의 일정을 열람하고 상대의 일정에 동참하는 등의 상호작용을 강화한 공유형 일정 관리 프로그램을 만드는 것을 목표로 한다.

3) 차별점

기존 프로그램들은 본인의 일정만을 관리할 뿐, 상대의 일정을 공유하고 동참하는 기능은 없었다. 일정을 공유하기 위해서는 일정표를 캡쳐하고 전송하는 등의 불편한 과정을 거쳐야만 했다. 우리는 그룹을 설정하고 그룹 내 멤버들 간 일정 열람을 자유롭게 하였다. 또한, 멤버의 일정에 동참해 멤버의 일정을 내 일정표에 추가하는 기능과 특정 날짜의 모든 멤버들의 일정을 출력하는 등의 기능들을 포함하여 일정 공유를 용이하게 할 뿐만 아니라 단순한 일정 관리 프로그램에서 SNS의 요소를 갖춘 프로그램으로 확장을 이뤄냈다는 데에 기존 프로그램과의 차별점이 있다. 또한, 다양한 조건의 검색과 루틴 일정 입력 기능을 가지고 있어 기존의 일정 관리 프로그램으로서의 기능 역시 강화하였다.

2. 기능 계획

1) 사용자 생성 및 전환

- 사용자를 생성하거나, 다른 사용자로 전환한다.

(1) 사용자 생성

- 프로그램 최초 실행 시, 사용자의 이름을 입력받아 사용자를 생성한다.

(1) 사용자 전환

- 사용자 목록에 사용자가 있는지 확인하고, 사용자를 전환한다. 사용자 목록에 없다면 신규 생성도 가능하다. 단, 이름은 중복되지 않아야 한다.

2) 일정 입력

- 고객에게 입력을 받아 일정표에 입력한 일정을 채워넣는다. 이 때, 입력은 파일을 통한 배치(batch) 입력을 지원한다. (기본은 수동으로 하나씩 입력하는 것이다.) 또한, 루틴한 일정의 자동 입력을 지원한다.

(1) 일정 수동 입력

- 일정 명과 일정의 날짜를 직접 입력하여, 하나씩 일정을 수동으로 추가한다.

(2) 루틴 일정 입력

- 일정 명을 포함해 루틴의 시작, 종료일과 간격, 반복 횟수를 입력받아 반복되는 루틴 일정을 한 번에 추가한다.

(3) 파일 배치 입력

- 파일의 경로를 입력해 텍스트 파일을 읽고, 텍스트 파일에 작성된 일정들을 한 번에 배치 입력한다.

3) 일정표 출력

- 고객이 설정한 대상과 날짜, 범위에 해당하는 일정들을 나타내는 일정표를 출력한다. 이 때, 대상은 본인 뿐만 아니라 같은 그룹 내의 다른 멤버들이 될 수도

있다. 또한, 월/주/일 또는 사용자 지정의 범위를 지원하며 표시 방법을 리스트형과 달력형을 선택할 수 있도록 한다.

(1) 일정 표시 대상 선택

- 일정 표시 대상을 선택한다. 본인 또는 멤버 중 한 명을 선택할 수 있다.

(2) 일정 표시 범위 선택

- 일정을 표시할 범위를 지정한다. 직접 지정할 수도, 월, 주, 일 단위의 선택도 가능하다.

(3) 일정 표시 형식 선택

- 일정 표시 형식을 선택한다. 리스트형과 달력형 중 하나를 선택한다. 리스트형은 한 줄씩 일정 날짜와 일정 명을 표시한다. 달력형은 선택한 달의 달력을 보여주며, 일정있는 날에 체크 표시를 하는 형태이다.

4) 그룹 관리

- 고객들은 그룹에 가입할 수 있다. 그룹 내 멤버들은 서로의 일정표를 공유한다. 그룹 내 멤버들을 조회할 수 있고, 가입과 탈퇴가 자유롭다.

5) 멤버의 일정에 동참

- 멤버의 일정을 조회하고 동참하고자 하는 일정을 선택해 나의 일정으로 추가할 수 있다.

6) 일정이 있는 멤버 조회

- 특정 일에 일정이 있는 멤버들을 알고 싶을 때, 일일이 멤버들의 일정표를 조회하는 것은 굉장히 번거로운 일이다. 따라서, 특정 일에 일정이 있는 멤버들을 조회하는 기능 역시 지원한다.

3. 진척사항

1) 기능 구현

(1) 사용자 생성

- 입출력

    A. 입력

        1. user_name : 생성할 사용자의 이름

    B. 출력

        1. 사용자 생성을 안내하는 문자열

        2. 이름 입력을 안내하는 문자열

        3. 생성 완료 안내 및 생성한 사용자의 이름을 포함한 문자열

- 설명

프로그램을 최초 실행한 경우, 사용자를 생성해야 한다. 이름을 문자열로 입력받고, 해당 문자열을 생성자의 인자로 해 User 객체를 생성한다. 생성된 객체는 current_user라는 User타입 포인터 변수에 저장된다. 또한, 생성된 사용자들의 목록을 저장하는 use_list 벡터에 해당 객체를 추가한다. 마지막으로, 생성한 사용자의 이름을 포함해 문자열을 출력하여, 생성이 완료되었음을 알린다.

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

    A. 클래스(User 클래스 객체 생성)

    B. 포인터(current_user 변수)

    C. 조건문

    D. 함수(GetUserName 호출)

    E. 벡터

- 코드 스크린샷

```
string input; // 입력된 명령어를 저장
User* current_user = nullptr; // 현재 사용중인 유저
vector<User*> user_list; // 생성된 유저들의 목록

while (true) {

    // 최초 실행 시 사용자 생성
    if (current_user == nullptr) {
        cout << "사용자를 생성합니다." << endl;
        string user_name; // 생성할 사용자 명
        cout << "이름을 입력하세요: ";
        cin >> user_name;
        current_user = new User(user_name);
        user_list.push_back(current_user);
        cout << "사용자가 생성되었습니다. ";
        cout << current_user->GetUserName() << "님, 환영합니다." << endl;
        // TODO: 그룹에 가입
    }
```

(2) 사용자 전환

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어

2. name : 전환할 사용자의 이름

3. add_user : 사용자 생성 여부(y/n)

B. 출력

1. 사용자 이름 입력을 위한 문자열

2. 입력한 이름과 일치하는 사용자가 없음을 안내하는 문자열

3. 사용자 생성 여부를 묻는 문자열

- 설명

사용자가 '사용자전환'을 명령어로 입력한 경우, 전환할 사용자의 이름을 입력받는다. 사용자 목록을 저장해둔 벡터 user_list에서 입력한 사용자의 이름과 일치하는 User 객체를 검색한다. 일치하는 객체가 존재한다면 current_user 변수가 가리키는 객체를 해당 객체로 바꾼다. 일치하는 객체가 존재하지 않는다면, 사용자 생성 여부를 묻고 입력한 이름으로 새로운 User 객체를 생성한다. 이후 current_user 변수를 해당 객체를 가리키도록 한 후 , user_list에 해당 객체를 추가한다.

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

    A. 클래스(User 클래스 객체 생성)

    B. 포인터(current_user 변수)

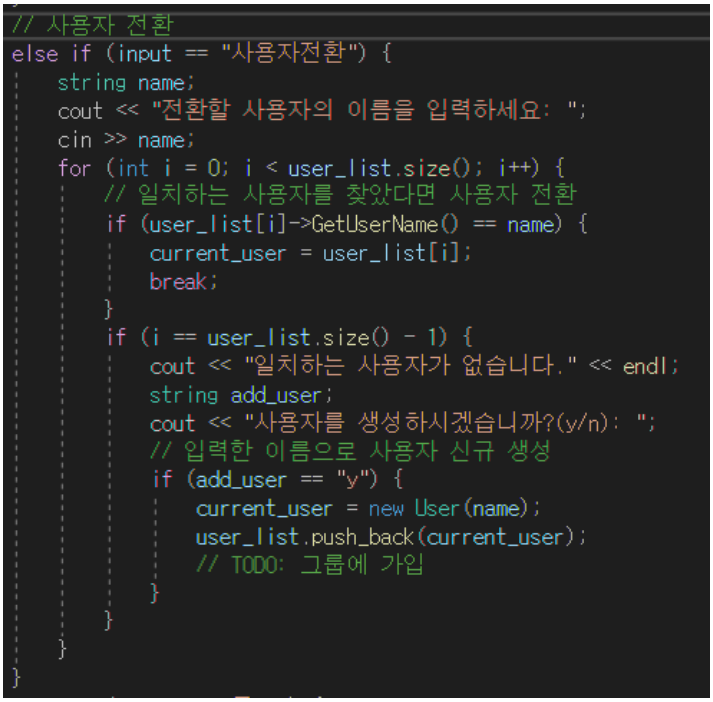
    C. 조건문

    D. 반복문

    E. 함수(GetUserName 호출)

    F. 벡터(User* 타입의 백터 user_list)

- 코드 스크린샷



```
// 사용자 전환
else if (input == "사용자전환") {
    string name;
    cout << "전환할 사용자의 이름을 입력하세요: ";
    cin >> name;
    for (int i = 0; i < user_list.size(); i++) {
        // 일치하는 사용자를 찾았다면 사용자 전환
        if (user_list[i]->GetUserName() == name) {
            current_user = user_list[i];
            break;
        }
        if (i == user_list.size() - 1) {
            cout << "일치하는 사용자가 없습니다." << endl;
            string add_user;
            cout << "사용자를 생성하시겠습니까?(y/n): ";
            // 입력한 이름으로 사용자 신규 생성
            if (add_user == "y") {
                current_user = new User(name);
                user_list.push_back(current_user);
                // TODO: 그룹에 가입
            }
        }
    }
}
```

(3) 일정 수동 입력

- 입출력

    A. 입력

        1. input : 사용자가 입력한 명령어

        2. option : 사용자가 선택한 일정 추가 옵션

        3. schd_name : 추가할 일정의 이름

4. schd_date : 추가할 일정의 날짜(yyyy-mm-dd-의 형태)

B. 출력

1. 옵션 입력을 안내하는 문자열

2. 일정 명 입력을 안내하는 문자열

3. 일정 날짜 입력을 안내하는 문자열

4. 일정이 추가되었음을 안내하는 문자열

- 설명

사용자가 '일정추가'를 명령어로 입력한 경우, 옵션을 입력하도록 한다. 옵션을 '수동'으로 입력했다면, 수동 일정 입력이 시작된다. 사용자로부터 일정 명과 날짜를 입력받는다. 입력받은 일정 명과 날짜를 인자로 현재 사용자 User객체의 멤버 함수인 AddSchedule 함수를 호출한다. 해당 함수는 내부에서 GetUserName 함수를 호출해 User 객체의 이름을 반환받고, 입력한 일정 날짜를 나타내는 문자열을 인자로 Date 객체를 생성한다. 마지막으로, 객체의 이름과 일정의 이름, 그리고 생성한 Date 객체를 인자로 Schedule 객체를 생성해 User 객체의 일정을 저장하는 멤버 변수인 user_schd_list 객체에 추가한다.

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

A. 클래스(Date, Schedule 클래스 객체 생성)

B. 포인터(current_user 변수)

C. 조건문

D. 함수(AddSchedule, GetUserName 호출)

E. 벡터(Schedule* 타입의 백터 user_schd_list)

- 코드 스크린샷

```
// 일정을 수동 입력하는 함수
void User::AddSchedule(string schd_name, string schd_date) {

    // Schedule 객체 생성 후 일정 목록에 추가
    string user_name = this->GetUserName();

    Date date = Date(schd_date);

    Schedule* schd = new Schedule(user_name, schd_name, date);
    this->user_schd_list.push_back(schd);
}
```

```
// 일정 추가
if (input == "일정추가") {
    string option;
    while (true) {
        cout << "일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): ";
        cin >> option;
        // 수동 입력
        if (option == "수동") {
            string schd_name;
            string schd_date;
            cout << "수동으로 일정을 추가합니다." << endl;
            cout << "일정 명: ";
            cin >> schd_name;
            cout << "일정 날짜(yyyy-mm-dd): ";
            cin >> schd_date;

            current_user->AddSchedule(schd_name, schd_date);
            cout << "일정이 추가되었습니다!" << endl;

            break;
        }
```

(4) 루틴 일정 입력

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어

2. option : 사용자가 선택한 일정 추가 옵션

3. schd_name : 추가할 루틴 일정의 이름

4. start_date : 추가할 루틴 일정의 시작 날짜(yyyy-mm-dd-의 형태)

5. interval : 루틴 일정의 날짜 간격

6. count : 루틴 반복 횟수

B. 출력

1. 옵션 입력을 안내하는 문자열
2. 일정 명 입력을 안내하는 문자열
3. 루틴 시작 날짜 입력을 안내하는 문자열
4. 루틴 날짜 간격 입력을 안내하는 문자열
5. 루틴 반복 횟수 입력을 안내하는 문자열
6. 일정이 추가되었음을 안내하는 문자열

- 설명

사용자가 '일정추가'를 명령어로 입력한 경우, 옵션을 입력하도록 한다. 옵션을 '루틴'으로 입력했다면, 루틴 일정 입력이 시작된다. 사용자로부터 일정 명, 시작 날짜, 날짜 간격, 반복 횟수를 입력받는다. 입력받은 일정 명, 시작 날짜, 날짜 간격, 반복횟수를 인자로 현재 사용자 User객체의 멤버 함수인 AddRoutineSchedule 함수를 호출한다. 해당 함수는 내부에서 GetUserName 함수를 호출해 User 객체의 이름을 반환받는다. 또한, Date 포인터 타입의 벡터인 date_list를 선언하고, 루틴 시작일을 포함해 시작일로부터 날짜 간격 만큼 떨어진 날짜들을 반복 횟수만큼 Date 객체로 생성한다. 이 때, Date 객체의 Arrange 함수를 호출하여, 해당 월의 일 또는 해당 연의 월을 초과한 경우에 대해 다음 월 또는 연으로 이동시켜주는 과정을 거친다. 이렇게 생성된 date_list의 요소들을 인자로 Schedule 객체를 생성해 user_schd_list에 추가한다.

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

A. 클래스(Date, Schedule 클래스 객체 생성)

B. 포인터(current_user, start, date 변수)

C. 조건문

D. 반복문

E. 함수(AddRoutineSchedule, GetUserName, Arrange 호출)

F. 벡터(Schedule* 타입의 백터 user_schd_list, Date* 타입의 date_list 벡터)

- 코드 스크린샷

```
// 루틴 입력
else if (option == "루틴") {
    string schd_name;
    string start_date;
    int interval;
    int count;
    cout << "루틴 일정을 추가합니다." << endl;
    cout << "일정 명: ";
    cin >> schd_name;
    cout << "루틴 시작 날짜(yyyy-mm-dd): ";
    cin >> start_date;
    cout << "루틴 날짜 간격: ";
    cin >> interval;
    cout << "반복 횟수: ";
    cin >> count;

    current_user->AddRoutineSchedule(schd_name, start_date, interval, count);
    cout << "루틴 일정이 추가되었습니다!" << endl;

    break;
}
```

```
// 루틴 일정을 입력하는 함수
void User::AddRoutineSchedule(string schd_name, string start_date, int interval, int count) {

    string user_name = this->GetUserName();

    vector<Date*> date_list; // 루틴 일정들의 날짜 목록

    // 루틴이 시작되는 날
    Date* start = new Date(start_date);
    date_list.push_back(start);

    // 반복 횟수만큼 간격을 적용해 날짜 목록에 추가
    for (int i = 0; i < count - 1; i++) {
        start = date_list.back();
        Date* date = new Date(start->year, start->month, start->day + interval);
        date->Arrange();
        date_list.push_back(date);
    }

    // 날짜 목록을 참조해 일정을 생성해 일정 목록에 추가
    for (int i = 0; i < date_list.size(); i++) {
        this->user_schd_list.push_back(new Schedule(user_name, schd_name, *date_list[i]));
    }
}
```

```cpp
// 초과한 날짜를 정리해 다음 연/월로 넘겨주는 함수
void Date::Arrange() {

    int days_in_month = 0; // 해당 월의 일 수

    // month에 해당하는 일 수 설정
    switch (this->month) {
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
        days_in_month = 31;
        break;
    case 4: case 6: case 9: case 11:
        days_in_month = 30;
        break;
    case 2: // 윤년은 고려하지 않음.
        days_in_month = 28;
        break;
    default:
        break;
    }

    // 날짜 정리
    if (this->day > days_in_month) {
        this->month++;
        this->day -= days_in_month;
    }
    if (this->month > 12) {
        this->month -= 12;
        this->year++;
    }
}
```

(5) 파일 배치 입력

- 입출력

    A. 입력

        1. input : 사용자가 입력한 명령어

        2. option : 사용자가 선택한 일정 추가 옵션

        3. route : 입력할 파일의 경로

    B. 출력

        1. 옵션 입력을 안내하는 문자열

        2. 파일 경로 입력을 안내하는 문자열

        3. 일정이 추가되었음을 안내하는 문자열

- 설명

사용자가 '일정추가'를 명령어로 입력한 경우, 옵션을 입력하도록 한다. 옵션을 '파일'로 입력했다면, 파일 배치 입력이 시작된다. 사용자로부터 파일 경로를 입력받는다. 입력한 파일 경로를 인자로 User 객체의 멤버 함수인 AddBatchSchedule 함수를 호출한다. 함수 내부에서 파일로부터 두 줄씩 입력받는다. 각각의 줄은 일정 명과 일정 날짜이다. 이들을 인자로 Schedule 객체를 생성하고 이를 파일 스트림이 끝날 때까지 반복한다. 또한, UTF-8 인코딩 방식을 가정하기 때문에 변환 과정을 포함한다.

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

    A. 클래스(Date, Schedule 클래스 객체 생성)

    B. 포인터(current_user, schedule변수)

    C. 조건문

    D. 반복문

    E. 함수(AddBatchSchedule, GetUserName 호출)

    F. 벡터(Schedule* 타입의 백터 user_schd_list)

- 코드 스크린샷

```
// 배치 입력
else if (option == "파일") {
    string route;

    cout << "파일을 통해 일정을 추가합니다." << endl;
    cout << "파열 경로: ";
    cin >> route;

    current_user->AddBatchSchedule(route);
    cout << "일정이 추가되었습니다!" << endl;

    break;
}
```

```cpp
// 파일을 통한 배치 일정 입력 함수
/* 파일의 형식 -> 두 라인에 하나의 일정을 갖는 .txt 파일
   각 라인(2줄)의 형식 -> schedule_name
                          yyyy-mm-dd  */
void User::AddBatchSchedule(string route) {

    // 경로 route의 파일 읽기
    wifstream ifs;
    ifs.imbue(locale(locale::empty(), new codecvt_utf8<wchar_t>));

    ifs.open(route);

    // 파일에서 일정 명과 날짜를 읽어와 일정 생성 및 목록에 추가
    wstring schd_name;
    wstring schd_date;

    while (!ifs.eof()) {
        getline(ifs, schd_name);
        getline(ifs, schd_date);

        // wstring을 string으로 변환
        wstring_convert<codecvt_utf8<wchar_t>, wchar_t> converter;
        string name_utf8 = converter.to_bytes(schd_name);
        string date_utf8 = converter.to_bytes(schd_date);

        Date date(date_utf8);
        Schedule* schedule = new Schedule(this->GetUserName(), name_utf8, date);
        this->user_schd_list.push_back(schedule);
    }
}
```

(6) 일정 표시 대상 선택 – 나 / 일정 표시 범위 선택 – 사용자 지정 / 일정 표시 형식 선택 – 리스트형

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어

2. option : 사용자가 선택한 일정 표시 대상 옵션

3. start : 일정을 표시할 시작일

4. end : 일정을 표시할 종료일

B. 출력

1. 일정 표시 대상 선택을 안내하는 문자열

2. 일정 표시 시작일 입력을 안내하는 문자열

3. 일정 표시 종료일 입력을 안내하는 문자열

4. 일정 출력을 안내하는 문자열

5. 선택한 대상/범위에 해당하는 일정들

- 설명

사용자가 '일정보기'를 명령어로 입력한 경우, 일정 표시 대상을 입력하도록 한다. 대상을 '나'로 입력했다면, 일정을 표시할 날짜 범위를 입력하도록 한다. 시작일과 종료일을 입력받았다면, 해당 일들을 인자로 Date 객체를 생성한다. 현재 유저를 가리키는 current_user와 시작일, 종료일로 생성한 Date 객체인 start, end를 인자로 User 객체의 멤버 함수인 PrintScheduleToList 함수를 호출한다. 해당 함수의 내부에서 SortSchedule 함수가 호출된다. SortSchedule 함수는 User 객체의 user_schd_list에 저장된 Schedule 객체들을 빠른 날짜 순으로 정렬해준다. (이를 위해 Schedule과 Date 클래스의 비교 연산자 오버라이딩을 선행했다.) 정렬 후 user_schd_list의 요소들을 start, end와의 비교를 통해 사용자가 지정한 날짜 범위에 해당하는 일정들만을 출력한다. 일정들이 날짜 순으로 정렬되었으므로, 벡터 전체를 탐색할 필요가 없어, 검색 성능이 개선된다.

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

A. 클래스(Date, Schedule 클래스)

B. 포인터(current_user)

C. 조건문

D. 반복문

E. 함수(PrintScheduleToList, SortSchedule 호출)

F. 벡터(Schedule* 타입의 백터 user_schd_list)

- 코드 스크린샷

```
// 일정 보기
else if (input == "일정보기") {
    string option;
    while (true) {
        cout << "누구의 일정을 보시겠습니까?(나|멤버): ";
        cin >> option;
        if (option == "나") {
            string start;
            string end;
            cout << "일정을 표시할 시작일을 입력하세요(yyyy-mm-dd): ";
            cin >> start;
            cout << "일정을 표시할 종료일을 입력하세요(yyyy-mm-dd): ";
            cin >> end;

            Date start_date(start);
            Date end_date(end);

            cout << "일정을 출력합니다." << endl;
            current_user->PrintScheduleToList(*current_user, start, end);

            break;
        }
        else if (option == "멤버") {
            // TODO: 멤버를 선택하고 그룹에 있는지 체크 후 일정을 보여주는 로직
        }
        else {
            cout << "잘못된 옵션입니다. 다시 입력해주세요." << endl;
            continue;
        }
    }

}
```

```
// 일정을 리스트 형태로 출력하는 함수
void User::PrintScheduleToList(User &user, Date start, Date end) {

    // 일정을 빠른 순으로 정렬
    user.SortSchedule();

    // start부터 end까지의 날짜에 존재하는 모든 일정을 출력
    for (int i = 0; i < user.user_schd_list.size(); i++) {
        if (user.user_schd_list[i]->schd_date > start || user.user_schd_list[i]->schd_date == start) {
            if (user.user_schd_list[i]->schd_date > end) {
                break;
            }

            int year = user.user_schd_list[i]->schd_date.year;
            int month = user.user_schd_list[i]->schd_date.month;
            int day = user.user_schd_list[i]->schd_date.day;
            string schd_name = user.user_schd_list[i]->schd_name;

            cout << year << "-" << month << "-" << day << " " << schd_name << endl;
        }
    }
}
```

```
// 유저의 일정을 빠른 날짜 순으로 정렬하는 함수
void User::SortSchedule() {

    sort(this->user_schd_list.begin(), this->user_schd_list.end());
}
```

```
// 비교 연산자 오버라이딩
bool operator < (const Schedule& schd);
bool operator > (const Schedule& schd);
bool operator == (const Schedule& schd);
```

```
// Date 클래스 연산자 오버라이딩
bool Date::operator<(const Date& date) const {
    if (this->year != date.year) {
        return this->year < date.year;
    }
    if (this->month != date.month) {
        return this->month < date.month;
    }
    return this->day < date.day;
}

bool Date::operator>(const Date& date) const {
    if (this->year != date.year) {
        return this->year > date.year;
    }
    if (this->month != date.month) {
        return this->month > date.month;
    }
    return this->day > date.day;
}

bool Date::operator==(const Date& date) const {
    return this->year == date.year && this->month == date.month && this->day == date.day;
}
```

2) 테스트 결과

(1) 사용자 생성

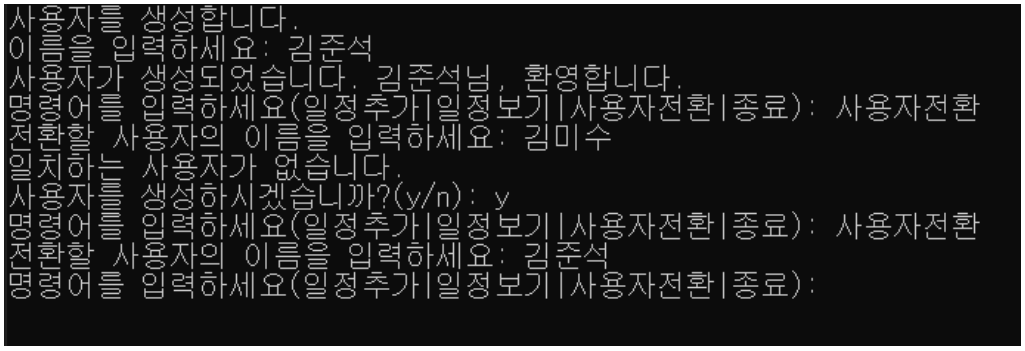
- 사용자 생성 검증(객체 생성, GetUserName 함수 호출 여부)

-

```
사용자를 생성합니다.
이름을 입력하세요: 김준석
사용자가 생성되었습니다. 김준석님, 환영합니다.
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료):
```

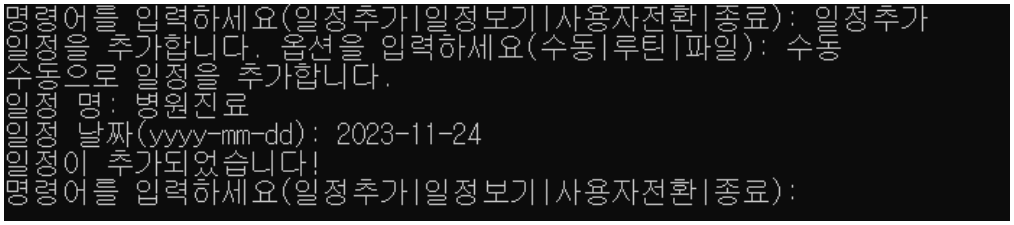
(2) 사용자 전환

- 존재하지 않는 사용자일 때의 사용자 생성 검증 및 존재하는 사용자일 때의 사용자 전환 여부 검증

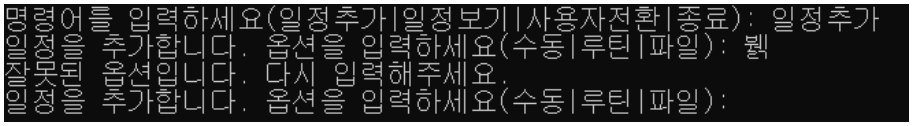
- 

(3) 일정 수동 입력

- 올바른 옵션을 입력했을 때, 올바른 일정 추가 검증

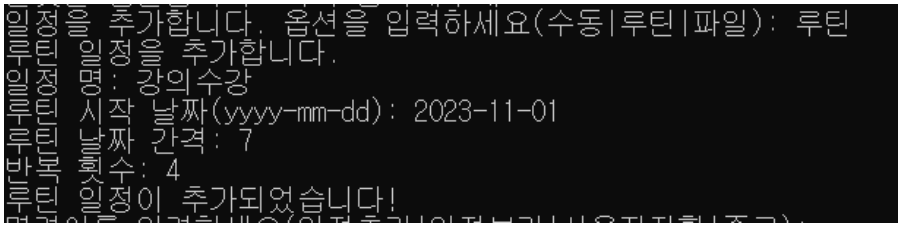
- 

- 올바르지 않은 옵션을 입력했을 때, 출력 검증

- 

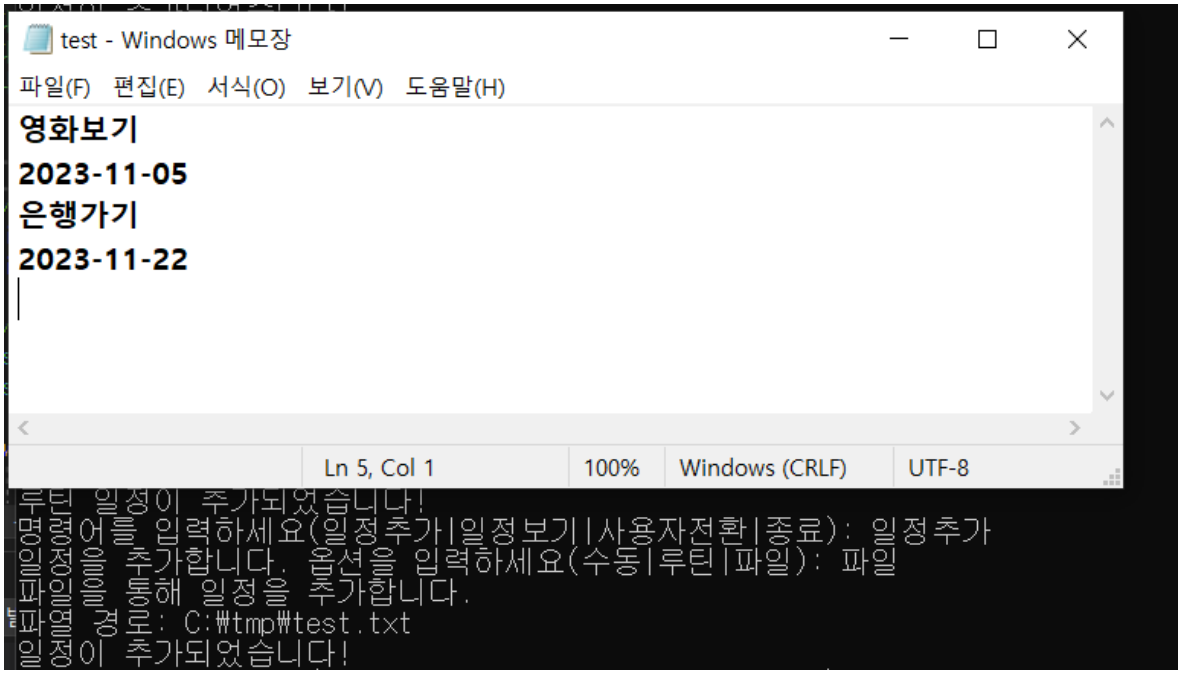
(4) 루틴 일정 입력

- 올바른 입력일 때, 올바른 일정 추가 검증

- 

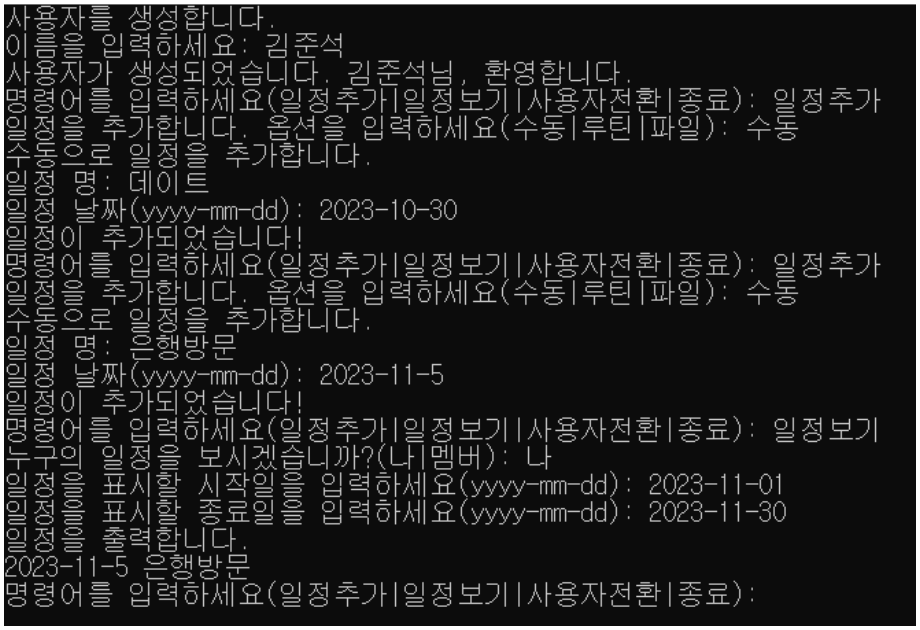
(5) 파일 배치 입력

- 올바른 경로일 때, 올바른 일정 추가 검증

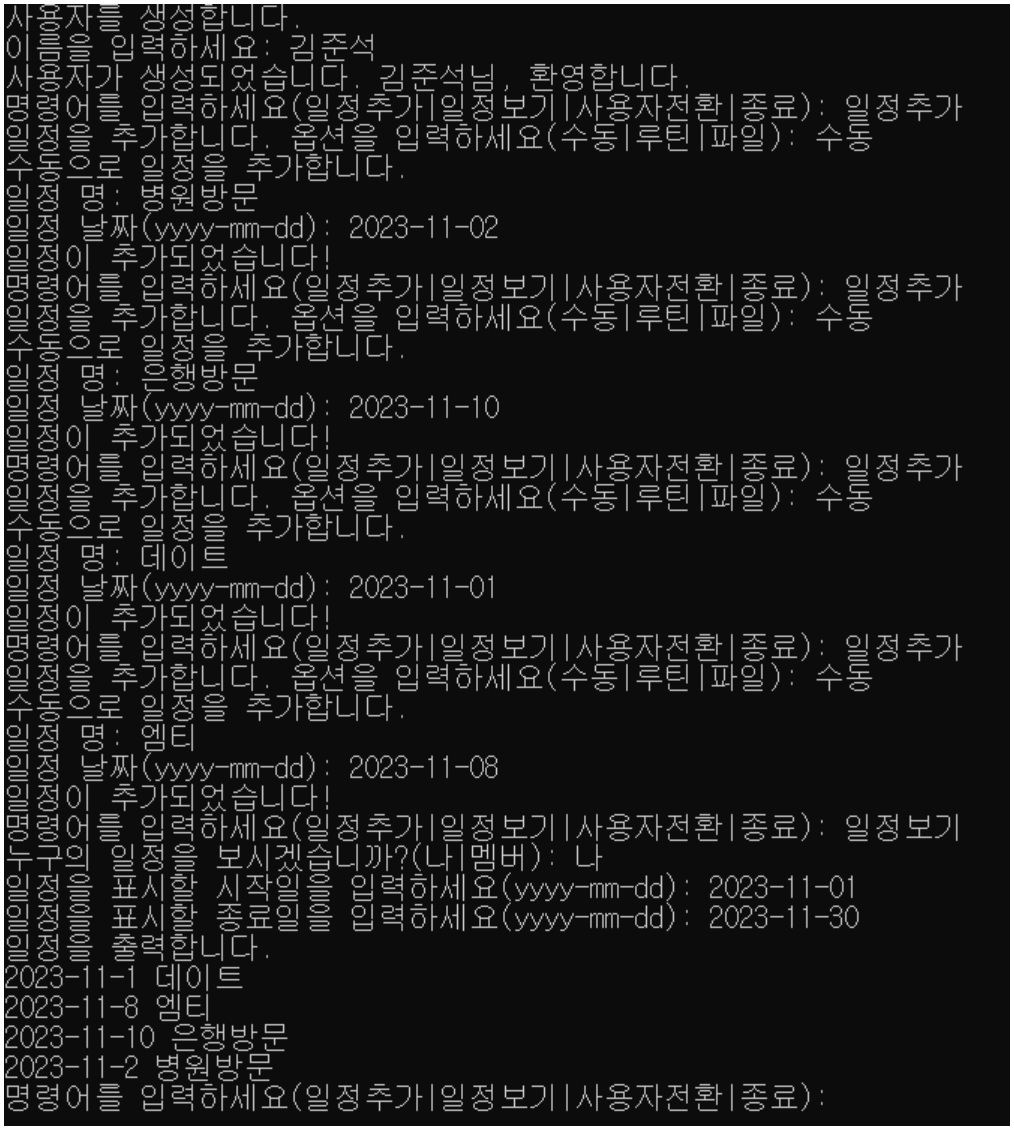
- 

(6) 일정표 출력 - 일정 표시 대상 선택 – 나 / 일정 표시 범위 선택 – 사용자 지정 / 일정 표시 형식 선택 – 리스트형

- 범위를 벗어나는 일정이 출력되지 않는지 검증

- 

- 범위 내의 일정들이 모두 출력되는지, 빠른 날짜 순으로 출력되는지 검증 (빠른 날짜 순으로 올바르게 출력되지 않아, 디버깅 필요)



4. 계획 대비 변경 사항

5. 프로젝트 일정

업무	11/3	11/14	11/24	11/31
제안서 작성	완료			
일정 입력		완료		
일정표 출력(리스트)			완료	

일정표 출력(달력)				---------->
업무	12/3	12/7	12/10	12/14
그룹 관리	---------->			
멤버의 일정에 동참		---------->		
일정이 있는 멤버 조회			---------->	
예외 처리 및 리팩토링				---------->