C++프로그래밍및실습

공유형 일정 관리 프로그램 개발

최종 보고서

제출일자: 2023-12-23
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. 또한, 사용자 전환 시 존재하지 않는 사용자라면 사용자를 생성한다.

(2) 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 ANSI방식으로 인코딩 된 텍스트 파일을 읽고, 텍스트 파일에 작성된 일정들을 한 번에 배치 입력한다.

3) 일정표 출력

- 고객이 설정한 대상과 날짜, 범위에 해당하는 일정들을 나타내는 일정표를 출

력한다. 이 때, 대상은 본인 뿐만 아니라 같은 그룹 내의 다른 멤버들이 될 수도 있다. 또한, 월/주/일 또는 사용자 지정의 범위를 지원한다.

(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 있고, 월, 주, 일 단위의 선택도 가능하다.

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. 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

    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

    A. 클래스(User 클래스 객체 생성)

    B. 포인터(current_user 변수)

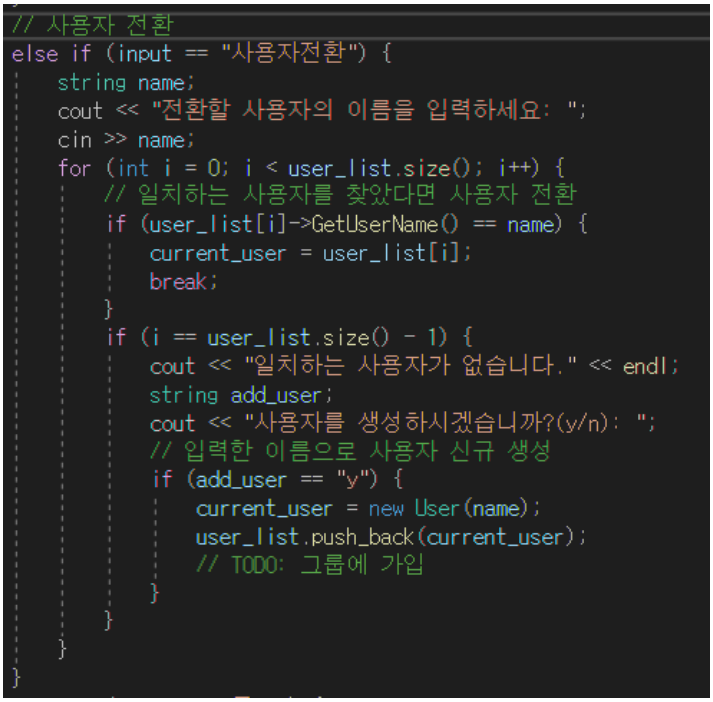
    C. 조건문

    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

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

```cpp
// 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);
}
```

```cpp
// 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

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

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)

G. 파일 입력(route 경로의 파일을 is로 읽기)

- 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
    ifstream is;
    is.open(route);

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

    while (is >> schd_name >> schd_date) {

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

(6) 사용자 입력 범위의 일정 보기 – 나의 일정(리팩토링됨)

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어

2. option : 사용자가 선택한 일정 표시 대상 옵션

3. print_option : 사용자가 선택한 일정 표시 범위 옵션

3. start : 일정을 표시할 시작일

4. end : 일정을 표시할 종료일

B. 출력

1. 일정 표시 대상 선택을 안내하는 문자열

2. 일정 표시 범위 옵션 선택을 안내하는 문자열

3. 일정 표시 시작일 입력을 안내하는 문자열

4. 일정 표시 종료일 입력을 안내하는 문자열

5. 일정 출력을 안내하는 문자열

6. 선택한 대상/범위에 해당하는 일정들

- 설명

사용자가 '일정보기'를 명령어로 입력한 경우, 일정 표시 대상을 입력하도록 한다. 대상을 '나'로 입력했다면, 현재 나를 가리키고 있는 current_user를 인자로 PrintSchdule 함수가 호출된다.

다음으로는 일정 표시 범위 옵션을 선택한다. 일정 표시 범위 옵션을 '사용자입력'으로 입력했다면, PrintSchdule 함수 내부에서 PrintRangeSchedule 함수가 호출되어 사용자입력 범위의 일정 표시가 시작된다.

다음으로는 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

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)

G. 함수 오버라이딩(Date 객체의 연산자 오버라이딩)

- 코드 스크린샷

```
// 일정 보기
else if (input == "일정보기") {
    string option;
    while (true) {
        cout << "누구의 일정을 보시겠습니까?(나|멤버): ";
        cin >> option;
        if (option == "나") {

            current_user->PrintSchedule(*current_user);
            break;
        }
```

```
// 일정을 출력하는 함수
void User::PrintSchedule(User& user) {

    while (true) {

        string print_option; // 일정 출력 범위 옵션
        cout << "범위 옵션을 선택하세요(월|주|일|사용자입력): ";
        cin >> print_option;

        if (print_option == "월") {
            PrintMonthSchedule(user);
            break;
        }
        else if (print_option == "주") {
            PrintWeekSchedule(user);
            break;
        }
        else if (print_option == "일") {
            PrintDaySchedule(user);
            break;
        }
        else if (print_option == "사용자입력") {
            PrintRangeSchedule(user);
            break;
        }
        else {
            cout << "잘못된 입력입니다. 다시 입력하세요." << endl;
            continue;
        }
    }
}
```

```
// 사용자가 지정한 날짜 범위 내 일정을 출력하는 함수
void User::PrintRangeSchedule(User &user) {

    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;
    user.PrintScheduleToList(user, start_date, end_date);
}
```

```cpp
// 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;
        }
    }
}
```

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

    sort(this->user_schd_list.begin(), this->user_schd_list.end(), [](Schedule* a, Schedule* b) {
        return a->schd_date < b->schd_date;
    });
}
```

```cpp
// 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;
}
```

(7) 월 범위의 일정 보기 – 나의 일정

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어

2. option : 사용자가 선택한 일정 표시 대상 옵션

3. print_option : 사용자가 선택한 일정 표시 범위 옵션

4. month : 일정을 표시할 월

B. 출력

1. 일정 표시 대상 선택을 안내하는 문자열

2. 일정 표시 범위 옵션 선택을 안내하는 문자열

3. 일정 표시 월 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, 현재 나를 가리키고 있는 current_user를 인자로 PrintSchdule 함수가 호출된다.

다음으로는 일정 표시 범위 옵션을 선택한다. 일정 표시 범위 옵션을 '월'으로 입력했다면, PrintSchdule 함수 내부에서 PrintMonthSchedule 함수가 호출되어 월 단위의 일정 표시가 시작된다.

다음으로는 일정을 표시할 월을 입력하도록 한다. 월을 입력받았다면, switch문을 통해 해당 월의 일 수를 days_in_month 변수에 저장한다. 이제 해당 달의 1일과 마지막날을 각각 Date 객체로 생성하고, 두 Date 객체를 인자로 PrintScheduleToList 함수를 호출한다. 마찬가지로 해당 함수의 내부에서 SortSchedule 함수가 호출된다.

- 적용된 배운 내용

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)

G. 함수 오버라이딩(Date 객체의 연산자 오버라이딩)

- 코드 스크린샷

중복되는 코드는 생략합니다.

```
// 특정 달(월)의 일정을 출력하는 함수
void User::PrintMonthSchedule(User& user) {

    int month;
    cout << "일정을 표시할 달을 입력하세요: ";
    cin >> month;

    int days_in_month;
    // month에 해당하는 일 수 설정
    switch (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;
    }

    Date start_date(2023, month, 1);
    Date end_date(2023, month, days_in_month);

    user.PrintScheduleToList(user, start_date, end_date);
}
```

(8) 주 범위의 일정 보기 – 나의 일정

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어
2. option : 사용자가 선택한 일정 표시 대상 옵션
3. print_option : 사용자가 선택한 일정 표시 범위 옵션
4. day : 일정을 표시할 주의 시작일

B. 출력

1. 일정 표시 대상 선택을 안내하는 문자열
2. 일정 표시 범위 옵션 선택을 안내하는 문자열
3. 일정 표시 주의 시작일 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, 현재 나를 가리키고 있는 current_user를 인자로 PrintSchdule 함수가 호출된다.

다음으로는 일정 표시 범위 옵션을 선택한다. 일정 표시 범위 옵션을 '주'로 입력했다면, PrintSchdule 함수 내부에서 PrintWeekSchedule 함수가 호출되어 주 단위의 일정 표시가 시작된다.

다음으로는 일정을 표시할 주의 시작일을 입력하도록 한다. 시작일을 입력받았다면, 해당 일을 인자로 Date 객체를 생성한다. 그리고 그 객체에 6을 더한 해당 주의 마지막 일의 Date 객체를 생성한다. (Date 객체의 +연산자를 오버라이딩 했기 때문에 해당 연산이 가능하다.) 이제 두 Date 객체를 인자로 PrintScheduleToList 함수를 호출한다. 마찬가지로 해당 함수의 내부에서 SortSchedule 함수가 호출된다.

- 적용된 배운 내용

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)

G. 함수 오버라이딩(Date 객체의 연산자 오버라이딩)

- 코드 스크린샷

```
// 특정 주의 일정을 출력하는 함수
void User::PrintWeekSchedule(User& user) {

    string day;
    cout << "일정을 표시할 주의 시작일을 입력하세요(yyyy-mm-dd): ";
    cin >> day;

    Date start_date(day);
    Date end_date = start_date + 6;

    user.PrintScheduleToList(user, start_date, end_date);
}
```

(9) 일 범위의 일정 보기 – 나의 일정

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어

2. option : 사용자가 선택한 일정 표시 대상 옵션

3. print_option : 사용자가 선택한 일정 표시 범위 옵션

4. day : 일정을 표시할 날짜

B. 출력

1. 일정 표시 대상 선택을 안내하는 문자열

2. 일정 표시 범위 옵션 선택을 안내하는 문자열

3. 일정 표시할 날짜 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, 현재 나를 가리키고 있는 current_user를 인자로 PrintSchdule 함수가 호출된다.

다음으로는 일정 표시 범위 옵션을 선택한다. 일정 표시 범위 옵션을 '일'로 입력했다면, PrintSchdule 함수 내부에서 PrintDaySchedule 함수가 호출되어 일 단위의 일정 표시가 시작된다.

다음으로는 일정을 표시할 날짜를 입력하도록 한다. 날짜를 입력받았다면, 해당 일을 인자로 Date 객체를 생성한다. 이제 해당 Date 객체를 인자로 PrintScheduleToList 함수를 호출한다. (시작일과 마지막일의 객체 모두 해당 일의 Date객체로 한다.) 마찬가지로 해당 함수의 내부에서 SortSchedule 함수가 호출된다.

- 적용된 배운 내용

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)

G. 함수 오버라이딩(Date 객체의 연산자 오버라이딩)

- 코드 스크린샷

```
// 특정 일의 일정을 출력하는 함수
void User::PrintDaySchedule(User& user) {

    string day;
    cout << "일정을 표시할 날짜를 입력하세요(yyyy-mm-dd): ";
    cin >> day;

    Date date(day);

    user.PrintScheduleToList(user, date, date);
}
```

(10) 그룹에 가입하기

- 입출력

A. 입력

1. group_option : 그룹에 가입할지의 여부
2. group_name : 가입할 그룹의 이름
3. add_group : 그룹을 생성할지의 여부

B. 출력

1. 그룹에 가일할지의 여부 입력을 안내하는 문자열
2. 가입 또는 생성할 그룹의 이름 입력을 안내하는 문자열
3. 그룹의 가입을 안내하는 문자열
4. 입력한 그룹이 존재하지 않음을 안내하는 문자열
5. 입력한 이름으로 그룹의 생성 여부를 묻는 문자열
6. 그룹의 생성을 안내하는 문자열

- 설명

프로그램을 최초로 실행하게 되면(current_user가 nullptr인 상태), 사용자가 생성된다. 이후 사용자는 그룹 가입 또는 생성여부를 입력한다. 최초 실행 시엔 생성된 그룹이 없으므로, 이 때엔 따로 묻지 않고, 그룹을 생성한다.

사용자 전환 시에도 그룹의 가입 여부를 묻는다. 사용자가 'y'를 입력해 그룹에 가입할 의사를 밝히면, 가입할 그룹의 이름을 입력받아 group_name에 저장한다. 이제 생성된

그룹의 목록인 group_list를 탐색해가며, group_name과 일치하는 그룹이 있는지를 따진다. 만일 일치하는 그룹이 발견되면, 해당 그룹 객체의 member_list에 current_user를 추가한다. 그리고 현재 그룹을 가리키는 current_group에 해당 그룹을 저장한다. 일치하는 그룹이 없다면, 입력한 그룹이 없음을 안내하고 생성 여부를 묻는다. 사용자가 'y'를 입력해 그룹을 생성하려고 하면, group_list에 group_name을 인자로 생성한 Group객체를 추가하고 해당 객체의 member_list에 current_user를 추가한다. 이후 마찬가지로 current_group에 해당 그룹을 저장한다.

- 적용된 배운 내용

A. 클래스(Group 클래스)

B. 포인터(current_user, current_group)

C. 조건문

D. 반복문

F. 벡터(Group* 타입의 백터 group_list, Group 객체의 필드 member_list)

- 코드 스크린샷

```
// 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;

    string group_option;
    cout << "그룹에 가입하시겠습니까?(y/n): ";
    cin >> group_option;
    if (group_option == "y") {
        string group_name;
        cout << "가입 또는 생성할 그룹의 이름을 입력하세요: ";
        cin >> group_name;
        // 그룹 생성
        group_list.push_back(new Group(group_name));
        cout << "그룹이 생성되었습니다." << endl;
        // 그룹 가입
        group_list[group_list.size() - 1]->member_list.push_back(current_user);
        cout << "그룹에 가입되었습니다." << endl;
        current_group = group_list[group_list.size() - 1];
    }
}
```

```
cout << "사용자가 생성되었습니다!" << endl;
// 그룹에 가입
string group_option;
cout << "그룹에 가입하시겠습니까?(y/n): ";
cin >> group_option;
if (group_option == "y") {
    string group_name;
    cout << "가입할 그룹의 이름을 입력하세요: ";
    cin >> group_name;
    for (int i = 0; i < group_list.size(); i++) {
        if (group_list[i]->group_name == group_name) {
            // 현재 유저를 그룹에 추가
            group_list[i]->member_list.push_back(current_user);
            cout << "그룹에 가입되었습니다." << endl;
            // 현재 그룹을 해당 그룹으로 변경
            current_group = group_list[i];
            break;
        }
        // 일치하는 그룹이 없는 경우
        if (i == group_list.size() - 1) {
            cout << "입력한 그룹이 존재하지 않습니다." << endl;
            string add_group;
            cout << "입력한 그룹을 생성하시겠습니까?(y/n): ";
            cin >> add_group;
            // 그룹 생성
            if (add_group == "y") {
                group_list.push_back(new Group(group_name));
                cout << "그룹이 생성되었습니다." << endl;
                group_list[group_list.size() - 1]->member_list.push_back(current_user);
                cout << "그룹에 가입되었습니다." << endl;
                current_group = group_list[group_list.size() - 1];
                break;
            }
        }
    }
    break;
}
else {
    break;
}
```

(11) 멤버의 일정 보기

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어

2. option : 사용자가 선택한 일정 표시 대상 옵션

3. members_name : 일정을 볼 대상 멤버의 이름

4. print_option : 사용자가 선택한 일정 표시 범위 옵션

B. 출력

1. 일정 표시 대상 선택을 안내하는 문자열

2. 일정을 볼 대싱 멤버의 이름 입력을 안내하는 문자열

3. 입력한 멤버가 그룹에 존재하지 않음을 안내하는 문자열

4. 일정 표시 범위 옵션 선택을 안내하는 문자열

5. 표시 옵션에 따라 시작일/종료일 또는 월/주 시작일/날짜 입력을 안내하는 문자열

6. 일정 출력을 안내하는 문자열

7. 선택한 대상/범위에 해당하는 일정들

- 설명

사용자가 '일정보기' 명령어를 입력하면, 일정을 볼 대상을 선택하게 된다. 이 때, '멤버'를 입력하면, 같은 그룹 내의 멤버를 지정해 해당 멤버의 일정을 볼 수 있다. 현재 사용자가 가입된 그룹이 없는지 current_group이 nullptr인지를 통해 확인하고 가입된 그룹이 없다면, 가입된 그룹이 없음을 안내한다. 가입된 그룹이 있다면, 일정을 볼 대상 멤버의 이름을 입력하게 되고 입력값은 members_name에 저장된다. 이제 members_name을 가지고 current_group의 member_list에 해당 이름이 존재하는지 탐색한다. 존재하지 않는다면, 입력한 멤버가 그룹에 존재하지 않음을 안내하고, 존재한다면 해당 멤버를 인자로 PrintSchedule 함수를 호출한다. 이후의 과정은 나의 일정을 볼 때와 동일한 로직을 수행하게 된다.

- 적용된 배운 내용

A. 클래스(Group, Schedule, User 클래스)

B. 포인터(current_user, current_group)

C. 조건문

D. 반복문

F. 벡터(Group 객체의 필드 member_list)

- 코드 스크린샷

```
else if (option == "멤버") {

    string members_name;
    cout << "일정을 볼 대상 멤버의 이름을 입력하세요: ";
    cin >> members_name;

    if (current_group != nullptr) {
        for (int i = 0; i < current_group->member_list.size(); i++) {
            // 입력한 멤버가 존재
            if (current_group->member_list[i]->GetUserName() == members_name) {
                current_user->PrintSchedule(*current_group->member_list[i]);
                break;
            }
            // 입력한 멤버가 없음
            if (i == current_group->member_list.size() - 1) {
                cout << "입력한 멤버가 그룹에 존재하지 않습니다." << endl;
                break;
            }
        }
        break;
    }
    else {
        cout << "현재 소속된 그룹이 없습니다." << endl;
        continue;
    }
}
```

(12) 멤버의 일정에 동참하기

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어
2. members_name : 동참하려는 대상 멤버의 이름
3. date : 동참하고 싶은 일정의 날짜
4. schd_name : 동참하고 싶은 일정의 이름

B. 출력

1. 가입된 그룹이 없음을 안내하는 문자열
2. 일정에 동참하고 싶은 멤버의 이름 입력을 안내하는 문자열
3. 입력한 멤버가 그룹에 존재하지 않음을 안내하는 문자열
4. 동참하고 싶은 일정의 날짜 입력을 안내하는 문자열
5. 해당 날짜의 대상 멤버의 일정들

6. 동참하고 싶은 일정의 이름 입력을 안내하는 문자열

7. 일정이 추가되었음을 안내하는 문자열

- 설명

사용자가 '일정동참' 명령어를 입력하면, 현재 사용자인 current_user가 가입된 그룹을 나타내는 current_group이 nullptr인지 확인한다. nullptr이면 가입된 그룹이 없음을 안내하고 nullptr이 아니면, 일정에 동참하고 싶은 멤버의 이름을 입력하게 된다.

입력한 이름은 members_name에 저장되고, current_group을 탐색하며 해당 이름이 존재하는지 찾는다. 존재하지 않는다면, 입력한 멤버가 존재하지 않음을 안내하고 존재한다면, member에 해당 멤버의 객체의 포인터를 저장하고, 동참하고 싶은 일정의 날짜를 입력한다.

입력한 날짜는 date에 저장되고, date를 인자로 Date 객체를 생성한다. 이제 이 Date 객체와 User타입의 포인터인 member를 인자로 PrintScheduleToList 함수를 호출해 해당 날짜의 대상 멤버의 일정들을 보여준다. 이후, 동참하고 싶은 일정의 이름 입력을 안내한다. 사용자는 출력된 일정 중 한 가지를 골라 대상 일정의 이름을 입력한다. 입력된 이름은 schd_name에 저장되고, schd_name과 schd_date(아까 생성한 Date 객체), 그리고 member를 인자로 JoinSchedule함수를 호출한다.

JoinSchedule 함수는 인자로 준 schd_date와 schd_name을 갖고 member의 user_schd_list를 탐색한다. 일치하는 일정이 나타나면, 현재 사용자인 current_user의 user_schd_list에 해당 일정을 추가한다.

- 적용된 배운 내용

A. 클래스(Group, Schedule, User, Date 클래스)

B. 포인터(current_user, current_group)

C. 조건문

D. 반복문

F. 벡터(Group 객체의 필드 member_list, User 객체의 필드 user_shcd_list)

- 코드 스크린샷

```
// 일정 동참
else if (input == "일정동참") {
    if (current_group == nullptr) {
        cout << "가입된 그룹이 없습니다!" << endl;
    }
    else {
        string members_name; // 대상 멤버의 이름
        cout << "일정에 동참하고 싶은 멤버의 이름을 입력하세요: ";
        cin >> members_name;
        for (int i = 0; i < current_group->member_list.size(); i++) {
            if (current_group->member_list[i]->GetUserName() == members_name) {
                User* member = current_group->member_list[i];
                // 일정 날짜 입력
                string date;
                cout << "동참하고 싶은 일정의 날짜를 입력하세요(yyyy-mm-dd): ";
                cin >> date;
                Date schd_date(date);
                cout << "해당 날짜의 멤버의 일정은 아래와 같습니다." << endl;
                member->PrintScheduleToList(*member, schd_date, schd_date);

                // 일정 이름 입력
                string schd_name;
                cout << "동참하고 싶은 일정의 이름을 입력하세요: ";
                cin >> schd_name;
                // 일정 동참
                current_user->JoinShedule(*member, schd_date, schd_name);
                break;
            }
            if (i == current_group->member_list.size() - 1) {
                cout << "입력한 멤버가 존재하지 않습니다." << endl;
            }
        }
    }
}
```

```
// 멤버의 일정을 선택해 내 일정에 추가하는 함수
void User::JoinShedule(User& member, Date schd_date, string schd_name) {

    for (Schedule* schd : member.user_schd_list) {
        // 날짜와 이름이 일치하면 일정 목록에 추가
        if (schd->schd_date == schd_date) {
            if (schd->schd_name == schd_name) {
                this->user_schd_list.push_back(schd);
                cout << "일정이 추가되었습니다!" << endl;
                break;
            }
        }
    }
}
```

(13) 특정 일에 일정이 있는 멤버 조회하기

- 입출력

A. 입력

1. input : 사용자가 입력한 명령어

2. date : 일정이 있는 멤버를 조회할 날짜

B. 출력

1. 가입된 그룹이 없음을 안내하는 문자열

2. 조회할 날짜 입력을 안내하는 문자열

3. 멤버 출력을 안내하는 문자열

4. 입력한 날짜에 일정이 있는 멤버들

5. 조회되는 멤버가 없는 경우 그것을 안내하는 문자열

- 설명

사용자가 '멤버조회' 명령어를 입력하면, 현재 사용자인 current_user가 가입된 그룹을 나타내는 current_group이 nullptr인지 확인한다. nullptr이면 가입된 그룹이 없음을 안내하고 nullptr이 아니면, 멤버를 조회할 날짜를 입력하게 된다.

입력한 날짜는 date에 저장되고, date를 매개변수로 Date 객체가 생성된다. 또한, 조회된 멤버의 이름을 저장하기 위한 output_buffer 벡터가 선언된다.

current_group의 필드인 member_list에 대해 반복문을 수행한다. 이 반복문 내부에서 이중으로 반복문이 수행된다. member_list의 요소인 각 User 객체의 필드 user_schd_list에 대해 탐색이 수행된다. user_schd_list에 입력한 날짜의 Schedule 객체가 있으면, output_buffer에 해당 멤버의 이름이 추가된다.

전체 member_list에 대해 탐색이 끝나면, output_buffer를 출력한다. 이 때, 비어있다면 조회되는 멤버가 없음을 나타내는 문자열을 출력한다.

- 적용된 배운 내용

A. 클래스(Group, Schedule, User, Date 클래스)

B. 포인터(current_user, current_group)

C. 조건문

D. 반복문

F. 벡터(Group 객체의 필드 member_list, User 객체의 필드 user_shcd_list,

멤버 출력을 위한 output_buffer)

- 코드 스크린샷

```
// 멤버 조회
else if (input == "멤버조회") {
    if (current_group == nullptr) {
        cout << "현재 소속된 그룹이 없습니다." << endl;
    }
    else {
        string date;
        cout << "일정이 있는 멤버를 조회할 날짜를 입력하세요(yyyy-mm-dd): ";
        cin >> date;
        Date schd_date(date);

        cout << date << "에 일정이 있는 멤버를 출력합니다." << endl;
        vector<string> output_buffer; // 멤버 이름 출력 버퍼
        // 그룹 내 멤버 탐색
        for (int i = 0; i < current_group->member_list.size(); i++) {
            // 멤버의 일정 탐색
            for (int j = 0; j < current_group->member_list[i]->user_schd_list.size(); j++) {
                if (current_group->member_list[i]->user_schd_list[j]->schd_date == schd_date)
                    // 출력 버퍼에 추가
                    output_buffer.push_back(current_group->member_list[i]->GetUserName());
                    break;
                }
            }
        }
        // 버퍼 출력
        if (output_buffer.size() < 1) {
            cout << "일정이 있는 멤버가 없습니다!" << endl;
        }
        else {
            for (int i = 0; i < output_buffer.size(); i++) {
                cout << output_buffer[i] << endl;
            }
        }
    }
}
```

4. 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

```
사용자를 생성합니다.
이름을 입력하세요: 김준석
사용자가 생성되었습니다. 김준석님, 환영합니다.
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료): 사용자전환
전환할 사용자의 이름을 입력하세요: 김미수
일치하는 사용자가 없습니다.
사용자를 생성하시겠습니까?(y/n): y
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료): 사용자전환
전환할 사용자의 이름을 입력하세요: 김준석
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료):
```

-

(3) 일정 수동 입력

- 올바른 옵션을 입력했을 때, 올바른 일정 추가 검증

```
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 병원진료
일정 날짜(yyyy-mm-dd): 2023-11-24
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료):
```

-

- 올바르지 않은 옵션을 입력했을 때, 출력 검증

```
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 뭬
잘못된 옵션입니다. 다시 입력해주세요.
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일):
```

-

(4) 루틴 일정 입력

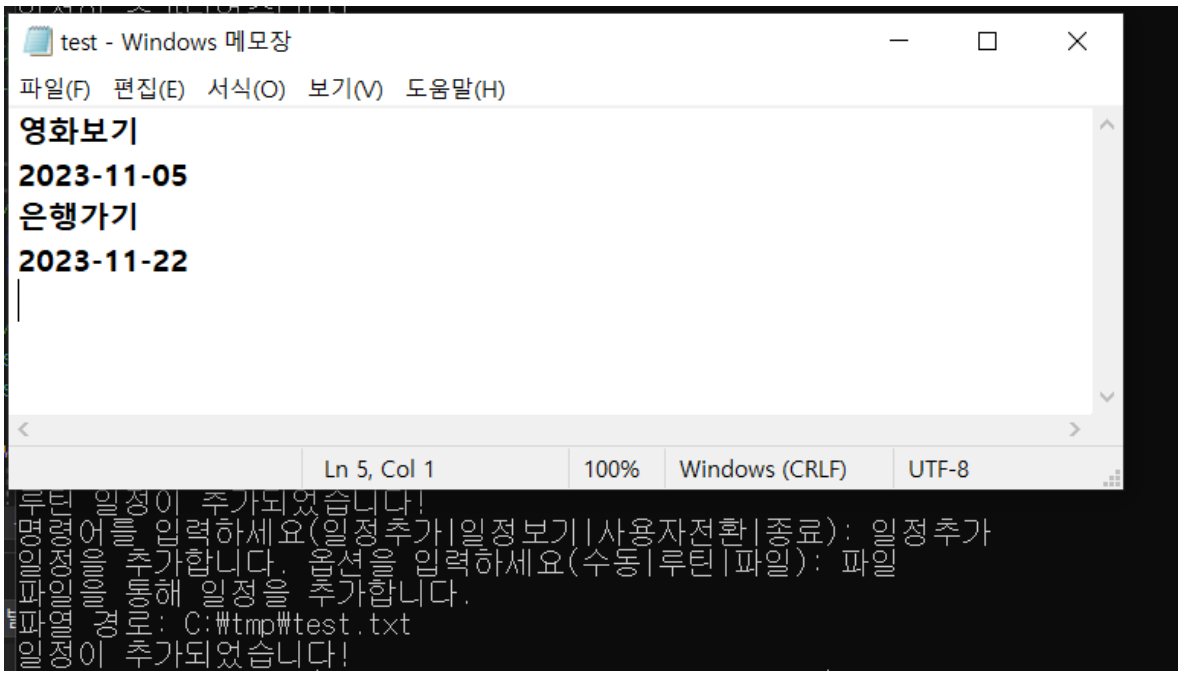
- 올바른 입력일 때, 올바른 일정 추가 검증

```
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 루틴
루틴 일정을 추가합니다.
일정 명: 강의수강
루틴 시작 날짜(yyyy-mm-dd): 2023-11-01
루틴 날짜 간격: 7
반복 횟수: 4
루틴 일정이 추가되었습니다!
```

-

(5) 파일 배치 입력

- 올바른 경로일 때, 올바른 일정 추가 검증



(6) 일정 입력

- 입력한 일정이 제대로 추가되었는지 검증

```
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정보기
누구의 일정을 보시겠습니까?(나|멤버): 나
범위 옵션을 선택하세요(월|주|일|사용자입력): 사용자입력
일정을 표시할 시작일을 입력하세요(yyyy-mm-dd): 2023-01-01
일정을 표시할 종료일을 입력하세요(yyyy-mm-dd): 2023-12-31
일정을 출력합니다.
2023-1-1 신년회
2023-1-5 러닝크루
2023-1-10 아침운동
2023-1-20 아침운동
2023-1-30 아침운동
2023-2-9 아침운동
2023-2-19 아침운동
2023-11-5 영화보기
2023-11-22 은행가기
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료):
```

(7) 나의 일정표 출력

- 범위를 벗어나는 일정이 출력되지 않는지 검증

-
```
사용자를 생성합니다.
이름을 입력하세요: 김준석
사용자가 생성되었습니다. 김준석님, 환영합니다.
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 데이트
일정 날짜(yyyy-mm-dd): 2023-10-30
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 은행방문
일정 날짜(yyyy-mm-dd): 2023-11-5
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료): 일정보기
누구의 일정을 보시겠습니까?(나|멤버): 나
일정을 표시할 시작일을 입력하세요(yyyy-mm-dd): 2023-11-01
일정을 표시할 종료일을 입력하세요(yyyy-mm-dd): 2023-11-30
일정을 출력합니다.
2023-11-5 은행방문
명령어를 입력하세요(일정추가|일정보기|사용자전환|종료):
```

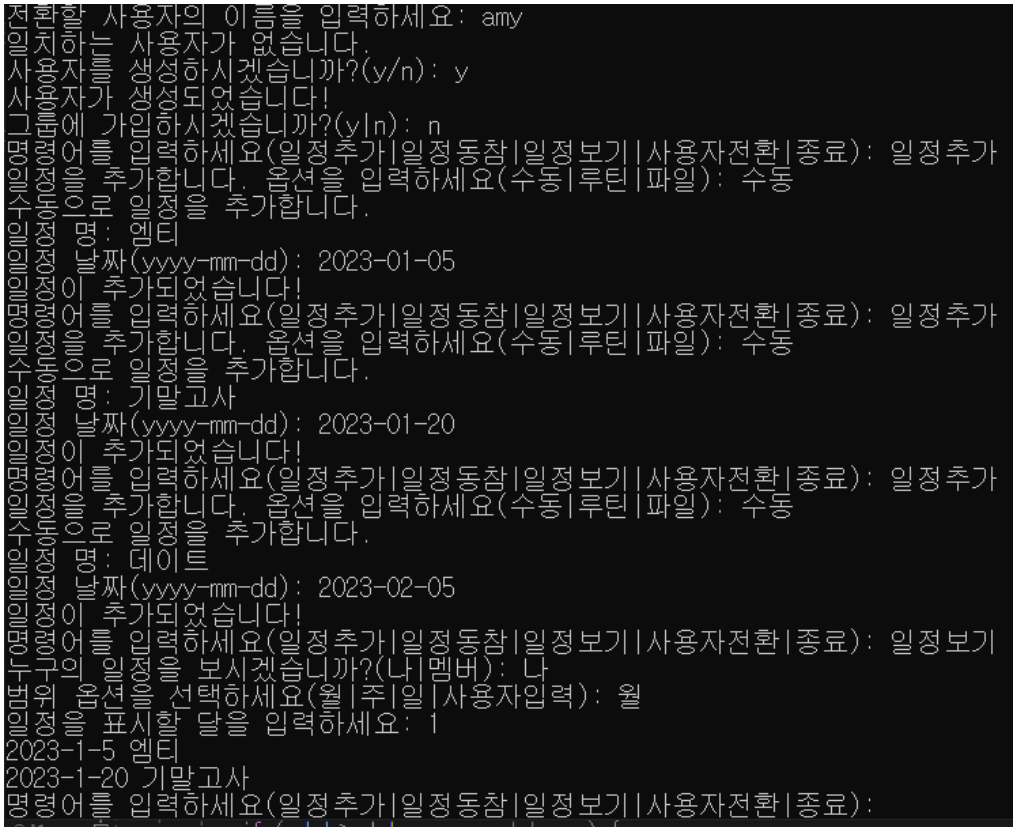
- 범위 내의 일정들이 모두 출력되는지, 빠른 날짜 순으로 출력되는지 검증

```
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 병원방문
일정 날짜(yyyy-mm-dd): 2023-11-02
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 은행방문
일정 날짜(yyyy-mm-dd): 2023-11-10
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 데이트
일정 날짜(yyyy-mm-dd): 2023-11-01
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 엠티
일정 날짜(yyyy-mm-dd): 2023-11-08
일정이 추가되었습니다!
```

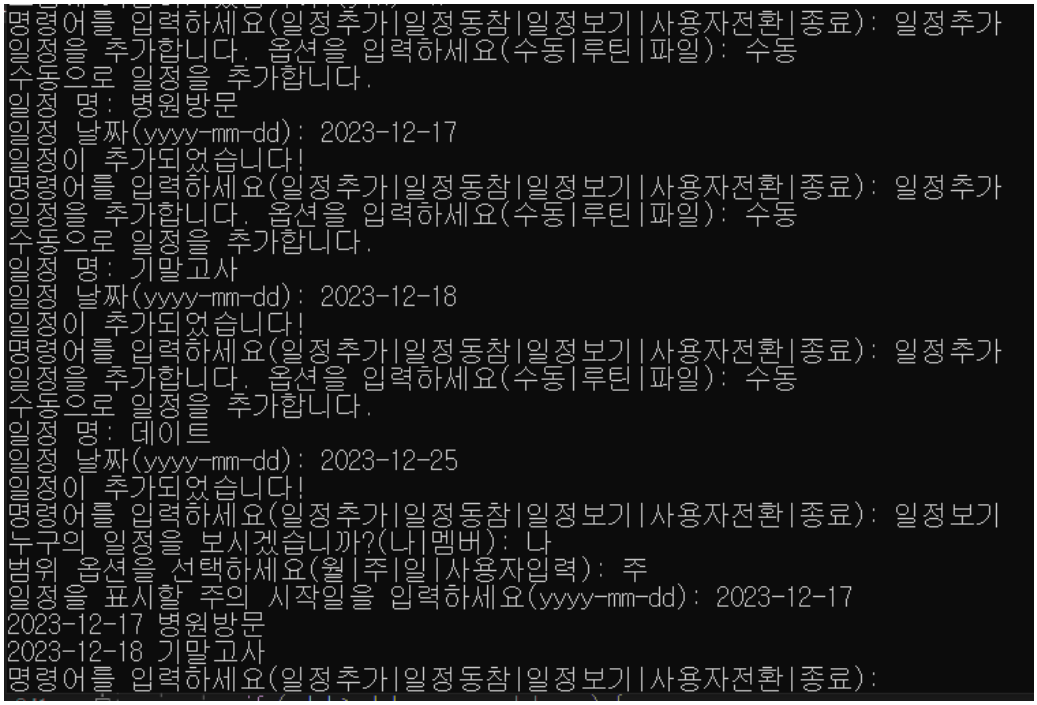
- 명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정보기

```
누구의 일정을 보시겠습니까?(나|멤버): 나
범위 옵션을 선택하세요(월|주|일|사용자입력): 2023-11-01
잘못된 입력입니다. 다시 입력하세요.
범위 옵션을 선택하세요(월|주|일|사용자입력): 2023-11-30
잘못된 입력입니다. 다시 입력하세요.
범위 옵션을 선택하세요(월|주|일|사용자입력): 사용자입력
일정을 표시할 시작일을 입력하세요(yyyy-mm-dd): 2023-11-01
일정을 표시할 종료일을 입력하세요(yyyy-mm-dd): 2023-11-30
일정을 출력합니다.
2023-11-1 데이트
2023-11-2 병원방문
2023-11-8 엠티
2023-11-10 은행방문
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료):
```

- 월 단위의 일정 출력에서 해당 월의 모든 일정이 출력되는지, 다른 월의 일정이 출력되지는 않는지 검증

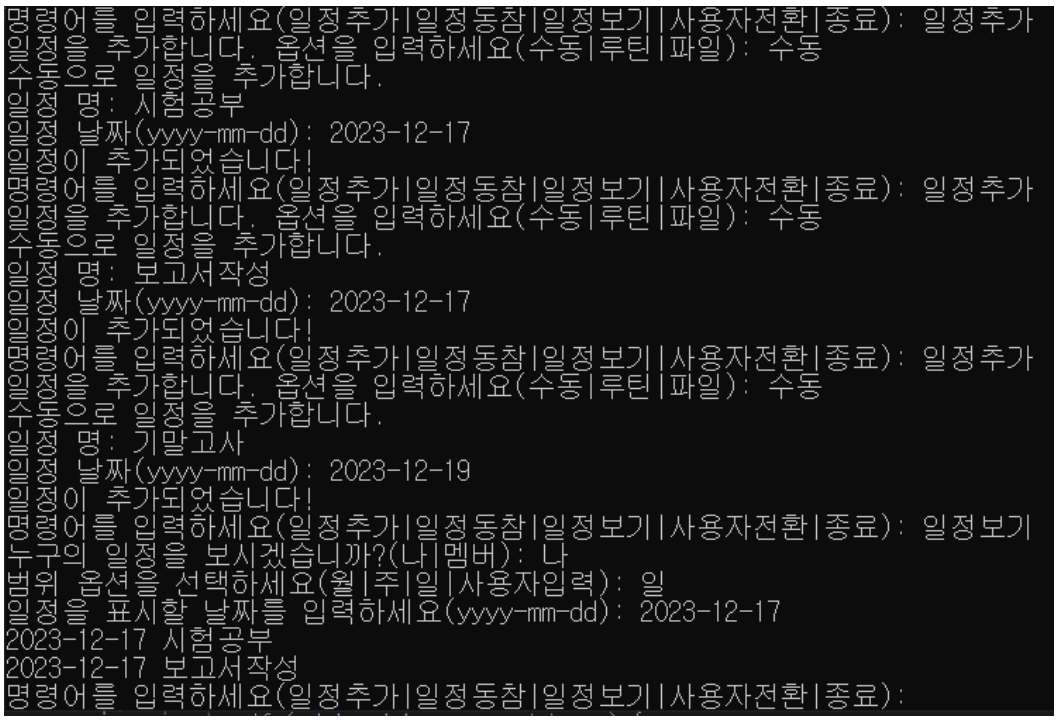
- 

- 주 단위의 일정 출력에서 해당 주의 모든 일정이 출력되는지, 다른 주의 일정이 출력되지는 않는지 검증

- 

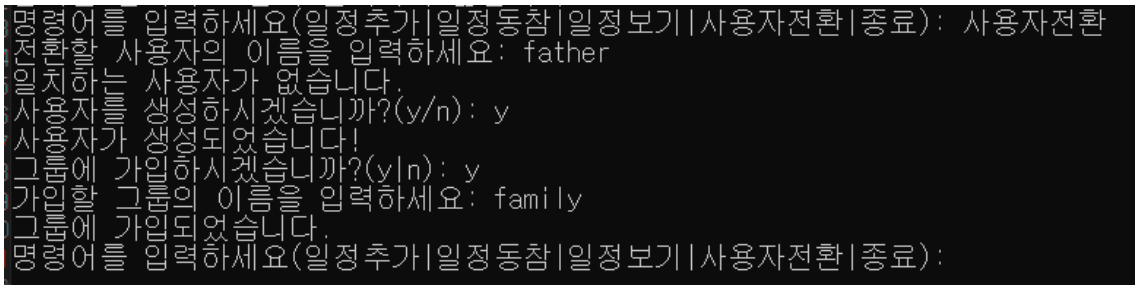
- 일 단위의 일정 출력에서 해당 일의 모든 일정이 출력되는지, 다른 날짜의 일정이 출

력되지는 않는지 검증

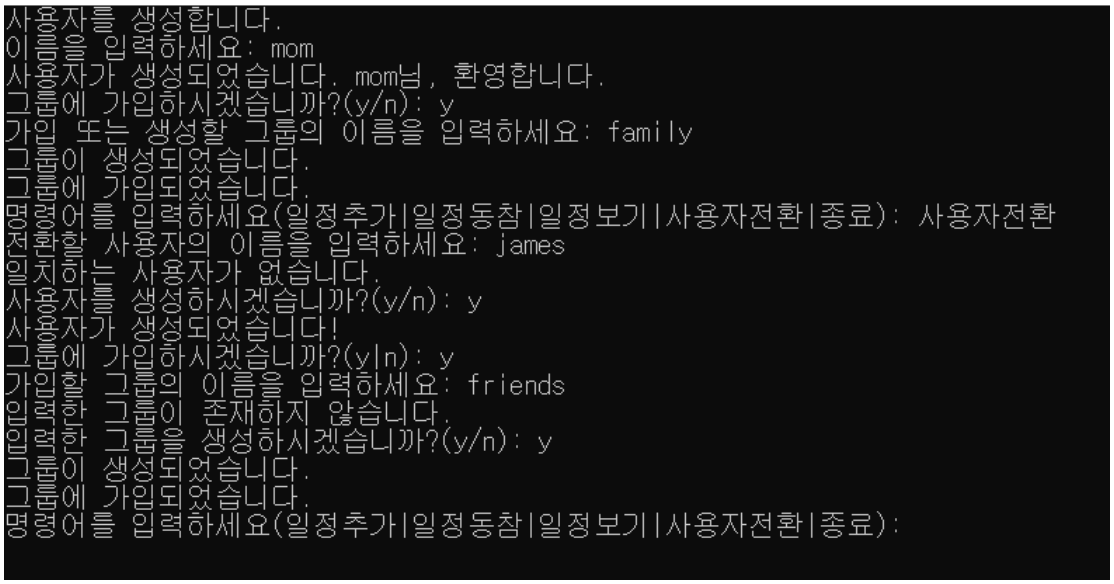
- 

(8) 그룹 가입

- 존재하는 그룹에 올바르게 가입되는지 검증

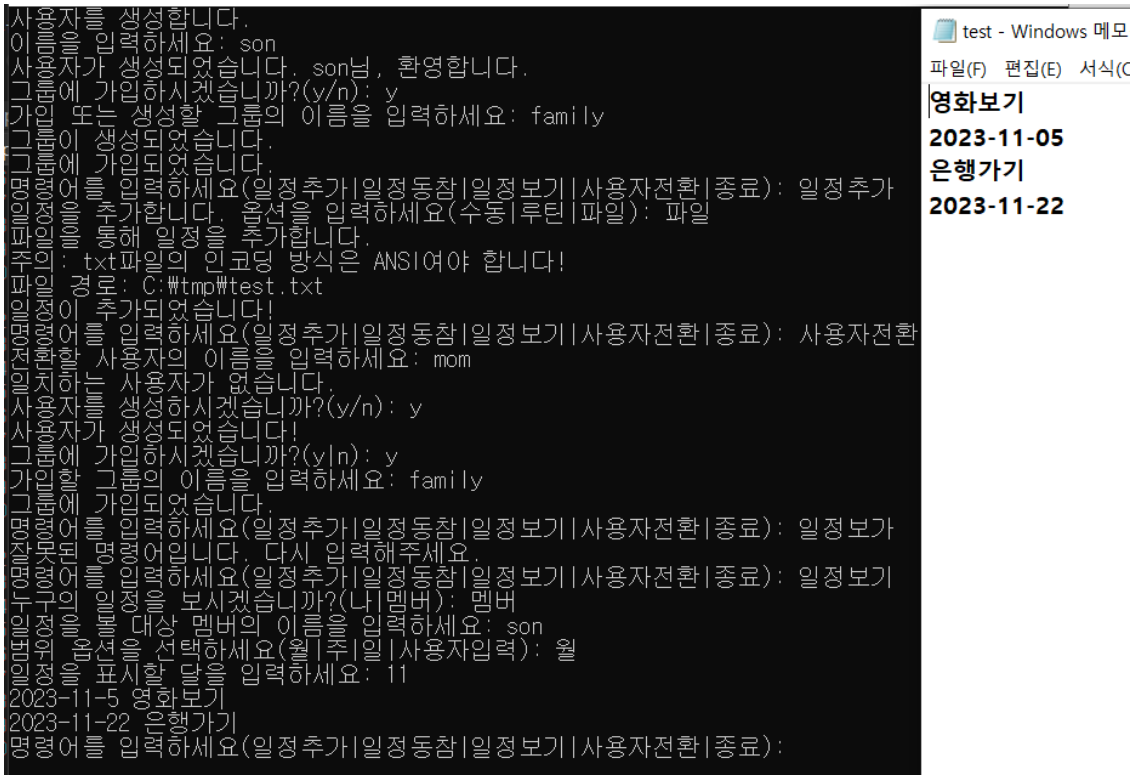
- 

- 존재하지 않는 그룹에 에러 메시지가 출력되는지, 이후 올바르게 그룹이 생성되는지 검증

- 

(9) 멤버의 일정표 출력

- 지정한 범위의 대상 멤버의 모든 일정이 출력되는지 검증

- 

- 그룹에 존재하지 않는 멤버의 이름을 입력했을 때, 에러 메시지 출력 검증

-
```
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정보기
누구의 일정을 보시겠습니까?(나|멤버): 멤버
일정을 볼 대상 멤버의 이름을 입력하세요: sister
입력한 멤버가 그룹에 존재하지 않습니다.
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료):
```

(10) 멤버의 일정에 동참

- 멤버의 일정에 동참한 후, 제대로 나의 일정에 추가되었는지 검증

-
```
사용자를 생성합니다.
이름을 입력하세요: son
사용자가 생성되었습니다. son님, 환영합니다.
그룹에 가입하시겠습니까?(y/n): y
가입 또는 생성할 그룹의 이름을 입력하세요: family
그룹이 생성되었습니다.
그룹에 가입되었습니다.
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 가족모임
일정 날짜(yyyy-mm-dd): 2023-01-01
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 신년회
일정 날짜(yyyy-mm-dd): 2023-01-01
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 사용자전환
전환할 사용자의 이름을 입력하세요: mom
일치하는 사용자가 없습니다.
사용자를 생성하시겠습니까?(y/n): y
사용자가 생성되었습니다!
그룹에 가입하시겠습니까?(y/n): y
가입할 그룹의 이름을 입력하세요: family
그룹에 가입되었습니다.
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정동참
일정에 동참하고 싶은 멤버의 이름을 입력하세요: son
동참하고 싶은 일정의 날짜를 입력하세요(yyyy-mm-dd): 2023-01-01
해당 날짜의 멤버의 일정은 아래와 같습니다.
2023-1-1 가족모임
2023-1-1 신년회
동참하고 싶은 일정의 이름을 입력하세요: 가족모임
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료): 일정보기
누구의 일정을 보시겠습니까?(나|멤버): 나
범위 옵션을 선택하세요(월|주|일|사용자입력): 월
일정을 표시할 달을 입력하세요: 1
2023-1-1 가족모임
명령어를 입력하세요(일정추가|일정동참|일정보기|사용자전환|종료):
```

(11) 특정 일에 일정이 있는 멤버 조회

- 입력한 날짜에 일정이 있는 멤버가 모두 출력되는지 검증

```
일정 날짜(yyyy-mm-dd): 2023-12-25
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|멤버조회|사용자전환|종료): 사용자전환
전환할 사용자의 이름을 입력하세요: james
일치하는 사용자가 없습니다.
사용자를 생성하시겠습니까?(y/n): y
사용자가 생성되었습니다!
그룹에 가입하시겠습니까?(y/n): y
가입할 그룹의 이름을 입력하세요: family
그룹에 가입되었습니다.
명령어를 입력하세요(일정추가|일정동참|일정보기|멤버조회|사용자전환|종료): 일정추가
일정을 추가합니다. 옵션을 입력하세요(수동|루틴|파일): 수동
수동으로 일정을 추가합니다.
일정 명: 특근
일정 날짜(yyyy-mm-dd): 2023-12-25
일정이 추가되었습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|멤버조회|사용자전환|종료): 사용자전환
전환할 사용자의 이름을 입력하세요: amy
일치하는 사용자가 없습니다.
사용자를 생성하시겠습니까?(y/n): y
사용자가 생성되었습니다!
그룹에 가입하시겠습니까?(y/n): y
가입할 그룹의 이름을 입력하세요: family
그룹에 가입되었습니다.
명령어를 입력하세요(일정추가|일정동참|일정보기|멤버조회|사용자전환|종료): 멤버조회
일정이 있는 멤버를 조회할 날짜를 입력하세요(yyyy-mm-dd): 2023-12-25
2023-12-25에 일정이 있는 멤버를 출력합니다.
jones
james
명령어를 입력하세요(일정추가|일정동참|일정보기|멤버조회|사용자전환|종료):
```

- 입력한 날짜에 일정이 없는 경우 문자열이 출력되는지 검증

```
명령어를 입력하세요(일정추가|일정동참|일정보기|멤버조회|사용자전환|종료): 멤버조회
일정이 있는 멤버를 조회할 날짜를 입력하세요(yyyy-mm-dd): 2023-12-31
2023-12-31에 일정이 있는 멤버를 출력합니다.
일정이 있는 멤버가 없습니다!
명령어를 입력하세요(일정추가|일정동참|일정보기|멤버조회|사용자전환|종료):
```

5. 계획 대비 변경 사항

1) 일정 출력 방식 축소

- 이전

일정을 리스트형과 달력형을 선택하여 출력하려고 하였다.

- 이후

일정 출력을 달력형을 제외한 리스트형만 지원하도록 하였다.

- 사유

GUI를 사용하지 않고 콘솔창에서 출력되는 점을 감안하였다. 콘솔창에서 달력을 문자열로 구현하기에는 결과물이 다소 난잡하고, 일정의 이름이 조금만 길어져도 달력 내에 채워 넣을 수 없는 등 표출할 수 있는 정보에 한계가 있어 큰 효용이 없을 것이라 판단했다.

6. 느낀점

객체 지향 언어인 C++로 프로젝트를 진행하면서, 좀 더 객체 지향적인 프로그래밍에 대해 고민해보게 되었다. 캡슐화를 어느 정도로 할 것인지, 객체에 포함시킬지 메인 함수에서 조작할 것인지를 선택하는 기준이나 고려 사항들에 대해 알고 싶어졌다. 또한, 프로젝트를 시작하기 이전에 클래스 구조를 충분히 고민해보고 코딩을 시작해야겠다는 교훈도 얻었다.

보고서를 작성하면서 스스로 요구사항들을 만들어 보았는데, 개발 과정에서 요구사항을 변경하거나 축소하게 될 경우들에 대해 고려해보게 되었고, 요구사항에 맞게 기능들을 구현함에 있어 내가 할 수 있는 정도인지에 대해 가늠하는 것도 중요하다는 것을 알게되었다.

또한, 테스트의 중요성에 대해 절감하게 되었다. 테스트 없이 무작정 기능들을 구현하다 보면, 에러가 발생했을 때 어느 부분에서 발생한 에러인지에 대해 파악하는 것이 정말 어려울 수 있겠다는 생각이 들었다. 기능 단위의 테스트를 반드시 해야겠다는 것과 테스트 케이스 작성의 중요성을 절감했다.

내가 작성한 코드를 남에게 설명하는 보고서를 작성하면서 개발자는 프로그래밍

역량 못지 않게 문서 작성 역량이 요구된다는 점을 알게 되었고, 그러한 부분에서 부족한 점을 많이 느낄 수 있었다.

마지막으로, 내가 주력으로 사용하는 언어는 Java인지라 C++로 프로젝트 단위의 개발은 처음 해보았는데, 나름 프로젝트를 진행하면서 C++의 문법과 Java와의 차이점들에 대해 많이 익숙해진 것 같아 성취감을 느꼈다. 열정적으로 강의해주신 교수님께도 매우 감사드린다.