c++프로그래밍 및 실습

하루 노트

진척 보고서 #번호

제출일자:

제출자명: 송 단

제출자학번: 215302

1. 프로젝트 목표

1) 배경 및 필요성

대학생활을 하면서 제한된 자원(돈, 시간 등)의 효율적인 분배의 필요성에 대해 느끼게 되었습니다. 강의, 과제, 시험, 아르바이트와 같은 학업 및 생활 일정을 비롯하여, 경제적 부담과 체력 관리까지 포함된 대학생의 일상은 계획과 적절한 자원의 분배 없이는 수면 부족, 시간부족, 경제적 어려움 등 다양한 문제에 직면할 수 있습니다. 이 문제를 해결하기 위해 시간 관리, 재정 관리, 그리고 개인 기록 기능이 통합된 하나의 프로그램이 필요하다고 생각했습니다.

(하루노트: 하루를 계획하고 기록하는 의미를 담은 노트)

2) 프로젝트 목표

하루의 시간과 경제적 계획을 보기 좋게 정리하고, 알림 기능을 통해 계획에 준수할 수 있게 돕는 것을 목표로 합니다.

3) 차별점

기존 프로그램들은 시간 관리 또는 재정 관리 중 하나에만 집중되어 있어, 하루에 대한 기록을 위해 여러 개의 프로그램을 사용해야 하는 불편함이 있습니다. 반면, 하루노트는 시간 관리, 재정 관리 기능을 하나의 프로그램에서 통합하여 제공한다는 점에서 차별점이 있습니다.

2. 기능 계획

1) 캘린더

- 달력에 간략한 일정 표기

(1) 캘린더 표기

- 달력에 간략한 일정 표기한다.

(2) 일정 추가

- 간략한 일정 입력 받아 저장한다.

(3) 일정 삭제

- 간략한 일정을 입력 받아 삭제한다.

2) 플래너

- 일 단위의 상세한 일정 또는 메모 표기

(1) 플래너 표기

- 상세 일정 표기을 표기한다.

(2) 일정 추가

- 상세한 일정 입력 받아 저장한다.

(3) 일정 삭제

- 상세한 일정을 입력 받아 삭제한다.

3) 가계부

-수입 및 소비 금액 관리하여 표기

(1) 가계부 표기

-수입 및 소비 내용 표기한다.

(2) 수입 및 소비 내용 추가

-수입 및 소비 내용을 입력 받아 추가한다.

(3) 수입 및 소비 내용 삭제

-수입 및 소비 내용을 입력 받아 삭제한다.

4) 암호화 노트

- 암호를 풀어야 볼 수 있는 노트

(1) 노트 표기

- 노트 표기하는 기능

(2) 세부 기능 1(암호 설정)

- 암호 설정하는 기능

(3) 세부 기능 2(암호 해제)

- 암호 해제하는 기능

3. 진척사항

1) 기능 구현

Schedule 클래스(일정 관리 클래스, 일 단위)

```h
#pragma once
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

class Schedule {
private:
    // 일정 벡터
    vector<string> schedule;
public:
    // 일정 추가
    void AddSchedule(string detail);

    // 일정 삭제
    void DelSchedule(string detail);

    // 일정vector 반환
    vector<string> GetScheduleVector();

    // index번째 일정 반환
    string GetScheduleString(int index);

    // 일정 개수 반환
    int CountSchedule();

    // 일정이 비어있는지 확인
    bool EmptySchedule();
};
```

```cpp
// 일정 추가
void Schedule::AddSchedule(string detail) {
    schedule.push_back(detail);
}
```

1. 입력
   · schedule: 일정 벡터
   · detail: 일정에 넣을 문자열
2. 결과
   · 일정을 일정 목록에 추가
3. 설명
   · detail(문자열)을 schedule(벡터)에 추가
4. 활용된 개념
   · 벡터, 함수, 클래스

```cpp
// 일정 삭제
void Schedule::DelSchedule(string detail) {
    schedule.erase(remove(schedule.begin(), schedule.end(), detail), schedule.end());
}
```

1. 입력
   · schedule: 일정 벡터
   · detail: 삭제할 일정 문자열
2. 결과
   · 일정을 일정 목록에 삭제
3. 설명
   · remove: detail(문자열)와 동일한 일정을 찾아 제일 뒤로 보내고 그것들을 제외한 벡터의 끝을 반환합니다.
   · erase: remove의 반환값으로부터 원래 schedule(벡터)의 끝까지의 항목을 삭제합니다.
4. 활용된 개념
   · 벡터, 함수, 클래스

<table>
<tr>
<td>

```
// index번째 일정 반환
string Schedule::GetScheduleString(int index) {
    return schedule[index];
}
```

</td>
<td>
1. 입력
· schedule: 일정 벡터
· index: 반환 받을 벡터의 인덱스값 변수

2. 결과
· schedule의 index번째 항목을 반환함.

3. 설명
· schedule의 index번째 항목을 반환함

4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
// 일정vector 반환
vector<string> Schedule::GetScheduleVector() {
    return schedule;
}
```

</td>
<td>
1. 입력
· schedule: 일정 벡터

2. 결과
· schedule 벡터를 반환함.

3. 설명
· schedule의 벡터를 반환함.

4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
// 일정 개수 반환
int Schedule::CountSchedule() {
    return schedule.size();
}
```

</td>
<td>
1. 입력
· schedule: 일정 벡터

2. 결과
· schedule의 원소 개수를 반환함.

3. 설명
· size: schedule의 원소 개수를 반환함.

4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
// 일정이 비어있는지 확인
bool Schedule::EmptySchedule() {
    return schedule.empty();
}
```

</td>
<td>
1. 입력
· schedule: 일정 벡터

2. 결과
· schedule가 비어있는지 여부를 bool값으로 반환

3. 설명
· empty: schedule가 비어있는지 여부를 bool값으로 반환함

4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
</table>

Transaction 클래스(거래 내역 관리 클래스, 일 단위)

```
class Transaction {
private:
    // 거래 상세내용 벡터
    vector<string> transactionDetail;
    // 거래 가격 벡터
    vector<string> transactionPrice;
public:
    // 거래 내역 추가
    void AddTransaction(string detail, int price);

    // 거래 내역 삭제
    void DelTransaction(string detail, string price);

    // 거래 상세내용 벡터 반환
    vector<string> GetDetailVector();

    // 거래 가격 벡터 반환
    vector<string> GetPriceVector();

    // index번째 거래 상세내용 반환
    string GetDetailString(int index);

    // index번째 거래 가격 반환
    string GetPriceString(int index);

    // 거래 개수 반환
    int CountTransaction();

    // 거래이 비어있는지 확인
    bool EmptyTransaction();
};
```

```
//거래 내역 추가
void Transaction::AddTransaction(string detail, int price) {
    transactionDetail.push_back(detail);
    transactionPrice.push_back(to_string(price));
}
```

1. 입력
   · transactionDetail: 거래 상세 내역 벡터
   · trasactionPrice: 거래 가격 벡터
   · detail: 추가할 거래 상세 내역
   · price: 추가할 거래 가격
2. 결과
   · 거래 상세 내역과 가격을 목록에 추가
3. 설명
   · detail(문자열)을 transactionDetail(벡터)에 추가
   · price(정수)을 transactionPrice(벡터)에 문자열로 추가
4. 활용된 개념
   · 벡터, 함수, 클래스

```
// 거래 내역 삭제
void Transaction::DelTransaction(string detail, string price) {
    transactionDetail.erase(remove(transactionDetail.begin(), transactionDetail.end(), detail), transactionDetail.end());
    transactionPrice.erase(remove(transactionPrice.begin(), transactionPrice.end(), price), transactionPrice.end());
}
```

1. 입력
   · transactionDetail: 거래 상세 내역 벡터
   · trasactionPrice: 거래 가격 벡터
   · detail: 삭제할 거래 상세 내역
   · price: 삭제할 거래 가격
2. 결과
   · 거래 상세 내역과 가격을 목록에 추가
3. 설명
   · detail(문자열)을 transactionDetail(벡터)에 추가
   · price(정수)을 transactionPrice(벡터)에 문자열로 추가
4. 활용된 개념
   · 벡터, 함수, 클래스

```
// 거래 상세내용 벡터 반환
vector<string> Transaction::GetDetailVector() {
    return transactionDetail;
}
```

1. 입력
   · transactionDetail: 거래 상세 내역 벡터
2. 결과
   · transactionDetail 벡터를 반환함.
3. 설명
   · transactionDetail의 벡터를 반환함.
4. 활용된 개념
   · 벡터, 함수, 클래스

<table>
<tr>
<td>

```
// 거래 가격 벡터 반환
vector<string> Transaction::GetPriceVector() {
    return transactionPrice;
}
```

</td>
<td>
1. 입력
· trasactionPrice: 거래 가격 벡터
2. 결과
· trasactionPrice 벡터를 반환함.
3. 설명
· trasactionPrice의 벡터를 반환함.
4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
// index번째 거래 상세내용 반환
string Transaction::GetDetailString(int index) {
    return transactionDetail[index];
}
```

</td>
<td>
1. 입력
· transactionDetail: 거래 상세 내역 벡터
· index: 반환 받을 벡터의 인덱스값 변수
2. 결과
· transactionDetail의 index번째 항목을 반환함.
3. 설명
· transactionDetail의 index번째 항목을 반환함
4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
// index번째 거래 가격 반환
string Transaction::GetPriceString(int index) {
    return transactionPrice[index];
}
```

</td>
<td>
1. 입력
· trasactionPrice: 거래 가격 벡터
· index: 반환 받을 벡터의 인덱스값 변수
2. 결과
· trasactionPrice의 index번째 항목을 반환함.
3. 설명
· trasactionPrice의 index번째 항목을 반환함
4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
//일정 개수 반환
int Transaction::CountTransaction() {
    return transactionDetail.size();
}
```

</td>
<td>
1. 입력
· transactionDetail: 거래 상세 내역 벡터
2. 결과
· transactionDetail의 원소 개수를 반환함.
3. 설명
· size: transactionDetail의 원소 개수를 반환함.
4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
//일정이 비어있는지 확인
bool Transaction::EmptyTransaction() {
    return transactionDetail.empty();
}
```

</td>
<td>
1. 입력
· transactionDetail: 거래 상세 내역 벡터
2. 결과
· transactionDetail가 비어있는지 여부를 bool값으로 반환
3. 설명
· empty: transactionDetail가 비어있는지 여부를 bool값으로 반환함
4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
</table>

Date 클래스(날짜 관리 클래스)

```cpp
#pragma once
#include <chrono>
#include <string>
using namespace std;

class Date {
public:
    static const int MAX_YEARS;             // 최대 연도수
    static const int MAX_MONTHS;            // 최대 개월수
    static const int MAX_DAYS;              // 최대 일수
    static const int kDays[12];             // 각 월의 최대 일수
    static const int kDaysLeafYear[12];     // 각 월의 최대 일수
    static const string kWeekdays[7];       // 요일 배열
    int current_year;                       // 현재 연도 변수
    int current_month;                      // 현재 월 변수
    int current_day;                        // 현재 일 변수
    int initial_year;                       // 0번 인덱스의 연도(시작 연도)
    int max_year_index = 0;                 // 프로그램 시작 시 할당할 연도의 인덱스 저장하는 변수

    //생성자
    Date();

    //윤년 체크(true: 윤년/ flase: 윤년x)
    bool CheckLeapYear(int year);

    //날짜 유효한지 체크(true: 날짜 유효/ flase: 날짜 유효x)
    bool CheckRange(int year, int month, int day);

    //해당 날짜의 요일 반환(Zeller의 공식)(0=일요일, 1=월요일, ..., 6=토요일)
    int GetDayOfWeek(int year, int month, int day);
};
```

```cpp
//static const 변수 초기화
const int Date::MAX_YEARS = 100;
const int Date::MAX_MONTHS = 12;
const int Date::MAX_DAYS = 31;
const int Date::kDays[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
const int Date::kDaysLeafYear[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
const string Date::kWeekdays[7] = {"일요일", "월요일", "화요일", "수요일", "목요일", "금요일", "토요일"};
```

```cpp
// 생성자(현재 시간 저장, 연도 단위 날짜 할당, 환경 설정 파일 가져오기)
Date::Date() {
    // 현재 시간 가져오기
    time_t t = time(nullptr);           // 현재 시간을 time_t 타입으로 얻기
    tm* now = localtime(&t);            // 현재 시간을 tm 구조체 포인터로 변환

    // 현재 년, 월, 일을 변수에 저장
    current_year = now->tm_year + 1900;  // tm_year는 1900년부터 시작하므로 1900을 더해 실제 연도 계산
    current_month = now->tm_mon + 1;     // tm_mon은 0부터 시작하므로 1을 더해 실제 월 계산
    current_day = now->tm_mday;          // 오늘의 일(day)
}
```

1. 입력
   · current_year: 현재 연도 변수
   · current_month: 현재 월 변수
   · current_day: 현재 일 변수
2. 결과
   · 현재 날짜 정보를 각 변수에 저장함.
3. 설명
   · chrono헤더 파일을 통해 time_t타입으로 현재 날짜를 구함. 그 후 각 연도, 월, 일을 각 변수에 저장함.
4. 활용된 개념
   · 클래스, 포인터, 생성자

```cpp
// 윤년 체크(true: 윤년/ flase: 윤년x)
bool Date::CheckLeapYear(int year) {
    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
        return true;
    return false;
}
```

1. 입력
   · year: 윤년인지 판단하고 싶은 연도 변수
2. 결과
   · 해당 연도가 윤년일 경우 true를 반환함.
3. 설명
   · if문을 통해 윤년일 경우 true를 반환하게 함.
4. 활용된 개념
   · 조건문, 함수, 클래스

```cpp
// 날짜 유효한지 체크(true: 날짜 유효/ flase: 날짜 유효x)
bool Date::CheckRange(int year, int month, int day) {
    if(initial_year <= year && year <= initial_year + 99) {     // 연도 체크
        if(1 <= month  && month <= MAX_MONTHS) {                // 월 체크
            if (CheckLeapYear(year)){                           // 윤년 판단(일 체크)
                if(1 <= day && day <= kDaysLeafYear[month-1])
                    return true;
            }
            else {
                if(1 <= day && day <= kDays[month-1])
                    return true;
            }
        }
    }
    return false;
}
```

1. 입력
   · year: 유효한지 판단할 연도 변수
   · month: 유효한지 판단할 월 변수
   · day: 유효한지 판단할 일 변수
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · MAX_MONTHS: 월의 최댓값 상수(12)
   · kDaysLeafYear: 윤년일 때 각 달의 최대 일수 배열
   · kDays: 윤년이 아닐 때 각 달의 최대 일수 배열
2. 결과
   · 날짜가 유효하면 true를 반환함.
3. 설명
   · if문을 통해 연도가 유효한지 판단.
   · if문을 통해 월이 유효한지 판단.
   · if문을 통해 윤년인지 판단하여 사용할 배열 선택
   · if문을 통해 일이 유효한지 판단.
4. 활용된 개념
   · 조건문, 함수, 클래스, 배열

```cpp
// 해당 날짜의 요일 반환(Zeller의 공식)(0=일요일, 1=월요일, ..., 6=토요일)
int Date::GetDayOfWeek(int year, int month, int day) {
    // 1월과 2월을 13월, 14월로 처리
    if (month < 3) {
        month += 12;
        year--;
    }
    int K = year % 100;         // 연도의 마지막 두 자리
    int J = year / 100;         // 연도의 첫 두 자리

    // Zeller의 공식
    int h = (day + (13 * (month + 1)) / 5 + K + K / 4 + J / 4 + 5 * J) % 7;
    h = (h + 6) % 7;            // h가 음수일 경우 양수로 변환

    return h;                   // 요일 인덱스값 반환
}
```

1. 입력
   · year: 요일을 판단할 연도 변수
   · month: 요일을 판단할 월 변수
   · day: 요일을 판단할 일 변수
2. 결과
   · 해당 날짜의 요일 인덱스값을 반환.
   (0=일, 1=월, ... , 6= 토)
3. 설명
   · zeller의 공식을 이용하여 요일을 판단함.
4. 활용된 개념
   · 조건문, 함수, 클래스

DateManagement 클래스(파일 관리 클래스, 싱글턴)

```hpp
#pragma once
#include <iostream>
#include <fstream>
#include <regex>
#include "DataType.hpp"
#include "Date.hpp"

class DataManagement: public Date {
private:
    Schedule*** calendar;           // 날짜별 Schedule 객체를 담을 3차원 배열의 포인터
    Schedule*** planner;            // 날짜별 Schedule 객체를 담을 3차원 배열의 포인터
    Transaction*** accountBook;     // 날짜별 Transaction 객체를 담을 3차원 배열의 포인터

    // 생성자(Date 생성자 호출, 연도 단위 날짜 할당, 환경 설정 파일 가져오기)(Singleton)
    DataManagement();
public:
    int year_index = -1;            // 현재 할당된 연도의 인덱스의 최댓값을 저장할 변수

    // 싱글턴 객체 반환
    static DataManagement& GetInstance();

    //calendar배열 반환
    Schedule*** GetArrayCalendar();

    //planner배열 반환
    Schedule*** GetArrayPlanner();

    //accountBook배열 반환
    Transaction*** GetArrayAccountBook();

    // 날짜 범위 1년 증가(다음 연도 할당)
    void AddYear();

    // 환경 설정 파일 저장
    void SaveConfig();

    // 환경 설정 파일 불러오기
    void LoadConfig();

    // 데이터 파일 저장
    void SaveToFile();

    // 데이터 파일 불러오기
    void LoadFromFile();
};

// 시작과 끝의 공백을 제거하는 함수
string Trim(const string& str);
int StringLength(const string& str);
```

4. 활용된 개념
   · 클래스, 싱글턴

```cpp
// 생성자(현재 시간 저장, 연도 단위 날짜 할당, 환경 설정 파일 가져오기)
DataManagement::DataManagement(): Date() {
    // 환경 설정 파일 가져오기
    initial_year = current_year;
    LoadConfig();

    // 초기 연도 할당
    if(current_year - initial_year > max_year_index)
        max_year_index = current_year - initial_year;

    calendar = new Schedule**[MAX_YEARS];
    planner = new Schedule**[MAX_YEARS];
    accountBook = new Transaction**[MAX_YEARS];
    while(year_index < max_year_index){
        AddYear();
    }

    //일정 불러오기
    LoadFromFile();
}
```

1. 입력
   · current_year: 현재 연도 변수
   · current_month: 현재 월 변수
   · current_day: 현재 일 변수
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
   · year_index: 현재 할당된 연도의 인덱스의 최댓값 변수
   · calendar: 캘린더의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · planner: 플래너의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · acountBook: 캘린더의 날짜별 Transaction(거래) 객체를 담을 3차원 포인터 배열

2. 결과
   · 현재 날짜 정보를 각 변수에 저장함.
   · 환경 설정 정보, 일정 데이터를 불러옴.
   · 환경 설정 정보에 따라 시작 시 연도 할당

3. 설명
   · Date 생성자를 호출하여 현재 날짜 정보를 각 변수에 저장함.
   · LoadConfig()를 통해 환경 설정 파일을 받아옴.
   · if문을 통해 현재 날짜가 할당된 날짜를 넘어간다면 AddYear메소드를 통해 연도 할당
   · 환경 설정 정보에 따라 할당된 연도의 개수를 판단하고 AddYear함수를 통해사용 중인 연도까지만 할당함.
   · LoadFromFile함수를 통해 일정 데이터를 불러옴

4. 활용된 개념
   · 조건문, 반복문, 함수, 클래스, 포인터, 동적할당

```cpp
//calendar배열 반환
Schedule*** DataManagement::GetArrayCalendar() {
    return calendar;
}
// 싱글턴 객체 반환
DataManagement& DataManagement::GetInstance() {
    static DataManagement instance;  // 메서드 내부에서 static 인스턴스 생성
    return instance;
}
```

1. 입력
   · calendar: 캘린더의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열

2. 결과
   · calendar 반환함

3. 설명
   · calendar 반환함

4. 활용된 개념
   · 벡터, 함수, 클래스

<table>
<tr>
<td>

```
//planner배열 반환
Schedule*** DataManagement::GetArrayPlanner() {
    return planner;
}
```

</td>
<td>
1. 입력
· planner: 플래너의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
2. 결과
· planner 반환함
3. 설명
· planner 반환함
4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
//accountBook배열 반환
Transaction*** DataManagement::GetArrayAccountBook() {
    return accountBook;
}
```

</td>
<td>
1. 입력
· acountBook: 캘린더의 날짜별 Transaction(거래) 객체를 담을 3차원 포인터 배열
2. 결과
· acountBook 반환함
3. 설명
· acountBook 반환함
4. 활용된 개념
· 벡터, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
// 달력 범위 1년 증가(다음 연도 할당)
void DataManagement::AddYear() {
    // 캘린더
    calendar[++year_index] = new Schedule*[MAX_MONTHS];          // 다음 연도에 대한 월 배열 생성
    for (int i = 0; i < MAX_MONTHS; i++)                         // 다음 연도의 월에 대한 일 배열 생성
        calendar[year_index][i] = new Schedule[MAX_DAYS];

    // 플래너
    planner[year_index] = new Schedule*[MAX_MONTHS];             // 다음 연도에 대한 월 배열 생성
    for (int i = 0; i < MAX_MONTHS; i++)                         // 다음 연도의 월에 대한 일 배열 생성
        planner[year_index][i] = new Schedule[MAX_DAYS];

    // 가계부
    accountBook[year_index] = new Transaction*[MAX_MONTHS];      // 다음 연도에 대한 월 배열 생성
    for (int i = 0; i < MAX_MONTHS; i++)                         // 다음 연도의 월에 대한 일 배열 생성
        accountBook[year_index][i] = new Transaction[MAX_DAYS];
}
```

</td>
<td>
1. 입력
· calendar: 캘린더의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
· planner: 플래너의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
· acountBook: 캘린더의 날짜별 Transaction(거래) 객체를 담을 3차원 포인터 배열
· year_index: 현재 할당된 연도의 인덱스의 최댓값 변수
· MAX_MONTHS: 월의 최댓값 상수(12)
· MAX_DAYS: 일의 최댓값 상수(31)
2. 결과
· 다음 연도 할당함.
3. 설명
· yearIndex를 1 증가
· calendar의 다음 인덱스에 크기가 12인 2차원 객체 배열을 할당함. (월)
· 위의 할당한 2차원 객체 배열에 크기가 31인 객체 배열을 할당함. (일)
· 위의 할당 과정으로 planner, accountBook도 배열을 할당함.
4. 활용된 개념
· 반복문, 함수, 클래스, 포인터, 동적할당
</td>
</tr>
<tr>
<td>

```
//환경 설정 파일 저장
void DataManagement::SaveConfig() {
    ofstream configFile("config");                              //config파일을 쓰기 모드로 열기

    if (configFile.is_open()) {                                 //파일이 정상적으로 열렸을 경우
        configFile << initial_year << " " << max_year_index;    //intitial_year maxYearIndex 저장
        configFile.close();                                     //파일 닫기
    } else {
        cout << "설정 파일을 저장할 수 없습니다." << endl;
    }
}
```

</td>
<td>
1. 입력
· initial_year: 프로그램을 처음 시작한 연도 변수
· max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
2. 결과
· initial_year, max_year_index 환경 설정 관련 변수를 파일로 저장함.
3. 설명
· 파일을 쓰기 모드로 열음
· if문을 통해 파일이 정상적으로 열렸는지 판단함.
· 파일이 정상적으로 열렸다면, initial_year, max_year_index를 파일에서 저장한 후 파일을 닫음.
· 파일이 정상적으로 열리지 않았다면 문구를 띄움.
4. 활용된 개념
· 조건문, 함수, 클래스
</td>
</tr>
<tr>
<td>

```
//환경 설정 파일 불러오기
void DataManagement::LoadConfig() {
    ifstream configFile("config");                              //config파일을 읽기 모드로 열기

    if (configFile.is_open()) {                                 //파일이 정상적으로 열렸을 경우
        configFile >> initial_year >> max_year_index;           //intitial_year, maxYearIndex 읽어오기
        configFile.close();                                     //파일 닫기
    } else {
        cout << "설정 파일이 없으므로 기본값을 사용합니다." << endl;
    }
}
```

</td>
<td>
1. 입력
· initial_year: 프로그램을 처음 시작한 연도 변수
· max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
2. 결과
· initial_year, max_year_index 환경 설정 관련 변수를 파일에서 읽어옴.
3. 설명
· 파일을 읽기 모드로 열음.
· if문을 통해 파일이 정상적으로 열렸는지 판단함.
· 파일이 정상적으로 열렸다면, initial_year, max_year_index를 파일에서 읽어온 후 파일을 닫음.
· 파일이 정상적으로 열리지 않았다면 문구를 띄움.
4. 활용된 개념
· 조건문, 함수, 클래스
</td>
</tr>
</table>

<table>
<tr>
<td>

```
// 데이터 파일 저장
void DataManagement::SaveToFile() {
    string fileName[3] = {"CalendarData", "PlannerData", "AccountBookData"};
    // CalendarData, PlannerData 저장
    for(int i = 0; i < 2; i++) {
        ofstream outFile(fileName[i]);
        Schedule*** data;
        if(i == 0)
            data = calendar;
        else
            data = planner;
        //파일이 정상적으로 열렸을 경우
        if (outFile.is_open()) {
            for (int y = 0; y <= max_year_index; ++y) {
                for (int m = 0; m < MAX_MONTHS; ++m) {
                    for (int d = 0; d < MAX_DAYS; ++d) {
                        // 해당 날짜의 일정이 비어있지 않다면 저장
                        if (!data[y][m][d].EmptySchedule()) {
                            // year month day schedule 형식의 저장
                            for (const string& schedule : data[y][m][d].GetScheduleVector()) {
                                outFile << (initial_year + y) << " "    // Year
                                        << (m + 1) << " "               // Month
                                        << (d + 1) << " "               // Day
                                        << schedule << "\n";            // Schedule details
                            }
                        }
                    }
                }
            }
            outFile.close();
        }
        //파일이 정상적으로 열리지 않았을 경우
        else {
            cout << "파일을 열 수 없습니다." << endl;
        }
    }
    // AccountBookData 저장
    ofstream outFile(fileName[2]);
    if (outFile.is_open()) {
        for (int y = 0; y <= max_year_index; ++y) {
            for (int m = 0; m < MAX_MONTHS; ++m) {
                for (int d = 0; d < MAX_DAYS; ++d) {
                    // 해당 날짜의 일정이 비어있지 않다면 저장
                    if (!accountBook[y][m][d].EmptyTransaction()) {
                        // year month day schedule 형식로 저장
                        for (int i = 0; i < accountBook[y][m][d].CountTransaction(); i++) {
                            outFile << (initial_year + y) << " "    // Year
                                    << (m + 1) << " "               // Month
                                    << (d + 1) << " "               // Day
                                    << accountBook[y][m][d].GetPriceString(i) << " "    // Transaction Price
                                    << accountBook[y][m][d].GetDetailString(i) << "\n";  // Transaction Detail
                        }
                    }
                }
            }
        }
        outFile.close();
    }
    //파일이 정상적으로 열리지 않았을 경우
    else {
        cout << "파일을 열 수 없습니다." << endl;
    }
}
```

</td>
<td>

1. 입력
   · calendar: 캘린더의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · planner: 플래너의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · acountBook: 캘린더의 날짜별 Transaction(거래) 객체를 담을 3차원 포인터 배열
   · max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
   · MAX_MONTHS
   · MAX_DAYS
   · initial_year: 프로그램을 처음 시작한 연도 변수
2. 결과
   · 날짜와 일정, 거래내역을 파일에 저장함.
3. 설명
   · 파일을 쓰기 모드로 열음.
   · if문을 통해 파일이 정상적으로 열렸는지 판단함.
   · 파일이 정상적으로 열렸다면, 반복문으로 각 날짜를 순회하여 날짜와 일정을 저장한다.
   · 파일이 정상적으로 열리지 않았다면 문구를 띄움.
   · 이 과정으로 calendar, planner, accountBook 데이터를 저장함.
4. 활용된 개념
   · 조건문, 반복문, 배열, 함수, 클래스, 벡터

</td>
</tr>
<tr>
<td>

```
// 파일에서 데이터를 불러오기
void DataManagement::LoadFromFile() {
    string fileName[3] = {"CalendarData", "PlannerData", "AccountBookData"};
    // CalendarData, PlannerData 불러오기
    for(int i = 0; i < 2; i++) {
        ifstream inFile(fileName[i]);
        Schedule*** data;
        if(i == 0)
            data = calendar;
        else
            data = planner;
        //파일이 정상적으로 열렸을 경우
        if (inFile.is_open()) {
            int year, month, day;                                   // 날짜 정보를 저장할 변수
            string detail;                                          // 일정을 저장할 변수

            // 파일에서 한 줄씩 읽으면서 연도, 월, 일, 세부 내용을 처리
            while (inFile >> year >> month >> day) {                // 연도, 월, 일 읽어오기
                inFile.ignore();                                    // 한 칸 건너띄어 공백 무기
                getline(inFile, detail);                            // 일정 읽어오기
                data[year-initial_year][month - 1][day - 1].AddSchedule(detail); // 읽어온 일정 추가
            }

            inFile.close();
        }
        //파일이 정상적으로 열리지 않았을 경우
        else {
            cout << "파일을 열 수 없습니다." << endl;
        }
    }

    // AccountBookData 불러오기
    ifstream inFile(fileName[2]);
    //파일이 정상적으로 열렸을 경우
    if (inFile.is_open()) {
        int year, month, day, price;                                // 날짜 정보를 저장할 변수
        string detail;                                              // 일정을 저장할 변수

        // 파일에서 한 줄씩 읽으면서 연도, 월, 일, 세부 내용을 처리
        while (inFile >> year >> month >> day >> price) {           // 연도, 월, 일 읽어오기
            inFile.ignore();                                        // 한 칸 건너띄어 공백 무기
            getline(inFile, detail);                                // 일정 읽어오기
            accountBook[year-initial_year][month - 1][day - 1].AddTransaction(detail, price); // 읽어온 일정 추가
        }

        inFile.close();
    }
    //파일이 정상적으로 열리지 않았을 경우
    else {
        cout << "파일을 열 수 없습니다." << endl;
    }
}
```

</td>
<td>

1. 입력
   · calendar: 캘린더의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · planner: 플래너의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · acountBook: 캘린더의 날짜별 Transaction(거래) 객체를 담을 3차원 포인터 배열
   · max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
2. 결과
   · 날짜와 일정을 파일에서 읽어옴.
3. 설명
   · 파일을 읽기 모드로 열음.
   · if문을 통해 파일이 정상적으로 열렸는지 판단함.
   · 파일이 정상적으로 열렸다면, 반복문으로 한 줄씩 날짜와 일정을 읽어와서 해당 날짜의 일정에 추가함.
   · 파일이 정상적으로 열리지 않았다면 문구를 띄움.
   · 이 과정으로 calendar, planner, accountBook 데이터를 불러옴.
4. 활용된 개념
   · 조건문, 반복문, 배열, 함수, 클래스, 벡터

</td>
</tr>
<tr>
<td colspan="2">함수</td>
</tr>
<tr>
<td>

```
// 시작과 끝의 공백을 제거하는 함수
string Trim(const string& str) {
    // 정규 표현식을 사용하여 시작과 끝의 공백 제거
    regex ws_re("^\\s+|\\s+$");
    return regex_replace(str, ws_re, "");
}
```

</td>
<td>

1. 입력
   · string: 공백을 제거할 문자열
2. 결과
   · 문자열의 시작과 끝의 공백을 제거함
3. 설명
   · 문자열의 시작의 공백 또는 문자열의 끝의 공백을 찾는 정규 표현식을 정의함.
   · 정규 표현식으로 찾은 것들을 빈 문자열로 대체하여 반환
4. 활용된 개념
   · 함수, 문자열

</td>
</tr>
<tr>
<td>

```
// 문자열 길이 반환 함수
int StringLength(const string& str) {
    int count_ascii = 0;
    int count_korean = 0;
    for(char c: str) {
        if(0 <= c && c <= 127)
            count_ascii++;
        else
            count_korean++;
    }
    return count_ascii + count_korean*2/3;
}
```

</td>
<td>

1. 입력
   · string: 공백을 제거할 문자열
2. 결과
   · 문자열의 시작과 끝의 공백을 제거함
3. 설명
   · for문으로 아스키 문자와 그렇지 않은 문자를 카운트함.
   · 글자수를 계산하여 반환함.
4. 활용된 개념
   · 함수, 문자열

</td>
</tr>
</table>

1. 캘린더

```
#pragma once
#include "Planner.hpp"
#include "AccountBook.hpp"

class Calendar {
private:
    DataManagement* bd;
    Schedule*** date;
    int current_year, current_month, current_day;
    Planner* plan;
    AccountBook* accountBook;
public:
    //생성자(Date 생성자 호출)
    Calendar();

    // 캘린더 일정 추가
    void AddSchedule(int year_s, int month_s, int day_s, int year_e, int month_e, int day_e, string detail);

    // 캘린더 일정 삭제
    void DelSchedule(int year_s, int month_s, int day_s, int year_e, int month_e, int day_e, string detail);

    // 캘린더 표시
    void PrintCalendar(int year, int month, int day = 1);

    // 일정 출력
    void PrintSchedule(int year, int month, int day_s, int day_e, int space_position = 0);

    // 가계부 일의 합계 출력
    void PrintTotle(int year, int month, int day_s, int day_e, int space_position = 0);

    //메뉴
    void Menu() ;

};
```

```
// 생성자(달력 해당 연도만 할당)
Calendar::Calendar() {
    bd = &DataManagement::GetInstance();
    date = bd->GetArrayCalendar();
    current_year = bd->current_year;
    current_month = bd->current_month;
    current_day = bd->current_day;
    plan = new Planner{};
    accountBook = new AccountBook{};
}
```

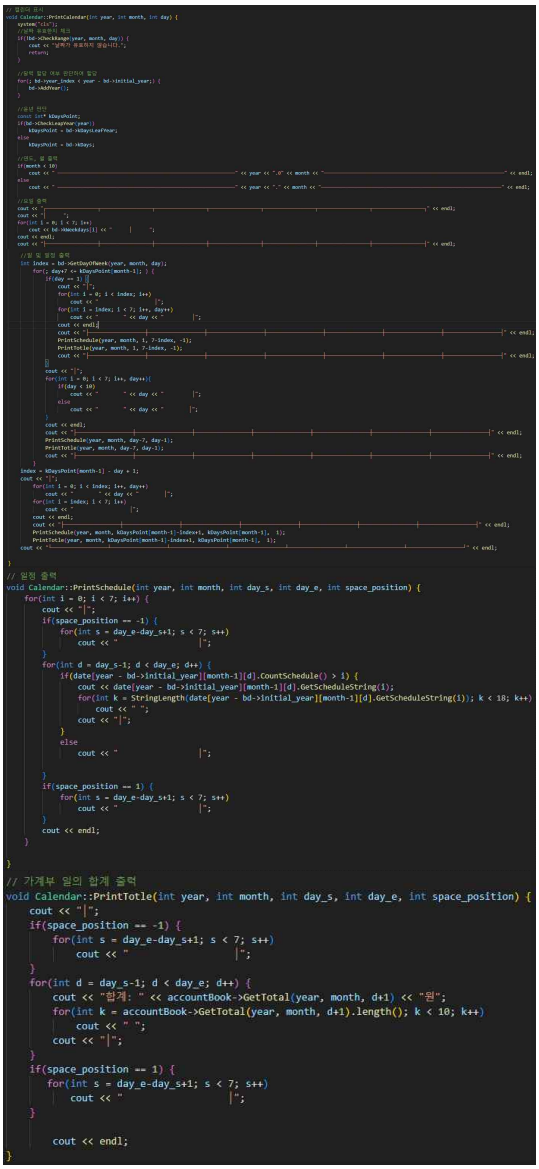
1. 입력
   · bd: DataManagement 싱글턴 객체를 담을 포인터 변수
   · date: 캘린더의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · currentYear: 현재 연도 변수
   · currentMonth: 현재 월 변수
   · currentDay: 현재 일 변수
   · plan: 플래너 객체를 담을 포인터 변수
   · accountBook: 가계부 객체를 담을 포인터 변수

2. 결과
   · 각종 변수 초기화 및 동적할당

3. 설명
   · DataMangement의 GetInstance()메소드를 통해 싱글턴 객체를 받아 주소값을 bd에 넣음.
   · bd의 GetArrayCalendar()메소드를 통해 date에 bd의 calendar배열을 받아 넣음.
   · bd의 현재 날짜를 받아와 변수에 저장함.
   · Planner, AccountBook 객체를 동적할당함.

4. 활용된 개념
   · 클래스, 생성자, 동적할당, 싱글턴, 포인터

1) 캘린더 표시



1. 입력
   · year: 캘린더를 표시할 연도
   · month: 캘린더를 표시할 월
   · day: 캘린더를 표시할 일
   · date: 캘린더의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · year_index: 현재 할당된 연도 인덱스의 최댓값 변수
   · kDaysLeafYear: 윤년일 때 각 달의 최대 일수 배열
   · kDays: 윤년이 아닐 때 각 달의 최대 일수 배열
   · kWeekdays: 요일 문자열이 들어있는 배열
   · day_s: 시작 날짜의 일
   · day_e: 종료 날짜의 일
   · space_position: 달력을 출력할 때 공백의 위치를 파악하기 위한 변수(-1: 공백 앞, +1: 공백 뒤)
   · GetDayOfWeek, CheckRange, CheckLeapYear, (Date 클래스 참고)
   · CountSchedule, GetScheduleString, GetScheduleVector (Schedule 클래스 참고)
   · AddYear (DataManagement 클래스 참고)

2. 결과
   · 입력 받은 날짜의 캘린더를 표시함.

3. 설명
   · if문과 CheckRange을 통해 날짜가 유효한지 판단.
   · if문으로 현재 날짜가 할당됐는지 판단하고 할당.
   · if문과 CheckLeapYear을 통해 윤년인지 판단하고 사용할 각 월별 최대 일수 배열을 선택한다.
   · 연도와 월을 출력한다.
   · 반복문을 사용하여 요일을 출력한다.
   · 일 및 일정, 가계부를 출력하는데 3단계로 나눈다. (PrintSchedule, PrintTotle에 공백 변수로 다르게 제공.)
      · 1단계: 첫 주(앞에 공백이 존재)
      · 2단계: 첫 주와 마지막주를 제외한 주
      · 3단계: 마지막 주(뒤에 공백이 존재

4. 활용된 개념
   · 조건문, 반복문, 배열, 함수, 클래스, 벡터

2) 일정 추가

```
// 캘린더 일정 추가
void Calendar::AddSchedule(int year_s, int month_s, int day_s, int year_e, int month_e, int day_e, string detail){
    // 날짜 유효성 체크(유효하지 않으면 함수 종료)
    if(!bd->CheckRange(year_s, month_s, day_s) || !bd->CheckRange(year_e, month_e, day_e)) {
        cout << "날짜가 유효하지 않습니다.";
        return;
    }
    // 시작 날짜 < 종료 날짜가 유효한지 체크
    if (year_s > year_e ||
    (year_s == year_e && month_s > month_e) ||
    (year_s == year_e && month_s == month_e && day_s > day_e)) {
    cout << "날짜가 유효하지 않습니다.";
    return;
}

    // 프로그램 시작 시 설정이 있는 날짜까지만 할당하기 위해 변수값 변경 및 달력 할당 여부 판단하여 할당
    if(bd->max_year_index < year_e - bd->initial_year) {
        for(; bd->year_index < year_e - bd->initial_year; bd->max_year_index++) {
            bd->AddYear();
        }
        // 프로그램 시작 시 설정이 있는 날짜까지만 할당하기 위해 변수값 변경
        bd->max_year_index = year_e - bd->initial_year;
    }

    //일정 추가
    if(year_s == year_e) {
        if(month_s == month_e) {
            //시작과 끝의 연도, 월이 같을 경우
            for(int d = day_s-1; d < day_e; d++) {
                date[year_s - bd->initial_year][month_s-1][d].AddSchedule(detail);
            }
        }
        else {
            //시작과 끝의 연도가 같고 월이 다를 경우
            for(int d = day_s-1; d < bd->kDaysLeafYear[month_s-1]; d++)                 //시작 달
                date[year_s - bd->initial_year][month_s-1][d].AddSchedule(detail);
            for(int m = month_s; m < month_e - 1; m++) {                                //중간 달
                for(int d = 0;  d < bd->kDaysLeafYear[m]; d++)
                    date[year_s - bd->initial_year][m][d].AddSchedule(detail);
            }
            for(int d = 0; d < day_e; d++)                                              //끝 달
                date[year_s - bd->initial_year][month_e-1][d].AddSchedule(detail);
        }
    }
    else {
        //시작과 끝의 연도가 다를 경우
        //시작 연도
        for(int d = day_s-1; d < bd->kDaysLeafYear[month_s-1]; d++)                     //시작 연도의 시작 달
            date[year_s - bd->initial_year][month_s-1][d].AddSchedule(detail);
        for(int m = month_s; m < bd->MAX_MONTHS; m++) {                                 //시작 연도의 나머지 달
            for(int d = 0;  d < bd->kDaysLeafYear[m]; d++)
                date[year_s - bd->initial_year][m][d].AddSchedule(detail);
        }
        //중간 연도
        for(int y = year_s - bd->initial_year + 1; y < year_e - bd->initial_year; y++){         //중간 연도의 전체
            for(int m = 0; m < bd->MAX_MONTHS; m++) {
                for(int d = 0; d < bd->kDaysLeafYear[m]; d++)
                    date[y][m][d].AddSchedule(detail);
            }
        }
        //마지막 연도
        for(int m = 0; m < month_e - 1; m++) {                                          //마지막 연도의 마지막 달 전까지의 달
            for(int d = 0;  d < bd->kDaysLeafYear[m]; d++)
                date[year_e - bd->initial_year][m][d].AddSchedule(detail);
        }
        for(int d = 0; d < day_e; d++)                                                  //마지막 연도의 끝 달
            date[year_e - bd->initial_year][month_e-1][d].AddSchedule(detail);
    }
    bd->SaveToFile();
    bd->SaveConfig();
}
```

1. 입력
   · date: 캘린더의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · year_s: 일정의 시작 날짜의 연도 변수
   · month_s: 일정의 시작 날짜의 월 변수
   · day_s: 일정의 시작 날짜의 일 변수
   · year_e: 일정의 종료 날짜의 연도 변수
   · month_e: 일정의 종료 날짜의 월 변수
   · day_e: 일정의 종료 날짜의 일 변수
   · detail: 일정 문자열을 저장할 변수
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
   · CheckRange (Date 클래스 참고)
   · AddSchedule (Schedule 클래스 참고)
   · AddYear, SaveToFile, SaveConfig (Date 클래스 참고)

2. 결과
   · 시작 날짜와 종료 날짜, 일정을 입력받아 해당 날짜들에 추가함.

3. 설명
   · if문과 CheckRange을 통해 날짜가 유효한지 판단.
   · if문으로 시작 날짜보다 종료날짜가 더 뒤인지 판단.
   · if문으로 종료 날짜까지 데이터 공간 할당되어 있는지 판단하고 할당되어 있지 않다면 for문으로 할당.
   · 일정 추가는 3가지로 분류하여 추가
   1) 1개의 월 안에서 끝나는 일정 추가
      · for문을 통해 day만 바꾸어 추가한다.
   2) 1개의 연도 안에서 끝나는 일정 추가
      · 시작 달: day_s ~ 마지막 일까지 일정 추가
      · 중간 달: 1 ~ 마지막 일까지 일정 추가
      · 마지막 달: 1 ~ day_s까지 일정을 추가한다.
   3) 그 외의 경우
      · 시작 연도
         · 시작 달: day_s ~ 마지막일까지 일정 추가
         · 그 외의 달: 1 ~ 마지막일까지 일정 추가
      · 중간 연도: 모든 날에 일정 추가
      · 마지막 연도
         · 마지막 전까지의 달: 1 ~ 마지막일까지 일정 추가
         · 마지막 달: 1 ~ day_e까지 일정 추가
   · 일정 데이터와 환경 설정 파일을 저장함.

4. 활용된 개념
   · 조건문, 반복문, 배열, 함수, 클래스, 벡터

3) 일정 삭제

```
// 캘린더 일정 삭제
void Calendar::DelSchedule(int year_s, int month_s, int day_s, int year_e, int month_e, int day_e, string detail) {
    // 날짜 유효성 체크(유효하지 않으면 함수 종료)
    if(!bd->CheckRange(year_s, month_s, day_s) || !bd->CheckRange(year_e, month_e, day_e)) {
        cout << "날짜가 유효하지 않습니다.";
        return;
    }
    // 시작 날짜 < 종료 날짜가 유효한지 체크
    if (year_s > year_e ||
    (year_s == year_e && month_s > month_e) ||
    (year_s == year_e && month_s == month_e && day_s > day_e)) {
    cout << "날짜가 유효하지 않습니다.";
    return;
}

    //끝 연도가 할당된 연도보다 범위가 클 경우 할당된 범위로 변경
    if(year_e - bd->initial_year > bd->max_year_index)
        year_e = bd->max_year_index + bd->initial_year;

    //일정 삭제
    if(year_s == year_e) {
        if(month_s == month_e) {
            //시작과 끝의 연도, 월이 같을 경우
            for(int d = day_s-1; d < day_e; d++) {
                date[year_s - bd->initial_year][month_s-1][d].DelSchedule(detail);
            }
        }
        else {
            //시작과 끝의 연도가 같고 월이 다를 경우
            for(int d = day_s-1; d < bd->kDaysLeafYear[month_s-1]; d++)                 //시작 달
                date[year_s - bd->initial_year][month_s-1][d].DelSchedule(detail);
            for(int m = month_s; m < month_e - 1; m++) {                                //중간 달
                for(int d = 0;  d < bd->kDaysLeafYear[m]; d++)
                    date[year_s - bd->initial_year][m][d].DelSchedule(detail);
            }
            for(int d = 0; d < day_e; d++)                                              //끝 달
                date[year_s - bd->initial_year][month_e-1][d].DelSchedule(detail);
        }
    }
    else {
        //시작과 끝의 연도가 다를 경우
        //시작 연도
        for(int d = day_s-1; d < bd->kDaysLeafYear[month_s-1]; d++)                     //시작 연도의 시작 달
            date[year_s - bd->initial_year][month_s-1][d].DelSchedule(detail);
        for(int m = month_s; m < bd->MAX_MONTHS; m++) {                                 //시작 연도의 나머지 달
            for(int d = 0;  d < bd->kDaysLeafYear[m]; d++)
                date[year_s - bd->initial_year][m][d].DelSchedule(detail);
        }
        //중간 연도
        for(int y = year_s - bd->initial_year + 1; y < year_e - bd->initial_year; y++){         //중간 연도의 전체
            for(int m = 0; m < bd->MAX_MONTHS; m++) {
                for(int d = 0; d < bd->kDaysLeafYear[m]; d++)
                    date[y][m][d].DelSchedule(detail);
            }
        }
        //마지막 연도
        for(int m = 0; m < month_e - 1; m++) {                                          //마지막 연도의 마지막 달 전까지의 달
            for(int d = 0;  d < bd->kDaysLeafYear[m]; d++)
                date[year_e - bd->initial_year][m][d].DelSchedule(detail);
        }
        for(int d = 0; d < day_e; d++)                                                  //마지막 연도의 끝 달
            date[year_e - bd->initial_year][month_e-1][d].DelSchedule(detail);
    }
    bd->SaveToFile();
    bd->SaveConfig();
}
```

1. 입력
   · date: 캘린더의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · year_s: 일정의 시작 날짜의 연도 변수
   · month_s: 일정의 시작 날짜의 월 변수
   · day_s: 일정의 시작 날짜의 일 변수
   · year_e: 일정의 종료 날짜의 연도 변수
   · month_e: 일정의 종료 날짜의 월 변수
   · day_e: 일정의 종료 날짜의 일 변수
   · detail: 일정 문자열을 저장할 변수
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
   · CheckRange (Date 클래스 참고)
   · DelSchedule (Schedule 클래스 참고)
   · AddYear, SaveToFile, SaveConfig (Date 클래스 참고)

2. 결과
   · 시작 날짜와 종료 날짜, 일정을 입력받아 해당 날짜들에서 삭제함.

3. 설명
   · if문과 CheckRange을 통해 날짜가 유효한지 판단.
   · if문으로 시작 날짜보다 종료날짜가 더 뒤인지 판단.
   · if문으로 할당된 날짜보다 종료날짜가 크다면 범위를 할당된 날짜로 지정함.
   · 일정 삭제는 3가지로 분류하여 추가
   1) 1개의 월 안에서 끝나는 일정 추가
      · for문을 통해 day만 바꾸어 삭제
   2) 1개의 연도 안에서 끝나는 일정 추가
      · 시작 달: day_s ~ 마지막 일까지 일정 삭제
      · 중간 달: 1 ~ 마지막 일까지 일정 삭제
      · 마지막 달: 1 ~ day_s까지 일정을 삭제
   3) 그 외의 경우
      · 시작 연도
         · 시작 달: day_s ~ 마지막일까지 일정 삭제
         · 그 외의 달: 1 ~ 마지막일까지 일정 삭제
      · 중간 연도: 모든 날에 일정 삭제
      · 마지막 연도
         · 마지막 전까지의 달: 1 ~ 마지막일까지 일정 삭제
         · 마지막 달: 1 ~ day_e까지 일정 삭제
   · 일정 데이터와 환경 설정 파일을 저장함.

4. 활용된 개념
   · 조건문, 반복문, 배열, 함수, 클래스, 벡터

4) 메뉴

```
// 메뉴
void Calendar::Menu() {
    string input;
    system("cls");
    PrintCalendar(current_year, current_month);
    while(true) {
        cout << "날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: ";
        getline(cin, input);
        input.erase(std::remove(input.begin(), input.end(), ' '), input.end());
        if(input == "날짜이동") {
            int year, month;
            cout << "0000 00 형태로 연도와 월을 입력해주세요: ";
            cin >> year >> month;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            PrintCalendar(year, month);
        }
        else if(input == "일정추가") {
            int year_s, month_s, day_s;
            int year_e, month_e, day_e;
            string detail;
            cout << "시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: ";
            cin >> year_s >> month_s >> day_s;
            cout << "종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: ";
            cin >> year_e >> month_e >> day_e;
            cout << "1라인 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): ";
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            getline(cin, detail);
            detail = Trim(detail);
            AddSchedule(year_s, month_s, day_s, year_e, month_e, day_e, detail);
            PrintCalendar(year_s, month_s);
        }
        else if(input == "일정삭제") {
            int year_s, month_s, day_s;
            int year_e, month_e, day_e;
            string detail;
            cout << "삭제할 일정의 시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: ";
            cin >> year_s >> month_s >> day_s;
            cout << "삭제할 일정의 종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: ";
            cin >> year_e >> month_e >> day_e;
            cout << "1라인 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): ";
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            getline(cin, detail);
            detail = Trim(detail);
            DelSchedule(year_s, month_s, day_s, year_e, month_e, day_e, detail);
            PrintCalendar(year_s, month_s);
        }
        else if(input == "플래너") {
            plan->Menu();
            system("cls");
            PrintCalendar(current_year, current_month);
        }
        else if(input == "암호화메모장") {

        }
        else if(input == "가계부") {
            accountBook->Menu();
            system("cls");
            PrintCalendar(current_year, current_month);
        }
        else if(input == "종료") {
            cout << "종료합니다.";
            break;
        }
        else {
            cout << "잘못된 입력입니다.";
        }
    }
}
```

1. 입력
   · input: 실행할 작업의 문자열 변수
   · current_year: 프로그램 실행한 연도
   · current_month: 프로그램 실행한 월
   · PrintCalendar(): 캘린더 출력 함수
   · AddSchedule(): 일정 추가 함수
   · DelSchedule(): 일정 삭제 함수

2. 결과
   · 수행할 작업을 입력 받고 그에 해당하는 값들을 입력 받아 수행해줌.

3. 설명
   · 프로그램 실행 시점의 날짜로 캘린더를 출력함 (캘린더 출력 전에는 화면이 clear 됨.)
   · getline으로 수행할 작업을 입력 받음 (입력 받은 작업의 다루기 싶게 공백 모두 제거)
   · if문을 작업에 따라 입력받고 해당 함수를 호출.
   · 반복문으로 종료가 입력되기 전까지 반복.

4. 활용된 개념
   · 조건문, 반복문, 함수, 클래스

2. 플래너

```cpp
#pragma once
#include "DataManagement.hpp"

class Planner {
private:
    DataManagement* bd;                             // DataManagement 포인터
    Schedule*** date;                               // 날짜별 Schedule 객체를 담을 3차원 배열의 포인터
    int current_year, current_month, current_day;   // 현재 작업 중인 날짜 저장하는 변수
public:
    // 생성자(Date 생성자 호출)
    Planner();

    // 플래너 일정 추가
    void AddSchedule(string detail);

    // 플래너 일정 삭제
    void DelSchedule(string detail);

    // 플래너 표시
    void PrintPlanner();

    // 일정 출력
    void PrintSchedule();

    // 메뉴
    void Menu() ;
};
```

```cpp
// 생성자(Date 클래스 생성자 호출)
Planner::Planner() {
    bd = &DataManagement::GetInstance();
    date = bd->GetArrayPlanner();
    current_year = bd->current_year;
    current_month = bd->current_month;
    current_day = bd->current_day;
}
```

1. 입력
   · bd: DataManagement 싱글턴 객체를 담을 포인터 변수
   · date: 플래너의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · currentYear: 현재 연도 변수
   · currentMonth: 현재 월 변수
   · currentDay: 현재 일 변수
2. 결과
   · 각종 변수 초기화
3. 설명
   · DataMangement의 GetInstance()메소드를 통해 싱글턴 객체를 받아 주소값을 bd에 넣음.
   · bd의 GetArrayPlanner()메소드를 통해 date에 bd의 calendar배열을 받아 넣음.
   · bd의 현재 날짜를 받아와 변수에 저장함.
4. 활용된 개념
   · 클래스, 생성자, 싱글턴, 포인터

1) 플래너 표시

```cpp
//플래너 표시
void Planner::PrintPlanner() {
    // 날짜 유효한지 체크
    if(!bd->CheckRange(current_year, current_month, current_day)) {
        cout << "날짜가 유효하지 않습니다.";
        return;
    }

    // 현재 표시할 날짜가 할당이 되어 있는지 체크하고 할당
    for(; bd->year_index < current_year - bd->initial_year;) {
        bd->AddYear();
    }

    // 연도, 월, 일, 요일 출력
    if(current_month < 10) {
        if(current_day < 10) {
            cout << " ————— " << current_year
            << ".0" << current_month
            << ".0" << current_day
            << "(" << Date::kWeekdays[bd->GetDayOfWeek(current_year, current_month, current_day)] << ")—————" << endl;
        }
        else {
            cout << " ————— " << current_year
            << ".0" << current_month
            << "." << current_day
            << "(" << Date::kWeekdays[bd->GetDayOfWeek(current_year, current_month, current_day)] << ")—————" << endl;
        }
    }
    else {
        if(current_day < 10) {
            cout << " ————— " << current_year
            << "." << current_month
            << ".0" << current_day
            << "(" << Date::kWeekdays[bd->GetDayOfWeek(current_year, current_month, current_day)] << ")—————" << endl;
        }
        else {
            cout << " ————— " << current_year
            << "." << current_month
            << "." << current_day
            << "(" << Date::kWeekdays[bd->GetDayOfWeek(current_year, current_month, current_day)] << ")—————" << endl;
        }
    }

    // 일정 출력
    PrintSchedule();
}
// 일정 출력
void Planner::PrintSchedule() {
    // 위 경계선 출력
    cout << "┌———————————————————————————————————————————————————————————┐" << endl;

    // 일정 출력 및 오른쪽 공백 출력
    for(int i = 0; i < date[current_year-bd->initial_year][current_month][current_day].CountSchedule(); i++) {
        cout << "| " << date[current_year-bd->initial_year][current_month][current_day].GetScheduleString(i);
        for(int j = StringLength(date[current_year-bd->initial_year][current_month][current_day].GetScheduleString(i)); j < 40; j++)
            cout << " ";
        cout << "|" << endl;
    }

    // 아래쪽 공백 출력
    for(int i = date[current_year-bd->initial_year][current_month][current_day].CountSchedule(); i < 24; i++)
        cout << "|                                                           |" <<endl;

    // 아래 경계선 출력
    cout << "└———————————————————————————————————————————————————————————┘" << endl;
}
```

1. 입력
   · date: 플래너의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · current_year: 현재 작업 중인 날짜의 연도
   · current_month: 현재 작업 중인 날짜의 월
   · current_day: 현재 작업 중인 날짜의 일
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · year_index: 현재 할당된 연도 인덱스의 최댓값 변수
   · kWeekdays: 요일 문자열이 들어있는 배열
   · GetDayOfWeek, CheckRange (Date 클래스 참고)
   · CountSchedule, GetScheduleString (Schedule 클래스 참고)
   · AddYear (DateManagement 클래스 참고)
2. 결과
   · 입력 받은 날짜의 플래너를 출력함.
3. 설명
   · 날짜를 출력함
   · 상부 경계선을 출력함
   · 일정을 출력하고 if문과 for문으로 오른쪽 공백 출력
   · if문과 for문으로 아래쪽 공백 출력
   · 하부 경계선을 출력함.
4. 활용된 개념
   · 조건문, 반복문, 함수, 클래스

2) 일정 추가

```
// 플래너 일정 추가
void Planner::AddSchedule(string detail){
    // 프로그램 시작 시 일정이 있는 날짜까지만 할당하기 위해 변수값 변경
    if(bd->max_year_index < current_year - bd->initial_year)
        bd->max_year_index = current_year-bd->initial_year;

    // 일정 추가
    date[current_year-bd->initial_year][current_month][current_day].AddSchedule(detail);

    // 변경 사항 저장
    bd->SaveToFile();
    bd->SaveConfig();
}
```

1. 입력
   · detail: 일정 문자열을 저장할 변수
   · max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · current_year: 현재 작업 중인 날짜의 연도
   · current_month: 현재 작업 중인 날짜의 월
   · current_day: 현재 작업 중인 날짜의 일
   · date: 플래너의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · SaveToFile, SaveConfig (DateManagement 클래스 참고)
   · AddSchedule (Schedule 클래스 참고)
2. 결과
   · 현재 작업 중인 날짜의 플래너에 일정 추가
3. 설명
   · if문으로 max_year_index를 업데이트함.
   · 일정을 현재 작업 중인 날짜의 플래너에 추가
   · 일정 데이터 파일, 환경 설정 파일 저장
4. 활용된 개념
   · 조건문, 배열, 함수, 클래스

3) 일정 삭제

```
//플래너 일정 삭제
void Planner::DelSchedule(string detail) {
    // 일정 삭제
    date[current_year-bd->initial_year][current_month][current_day].DelSchedule(detail);

    // 변경 사항 저장
    bd->SaveToFile();
    bd->SaveConfig();
}
```

1. 입력
   · detail: 일정 문자열을 저장할 변수
   · max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · current_year: 현재 작업 중인 날짜의 연도
   · current_month: 현재 작업 중인 날짜의 월
   · current_day: 현재 작업 중인 날짜의 일
   · date: 플래너의 날짜별 Schedule(일정) 객체를 담을 3차원 포인터 배열
   · SaveToFile, SaveConfig (DateManagement 클래스 참고)
   · DelSchedule (Schedule 클래스 참고)
2. 결과
   · 현재 작업 중인 날짜의 플래너에 일정 삭제
3. 설명
   · 일정을 현재 작업 중인 날짜의 플래너에 삭제
   · 일정 데이터 파일, 환경 설정 파일 저장
4. 활용된 개념
   · 조건문, 배열, 함수, 클래스

4) 메뉴

```
// 메뉴(사용자 입력 받아 작업 실행)
void Planner::Menu() {
    string input;
            system("cls");
            PrintPlanner();
    while(true) {
        // 날짜 이동, 일정 추가, 일정 삭제, 캘린더 중 하나 사용자 입력 받기
        cout << "날짜 이동, 일정 추가, 일정 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): ";
        getline(cin, input);
        input.erase(std::remove(input.begin(), input.end(), ' '), input.end());
        // 날짜 이동
        if(input == "날짜이동") {
            cout << "0000 00 00형태로 연도, 월, 일을 입력해주세요: ";
            cin >> current_year >> current_month >> current_day;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            system("cls");
            PrintPlanner();
        }
        // 일정 추가
        else if(input == "일정추가") {
            string detail;
            cout << "40글자 이내로 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): ";
            getline(cin, detail);
            detail = Trim(detail);
            AddSchedule(detail);
            system("cls");
            PrintPlanner();
        }
        // 일정 삭제
        else if(input == "일정삭제") {
            string detail;
            cout << "40글자 이내로 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): ";
            getline(cin, detail);
            detail = Trim(detail);
            DelSchedule(detail);
            system("cls");
            PrintPlanner();
        }
        // 뒤로 가기
        else if(input == "뒤로가기") {
            break;
        }
        // 잘못된 입력
        else {
            cout << "잘못된 입력입니다.";
        }
    }
}
```

1. 입력
   · input: 실행할 작업의 문자열 변수
   · current_year: 현재 작업 중인 날짜의 연도
   · current_month: 현재 작업 중인 날짜의 월
   · current_day: 현재 작업 중인 날짜의 일
   · PrintPlanner(): 플래너 출력 함수
   · AddSchedule(): 일정 추가 함수
   · DelSchedule(): 일정 삭제 함수
2. 결과
   · 수행할 작업을 입력 받고 그에 해당하는 값들을 입력 받아 수행해줌.
3. 설명
   · 프로그램 실행 시점의 날짜로 플래너를 출력함 (플래너 출력 전에는 화면이 clear 됨.)
   · getline으로 수행할 작업을 입력 받음 (입력 받은 작업의 다루기 싶게 공백 모두 제거)
   · if문을 작업에 따라 입력받고 해당 함수를 호출.
   · 반복문으로 종료가 입력되기 전까지 반복.
4. 활용된 개념
   · 조건문, 함수, 클래스

3. 가계부

```
#pragma once
#include "DataManagement.hpp"

class AccountBook {
private:
    DataManagement* bd;                          // DataManagement 포인터
    Transaction*** date;                         // 날짜별 Schedule 객체를 담을 3차원 배열의 포인터
    int current_year, current_month, current_day;   // 현재 작업 중인 날짜 저장하는 변수
public:
    // 생성자(Date 생성자 호출)
    AccountBook();

    // 가계부 내역 추가
    void AddTransaction(string detail, int price);

    // 가계부 내역 삭제
    void DelTransaction(string detail, int price);

    // 합계 반환
    string GetTotal(int year, int month, int day);

    // 가계부 표시
    void PrintAccountBook();

    // 내역 출력
    void PrintTransaction();

    // 메뉴
    void Menu();
};
```

```
// 생성자(Date 생성자 호출)
AccountBook::AccountBook() {
    bd = &DataManagement::GetInstance();
    date = bd->GetArrayAccountBook();
    current_year = bd->current_year;
    current_month = bd->current_month;
    current_day = bd->current_day;
}
```

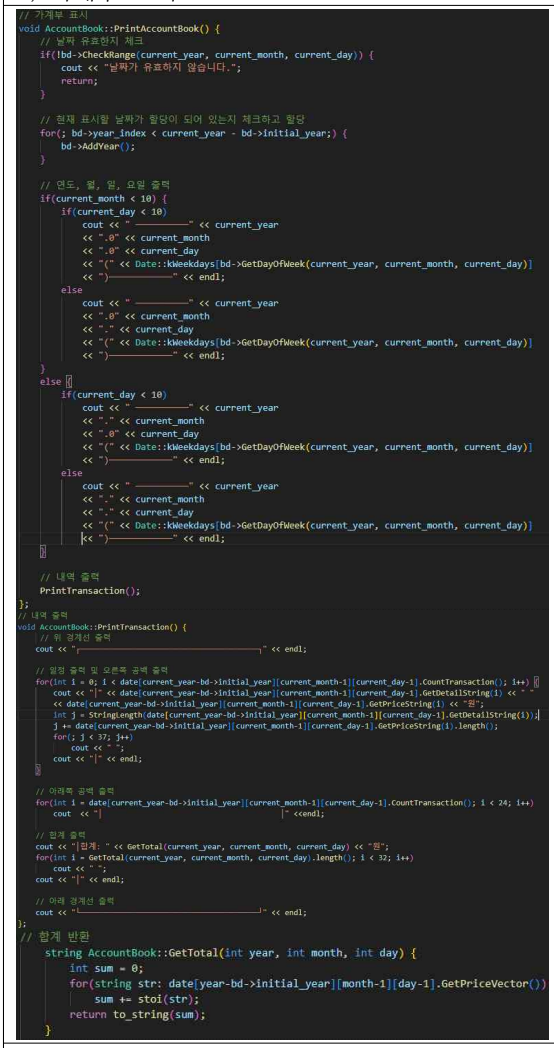
1. 입력
   · bd: DataManagement 싱글턴 객체를 담을 포인터 변수
   · date: 가계부의 날짜별 Transaction(일정) 객체를 담을 3차원 포인터 배열
   · currentYear: 현재 연도 변수
   · currentMonth: 현재 월 변수
   · currentDay: 현재 일 변수

2. 결과
   · 각종 변수 초기화

3. 설명
   · DataMangement의 GetInstance()메소드를 통해 싱글턴 객체를 받아 주소값을 bd에 넣음.
   · bd의 GetArrayAccountBook()메소드를 통해 date에 bd의 calendar배열을 받아 넣음.
   · bd의 현재 날짜를 받아와 변수에 저장함.

4. 활용된 개념
   · 클래스, 생성자, 싱글턴, 포인터

1) 가계부 표시



```
// 가계부 표시
void AccountBook::PrintAccountBook() {
    // 날짜 유효한지 체크
    if(!bd->CheckRange(current_year, current_month, current_day)) {
        cout << "날짜가 유효하지 않습니다.";
        return;
    }

    // 현재 표시할 날짜가 할당이 되어 있는지 체크하고 할당
    for(; bd->year_index < current_year - bd->initial_year;) {
        bd->AddYear();
    }

    // 연도, 월, 일, 요일 출력
    if(current_month < 10) {
        if(current_day < 10)
            cout << " ————————" << current_year
            << ".0" << current_month
            << ".0" << current_day
            << "(" << Date::kWeekdays[bd->GetDayOfWeek(current_year, current_month, current_day)]
            << ")————————" << endl;
        else
            cout << " ————————" << current_year
            << ".0" << current_month
            << "." << current_day
            << "(" << Date::kWeekdays[bd->GetDayOfWeek(current_year, current_month, current_day)]
            << ")————————" << endl;
    }
    else {
        if(current_day < 10)
            cout << " ————————" << current_year
            << "." << current_month
            << ".0" << current_day
            << "(" << Date::kWeekdays[bd->GetDayOfWeek(current_year, current_month, current_day)]
            << ")————————" << endl;
        else
            cout << " ————————" << current_year
            << "." << current_month
            << "." << current_day
            << "(" << Date::kWeekdays[bd->GetDayOfWeek(current_year, current_month, current_day)]
            << ")————————" << endl;
    }

    // 내역 출력
    PrintTransaction();
};
// 내역 출력
void AccountBook::PrintTransaction() {
    // 위 경계선 출력
    cout << "┌──────────────────────────────────────┐" << endl;

    // 일정 출력 및 오른쪽 공백 출력
    for(int i = 0; i < date[current_year-bd->initial_year][current_month-1][current_day-1].CountTransaction(); i++) {
        cout << "|" << date[current_year-bd->initial_year][current_month-1][current_day-1].GetDetailString(i) << " "
        << date[current_year-bd->initial_year][current_month-1][current_day-1].GetPriceString(i) << "원";
        int j = StringLength(date[current_year-bd->initial_year][current_month-1][current_day-1].GetDetailString(i));
        j += date[current_year-bd->initial_year][current_month-1][current_day-1].GetPriceString(i).length();
        for(; j < 32; j++)
            cout << " ";
        cout << "|" << endl;
    }

    // 아래쪽 공백 출력
    for(int i = date[current_year-bd->initial_year][current_month-1][current_day-1].CountTransaction(); i < 24; i++)
        cout  << "|                                    |" <<endl;

    // 합계 출력
    cout << "|합계: " << GetTotal(current_year, current_month, current_day) << "원";
    for(int i = GetTotal(current_year, current_month, current_day).length(); i < 32; i++)
        cout << " ";
    cout << "|" << endl;

    // 아래 경계선 출력
    cout << "└──────────────────────────────────────┘" << endl;
};
// 합계 반환
 string AccountBook::GetTotal(int year, int month, int day) {
     int sum = 0;
     for(string str: date[year-bd->initial_year][month-1][day-1].GetPriceVector())
         sum += stoi(str);
     return to_string(sum);
 }
```

1. 입력
   · date: 가계부의 날짜별 Transaction(일정) 객체를 담을 3차원 포인터 배열
   · current_year: 현재 작업 중인 날짜의 연도
   · current_month: 현재 작업 중인 날짜의 월
   · current_day: 현재 작업 중인 날짜의 일
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · year_index: 현재 할당된 연도 인덱스의 최댓값 변수
   · kWeekdays: 요일 문자열이 들어있는 배열
   · GetDayOfWeek, CheckRange (Date 클래스 참고)
   · CountTransaction, GetDetailString, GetPriceString, GetPriceVector (Schedule 클래스 참고)
   · AddYear (DateManagement 클래스 참고)

2. 결과
   · 입력 받은 날짜의 플래너를 출력함.

3. 설명
   · 날짜를 출력함
   · 상부 경계선을 출력함
   · 일정을 출력하고 if문과 for문으로 오른쪽 공백 출력
   · if문과 for문으로 아래쪽 공백 출력
   · 금일의 금액의 합계를 출력함.
   · 하부 경계선을 출력함.

4. 활용된 개념
   · 조건문, 반복문, 함수, 클래스

2) 수입 및 소비 내용 추가

```
// 가계부 내역 추가
void AccountBook::AddTransaction(string detail, int price) {
    // 프로그램 시작 시 일정이 있는 날짜까지만 할당하기 위해 변수값 변경
    if(bd->max_year_index < current_year - bd->initial_year)
        bd->max_year_index = current_year-bd->initial_year;

    // 일정 추가
    date[current_year-bd->initial_year][current_month-1][current_day-1].AddTransaction(detail, price);

    // 변경 사항 저장
    bd->SaveToFile();
    bd->SaveConfig();
}
```

1. 입력
   · detail: 거래 상세 내역 문자열을 저장할 변수
   · price: 거래 가격을 저장할 변수
   · max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · current_year: 현재 작업 중인 날짜의 연도
   · current_month: 현재 작업 중인 날짜의 월
   · current_day: 현재 작업 중인 날짜의 일
   · date: 가계부의 날짜별 Transaction(일정) 객체를 담을 3차원 포인터 배열
   · SaveToFile, SaveConfig (DateManagement 클래스 참고)
   · AddTransaction (Transaction 클래스 참고)

2. 결과
   · 현재 작업 중인 날짜의 가계부에 거래 내역 추가

3. 설명
   · if문으로 max_year_index를 업데이트함.
   · 일정을 현재 작업 중인 날짜의 가계부에 추가
   · 일정 데이터 파일, 환경 설정 파일 저장

4. 활용된 개념
   · 조건문, 배열, 함수, 클래스

3) 수입 및 소비 내용 삭제

```
// 가계부 내역 삭제
void AccountBook::DelTransaction(string detail, int price) {
    // 일정 삭제
    date[current_year-bd->initial_year][current_month-1][current_day-1].DelTransaction(detail, to_string(price));

    // 변경 사항 저장
    bd->SaveToFile();
    bd->SaveConfig();
};
```

1. 입력
   · detail: 거래 상세 내역 문자열을 저장할 변수
   · price: 거래 가격을 저장할 변수
   · max_year_index: 사용 중인 연도 인덱스의 최댓값 변수
   · initial_year: 프로그램을 처음 시작한 연도 변수
   · current_year: 현재 작업 중인 날짜의 연도
   · current_month: 현재 작업 중인 날짜의 월
   · current_day: 현재 작업 중인 날짜의 일
   · date: 가계부의 날짜별 Transaction(일정) 객체를 담을 3차원 포인터 배열
   · SaveToFile, SaveConfig (DateManagement 클래스 참고)
   · DelTransaction (Transaction 클래스 참고)
2. 결과
   · 현재 작업 중인 날짜의 가계부에 거래 내역 추가
3. 설명
   · if문으로 max_year_index를 업데이트함.
   · 일정을 현재 작업 중인 날짜의 가계부에 추가
   · 일정 데이터 파일, 환경 설정 파일 저장
4. 활용된 개념
   · 조건문, 배열, 함수, 클래스

4) 메뉴

```
// 메뉴
void AccountBook::Menu() {
    string input;
        system("cls");
        PrintAccountBook();
    while(true) {
        // 날짜 이동, 일정 추가, 일정 삭제, 캘린더 중 하나 사용자 입력 받기
        cout << "날짜 이동, 거래 내역 추가, 거래 내역 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): ";
        getline(cin, input);
        input.erase(std::remove(input.begin(), input.end(), ' '), input.end());
        // 날짜 이동
        if(input == "날짜이동") {
            cout << "0000 00 00형태로 연도, 월, 일을 입력해주세요: ";
            cin >> current_year >> current_month >> current_day;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            system("cls");
            PrintAccountBook();
        }
        // 일정 추가
        else if(input == "거래내역추가") {
            string detail;
            int price;
            cout << "30글자 이내로 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): ";
            getline(cin, detail);
            cout << "9글자 이내의 거래 가격을 입력해주세요: ";
            cin >> price;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            detail = Trim(detail);
            AddTransaction(detail, price);
            system("cls");
            PrintAccountBook();
        }
        // 일정 삭제
        else if(input == "거래내역삭제") {
            string detail;
            int price;
            cout << "30글자 이내로 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): ";
            getline(cin, detail);
            cout << "9글자 이내의 거래 가격을 입력해주세요: ";
            cin >> price;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            detail = Trim(detail);
            DelTransaction(detail,price);
            system("cls");
            PrintAccountBook();
        }
        // 뒤로 가기
        else if(input == "뒤로가기") {
            break;
        }
        // 잘못된 입력
        else {
            cout << "잘못된 입력입니다.";
        }
    }
};
```

1. 입력
   · input: 실행할 작업의 문자열 변수
   · current_year: 현재 작업 중인 날짜의 연도
   · current_month: 현재 작업 중인 날짜의 월
   · current_day: 현재 작업 중인 날짜의 일
   · PrintAccountBook(): 플래너 출력 함수
   · AddTransaction(): 일정 추가 함수
   · DelTransaction(): 일정 삭제 함수
2. 결과
   · 수행할 작업을 입력 받고 그에 해당하는 값들을 입력 받아 수행해줌.
3. 설명
   · 프로그램 실행 시점의 날짜로 가계부를 출력함 (가계부 출력 전에는 화면이 clear 됨.)
   · getline으로 수행할 작업을 입력 받음 (입력 받은 작업의 다루기 싶게 공백 모두 제거)
   · if문을 작업에 따라 입력받고 해당 함수를 호출.
   · 반복문으로 종료가 입력되기 전까지 반복.
4. 활용된 개념
   · 조건문, 반복문, 함수, 클래스

2) 테스트 결과

1. 캘린더

1) 캘린더 표시

2025.01

일요일	월요일	화요일	수요일	목요일	금요일	토요일
			1	2	3	4
			테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원
5	6	7	8	9	10	11
테스트3 합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
12	13	14	15	16	17	18
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
19	20	21	22	23	24	25
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
26	27	28	29	30	31	
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	

날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요:

2) 일정 추가

```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 일정 추가
시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 1
종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 4
18글자 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): 테스트1
```

2024.11

일요일	월요일	화요일	수요일	목요일	금요일	토요일
					1	2
					테스트1 합계: 0원	테스트1 합계: 0원
3	4	5	6	7	8	9
테스트1 합계: 0원	테스트1 합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원

```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 일정추가
시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 25
종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 12 5
18글자 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): 테스트2
```

2024.11

일요일	월요일	화요일	수요일	목요일	금요일	토요일
					1	2
					테스트1 합계: 0원	테스트1 합계: 0원
3	4	5	6	7	8	9
테스트1 합계: 0원	테스트1 합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
10	11	12	13	14	15	16
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
17	18	19	20	21	22	23
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
24	25	26	27	28	29	30
합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원

2024.12

일요일	월요일	화요일	수요일	목요일	금요일	토요일
1	2	3	4	5	6	7
테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원	합계: 0원	합계: 0원
8	9	10	11	12	13	14
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
15	16	17	18	19	20	21
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
22	23	24	25	26	27	28
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
29	30	31				
합계: 0원	합계: 0원	합계: 0원				

```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 일정 추가
시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 12 5
종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2025 1 5
18글자 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): 테스트3
```

2024.12

일요일	월요일	화요일	수요일	목요일	금요일	토요일
1	2	3	4	5	6	7
테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 합계: 0원	테스트2\ 테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원
8	9	10	11	12	13	14
테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원
15	16	17	18	19	20	21
테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원
22	23	24	25	26	27	28
테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원
29	30	31				
테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원				

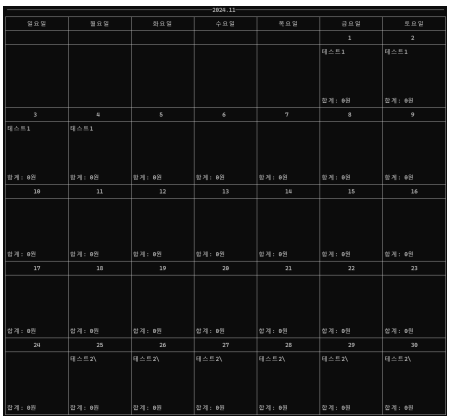
2025.01

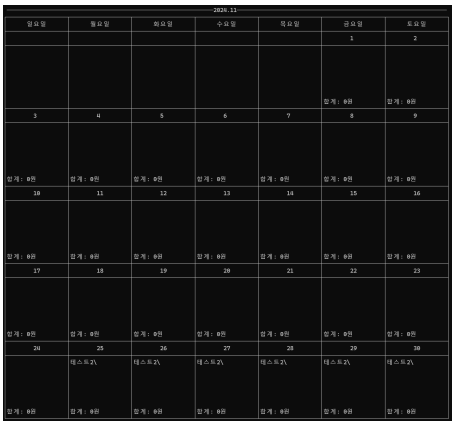
일요일	월요일	화요일	수요일	목요일	금요일	토요일
			1	2	3	4
			테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원
5	6	7	8	9	10	11
테스트3 합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
12	13	14	15	16	17	18
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
19	20	21	22	23	24	25
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
26	27	28	29	30	31	
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	

날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: |

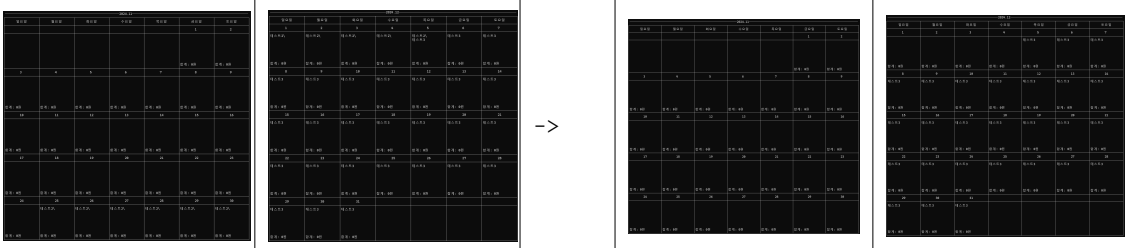
3) 일정 삭제

```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 일정 삭제
삭제할 일정의 시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 1
삭제할 일정의 종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 4
18글자 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): 테스트1
```

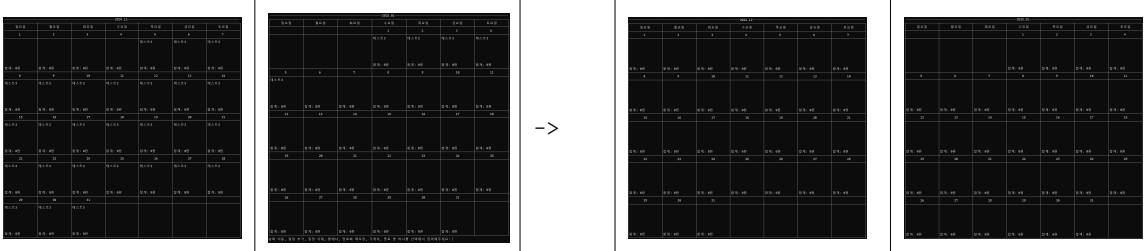




```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 일정 삭제
삭제할 일정의 시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 25
삭제할 일정의 종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 12 5
18글자 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): 테스트2\
```



```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 일정 삭제
삭제할 일정의 시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 12 5
삭제할 일정의 종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2025 1 5
18글자 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): 테스트3
```



4) 메뉴

```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 날짜이동
0000 00 형태로 연도와 월을 입력해주세요: 2025 1
```

2025.01

일요일	월요일	화요일	수요일	목요일	금요일	토요일
			1	2	3	4
			테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원	테스트3 합계: 0원
5	6	7	8	9	10	11
테스트3 합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
12	13	14	15	16	17	18
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
19	20	21	22	23	24	25
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
26	27	28	29	30	31	
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	

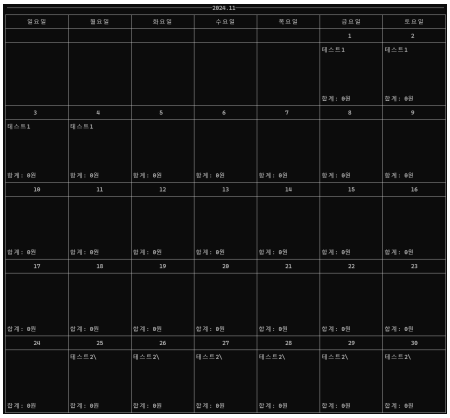
```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요:
```

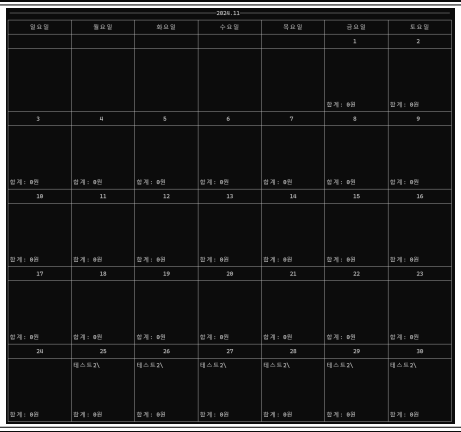
```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 일정 추가
시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 1
종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 4
18글자 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): 테스트1
```

2024.11

일요일	월요일	화요일	수요일	목요일	금요일	토요일
					1	2
					테스트1 합계: 0원	테스트1 합계: 0원
3	4	5	6	7	8	9
테스트1 합계: 0원	테스트1 합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원

```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 일정 삭제
삭제할 일정의 시작 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 1
삭제할 일정의 종료 날짜를 0000 00 00 형태로 연도, 월, 일을 입력해주세요: 2024 11 4
18글자 이내의 일정을 입력해주세요(한글: 2칸, 그 외: 1칸): 테스트1
```



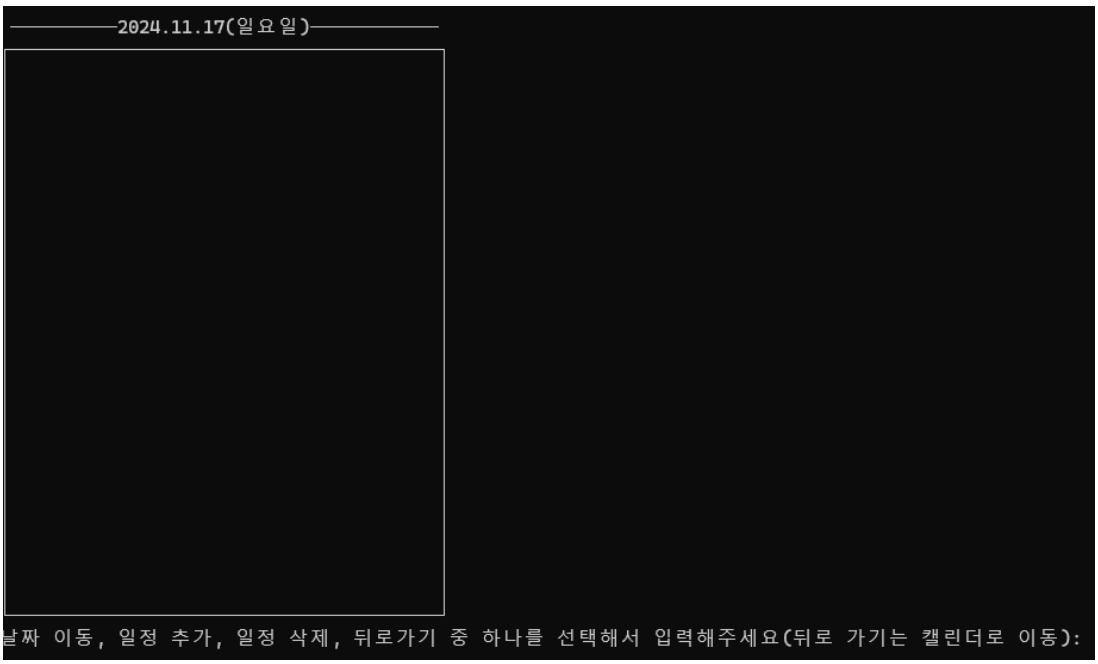



```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 플래너
──────────2024.11.17(일요일)──────────
┌──────────────────────────────────────┐
└──────────────────────────────────────┘
날짜 이동, 일정 추가, 일정 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동):
```

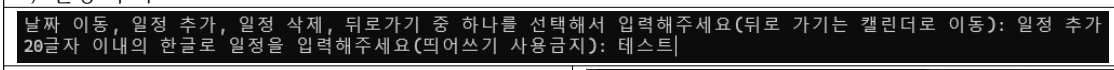
```
날짜 이동, 일정 추가, 일정 삭제, 플래너, 암호화 메모장, 가계부, 종료 중 하나를 선택해서 입력해주세요: 가계부
──────────2024.11.29(금요일)──────────
┌──────────────────────────────────────┐
│합계: 0원                             │
└──────────────────────────────────────┘
날짜 이동, 거래 내역 추가, 거래 내역 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동):
```

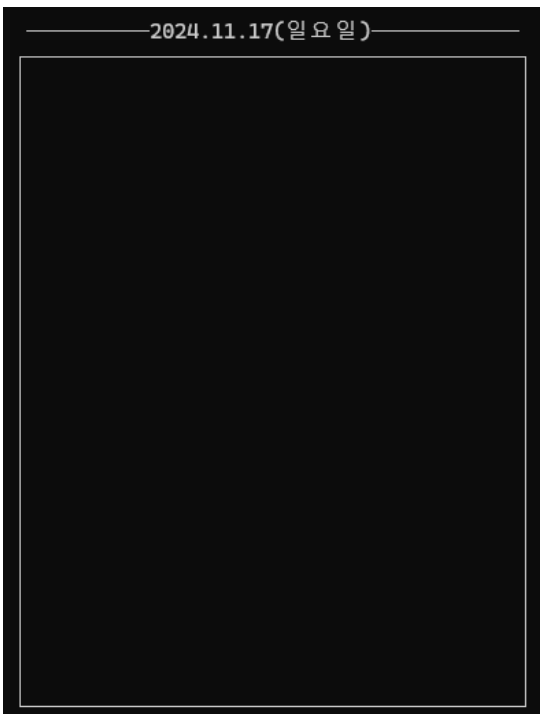
2. 플래너

1) 플래너 표시



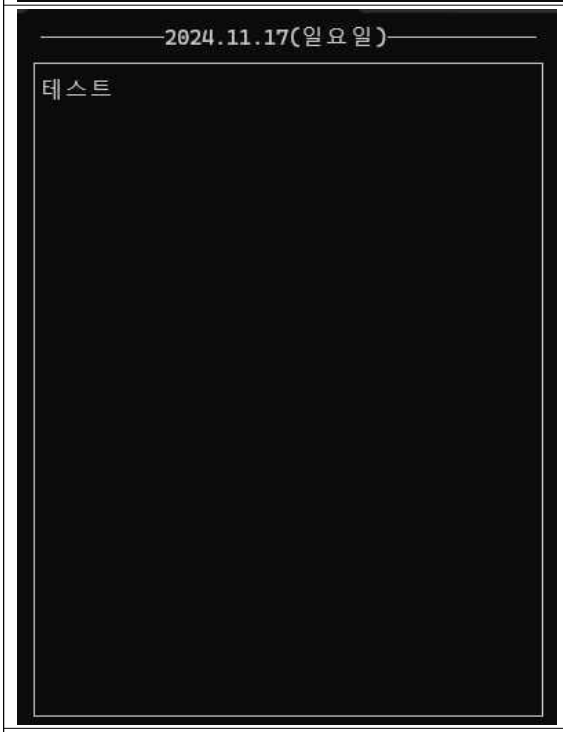
2) 일정 추가





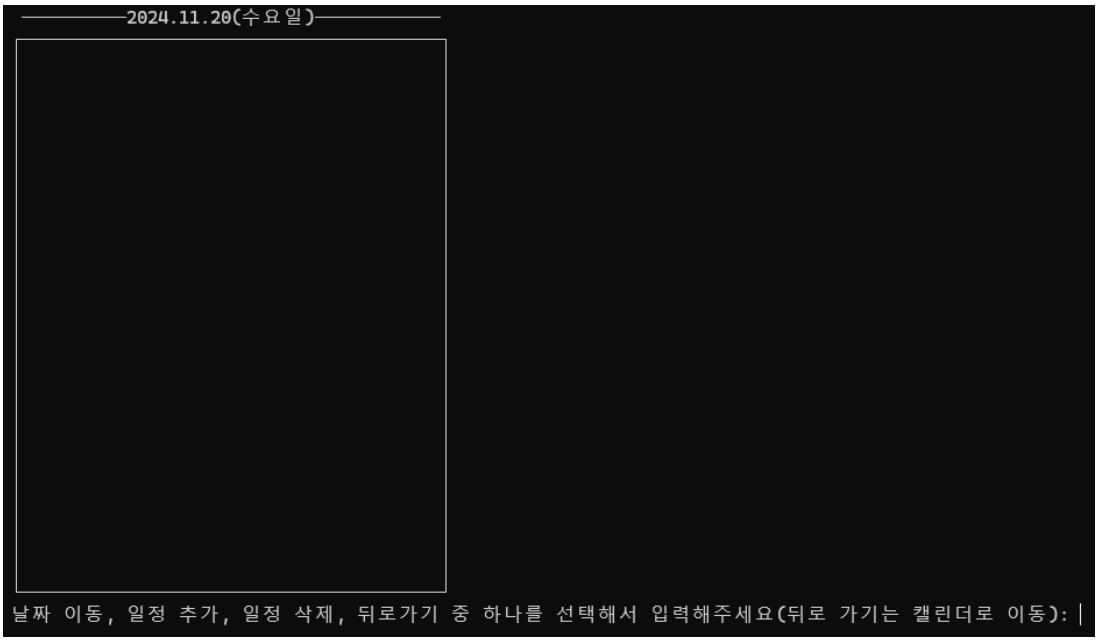
3) 일정 삭제

```
날짜 이동, 일정 추가, 일정 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): 일정 삭제
9글자 이내의 한글로 일정을 입력해주세요(띄어쓰기 사용금지): 테스트
```

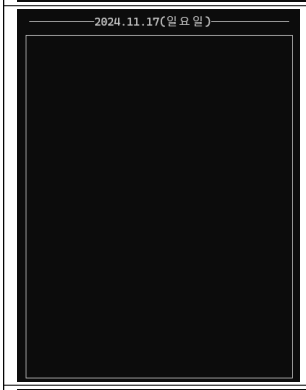


4) 메뉴

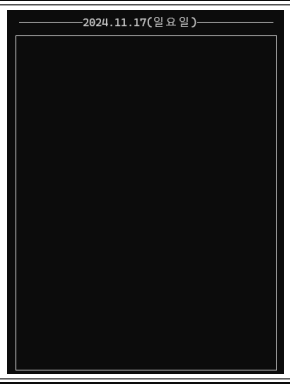
```
날짜 이동, 일정 추가, 일정 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): 날짜이동
0000 00 00형태로 연도, 월, 일을 입력해주세요: 2024 11 20
```



```
날짜 이동, 일정 추가, 일정 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): 일정 추가
20글자 이내의 한글로 일정을 입력해주세요(띄어쓰기 사용금지): 테스트
```



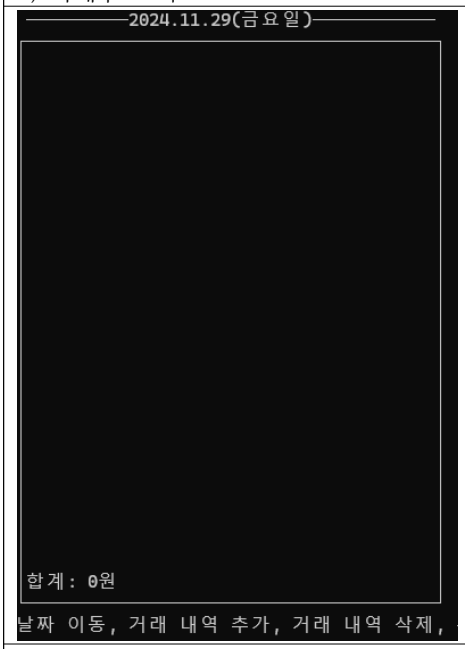

```
날짜 이동, 일정 추가, 일정 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): 일정 삭제
9글자 이내의 한글로 일정을 입력해주세요(띄어쓰기 사용금지): 테스트
```

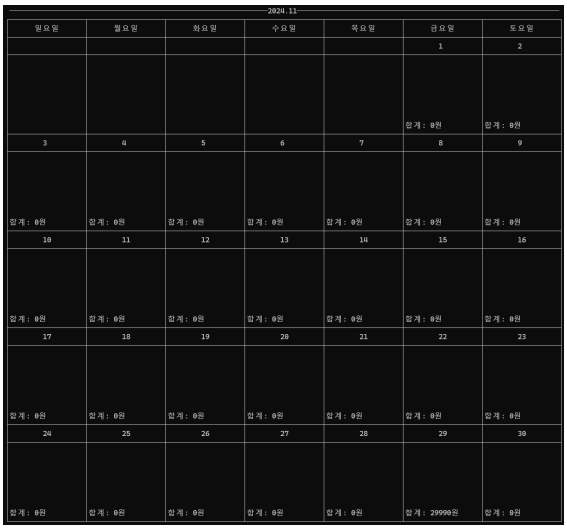



```
날짜 이동, 일정 추가, 일정 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): 뒤로가기
```

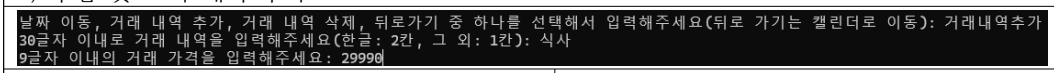
3. 가계부

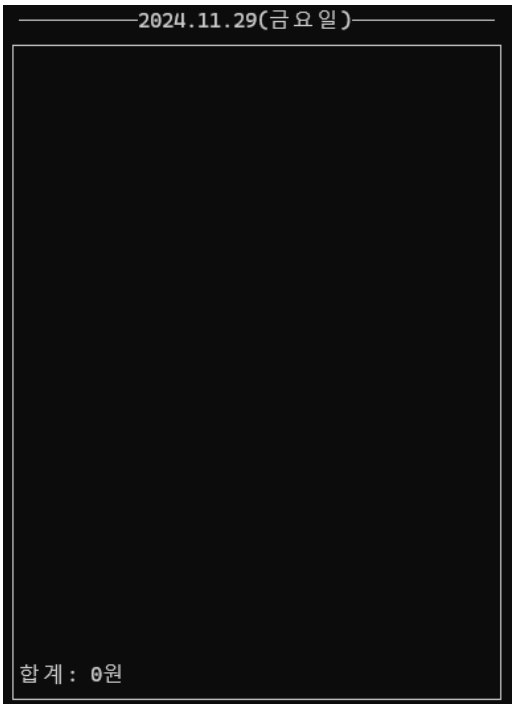
1) 가계부 표시

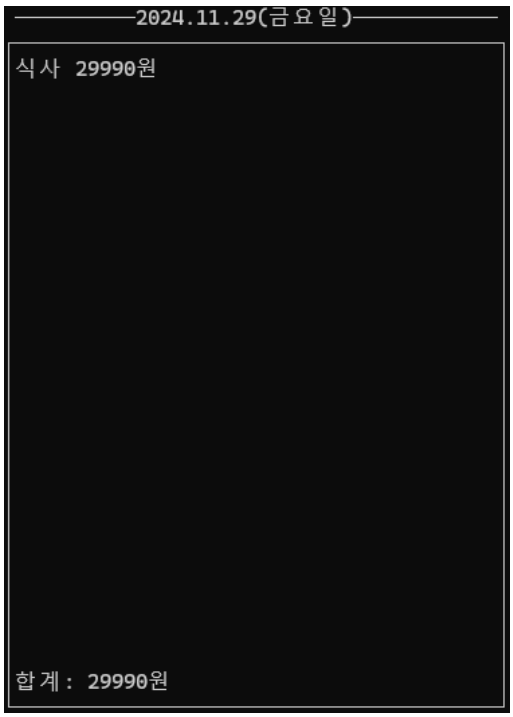




2) 수입 및 소비 내역 추가

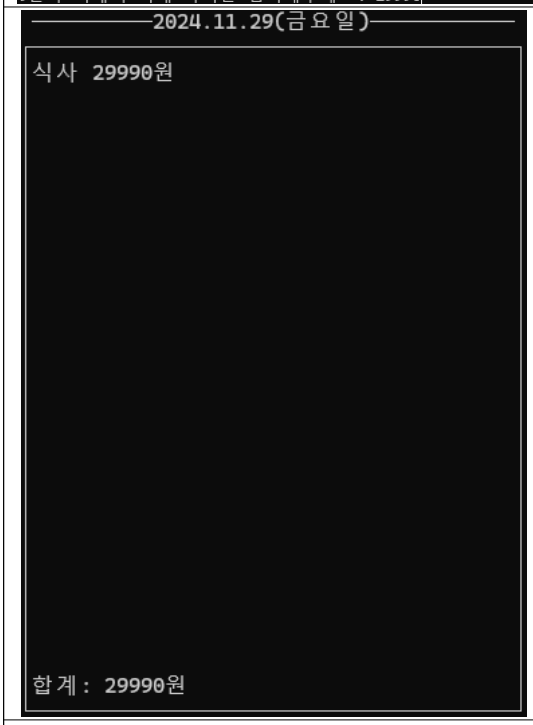


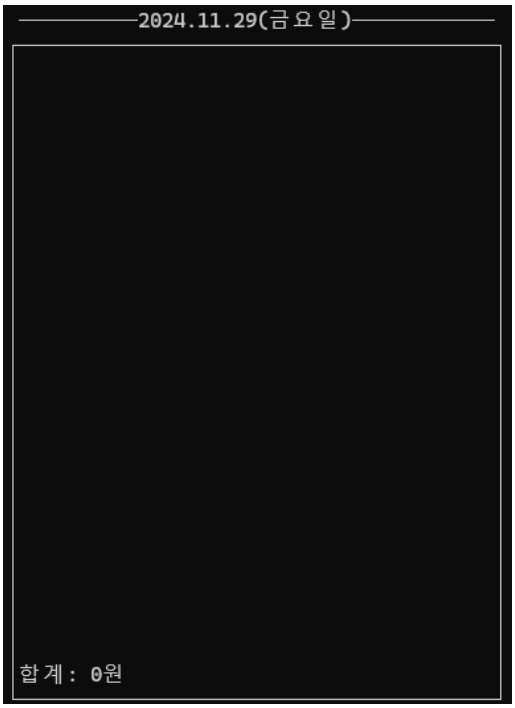




3) 수입 및 소비 내역 삭제

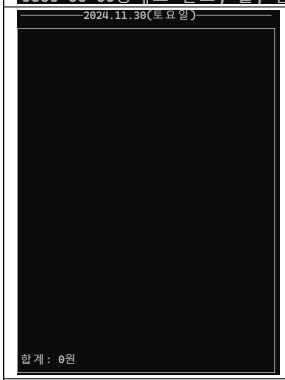
```
날짜 이동, 거래 내역 추가, 거래 내역 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): 거래내역삭제
30글자 이내로 거래 내역을 입력해주세요(한글: 2칸, 그 외: 1칸): 식사
9글자 이내의 거래 가격을 입력해주세요: 29990
```



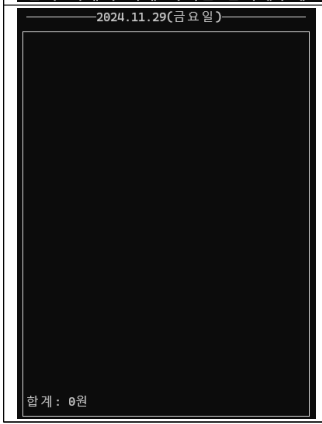


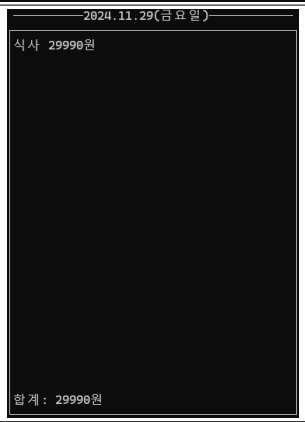
4) 메뉴

```
날짜 이동, 거래 내역 추가, 거래 내역 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): 날짜 이동
0000 00 00형태로 연도, 월, 일을 입력해주세요: 2024 11 30
```




```
날짜 이동, 거래 내역 추가, 거래 내역 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): 거래내역추가
30글자 이내로 거래 내역을 입력해주세요(한글: 2칸, 그 외: 1칸): 식사
9글자 이내의 거래 가격을 입력해주세요: 29990
```





```
날짜 이동, 거래 내역 추가, 거래 내역 삭제, 뒤로가기 중 하나를 선택해서 입력해주세요(뒤로 가기는 캘린더로 이동): 거래내역삭제
30글자 이내로 거래 내역을 입력해주세요(한글: 2칸, 그 외: 1칸): 식사
9글자 이내의 거래 가격을 입력해주세요: 29990
```

2024.11.29(금요일)

식사 29990원

합계: 29990원

2024.11.29(금요일)

합계: 0원

2024.11

일요일	월요일	화요일	수요일	목요일	금요일	토요일
					1	2
					합계: 0원	합계: 0원
3	4	5	6	7	8	9
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
10	11	12	13	14	15	16
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
17	18	19	20	21	22	23
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원
24	25	26	27	28	29	30
합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원	합계: 0원

4. 계획 대비 변경

- 없음

5. 프로젝트 일정

업무		~ 11/3	~ 11/10	~ 11/17	~ 11/24	~ 12/1	~ 12/8	~ 12/15	~ 12/22
제안서 작성		완료							
기능1	세부 기능1		진행 중						
	세부 기능2		완료						
	세부 기능3		완료						
기능2	세부 기능1		진행 중						
	세부 기능2			완료					
	세부 기능3			완료					
기능3	세부 기능1		완료						
	세부 기능2				완료				
	세부 기능3				완료				
기능4	세부 기능1								
	세부 기능2								
	세부 기능3								
오류 수정 및 코드 최적화									
최종 보고서									