C++프로그래밍및실습

예금 상품 제작 프로그램 개발

최종 보고서

제출일자: 2023. 12. 24.일
제출자명: 김지웅
제출자학번: 203185

1. 프로젝트 목표

1) 배경 및 필요성

금융시장이 발달함에 따라 금융상품 또한 더욱 복잡해지고 개수도 많아지고 있다. 우리는 금융상품 중 어떠한 상품이 본인에게 맞는지 파악하기 어려울 뿐만 아니라 금융상품을 이해하기 위한 지식도 부족한 것이 현 상황이다.

따라서 우리는 이러한 문제를 극복하기 위해 금융지식을 알려주며 고객이 직접 금융상품을 제작하고 그에 따른 수익을 확인시켜주는 프로그램을 만들고자 한다.

2) 프로젝트 목표

고객이 원하는 목표 수익에 맞춰 이자율과 납입 금액, 만기일을 직접 설정하고 해당 상품을 등록하는 프로그램을 만든다.

3) 차별점

기존 금융상품의 경우, 은행에서 정한 이자율과 만기에 따라 고객은 일정 금액을 납입하여 이자를 받는 구조였다. 하지만 이와 같은 구조의 상품들이 대량으로 쏟아지고 있어 고객은 어떠한 상품이 본인에게 맞는지 모르는 상황이 발생하고 있다. 우리가 만들고자 하는 프로그램은 고객의 니즈에 맞춰 이자율과 만기일을 설정하고 목표 수익을 달성하는 상품을 직접 제작한다는 것이 기존과의 차별점이다.

2. 기능 계획

1) 기능 1 : 원하는 목표 수익에 맞춘 상품 제작

- 설명 : 사용자가 직접 상품의 정보를 설정하여 금융 상품을 제작하는 기능이다.

(1) 세부 기능 1 : 사용자가 원하는 금융 상품을 제작

- 설명 : 사용자는 납입액은 얼마인지, 만기는 언제까지인지, 이자율을 몇 퍼센트인지를 직접 설정하고 만든 상품의 수익을 확인하는 기능을 제공한다.

(2) 세부 기능 2 : 사용자의 정보 저장 후 확인

- 설명 : 사용자가 만든 상품을 저장하여 필요에 따라 상품의 정보를 확인하는 기능을 제공한다.

2) 기능 2 : 금융 상품 정보 제공

- 설명 : 고객이 원하는 상품과 유사한 상품을 검색할 수 있도록 정보를 제공하는 기능이다.

(1) 세부 기능 1 : 금융 상품 정보 제공

- 설명 : 시중에서 제공하는 상품을 데이터화하여 저장한 후 기능 1 에서 사용자가 만든 상품과 유사한 상품을 찾아 사용자에게 추천하는 기능을 제공한다.

3. 기능 구현

(1) 예금 상품 제작 기능

- 입출력 : 사용자로부터 이름/생년월일/원금/이자율/만기/납입액을 입력 받아 이를 예금 계산식에 대입하여 최종지불액을 출력함.

- 설명 : DepositProduct클래스를 만들어 예금 상품을 만드는 멤버 함수를 정의함으로써 createDepositProduct 함수를 정의할 때 이를 사용하여 입력 받은 정보를 계산하여 최종 지불액을 계산함.

- 적용된 배운 내용

<클래스 및 객체 지향 프로그래밍>

먼저 예금 상품을 제작하기 위해 사용자 이름과 생년월일, 원금, 이자율, 만기, 납입액을 클래스 내에서 멤버 변수로 선언함. 이후 생성자를 만들고 예금을 계산하는 멤버 함수와 결과를 출력해주는 멤버 함수를 정의함.

<반복문과 조건문 활용>

예금 상품을 계산해주는 함수에서 매월 납입액 추가하는 구문과 매월 이자를 계산하는 구문을 반복문을 활용하여 계산해줌.

메인함수에서 초기화면을 출력 후 do-while문을 통해 업무 수행 후 다시 초기 화면으로 돌아가는 것을 구현하였으며 switch 조건문을 활용하여 번호에 맞게 임무를 실행함.

<함수를 활용하여 적용>

프로그램에서 활용한 함수는 예금 계산 함수와 결과 출력 함수 그리고 사용자에게 정보를 입력 받아 예금 상품을 만드는 함수를 정의함.

예금 계산 함수의 경우 반환갑이 존재하므로 Double 자료형으로 설정하였음

예금 상품 제작 함수의 경우에는 클래스에서 만든 함수들을 활용하여 결과를 출력하고 저장하는 방식의 함수를 제작함.

- 코드 스크린샷

```
class DepositProduct {
public:
    string name;            // 사용자 이름
    string birthdate;       // 사용자 생년월일
    double principal;       // 원금
    double interestRate;    // 이자율
    int maturityPeriod;     // 만기 기간 (월 단위)
    double monthlyDeposit;  // 월 납입액
```

```
public:
    // 생성자
    DepositProduct(const string& n, const string& bd, double p, double rate, int period, double deposit)
            : name(n), birthdate(bd), principal(p), interestRate(rate / 100.0), maturityPeriod(period), monthlyDeposit(deposit) {}

    // 예금 계산 메서드
    double calculateTotalAmount() const {
        double totalAmount = principal; // 최종 지불액 초기화

        for (int month = 1; month <= maturityPeriod; ++month) {
            totalAmount += monthlyDeposit; // 월 납입액 추가
            totalAmount *= (1 + interestRate / 12); // 월 이자 계산
        }

        return totalAmount;
    }

    // 결과 출력 메서드
    void displayResult() const {
        cout << "\n정기예금 계좌 개설이 완료되었습니다.\n";
        cout << "이름: " << name << "\t생년월일: " << birthdate << endl;
        cout << "최종 지불액: " << fixed << setprecision(2) << calculateTotalAmount() << " 원\n";
    }
```

```
void createDepositProduct() {
    // 사용자로부터 입력 받기
    string name, birthdate;
    double principal, interestRate, monthlyDeposit;
    int maturityPeriod;

    cout << "이름을 입력하세요: ";
    cin >> name;

    cout << "생년월일을 입력하세요 (YYYYMMDD 형식): ";
    cin >> birthdate;

    cout << "원금을 입력하세요: ";
    cin >> principal;

    cout << "이자율을 입력하세요 (%): ";
    cin >> interestRate;

    cout << "만기 기간을 입력하세요 (월 단위): ";
    cin >> maturityPeriod;

    cout << "월 납입액을 입력하세요: ";
    cin >> monthlyDeposit;

    // DepositProduct 클래스를 사용하여 예금 계좌 개설 및 결과 출력
    DepositProduct depositAccount(name, birthdate, principal, interestRate, maturityPeriod, monthlyDeposit);
    depositAccount.displayResult();
    depositAccount.saveToFile();  // 파일에 정보 저장
}
```

(2) 본인 상품 확인 기능

- 입출력 : 입력 받은 사용자 정보를 파일에 저장한 뒤 이름과 생년월일을 입력하면 해당 인원의 상품 정보를 출력함.

- 설명 : Deposit클래스에서 사용자의 정보를 저장하고 불러오는 멤버함수를 정의하고 CheckOwnProduct 함수를 정의할 때 사용하여 본인의 정보를 입력하면 본인이 만든 상품의 정보를 출력하도록 함.

- 적용된 배운 내용

<클래스 및 객체 지향 프로그래밍>

개인 정보를 파일에 저장하고 불러오기 위해 처음에 만든 DepositProduct클래스에서 선언된 멤버변수를 활용하여 계좌 정보를 저장하는 멤버 함수와 계좌 정보를 불러오는 멤버함수를 정의함. 이를 통해

<반복문과 조건문 활용>

사용자의 정보를 불러오는 함수를 정의하는 과정에서 파일에 있는 정보를 전부 확인하기 위해 while문과 if-else문을 활용하여 파일 끝까지 확인하는 구문을 작성함.

또한 사용자의 정보를 입력 받고 출력하는 함수에서 사용자의 정보가 있는지를 확인하는 과정을 조건문을 활용하여 작성함.

<벡터 동적 배열 활용>

사용자의 정보를 불러오는 함수에서 동적 배열 accounts를 선언하고 push_back를 활용하여 원하는 계좌 정보를 벡터에 넣어 보관함.

<함수를 활용하여 적용>

사용자에게 이름과 생년월일을 입력 받아 해당 계좌를 불러오기 위해 클래스에서 정의한 멤버 함수를 바탕으로 작성한 뒤 메임 함수에 적용함.

<정적 멤버 함수 활용>

파일에서 계좌를 불러오는 멤버 함수를 정의하는 과정에서 정적 함수를 이용하여 클래스에 의존하지 않고 활용이 가능하도록 함.

- 코드 스크린샷

```
    // 계좌 정보를 파일에 저장
    void saveToFile() const {
        ofstream outFile("deposit_info.txt", ios::app);
        if (outFile.is_open()) {
            outFile << name << " " << birthdate << " " << principal << " " << interestRate << " " << maturityPeriod << " " << monthlyDeposit << endl;
            outFile.close();
        } else {
            cout << "파일을 열 수 없습니다." << endl;
        }
    }

    // 파일에서 계좌 정보를 불러오기
    static vector<DepositProduct> loadFromFile() {
        vector<DepositProduct> accounts;
        ifstream inFile("deposit_info.txt");
        if (inFile.is_open()) {
            while (true) {
                string n, bd;
                double p, rate, deposit;
                int period;
                if (!(inFile >> n >> bd >> p >> rate >> period >> deposit)) {
                    break;  // 파일 끝에 도달하면 종료
                }
                DepositProduct account(n, bd, p, rate, period, deposit);
                accounts.push_back(account);
            }
            inFile.close();
        } else {
            cout << "파일을 열 수 없습니다." << endl;
        }
        return accounts;
    }
};
```

```
void checkOwnProduct() {
    // 파일에서 계좌 정보 불러오기
    vector<DepositProduct> accounts = DepositProduct::loadFromFile();

    if (accounts.empty()) {
        cout << "아직 개설된 계좌가 없습니다." << endl;
    } else {
        // 사용자로부터 이름과 생년월일 입력 받기
        string name, birthdate;
        cout << "이름을 입력하세요: ";
        cin >> name;
        cout << "생년월일을 입력하세요 (YYYYMMDD 형식): ";
        cin >> birthdate;

        // 해당 사용자의 계좌 찾아서 출력
        bool found = false;
        for (const auto& account : accounts) {
            if (account.name == name && account.birthdate == birthdate) {
                account.displayResult();
                found = true;
                break;
            }
        }

        if (!found) {
            cout << "해당하는 사용자의 계좌가 없습니다." << endl;
        }
    }
}
```

```
int main() {
    int mainMenuChoice;

    do {
        cout << "-----------------------------------예금 상품 제작-------------------------------" << endl;
        cout << "1. 예금 상품 만들기 \t 2. 본인 상품 확인하기  \t 3. 추천 상품 확인하기  \t 4. 종료"<< endl;
        cout << "-------------------------------------------------------------------------------" << endl;
        cout << "하시고자 하는 업무를 선택하여 주세요 : ";
        cin >> mainMenuChoice;

        switch (mainMenuChoice) {
            case 1:
                createDepositProduct();
                break;
            case 2:
                checkOwnProduct();
                break;
            case 3:
                financialInformation();
                break;
            case 4:
                cout << "프로그램을 종료합니다." << endl;
                break;
            default:
                cout << "올바르지 않은 선택입니다. 다시 선택하세요." << endl;
                break;
        }
    } while (mainMenuChoice != 4);

    return 0;
}
```

(3) 시중 은행 예금 상품

- 입출력 : 사용자가 만든 예금 상품을 입력 받아 사용자의 상품과 유사한 시중 상품을 찾아 해당 상품을 출력(추천)해주는 기능을 함.

- 설명 : financialInformation 함수를 만들어 시중에서 거래되고 있는 실제 예금 상품들을 텍스트 파일로 저장하여 파일 입출력을 통해 3번 항목을 실행 시 상품의 정보를 확인할 수 있음.

- 적용된 배운 내용

<함수를 활용하여 적용>

financialInformation 함수를 만들어 미리 만든 "bank_product.txt" 파일을 불러오는 함수를 제작함. 더불어 파일을 가져오는 과정에서 오류가 발생하는 경우, 예외처리를 활용하여 오류가 발생했음을 알려주는 함수를 만들어 메인 함수에 적용함으로써 프로그램의 모듈성과 관리성이 향상되도록 함.

<파일 입출력>

저장되어 있는 파일을 불러오기 위해서 ifstream 함수를 사용하여 "bank_product.txt" 텍스트 파일을 가져옴. 또한 파일의 특성상 한 줄씩 정보가 입력되어 있어 getline 함수를 사용하여 파일의 정보를 한 줄씩 가져와 출력시켜줌. 이후 close()를 하여 파일을 닫음.

<예외 처리>

try/catch문을 활용하여 try에서 예외가 발생하면 catch에서 해당되는 오류를 찾고 예외를 처리하는 과정을 구현함. 또한 표준 예외 클래스를 사용하여 runtime_error를 활용한 예외 처리 내용을 적용할 수 있었음. 예외 처리를 사용함으로써 파일을 여는 과정에서 발생하는 오류를 확인할 수 있었으며 코드의 가독성이 향상됨.

- 코드 스크린샷

```
147 void financialInformation() {
148     try{
149         ifstream bankfile("bank_product.txt");
150         if(!bankfile)
151             throw runtime_error("파일을 여는데 실패했습니다.");
152         string line;
153         while(getline(bankfile, line)){
154             cout << line << endl;
155         }
156         bankfile.close();
157     } catch(exception &e){
158         cerr << "에러 : " << e.what() << endl;
159     }
160 }
161
162 int main() {
163     int mainMenuChoice;
164
165     do {
166         cout << "-----------------------------------<예금 상품 제작 프로그램>-------------------------------" << endl;
167         cout << "1. 예금 상품 만들기 \t 2. 본인 상품 확인하기  \t 3. 추천 상품 확인하기  \t 4. 종료"<< endl;
168         cout << "-------------------------------------------------------------------------------------------" << endl;
169         cout << "하시고자 하는 업무를 선택하여 주세요 : ";
170         cin >> mainMenuChoice;
171
172         switch (mainMenuChoice) {
173             case 1:
174                 createDepositProduct();
175                 break;
176             case 2:
177                 checkOwnProduct();
178                 break;
179             case 3:
180                 financialInformation();
181                 break;
```

(4) 사용자의 상품과 유사한 상품 추천

- 입출력 : 사용자가 만든 예금 상품의 정보를 입력하면 만들어 놓은 파일에서 사용자의 상품과 유사한 상품을 찾아 출력함.

- 설명 : "bank_product.txt" 파일의 요소들을 불러와 변수로 선언한 뒤 이를 "deposit_info.txt"에 있는 사용자의 상품과 비교하여 이자율 차이가 0.5 이내인 상품을 찾아냄.

- 적용된 배운 내용

<클래스 및 객체 지향 프로그래밍>

"bank_product.txt"에 있는 요소들을 다루기 위해 Bankproduct라는 클래스를 만들어 변수들을 선언하고 이를 활용하여 동적 배열, 알고리즘 등등의 기능들에서 활용할 수 있도록 함.

<벡터 동적 배열 활용>

은행 상품 파일에서 요소들의 정보를 가져오는 과정에서 동적 배열 bankProducts를 선언하고 push_back를 활용하여 product에 정보를 넣음.

<STL 알고리즘 활용>

Sort를 사용하여 사용자가 만든 상품과 유사한 상품을 찾는 과정에서 파일에 있는 이자율 요소를 정렬하여 원하고자 하는 요소를 찾아냄.

<반복문과 조건문 활용>

"bank_product.txt"에서 사용자가 원하는 이자율과 유사한 이자율을 찾기 위해 if문과 for문을 결합하여 사용함. 정렬된 데이터를 통해 사용자의 이자율과 상품의 이자율의 절댓값 차이가 0.5 이내인 조건을 만들고 파일의 끝에서 끝까지 체크하여 해당 상품을 찾아내도록 함.

- 코드 스크린샷

```
    static vector<DepositProduct> createdProducts;

    static const DepositProduct& getLastCreatedProduct() {
        if (!createdProducts.empty()) {
            return createdProducts.back();
        } else {
            throw logic_error("No products created yet.");
        }
    }

class BankProduct {
public:
    string bankName;
    string productName;
    double maxInterestRate;
    double basicInterestRate;

    BankProduct(const string& bn, const string& pn, double maxRate, double basicRate)
        : bankName(bn), productName(pn), maxInterestRate(maxRate), basicInterestRate(basicRate){}
};

vector<BankProduct> readBankProductsFromFile(){
    vector<BankProduct> bankProducts;

    ifstream inFile("bank_product.txt");
    if(inFile.is_open()){
        while(true){
            string bankName, productName;
            double maxInterestRate, basicInterestRate;

            if(!(inFile >> bankName >> productName >> maxInterestRate >> basicInterestRate)){
                break;
            }

            BankProduct product(bankName, productName, maxInterestRate, basicInterestRate);
            bankProducts.push_back(product);
        }
        inFile.close();
    }else {
        cout << "파일을 열 수 없습니다." << endl;
    }
    return bankProducts;
}

vector<DepositProduct> DepositProduct::createdProducts;

void recommendSimilarProducts(const DepositProduct& userProduct, const vector<BankProduct>& bankProducts) {
    cout << "\n유사한 은행 상품 추천:\n";

    vector<BankProduct> similarProducts;
    for (const auto& bankProduct : bankProducts) {
        if (abs(userProduct.interestRate - bankProduct.maxInterestRate) < 0.5 &&
            userProduct.maturityPeriod == 12) {  // 예시로 만기 기간을 12로 설정
            similarProducts.push_back(bankProduct);
        }
    }

    sort(similarProducts.begin(), similarProducts.end(),
              [](const BankProduct& a, const BankProduct& b) {
                  return a.maxInterestRate > b.maxInterestRate;
              });

    const int numRecommendations = 3;
    for (int i = 0; i < min(numRecommendations, static_cast<int>(similarProducts.size())); ++i) {
        cout << similarProducts[i].bankName << "/" << similarProducts[i].productName
                  << "/최고 " << similarProducts[i].maxInterestRate << "%/기본 " << similarProducts[i].basicInterestRate << "%" << endl;
    }
}
```

- 비고

  Switch문의 점프 오류로 인해 프로그램 실행이 안됨.

```
            case 3:
                /*
                // 사용자가 만든 상품 정보 로드
                DepositProduct::loadFromFile();

                // 은행 상품 정보 로드
                vector<BankProduct> bankProducts = readBankProductsFromFile("bank_product.txt");   Jump bypasses variable initialization

                if (!bankProducts.empty() && !DepositProduct::createdProducts.empty()) {
                    const DepositProduct& userProduct = DepositProduct::getLastCreatedProduct();
                    recommendSimilarProducts(userProduct, bankProducts);
                } else {
                    cout << "은행 상품 정보 또는 사용자 상품 정보가 없습니다." << endl;
                }
                 */
                financialInformation();
                break;
```

4. 테스트 결과

(1) 예금 상품 제작

- 설명 : 사용자가 입력 상품의 정보를 입력하고 이를 계산하여 출력되는지를 확인하는 테스트임.

- 테스트 결과 스크린샷

```
-----------------------------------예금 상품 제작--------------------------------
1. 예금 상품 만들기  2. 본인 상품 확인하기    3. 추천 상품 확인하기    4. 종료
-------------------------------------------------------------------------------
하시고자 하는 업무를 선택하여 주세요 : 1
이름을 입력하세요: KimJiung
생년월일을 입력하세요 (YYYYMMDD 형식): 20020118
원금을 입력하세요: 1000000
이자율을 입력하세요 (%): 10
만기 기간을 입력하세요 (월 단위): 24
월 납입액을 입력하세요: 50000

정기예금 계좌 개설이 완료되었습니다.
이름: \343\205KimJiung    생년월일: 20020118
최종 지불액: 2553756.28 원
```

(2) 본인 상품 확인

- 설명 : 사용자가 입력한 상품 정보를 확인하기 위해 이름과 생년월일을 입력 후 상품의 정보가 나오는지를 확인한 테스트

- 비고 : 입력한 정보가 파일에 저장되는 과정에서 오류인지, 불러오는 과정에서 발생하는 오류인지를 모르겠지만 이름과 생년월일을 입력 후 해당 사용자의 계좌가 없다는 오류가 발생함. 추후 확인하여 수정할 예정임.

- 테스트 결과 스크린샷

```
-----------------------------------예금 상품 제작--------------------------------
1. 예금 상품 만들기  2. 본인 상품 확인하기    3. 추천 상품 확인하기    4. 종료
-------------------------------------------------------------------------------
하시고자 하는 업무를 선택하여 주세요 : 2
이름을 입력하세요: KimJiung
생년월일을 입력하세요 (YYYYMMDD 형식): 20020118
해당하는 사용자의 계좌가 없습니다.
```

(3) 종료

- 설명 : 프로그램을 종료하기 위해 입력한 값이 잘 구현 되는지를 확인하는 테스트

- 테스트 결과 스크린샷

```
----------------------------------예금 상품 제작-------------------------------
1. 예금 상품 만들기   2. 본인 상품 확인하기    3. 추천 상품 확인하기    4. 종료
--------------------------------------------------------------------------
하시고자 하는 업무를 선택하여 주세요 : 4
프로그램을 종료합니다.
Program ended with exit code: 0
```

(4) 시중 은행 예금 상품

- 설명 : 사용자를 위한 시중 은행 상품 정보를 텍스트 파일로 불러들이는 코드가 정상적으로 작동하는지를 확인하는 테스트

- 테스트 결과 스크린샷

```
--------------------------------------------------------------------------------
하시고자 하는 업무를 선택하여 주세요 : 3
은행별 적금 상품 (최고금리순)

전북은행/JB슈퍼씨드 적금/최고 13.60%/기본 3.60%

광주은행/광주은행제휴적금with유플러스닷컴/최고 13.00%/기본 3.00%

우리은행/데일리 워킹 적금/최고 11.00%/기본 1.00%

우리은행/우리 사장님 활짝 핀 적금/최고 10.00%/기본 3.00%

KB국민은행/온국민 건강적금-골든라이프/최고 10.00%/기본 2.00%

신한은행/패밀리 상생 적금/최고 9.00%/기본 3.00%

부산은행/너만 Solo 적금/최고 8.90%/기본 2.40%

하나은행/하나 아이키움 적금/최고 8.00%/기본 2.00%

카카오뱅크/카카오뱅크 한달적금/최고 8.00%/기본 2.50%

SH수협은행/Sh수산물을 좋아海 적금/최고 7.00%/기본 3.00%

신한은행/신한 SK LPG 쏠쏠한 행복 적금/최고 7.00%/기본 3.00%

우리은행/우리페이 적금/최고 7.00%/기본 2.00%

IBK기업은행/IBK탄소제로적금/최고 7.00%/기본 3.00%

광주은행/새희망적금최고/6.95%/기본 2.95%

NH농협은행/NH진짜사나이(군간부)적금/최고 6.80%/기본 3.10%

NH농협은행/NH아동수당 우대적금/최고 6.70%/기본 3.20%

NH농협은행/NH1418스윙적금/최고 6.60%/기본 3.10%

경남은행/오늘도 세이브 적금/최고 6.50%/기본 3.45%

IBK기업은행/IBK부모급여우대적금/최고 6.50%/기본 2.50%
```

5. 계획 대비 변경 사항

1) 사용자 상품 추천

- 이전 : 기능(1) 부분에서 제작한 사용자의 상품을 바탕으로 하여 시중에서 제공하는 실제 금융 상품과 비교한 뒤 사용자의 상품과 유사한 상품을 추천해주는 기능으로 변경함.

- 이후 : 코드는 작성하였으나 실행 불가로 추천 하는 기능을 구현해내지 못함.

- 사유 : 기능을 구현하기 위해 코드를 작성하고 실행하려고 하였지만 프로그램에 대한 이해도 부족과 부족한 실력으로 정확한 코드를 작성을 하지 못하여 switch문에서 다음 case로의 점프가 불가하다는 오류가 발생함.

6. 느낀점

이번 프로젝트를 진행하면서 정말 많은 것을 배우고 느꼈습니다.

먼저, 소프트웨어 전공을 처음 시작하는 타과학생으로 컴퓨터 언어를 이용한 프로그램 개발에 첫 걸음이 되었다는 것입니다. 가장 먼저 접한 언어는 파이썬으로 자세하게는 배우지 않았지만 간단한 기능들과 이를 구현하는 것에 흥미와 호기심을 느껴 소프트웨어 전공을 희망하게 되었습니다. 하지만 수업 초반에 파이썬과는 다른 코드 작성 방식과 더불어 이전에는 경험하지 못했던 더 다양한 기능들을 배우고 적용하기가 쉽지 않았습니다.

그런데 이번 프로젝트를 하면서 내가 만들고 싶은 프로그램을 수업 시간에 배운 내용을 바탕으로 하여 직접 만들어 보니 더 흥미롭고 배움에 대한 열망 또한 커졌습니다.

다만 프로그램에 대한 이해와 능력 부족으로 인해 자꾸만 chatGPT의 도움을 받아 문제를 해결하려고 했던 점들이 아쉽게 느껴졌습니다. 혼자서 문제를 해결하여 프로젝트를 진행해야 진정한 나의 프로젝트가 되는 것이라고 생각하지만, 내가 구현하고 싶은 코드들을 자세히 알려고 하지 않고 잠깐 이번만 넘기려는 식의 프로젝트 진행들이 실력 향상에 도움이 되지 못한 것 같아 후회가 남습니다.

그래도 한 번의 프로젝트지만 앞으로 이러한 경험이 또 있다면 이때의 기억을 토대로 더 완성된 결과물을 만드는 데에 좋은 밑거름이 될 것이라고 생각합니다.

이번 학기, 교수님께 수업을 듣게 되어 기쁘고 항상 열정적으로 임해주셔서 감사합니다.