은행 프로그램 분석

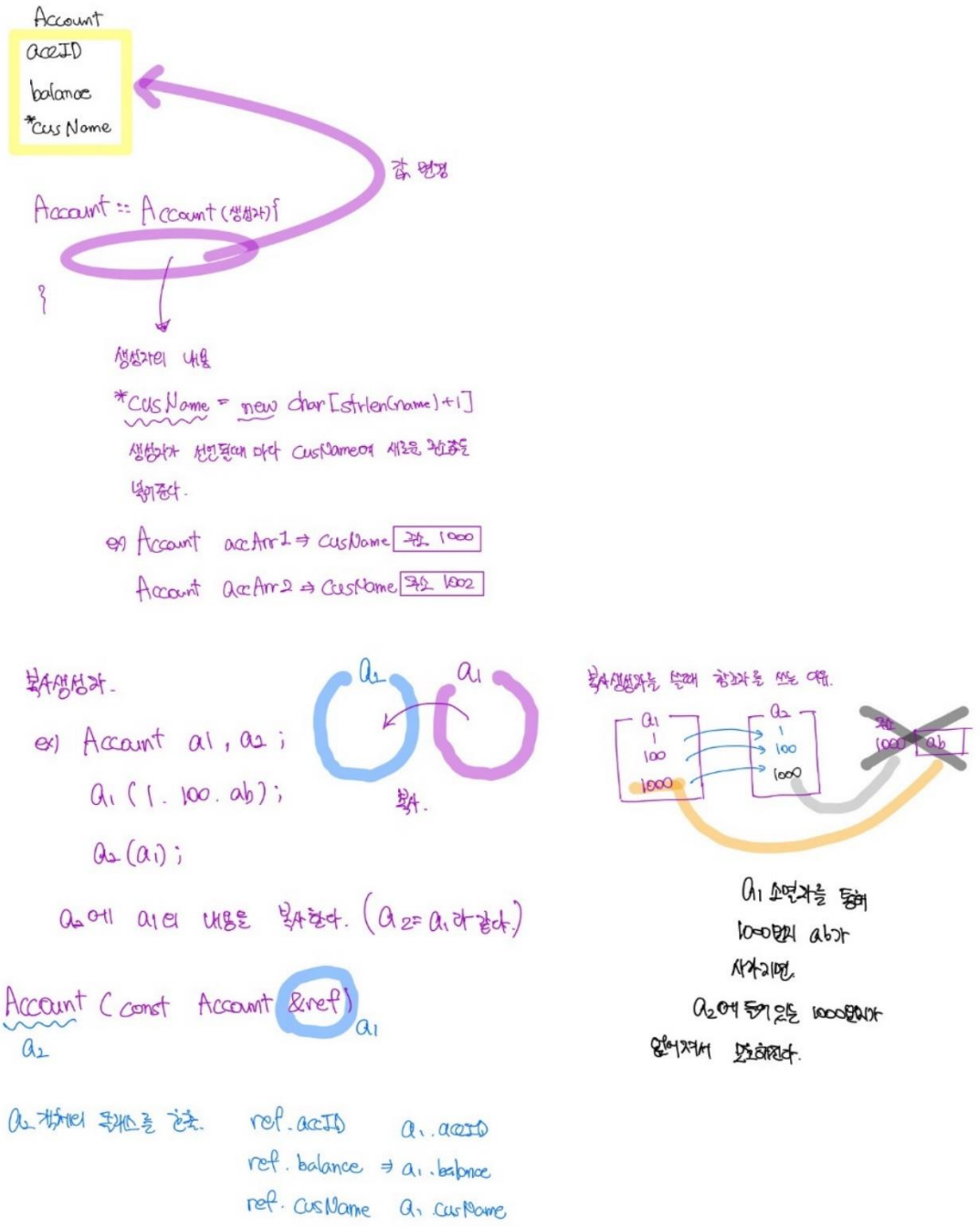
본 보고서를 최종보고서로 제출합니다.

이　　름: 이지호

학　　번: 20163290

1. 프로그램 1 분석

1.1 클래스 분석



멤버변수 : accID : 계좌번호, balance : 잔액,
*cusName : 고객이름을 저장하는 포인터변수(문자 배열의 주소를 저장한다)
생성자 : private로 감춰진 멤버변수에 접근하여 값을 변화시켜주는데 필요한 함수
객체를 생성할 때 호출되어 사용된다.
복사생성자 : 생성자로 만들어진 객체의 정보를
다른 객체에 복사하기 위해 필요한 함수이다.
GetAccID() : 생성된 객체의 accID값을 반환해준다.
Deposit() : 생성된 객체의 balance변수에 매개변수로 money를 받아 더해준다.

withDraw() : 출금액을 반환하는 함수이다. 출금액보다 저금된 금액이 적으면 0을 반환

해준다.

showInfo() : 객체의 멤버변수를 출력해주는 함수이다. Const를 붙임으로써 값의 변화가 없음을 알려준다.

~Account() : 포인터로 된 멤버변수를 사용했을 때 그 변수는 클래스가 종료되면 사라져야 하는데 사라지지 않고 메모리에 남아있게 된다. 그래서 소멸자로 없애주는 함수를 만들어줘야 한다.

1.2 main 함수 분석

```
int
main(void)
        {
                int choice;
                //choice 라는 정수형 변수를 생성해준다.

                while (1)
                {
                        ShowMenu();
                        //ShowMenu 함수를 사용해서 메뉴들을 출력하여 보여준다.
                        cout << "선택: ";
                        cin >> choice;
                        cout << endl;

                        switch (choice)
        //enum 으로 make = 1, deposit = 2, withdraw = 3, inquire = 4, exit = 5 로 지정을 해뒀다.
                        {
                        case MAKE:
                                //1 을 누른경우 make 의 함수를 실행
                                MakeAccount();
                                break;
                        case DEPOSIT:
                                //2 를 누른경우 depositmoney 의 함수를 실행
                                DepositMoney();
                                break;
                        case WITHDRAW:
                                //3 을 누른경우 WithdrawMoney 의 함수를 실행
                                WithdrawMoney();
                                break;
                        case INQUIRE:
                                //4 를 누른경우 ShowAllAccInfo 의 함수를 실행
                                ShowAllAccInfo();
                                break;
                        case EXIT:
                                //5 를 누른경우 exit 함수를 실행
                                return 0;
                        default:
        //위의 경우를 벗어난 수를 입력하면 잘못된 선택이라고 출력해주고 다시 입력을 요구한다.
                                cout << "Illegal selection.." << endl;
```

```
                }
        }

        for (int i = 0; i < accNum; i++)
                delete accArr[i];
//생성된 객체의 배열은 main 문을 종료하더라도 남아있기 때문에 삭제해줘야한다.
        return 0;
}
```

1.3 전체 코드 분석

```
#include
<iostream>
        #include <cstring>

        using namespace std;
        const int NAME_LEN = 20;

        void ShowMenu(void);        // 메뉴출력
        void MakeAccount(void);     // 계좌개설을 위한 함수
        void DepositMoney(void);        // 입    금
        void WithdrawMoney(void);       // 출    금
        void ShowAllAccInfo(void);      // 잔액조회

        enum { MAKE = 1, DEPOSIT, WITHDRAW, INQUIRE, EXIT };
//열거형 순서대로 나타나는 상수를 보기 쉽게   #define 과 비슷하다.

        class Account
        {
        private:            //private 으로 지정해놓으면 클래스 외부에서 접근할 수 없다
                int accID;      // 계좌번호
                int balance;    // 잔    액
                char* cusName;  // 고객이름

        public:  //외부에서도 접근이 가능하다.
                Account(int ID, int money, char* name)

//멤버변수인 accID, balance, cunName 에 접근해서 값을 할당해줄때 필요한 생성자이다.
                        : accID(ID), balance(money)

                        // accID = ID  accID 에 매개변수로 받아온 ID 의 값을 대입해준다
                        //balance = money  balance 에 매개변수로 받아온 money 값을
대입해준다.
                {
                        cusName = new char[strlen(name) + 1];

                //cusName 은 포인터 변수로 주소값을 저장할수 있는 character 형 변수이다.
```

```
//이 변수에 new 를 사용해서 새로운 character 배열을 만들고 그 주소를 넣어준다.

//char *name 은 아래의 makeAccount 함수에서 객체를 생성할때

// 문자배열에 의해 생성된 것을 넣어주는데 그 배열의 주소를 *name 에 저장한다.

// 저장된 주소값을 타고 들어가서 name 의 길이를 strlen 을 통해 반환받는다

//배열의 길이는 매개변수로 받아온 name 배열의 길이를 사용한다.

//마지막의 null 문자까지 계산해서 +1 을 해준다.

        strcpy_s(cusName, (strlen(name) + 1), name);

        //strcpy  string copy 문자열 복사함수를 사용해서 name 의 문자를
cusName 에 복사한다.
    }

    Account(const Account& ref)
        //매개변수로 참조자를 받는 생성자는 복사생성자이다.

        //복사생성자는 클래스를 통해 객체를 생성하고 그 객체에 값을 넣을때

        //기존에 있던 객체의 값을 복사해서 넣어주고 싶을때 사용한다


        : accID(ref.accID), balance(ref.balance)

//기존에 있던 클래스의 객체를 ref 라는 참조자로 받아왔을 때 기존 클래스의 별명을 ref 라고
하는것과 같다.

        //복사된 값을 넣으려는 객체의 accID 에 ref 의 accID 를 복사해서 넣고

    //복사된 값을 넣으려는 객체의 balance 에 ref 의 balance 를 복사해서 넣어준다.
    {
```

```
		cusName = new char[strlen(ref.cusName) + 1];

		//cusName 은 포인터변수로 주소를 저장하는 변수이다
		//그러므로 new 를 사용해서 character 배열을 만들어주고 그 배열의 주소를 cusName 에 저장해준다.

		//배열의 크기는 null 문자 포함 +1 이 된다.
		strcpy_s(cusName, (strlen(ref.cusName) + 1), ref.cusName);
	}

	int GetAccID() const { return accID; }

	//생성된 객체의 accID 의 값을 반환해준다.
	//이때 const 를 앞에 붙이므로써 accID 의 값은 바꿀수 없음을 알려준다.

	void Deposit(int money)
	{
		balance += money;
		//생성된 객체의 balance 멤버변수에 매개변수로 받아온 money 의 값을 더해준다.
	}

	int Withdraw(int money)   // 출금액 반환, 부족 시 0 반환
	{
		if (balance < money)
			// 생성된 객체의 balance 에 저장된 값이 매개변수로 받아온 money 의 값보다 작다면
			//0 을 반환해준다.
			return 0;

		balance -= money;
		//balance 의 값에 money 의 값을 빼준다.
		return money;
		//출금하려고 한 money 값을 반환해준다.
	}

	void ShowAccInfo() const
		//생성된 객체의 정보를 출력해주는 함수이다.
	{
		cout << "계좌 ID: " << accID << endl;
		cout << "이  름: " << cusName << endl;
		cout << "잔  액: " << balance << endl;
	}

	~Account()
		//포인터로된 멤버변수에 배열을 생성했을경우 객체가 클래스를 빠져나가더라도 소멸되지 않는다.
		//따라서 delete 를 사용해서 배열을 지워줘야한다.
```

```
	{
		delete[]cusName;
	}
};

Account* accArr[100];  // Account 저장을 위한 배열
int accNum = 0;        // 저장된 Account 수

int main(void)
{
	int choice;
	//choice 라는 정수형 변수를 생성해준다.

	while (1)
	{
		ShowMenu();
		//ShowMenu 함수를 사용해서 메뉴들을 출력하여 보여준다.
		cout << "선택: ";
		cin >> choice;
		cout << endl;

		switch (choice)
			//enum 으로 make = 1, deposit = 2, withdraw = 3, inquire = 4, exit = 5 로 지정을 해뒀다.
		{
		case MAKE:
			//1 을 누른경우 make 의 함수를 실행
			MakeAccount();
			break;
		case DEPOSIT:
			//2 를 누른경우 depositmoney 의 함수를 실행
			DepositMoney();
			break;
		case WITHDRAW:
			//3 을 누른경우 WithdrawMoney 의 함수를 실행
			WithdrawMoney();
			break;
		case INQUIRE:
			//4 를 누른경우 ShowAllAccInfo 의 함수를 실행
			ShowAllAccInfo();
			break;
		case EXIT:
			//5 를 누른경우 exit 함수를 실행
			return 0;
		default:
			//위의 경우를 벗어난 수를 입력하면 잘못된 선택이라고 출력해주고 다시 입력을 요구한다.
			cout << "Illegal selection.." << endl;
```

```
                }
        }

        for (int i = 0; i < accNum; i++)
                delete accArr[i];
                //생성된 객체의 배열은 main 문을 종료하더라도 남아있기 때문에
삭제해줘야한다.
        return 0;
}

void ShowMenu(void)
//아래의 출력부분을 실행한다.
{
        cout << "-----Menu------" << endl;
        cout << "1. 계좌개설" << endl;
        cout << "2. 입    금" << endl;
        cout << "3. 출    금" << endl;
        cout << "4. 계좌정보 전체 출력" << endl;
        cout << "5. 프로그램 종료" << endl;
}

void MakeAccount(void)
//계좌를 만들어주는 함수이다.
{
        int id;
        char name[NAME_LEN];
        int balance;

        cout << "[계좌개설]" << endl;
        cout << "계좌 ID: ";     cin >> id;
        cout << "이  름: ";      cin >> name;
        cout << "입금액: ";      cin >> balance;
        cout << endl;
        //id, name, balance 를 입력받는다.

        accArr[accNum++] = new Account(id, balance, name);
        //입력받은 내용들을 Account 저장을 위해 만든 포인터배열인 accArr 에 넣어준다.
        //ex) accArr[0] = new Account(1, 10, abc)
        //    accArr[1] = new Account(2, 100, efg)
        //와 같이 각각의 배열이 서로다른 객체를 생성하게 된다.
}

void DepositMoney(void)
{
        int money;
        int id;
        cout << "[입    금]" << endl;
        cout << "계좌 ID: ";     cin >> id;
```

```
	cout << "입금액: ";	cin >> money;
	//money 와 id 의 값을 입력받는다.

	for (int i = 0; i < accNum; i++)
	{
		if (accArr[i]->GetAccID() == id)
			//accArr[i]의 객체에 들어있는 accID 가 내가 입력한 id 와
같다면
		{
			accArr[i]->Deposit(money);
			//accArr[i]의 객체가 속한 클래스의 Deposit 함수를 수행한다
			//이때 들어가는 매개변수로는 money 가 들어간다.
			cout << "입금완료" << endl << endl;
			return;
		}
	}
	cout << "유효하지 않은 ID 입니다." << endl << endl;
	//내가 입력한 id 와 accArr[i]에 들어있는 accID 가 다르면 위의 문장을 출력한다.
}

void WithdrawMoney(void)
{
	int money;
	int id;
	cout << "[출    금]" << endl;
	cout << "계좌 ID: ";
	cin >> id;
	cout << "출금액: ";
	cin >> money;

	for (int i = 0; i < accNum; i++)
	{
		if (accArr[i]->GetAccID() == id)
			//accArr[i]객체가 포함한 class 의 GetAccID 멤버함수에
접근해서
			//return 받은 값이 id 와 같을때
		{
			if (accArr[i]->Withdraw(money) == 0)
				//accArr[i]객체의 클래스에 withdraw 함수에 접근해서
money 를
				//매개변수로 넣었을때 그 값이 0 으로 반환받았으면
				//입금되어있는 금액 즉,accArr[i]에 저장된
balance 값이 money 보다 작을때
			{
				cout << "잔액부족" << endl << endl;
				//잔액 부족을 출력한다.
				return;
			}
```

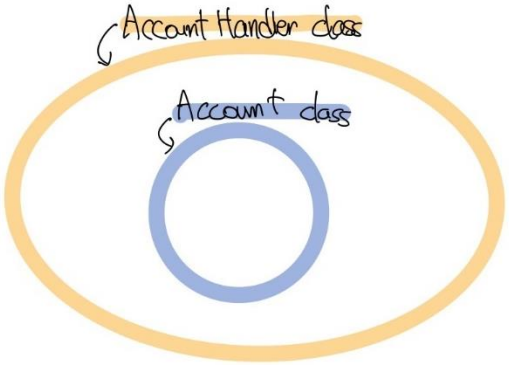
```
                        cout << "출금완료" << endl << endl;
                        return;
                }
        }
        cout << "유효하지 않은 ID 입니다." << endl << endl;
}

void ShowAllAccInfo(void)
{
        for (int i = 0; i < accNum; i++)
        {
                accArr[i]->ShowAccInfo();
                //accArr[i]의 클래스에 접근해서 showaccinfo 함수를 실행해준다.
                cout << endl;
        }
}
```

2. 프로그램 2 분석

1.1 클래스 분석

Account class 와 AccountHandler class 는

Has-a 관계이다.



AccountHandler 클래스의 멤버변수인 Account * accArr[100]

선언 : AccountHandler :: MakeAccount 에서

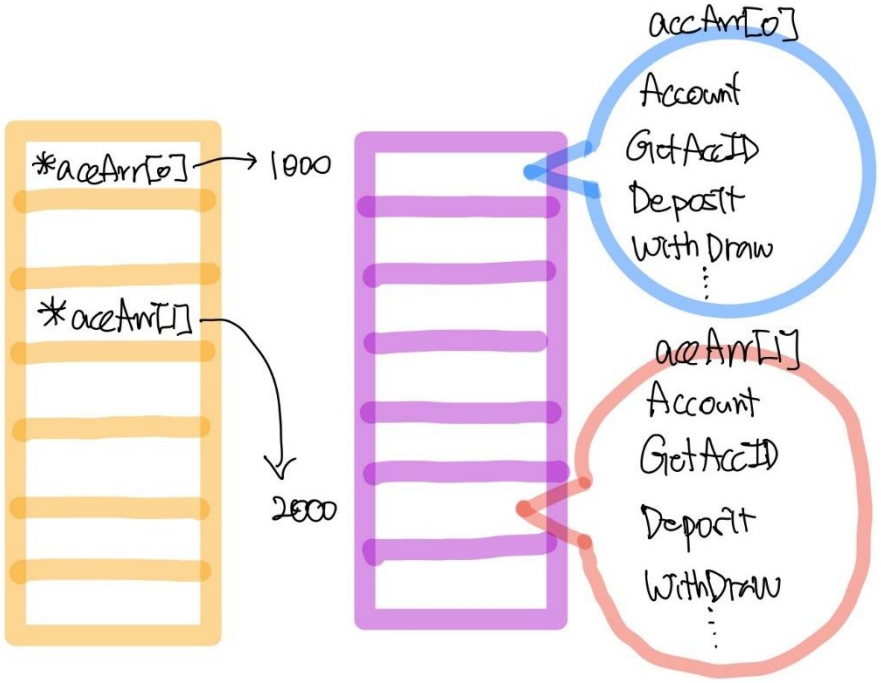
accArr[accNum++] = new Account (id, balance, name)

↳ accArr 은 Account 의 포인터형에 배열이름으로 생성되는 객체들을 동적으로 생성하고 그 주소값들을 배열의 원소로 가지는 배열이다.

즉 배열의 각 원소는 서로 다른 객체이다.

예) accArr[0] = new Account (....)
accArr[1] = new Account (....)
] 서로 다르다.

Account 객체를 생성하고 멤버함수에 접근하는 것 또한 포인터 배열이 가지는 주소를 이용하여 참조가능하다.



Account 클래스 :

Account() : Account객체를 생성하기 위한 생성자이다. 멤버변수의 값들이 private
으로 감쳐줘 있기 때문에 생성자를 통해 값을 변경해줘야 한다.
Account(const Account &ref) : 복사생성자이다. 이미 생성된 객체의 정보를 다른
객체에 복사하여 값을 넣어주기 위해 사용된다.
GetAccID() const : 상수의 의미를 가지는 const를 붙임으로써 함수 내에서 값의
변화가 없음을 알려준다. accID의 값을 반환해주는 함수이다.
Deposit(int money) : money의 값을 매개변수로 받아 객체가 가진 balance의 값
에서 money를 빼고 값을 저장하는 함수이다.
WithDraw(int money) : 출금을 하는 함수이다. 객체에 저장된 balance의 값이
매개변수로 받은 money의 값보다 작으면 0을 리턴
위의 경우가 아닌경우 balance에 money의 값을 빼주고 반환한다.
ShowAccInfo() const : const를 붙임으로써 함수 내에서의 값의 변화가 없음을 알림
객체의 데이터를 출력해주는 함수이다.
~Account() : 동적으로 생성된 문자배열(포인터 배열)은 클래스를 벗어나도 메모리에
남아있기 때문에 삭제 해줘야 한다.

AccountHandler 클래스 :

ShowMenu(void) const : const를 사용함으로써 함수내의 값 변화 없음을 알림
(void)로 매개변수로 들어오는 값이 없음을 알려주고
사용자로부터 입력할 메뉴들의 항목을 보여주는 함수이다.
MakeAccount(void) : 정수형 변수인 id와 balance를 입력받고 이름을 저장할
문자열 배열인 char[NAME_LEN]을 만들어준다.
사용자로부터 값들을 입력받고 그 값을 이용해 Account클래스의
객체를 만들어준다. 이때 만들어진 객체의 주소를 인덱스로 하게끔
포인터 배열에 값을 넣어준다.
DepositMoney(void) : 계좌ID와 입금액을 입력하는 함수이다.

입력받은 계좌ID가 접근하고자 하는 객체와 일치한다면 해당객체의 Deposit함수를 실행해준다.

WithDrawMoney(void) : 계좌ID와 출금액을 입력받는 함수이다.
객체의 아이디와 입력받은 아이디가 같다면 해당 객체의 WithDraw함수를 실행해준다 이때 반환된 값이 0 이라면 잔액부족을 출력해주고 아니라면 출력완료를 출력해준다.

AccountHandler() : accNum(0) : AccountHandler객체를 생성할 때 아무것도 적어주지 않았을때 이 생성자를 실행하고 accNum을 0으로 초기화 해준다.

ShowAccInfo(void) const : const를 사용함으로써 함수 내에서의 값 변화가 없음을 알려주고 모든 객체의 정보를 출력해주는 함수이다.

~AccountHandler() : 동적으로 생성된 Account정보를 주소값을 원소로 가지는 포인터 배열 accArr[i]를 삭제해줘야 한다.
Class를 벗어나도 포인터로 만든 변수는 메모리상에 남아있기 때문에 없애줘야한다.

1.2 main 함수 분석

```
int
main(void)
	{
		AccountHandler manager;
	//manager 로 AccountHandler 객체를 생성해준다.
		int choice;
	//정수형 변수인 choice 를 만들어준다.
		while (1)
	//while(1)은 항상 참이므로 while 문 내부의 조건이 없으면 계속 반복한다.
		{
			manager.ShowMenu();
	//manager 를 통해 ShowMenu 함수를 실행해준다.
			cout << "선택: ";
			cin >> choice;
	//choice 값을 입력받는다.
			cout << endl;

			switch (choice)
			{
	//choice 는 정수형인다 switch case 문에서는 MAKE, DEPOSIT 등이 사용된다.
	//이게 가능한 이유는 enum 에서 MAKE 는 1, DEPOSIT 은 2, WithDraw 는 3 이런식으로
	//정의를 해놓았기 때문에 case 1 : 는 case Make : 와 같은 의미이다.
			case MAKE:
				manager.MakeAccount();
				break;
	//Make(1)이 입력됐을 때 MakeAccount()함수를 실행해준다.
```

```
                case DEPOSIT:
                        manager.DepositMoney();
                        break;
        //Deposit(2)가 입력됐을 때 DepositMoney()함수를 실행해준다.
                case WITHDRAW:
                        manager.WithdrawMoney();
                        break;
        //WithDraw(3)이 입력됐을 때 WithDrawMoney()함수를 실행해준다.
                case INQUIRE:
                        manager.ShowAllAccInfo();
                        break;
        //INQUIRE(4)가 입력됐을 때 ShowAllInfo() 함수를 실행해준다.
                case EXIT:
                        return 0;
        //EXIT(5)를 입력했을 때 0을 while문에 넣고 반복문을 종료해준다.
                default:
                        cout << "Illegal selection.." << endl;
        //지정된 범위의 수가 입력되지 않았을 때 Illegal section을 출력해주고
        //다시 입력을 받는다.
                }
        }

        return 0;
}
```

1.2 전체 코드 분석

```
#include
<iostream>
        #include <cstring>

        using namespace std;
        const int NAME_LEN = 20;

        enum { MAKE = 1, DEPOSIT, WITHDRAW, INQUIRE, EXIT };
        //열거형 순서대로 나타나는 상수를 보기 쉽게  #define과 비슷하다.


        /*
         * 클래스 이름: Account
         * 클래스 유형: Entity 클래스
         */

        class Account
        {
        private:
        //private로 멤버변수의 값을 숨겨놓았다 외부에서 접근할 수 없다.
```

```
        int accID;
        int balance;
        char* cusName;

public:
//public 으로 지정되어 있으면 외부에서 접근이 가능하다.
//함수 원형으로 정의를 한다.
        Account(int ID, int money, char* name); //기본생성자
        Account(const Account& ref);  //복사생성자

        int GetAccID() const;
        void Deposit(int money);
        int Withdraw(int money);
        void ShowAccInfo() const;
        ~Account();
};
//멤버함수의 외부정의
Account::Account(int ID, int money, char* name)
        : accID(ID), balance(money)
//멤버변수인 accID, balance, cunName 에 접근해서 값을 할당해줄때 필요한 생성자이다.
// accID = ID  accID 에 매개변수로 받아온 ID 의 값을 대입해준다
//balance = money  balance 에 매개변수로 받아온 money 값을 대입해준다.
{
        cusName = new char[strlen(name) + 1];
//cusName 은 포인터 변수로 주소값을 저장할수 있는 character 형 변수이다.

//이 변수에 new 를 사용해서 새로운 character 배열을 만들고 그 주소를 넣어준다.
//char *name 은 아래의 makeAccount 함수에서 객체를 생성할때
// 문자배열에 의해 생성된 것을 넣어주는데 그 배열의 주소를 *name 에 저장한다.
// 저장된 주소값을 타고 들어가서 name 의 길이를 strlen 을 통해 반환받는다
//배열의 길이는 매개변수로 받아온 name 배열의 길이를 사용한다.
//마지막의 null 문자까지 계산해서 +1 을 해준다.

strcpy_s(cusName, (strlen(name) + 1), name);
//strcpy  string copy 문자열 복사함수를 사용해서 name 의 문자를 cusName 에 복사한다.
}

Account::Account(const Account& ref)
//매개변수로 참조자를 받는 생성자는 복사생성자이다.
//복사생성자는 클래스를 통해 객체를 생성하고 그 객체에 값을 넣을때
//기존에 있던 객체의 값을 복사해서 넣어주고 싶을때 사용한다
        : accID(ref.accID), balance(ref.balance)
//기존에 있던 클래스의 객체를 ref 라는 참조자로 받아왔을때 기존 클래스의 별명을 ref 라고
하는것과 같다.
//복사된 값을 넣으려는 객체의 accID 에 ref 의 accID 를 복사해서 넣고
//복사된 값을 넣으려는 객체의 balance 에 ref 의 balance 를 복사해서 넣어준다.
{
        cusName = new char[strlen(ref.cusName) + 1];
```

```
//cusName 은 포인터변수로 주소를 저장하는 변수이다
//그러므로 new 를 사용해서 character 배열을 만들어주고 그 배열의 주소를 cusName 에
저장해준다.
//배열의 크기는 null 문자 포함 +1 이 된다.
        strcpy_s(cusName, (strlen(ref.cusName) + 1), ref.cusName);
}

int Account::GetAccID() const { return accID; }
//생성된 객체의 accID 의 값을 반환해준다.
//이때 const 를 앞에 붙이므로써 accID 의 값은 바꿀수 없음을 알려준다.

void Account::Deposit(int money)
{
        balance += money;
//생성된 객체의 balance 멤버변수에 매개변수로 받아온 money 의 값을 더해준다.
}

int Account::Withdraw(int money)
{
        if (balance < money)
                return 0;
// 생성된 객체의 balance 에 저장된 값이 매개변수로 받아온 money 의 값보다 작다면
//0 을 반환해준다.

        balance -= money;
        return money;
//balance 의 값에 money 의 값을 빼준다.
//출금하려고 한 money 값을 반환해준다.
}

void Account::ShowAccInfo() const
//생성된 객체의 정보를 출력해주는 함수이다.
{
        cout << "계좌 ID: " << accID << endl;
        cout << "이  름: " << cusName << endl;
        cout << "잔  액: " << balance << endl;
}

Account::~Account()
//포인터로 된 멤버변수에 배열을 생성했을 경우 객체가 클래스를 빠져나가더라도 소멸되지
않는다.
//따라서 delete 를 사용해서 배열을 지워줘야 한다.
{
        delete[]cusName;
}


/*
```

```
 * 클래스 이름: AccountHandler
 * 클래스 유형: 컨트롤(Control) 클래스
 */

class AccountHandler
//Account 클래스를 구성품으로 가지는 클래스
//실질적인 메인문의 작동을 실행하는 클래스이다.
{
private:
        Account* accArr[100];
        int accNum;

public:
        AccountHandler();
        void ShowMenu(void) const;
        void MakeAccount(void);
        void DepositMoney(void);
        void WithdrawMoney(void);
        void ShowAllAccInfo(void) const;
        ~AccountHandler();
};

void AccountHandler::ShowMenu(void) const
{
        cout << "-----Menu------" << endl;
        cout << "1. 계좌개설" << endl;
        cout << "2. 입    금" << endl;
        cout << "3. 출    금" << endl;
        cout << "4. 계좌정보 전체 출력" << endl;
        cout << "5. 프로그램 종료" << endl;
}

void AccountHandler::MakeAccount(void)  //사용자의 정보를 저장하는 함수이다.
{
        int id;
        char name[NAME_LEN];
        int balance;

        cout << "[계좌개설]" << endl;
        cout << "계좌 ID: ";     cin >> id;
        cout << "이  름: ";      cin >> name;
        cout << "입금액: ";      cin >> balance;
        cout << endl;

        accArr[accNum++] = new Account(id, balance, name);
        //accArr 은 Account 의 포인터객체 배열이므로 생성되는 객체들을 동적으로 생성하고
        // 그 객체의 주소값을 저장하는 배열이다.
        //new 를 통해 Account 클래스의 생성자가 호출된다.
```

```
        //생성된 객체의 주소를 accArr[accNum]공간에 저장하고 accNum 의 값을 1
증가시킨다.
}

void AccountHandler::DepositMoney(void)  //계좌 ID 와 입금액을 입력하는 함수이다.
{
        int money;
        int id;
        cout << "[입    금]" << endl;
        cout << "계좌 ID: ";      cin >> id;
        cout << "입금액: ";       cin >> money;

        for (int i = 0; i < accNum; i++)
        {
                if (accArr[i]->GetAccID() == id)
//accArr[i]에 저장된 주소가 가리키는 객체의 getaccid 멤버함수를 실행한다.
//멤버함수에서 반환되는 값이 id 와 같다면
//아래의 문장을 실행해준다.
                {
                        accArr[i]->Deposit(money);
//accArr[i]에 저장된 주소가 가리키는 객체의 deposit 멤버함수에
//money 값을 매개변수로 넣어준다.
                        cout << "입금완료" << endl << endl;
                        return;
                }
        }
        cout << "유효하지 않은 ID 입니다." << endl << endl;
}

void AccountHandler::WithdrawMoney(void)  //계좌 id 와 출금액을 입력받는 함수
{
        int money;
        int id;
        cout << "[출    금]" << endl;
        cout << "계좌 ID: ";      cin >> id;
        cout << "출금액: ";       cin >> money;

        for (int i = 0; i < accNum; i++)
        {
                if (accArr[i]->GetAccID() == id)
//accArr[i]에 저장된 주소가 가리키는 객체의 getaccid 멤버함수에서
//반환되는 값과 id 가 같다면
//아래의 문장을 실행해준다.
                {
                        if (accArr[i]->Withdraw(money) == 0)
//withdraw 함수에 money 매개변수를 넣어서 반환되는 값이 0 이라면
                        {
                                cout << "잔액부족" << endl << endl;
```

```
                                //잔액 부족을 출력해준다.
                                return;
                        }

                        cout << "출금완료" << endl << endl;
                        return;
                }
        }
        cout << "유효하지 않은 ID 입니다." << endl << endl;
}

AccountHandler::AccountHandler() : accNum(0)
//AccountHandler 객체를 생성할때 아무것도 적어주지 않았을때
//이 생성자를 실행하고 accNum 을 0 으로 초기화 해준다.
{  }

void AccountHandler::ShowAllAccInfo(void) const
{
        for (int i = 0; i < accNum; i++)
        {
                accArr[i]->ShowAccInfo();
//accArr[i]에 저장된 주소의 showaccinfo 멤버함수에 접근하여 실행해준다.
                cout << endl;
        }
}

AccountHandler::~AccountHandler()  //소멸자
{
        for (int i = 0; i < accNum; i++)
                delete accArr[i];
//동적으로 생성된 Account 의 정보가 저장된 배열의 주소값을 원소로 가지는
//포인터 배열 accArr[i]를 삭제해줘야한다.
//class 를 벗어나도 포인터로 만든 변수는 메모리상에 남아있기 때문에 없애줘야한다.
}



/*
 * 컨트롤 클래스 AccountHandler 중심으로 변경된 main 함수
 */

int main(void)
{
        AccountHandler manager;
        int choice;

        while (1)
        {
                manager.ShowMenu();
```

```
                cout << "선택: ";
                cin >> choice;
                cout << endl;

                switch (choice)
                {
                case MAKE:
                        manager.MakeAccount();
                        break;
                case DEPOSIT:
                        manager.DepositMoney();
                        break;
                case WITHDRAW:
                        manager.WithdrawMoney();
                        break;
                case INQUIRE:
                        manager.ShowAllAccInfo();
                        break;
                case EXIT:
                        return 0;
                default:
                        cout << "Illegal selection.." << endl;
                }
        }

        return 0;
}
```

1.3 프로그램 1과의 차이점 분석

1.1의 프로그램은 class를 한 개를 사용하여 값의 반환을 해주고 실제 함수는 외부에 만들어서 접근하는 방식 이었다면 2의 프로그램은 Account클래스와 AccountHandler함수를 has-a관계로 묶어서 사용하여서 함수를 외부에 따로 만드는 방식이 아닌 클래스로 접근하여 코드를 더욱 깔끔하게 만들었습니다.

2.전역변수로 배열을 만들지 않고 class내부에서 배열을 만듦으로써 코드 간의 응집력을 높여주었다.

3.포인터 배열에 대한 소멸자를 main문에서 하지 않고 class내부에서 하게 되었다.

4.포인터 배열에 대한 접근이 어려워졌지만 정보를 숨길 수 있기 때문에 더 좋은 코드가 되었다.

5.멤버함수의 외부정의로 가독성을 향상시켜줬다.

3. 결론 및 소감

Class를 통해 구조화된 프로그램의 작성에 대해 배울 수 있어서 좋았습니다.

객체의 포인터 배열을 통해 한번에 100개의 객체를 저장할 수 있는 것에 대해 알게 되었습니다.

앞으로의 공부 방향을 class의 구현을 통한 객체지향 프로그래밍에 초점을 잡아야겠다 라는 생각을 했습니다.

코드를 분석하는데 있어서 이 코드의 포인터가 어떤 내용을 담고 있는데 어디서 어떻게 호출하는지에 대해 직관적으로 아는데 어려움이 있었고 디버깅을 통해서 알 수 있었기 때문입니다.

잘 경험해보지 못하는 객체지향 프로그래밍의 코드를 보고 분석할 수 있는 좋은 기회였던 것 같습니다.

감사합니다.