2023 – 2학기 DB응용 프로젝트 결과보고서

기본 사항			
분반 및 조	(9) 조	**담당교수**	이중화
프로젝트 주제	동의 컴소 쇼핑몰		
팀구성원 (학번/성명)	20192336 김남주 20193172 김민곤 20193174 이종민 20193183 오승헌		
개발 환경	Visual Studio 2019, Proc, Oracle SQL Developer19.2.1		

1. 프로젝트 개요

- 전 세계적으로 온라인 쇼핑은 계속해서 성장하고 있다. 특히 최근 몇 년간 COVID-19의 영향으로 인해 온라인 쇼핑은 더욱 중요해지고 있다. 이에 따라 쇼핑몰 관련 시장의 성장과 잠재력이 있는 주제이며 다양한 종류의 데이터를 다루어야 한다. 제품 정보, 주문 내역, 고객 정보, 재고 관리 등 다양한 데이터 요구 사항이 있고, 이를 효율적으로 관리하기 위해 데이터베이스를 구축하기 위해 이번 프로젝트에 알맞은 주제라 생각하였음

2. 프로젝트 구성요소

목표/기준 설정	본 프로젝트의 최종 목표는 쇼핑몰 관련 시장의 성장과 잠재력을 고려하여 온라인 쇼핑의 성장 추세에 맞춰 프로그램을 개발하는 것이 목표이다. 사용자에게 상품의 정보 및 재고 등을 제공하고 구매와 주문내역을 확인 할 수 있으며 관리자는 상품에 대하여 추가, 수정, 조회가 가능하고 사용자의 정보 또한 조회 가능하도록 한다.
분 석	총 프로젝트 진행 기간은 24일로 계획한다. 주제를 선정하고 관련 자료 수집을 시작으로 프로젝트를 진행한다. 이후 DB설계 및 화면 구성을 진행하고, 기능별로 역할을 분담하여 DB 및 소스 코드를 구현한다. 마지막으로 단위 테스트를 진행하면서 하난의 프로그램으로 통합한다.
제 작	총 24일에 걸쳐 프로젝트를 진행하였으며 매주 회의를 통해 금주의 목표치를 설정하여 그 일정을 바탕으로 프로젝트를 진행한다.
결과도출	화면 구성에 필요한 텍스트 기반 파일, 프로그램 구현을 위한 Pro*c 파일 및 프로그램 작동을 위한 C 파일을 도출한다.
평 가	설계 과정에서 도출한 기능 요구사항을 단위 테스트를 진행하여 초기 목표치 대비 완성도를 비교하여 프로젝트를 평가한다.

3. 현실적제한조건

신뢰성	DB를 구현할 때, 테이블에서 중복적으로 사용되는 스키마가 없어야하며 테이블에 외래키 설정을 하여 데이터에 오류가 없도록 한다.
미 학	사용자가 프로그램을 이용할 때, 화면의 이동이 자연스러워야하며 관리자의 경우 여러 작업들에 대해 헷갈리지 않게 명확히 구현해야한다.
생산성	본 프로젝트를 진행하여 하나의 개별적인 쇼핑몰 프로그램을 구현하였음.

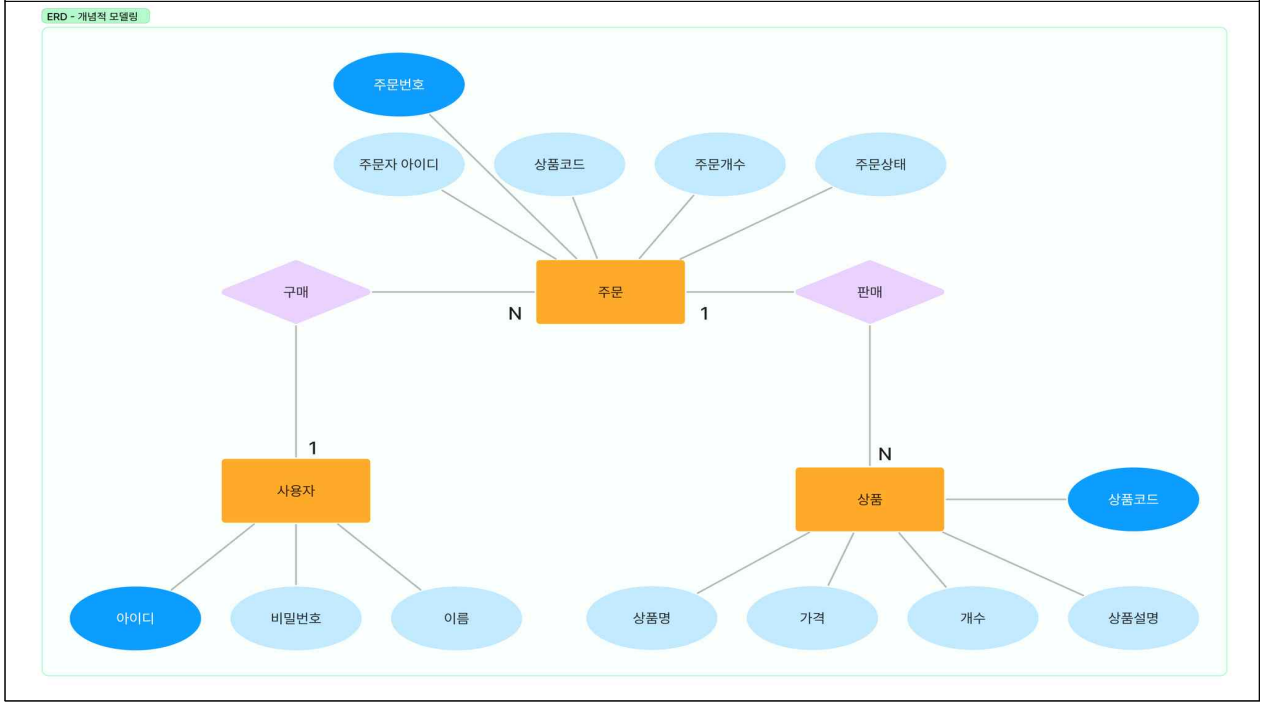
4. 프로젝트 설계 및 구현 내용

4.1 응용분석

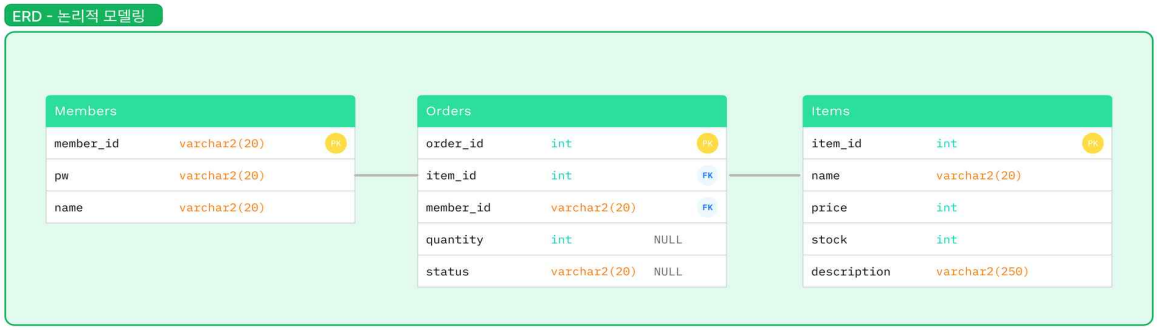
본 프로젝트에서 구축하고자 하는 쇼핑몰 시스템은 사용자에게 회원으로 가입 할 수 있어야 하며 제품의 재고와 가격을 제공받으며 관리자는 상품에 대해 등록, 수정, 삭제 조회가 가능해야하고, 사용자의 주문에 대해 상태를 업데이트 할 수 있어야 하며 주문 내역을 확인할 수 있어야 한다

4.2 E-R 다이어그램

ERD – 개념적 모델링



ERD – 논리적 모델링



4.3 테이블명세서

테이블명	MEMBERS	Table 기술서	작성일	23.12.07	Page
System	MASSAGE		작성자	오승헌	/
테이블 설명	고객 정보를 관리한다.				

No	Attribute	Data Type	NN	Ky	Default	Description
1	member_id	VARCHAR2(20)	Y	PK	-	고객아이디
2	pw	VARCHAR2(20)	N	-	-	고객비밀번호
3	name	VARCHAR2(20)	N	-	-	고객이름

비 고

PRIMARY KEY member_id)

테이블명	ITEMS	Table 기술서	작성일	23.12.07	Page
System	MASSAGE		작성자	오승헌	/
테이블 설명	상품의 정보를 관리한다.				

No	Attribute	Data Type	NN	Ky	Default	Description
1	item_id	NUMBER	Y	PK	-	상품번호
2	name	VARCHAR2(20)	N	-	-	상품명
3	price	INT	N	-	-	상품가격
4	stock	INT	N	-	-	상품수량
5	description	VARCHAR2(250)	N	-	-	상품설명

비 고

PRIMARY KEY (item_id)

테이블명	ORDER	Table 기술서	작성일	23.12.07	Page
System	MASSAGE		작성자	오승헌	/

테이블 설명	주문내역을 관리한다.

No	Attribute	Data Type	NN	Ky	Default	Description
1	order_id	NUMBER	Y	PK	-	주문번호
2	item_id	INT	N	-	-	상품번호
3	member_id	VARCHAR2(20)	N	-	-	고객아이디
4	quantity	INT	Y	FK	NULL	주문수량
5	status	VARCHAR2(20)	Y	FK	NULL	주문상태

비 고

PRIMARY KEY (order_id)

FOREIGN KEY (item_id) REFERENCES Items(item_id)

FOREIGN KEY (member_id) REFERENCES Members(member_id)

4.4 기능분석

ID	기능 요구사항	기능 상세 설명	개발자
SFR-100	고객 관리		
SFR-101	로그인 진행	- 고객은 주어진 아이디와 패스워드를 통해 로그인을 진행한다.	이종민
SFR-102	회원가입 진행	- 고객은 아이디, 패스워드, 이름을 입력하여 회원가입을 진행한다. - 아이디 중복확인 후 중복일 시 오류 메시지 출력 - 패스워드 입력 시 비밀번호 확인을 위한 입력을 받는다.	김민곤
SFR-200	관리자 메뉴		
SFR-201	상품 추가	- 상품명, 가격, 상품설명, 개수를 입력받고, 테이블에 반영할 수 있다.	김남주
SFR-202	상품 조회	- 모든 상품의 정보를 조회 가능하다. - 상품의 정보를 변경 가능하다. - 상품의 정보를 삭제 가능하다.	오승헌
SFR-203	사용자 관리	- 사용자의 정보를 불러올 수 있다.	김민곤
SFR-204	매출 관리	- 총 매출액 정보를 불러올 수 있다.	이종민
SFR-205	주문 관리	- 주문 리스트를 불러올 수 있다. - 요청한 주문에 대하여 상태를 처리할 수 있다.	오승헌
SFR-300	쇼핑 관리		
SFR-301	상품 상세	- 수량을 입력 받고, 상품을 구매할 수 있다.	김남주
SFR-302	주문 조회	- 주문한 상품의 리스트를 출력한다. - 주문한 상품의 배송상태를 출력한다.	이종민

4.5 프로그램 수행 내용

AddMenu.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "screenControl.h"
#include "MainMenu.h"
#include "AddMenu.h"
#include "ManagerMenu.h"
#include "ProductSearch.h"
#include "UserfileMenu.h"
#include "SaleMenu.h"
#include "OrderMenu.h"
void ManagerMenu::printSrc()
{
	while (1) {
		clrscr();
		print_screen("./screen/manager_screen.txt");

		int y = cursorControl(33, 9, 19, 2);
		switch (y) {
		case P_ADD:
			AddMenu addmenu;
			addmenu.printSrc();
			break;
		case P_SEARCH:
			ProductSearch productsearch;
			productsearch.printSrc();
			break;
		case USERS:
			UserfileMenu userfilemenu;
			userfilemenu.printSrc();
			break;
		case SALE:
			SaleMenu salemenu;
			salemenu.printSrc();
			break;
		case ORDER:
			OrderMenu ordermenu;
			ordermenu.printSrc();
```

```
                        break;
                case FIRST:
                        MainMenu mainmenu;
                        mainmenu.printSrc();
                }
        }
}
```

코드 설명

사용자로부터 상품의 이름, 가격, 설명, 재고를 입력받는다.
gets_s() 함수를 사용하여 이름과 설명을 입력받고, cin을 사용하여 가격과 재고를 입력받는다.
cin.ignore() 함수로 입력 버퍼를 비워준다.
cursorControl() 함수를 호출하여 사용자로부터 COMPLETE 또는 BACK 값을 입력받는다.
ItemsDto라는 구조체 변수 item을 선언하고, 입력받은 정보들을 해당 변수에 저장한다.
cursorControl() 함수의 반환값에 따라 switch 문을 통해 .COMPLETE일 경우 addItem() 함수를 호출하여 상품을 추가하고, BACK일 경우 아무 동작도 수행하지 않고 종료합니다.

실행 결과

ManagerMenu.cpp

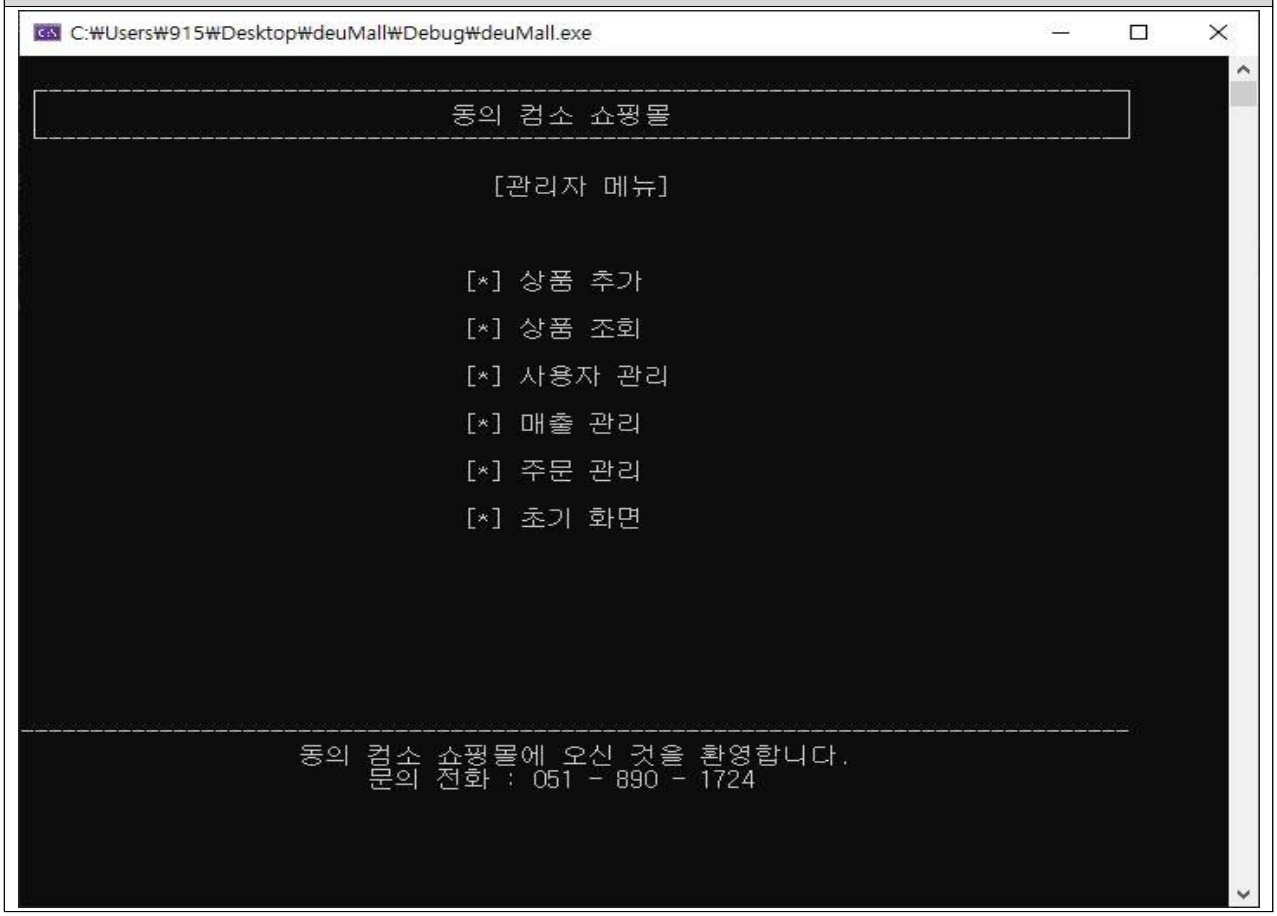
```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "screenControl.h"
#include "MainMenu.h"
#include "AddMenu.h"
#include "ManagerMenu.h"
#include "ProductSearch.h"
#include "UserfileMenu.h"
#include "SaleMenu.h"
#include "OrderMenu.h"
void ManagerMenu::printSrc()
{
	while (1) {
		clrscr();
		print_screen("./screen/manager_screen.txt");

		int y = cursorControl(33, 9, 19, 2);
		switch (y) {
		case P_ADD:
			AddMenu addmenu;
			addmenu.printSrc();
			break;
		case P_SEARCH:
			ProductSearch productsearch;
			productsearch.printSrc();
			break;
		case USERS:
			UserfileMenu userfilemenu;
			userfilemenu.printSrc();
			break;
		case SALE:
			SaleMenu salemenu;
			salemenu.printSrc();
			break;
		case ORDER:
			OrderMenu ordermenu;
			ordermenu.printSrc();
			break;
		case FIRST:
```

```
                    MainMenu mainmenu;
                    mainmenu.printSrc();
                }
            }
}
```

코드 설명

selectOrder(user.member_id) 함수를 호출하여 해당 사용자의 주문 목록을 가져온다. 이 목록은 vector<OrdersDto> 형태로 저장된다.
주문 목록을 순회하면서 각 주문의 회원 정보와 상품 정보를 가져온다. 이를 위해 findByMemberId() 함수와 findByItemId() 함수를 호출하여 주문에 해당하는 회원과 상품을 찾는다.
화면의 특정 위치에 주문 정보를 출력한다. 이를 위해 gotoxy() 함수와 printf() 함수를 사용한다.
cursorControl() 함수를 호출하여 사용자로부터 EXIT 값을 입력받는다.
switch 문을 통해 입력된 값에 따라 다음 동작을 수행한다.
EXIT일 경우, 함수를 종료하고 이전 메뉴로 돌아간다.
"setUser" 함수는 매개변수로 전달된 회원 정보를 클래스 내부의 멤버 변수인 "user"에 저장하는 역할을 한다.

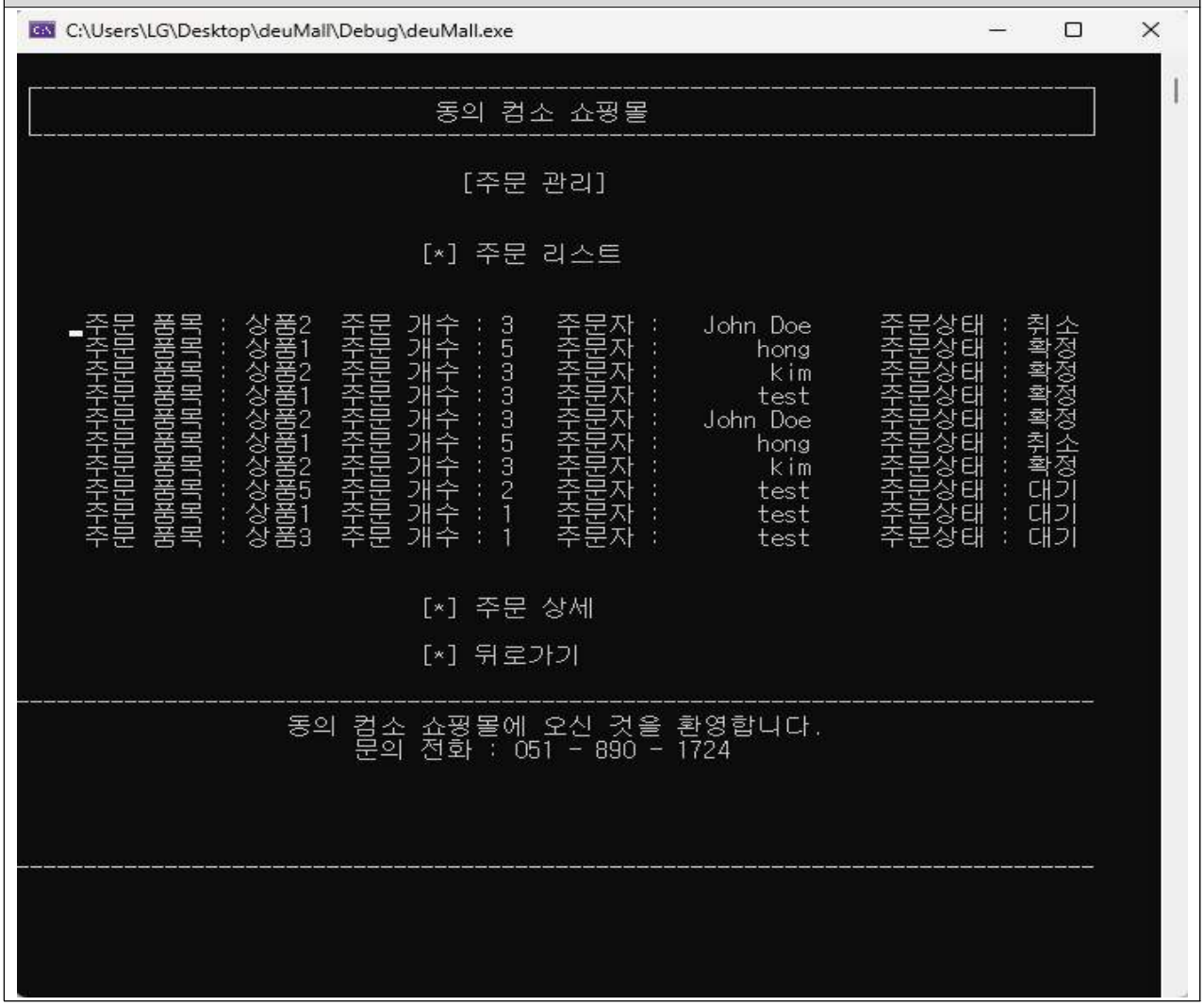
실행 결과



OrderMenu.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <vector>
#include "screenControl.h"
#include "Dao.h"
#include "OrderManage.h"
#include "OrderMenu.h"
using namespace std;
void OrderMenu::printSrc()
{
	while (1) {
		clrscr();
		print_screen("./screen/ordermanagement_screen.txt");
		vector<OrdersDto> orders = selectOrders();
		int i =0;
		for (OrdersDto order : orders) {
			MembersDto member = findByMemberId(order.member_id);
			ItemsDto item = findByItemId(order.item_id);
			gotoxy(5, 11 + i++);
			printf("주문 품목 : %s₩t주문 개수 : %d₩t주문자 : %10s₩t주문상태 : %s",
item.name, order.quantity, member.name, order.status);
		}
		int s = cursorControl(4, 11, 11 + orders.size() -1, 1);
		gotoxy(3, s);
		printf("*");
		int orderIndex = s -11;
		int y = cursorControl(31, 23, 25, 2);
		switch (y) {
		case CLICK:
			OrderManage ordermanage;
			ordermanage.setOrderIndex(orderIndex);
			ordermanage.printSrc();
			break;
		case BACK:
			return;
		}
	}
}
```

코드 설명
selectOrders() 함수를 호출하여 주문 목록을 가져온다. 이 목록은 vector<OrdersDto> 형태로 저장한 후 orderIndex에 해당하는 주문 정보를 orders에서 가져온다 gotoxy() 함수를 사용하여 주문 정보를 출력하고, 각 정보는 printf() 함수를 통해 출력됩니다. cursorControl() 함수를 호출하여 사용자로부터 COMPLETE 또는 CANCLE 중 하나의 값을 입력받습니다. switch 문을 통해 입력된 값에 따라 COMPLETE일 경우, orderStatus() 함수를 호출하여 해당 주문의 상태를 "확정"으로 변경한다. CANCLE일 경우, orderStatus() 함수를 호출하여 해당 주문의 상태를 "취소"로 변경한다 "setOrderIndex" 함수는 인자로 받은 값을 orderIndex 멤버 변수에 저장한다. 이 함수를 사용하여 orderIndex 값을 설정할 수 있다.
실행 결과



ProductChangecpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <vector>
#include <iostream>
#include "screenControl.h"
#include "ProductChange.h"
#include "ManagerMenu.h"
#include "Dao.h"
using namespace std;
void ProductChange::printSrc()
{
	clrscr();
	print_screen("./screen/productchange_screen.txt");
	vector<ItemsDto> items = selectItems();
	ItemsDto item = items.at(itemIndex);
	gotoxy(13, 10);
	printf("%s", item.name);
	gotoxy(13, 12);
	printf("%d", item.price);
	gotoxy(13, 14);
	printf("%s", item.description);
	gotoxy(13, 16);
	printf("%d", item.stock);

	gotoxy(47, 10);
	gets_s(item.name);
	gotoxy(47, 12);
	scanf("%d", &item.price);
	getchar();
	gotoxy(47, 14);
	gets_s(item.description);
	gotoxy(47, 16);
	scanf("%d", &item.stock);
	getchar();
	int y = cursorControl(35, 21, 23, 2);
	switch (y) {
	case COMPLETE:
		updateItem(item.item_id, item);
```

```
                break;
        case BACK:
                break;
        }
}
void ProductChange::setItemIndex(int i) {
        this->itemIndex = i;
}
```

코드 설명

selectItems() 함수를 호출하여 상품 목록을 가져오며 이 목록은 vector<ItemsDto> 형태로 저장된다.
itemIndex에 해당하는 상품 정보를 items에서 가져온다.
gotoxy() 함수를 사용하여 상품 정보를 출력한다. 각 정보는 printf() 함수를 통해 출력된다.
gets_s()와 scanf() 함수를 사용하여 사용자로부터 새로운 상품 정보를 입력받는다.
gets_s() 함수를 사용하여 상품 이름과 상품 설명을 입력받는다.
scanf() 함수를 사용하여 상품 가격과 상품 재고를 입력는다.
cursorControl() 함수를 호출하여 사용자로부터 COMPLETE 또는 BACK 중 하나의 값을 입력받는다.
switch 문을 통해 입력된 값에 따라 COMPLETE일 경우, updateItem() 함수를 호출하여 상품의 정보를 업데이트한다.. BACK일 경우, 아무 동작도 수행하지 않고 함수를 종료한다.
"setItemIndex" 함수는 인자로 받은 값을 itemIndex 멤버 변수에 저장한다. 이 함수를 사용하여 itemIndex 값을 설정할 수 있다.

실행 결과

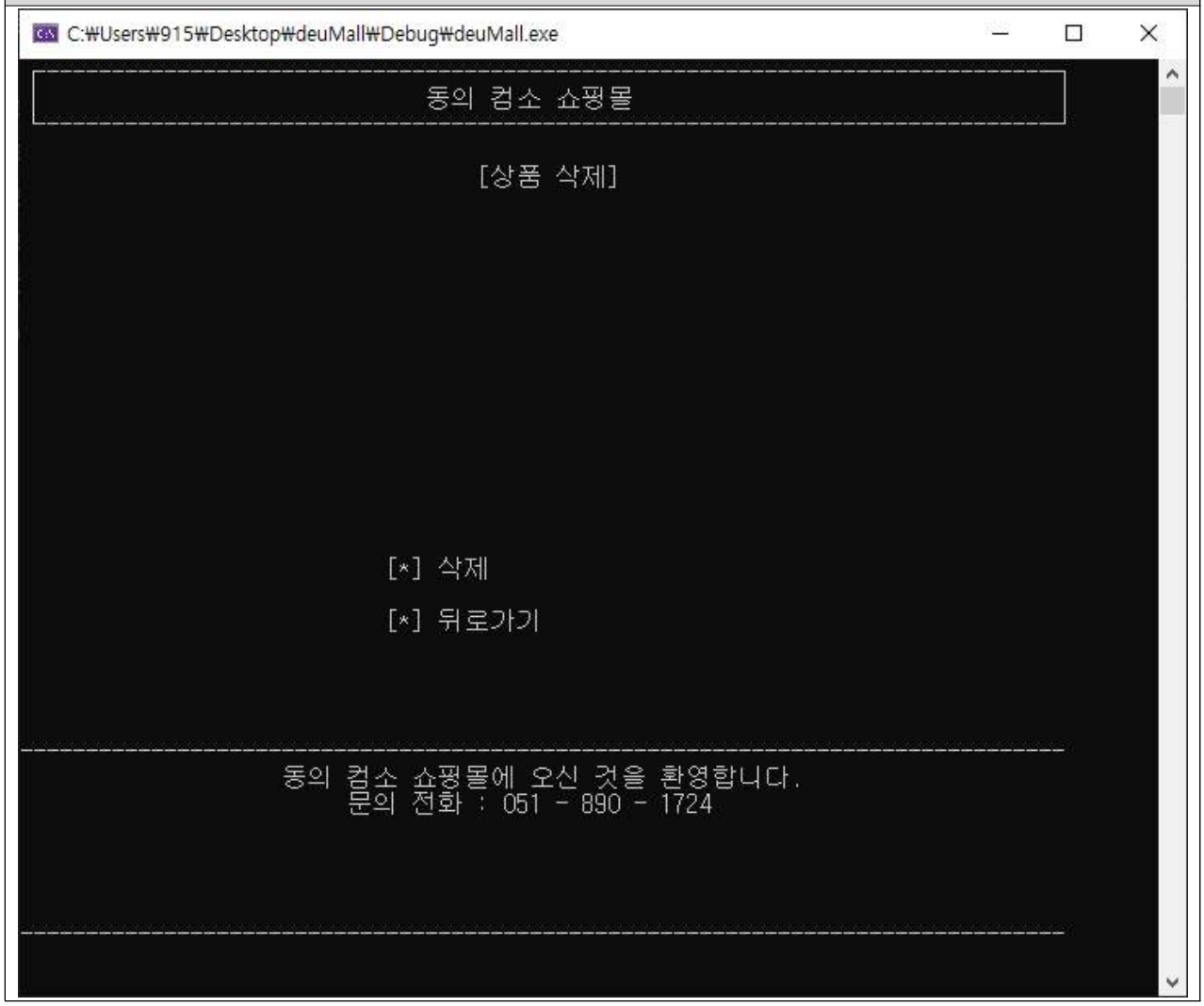
ProductDeletecpp

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "screenControl.h"
#include "ProductDelete.h"
#include "ManagerMenu.h"
#include "Dao.h"
void ProductDelete::printSrc()
{
	clrscr();
	print_screen("./screen/productdelete_screen.txt");
	vector<ItemsDto> items = selectItems();
	ItemsDto item = items.at(itemIndex);
	int y = cursorControl(29, 20, 22, 2);
	switch (y) {
		// 디비 연결 후 상품 리스트 클릭 시 상품 변경 및 삭제 화면 구성해야함
	case REMOVE:
		deleteItem(item.item_id);
		break;
	case BACK:
		break;
	}
}
void ProductDelete::setItemIndex(int i) {
	this->itemIndex = i;
}
```

코드 설명

selectItems() 함수를 호출하여 상품 목록을 가져온다. 이 목록은 vector<ItemsDto> 형태로 저장된다. itemIndex에 해당하는 상품 정보를 items에서 가져온다.
cursorControl() 함수를 호출하여 사용자로부터 REMOVE 또는 BACK 중 하나의 값을 입력받는다.
switch 문을 통해 입력된 값에 따라 REMOVE일 경우, deleteItem() 함수를 호출하여 상품을 삭제한다.
BACK일 경우, 아무 동작도 수행하지 않고 함수를 종료한다.
"setItemIndex" 함수는 인자로 받은 값을 itemIndex 멤버 변수에 저장한다. 이 함수를 사용하여 itemIndex 값을 설정할 수 있다.

실행 결과


C:₩Users₩915₩Desktop₩deuMall₩Debug₩deuMall.exe
동의 컴소 쇼핑몰
[상품 삭제]
[*] 삭제
[*] 뒤로가기
동의 컴소 쇼핑몰에 오신 것을 환영합니다.
문의 전화 : 051 - 890 - 1724

ProductDetails.cpp

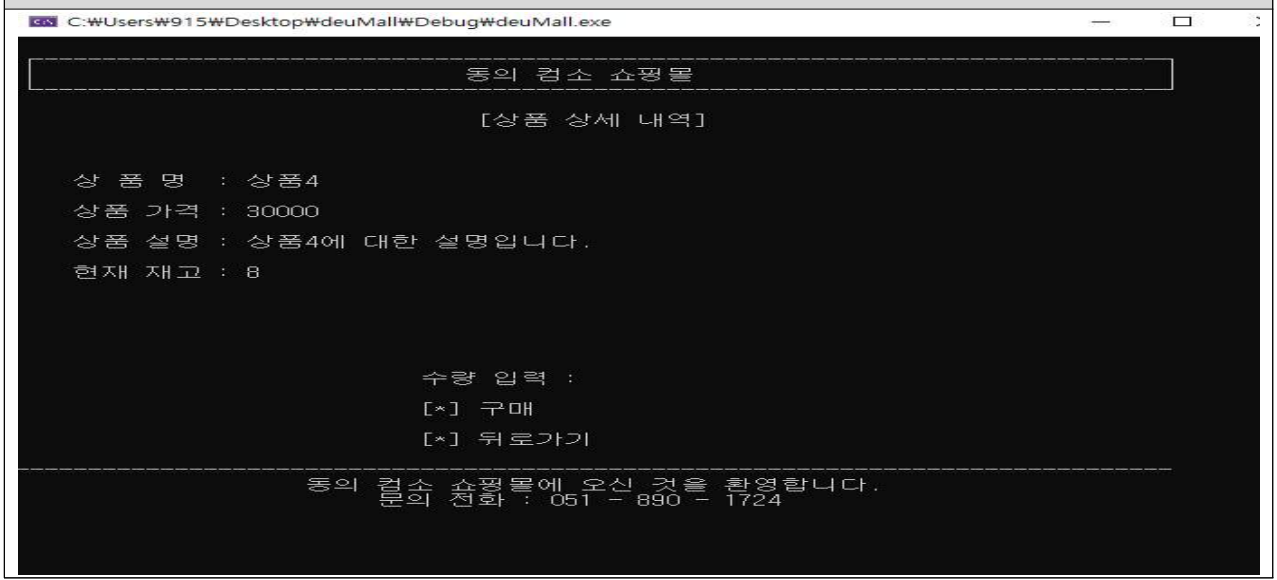
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <vector>
#include "screenControl.h"
#include "ProductDetails.h"
#include "ManagerMenu.h"
#include "TrackingOrder.h"
#include "Dao.h"
void ProductDetails::printSrc()
{
	clrscr();
	print_screen("./screen/productdetails_screen.txt");
	vector<ItemsDto> items = selectItems();
	ItemsDto item = items.at(index);

	gotoxy(16, 9);
	printf("%s", item.name);
	gotoxy(16, 11);
	printf("%d", item.price);
	gotoxy(16, 13);
	printf("%s", item.description);
	gotoxy(16, 15);
	printf("%d", item.stock);
	int quantity;
	gotoxy(42, 22);
	scanf("%d", &quantity);
	getchar();
	int y = cursorControl(29, 24, 26, 2);
	OrdersDto order;
	order.item_id = item.item_id;
	strcpy(order.member_id, user.member_id);
	order.quantity = quantity;
	strcpy(order.status, "대기");
	switch (y) {
	case BUY:
		item.stock -= quantity;
		updateItem(item.item_id, item);
		adddorder(order);
```

```
                break;
        case BACK:
                break;
        }
}
void ProductDetails::setIndex(int i) {
        this->index = i;
}
void ProductDetails::setUser(MembersDto u) {
        this->user = u;
}
```

코드 설명

selectItems() 함수를 호출하여 상품 목록을 가져온다. 이 목록은 vector<ItemsDto> 형태로 저장된다. index에 해당하는 상품 정보를 items에서 가져온다.
gotoxy() 함수를 사용하여 상품 정보를 출력한다. 각 정보는 printf() 함수를 통해 출력된다.
scanf() 함수를 사용하여 사용자로부터 상품의 수량을 입력받는다.
cursorControl() 함수를 호출하여 사용자로부터 BUY 또는 BACK 중 하나의 값을 입력받는다.
switch 문을 통해 입력된 값에 따라 BUY일 경우, 상품의 재고를 감소시키고, 상품을 주문 목록에 추가하며. BACK일 경우, 아무 동작도 수행하지 않고 함수를 종료한다.
"setIndex" 함수는 인자로 받은 값을 index 멤버 변수에 저장한다. 이 함수를 사용하여 index 값을 설정할 수 있다,
"setUser" 함수는 인자로 받은 MembersDto 타입의 값을 user 멤버 변수에 저장한다. 이 함수를 사용하여 user 값을 설정할 수 있다.

실행 결과



ProducuctSearch.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <vector>
#include "screenControl.h"
#include "ProductSearch.h"
#include "ManagerMenu.h"
#include "ProductChange.h"
#include "ProductDelete.h"
#include "Dao.h"
using namespace std;
void ProductSearch::printSrc()
{
	while (1) {
		clrscr();
		print_screen("./screen/inquiryproduct_screen.txt");
		vector<ItemsDto> items = selectItems();  //아이템 리스트 받아옴
		int size = items.size();
		int i =0;
		for (ItemsDto item : items) {	//리스트 돌면서 하나씩 출력
			gotoxy(20, 11 + i++);
			printf("%s₩t%7d 원₩t%4d 개", item.name, item.price, item.stock);
		}
		int prodY = cursorControl(19, 11, 11 + (size -1), 1); //선택한 아이템의 Y 값 반환
		int itemIndex = prodY -11; //아이템 인덱스값 items.at(itemIndex)로 접근하면 선택한 아이템 접근 가능
		gotoxy(18, prodY);  //선택한 아이템 값에 마킹
		printf("*");
		int y = cursorControl(31, 22, 26, 2);
		switch (y) {
		case CHANGE:
			ProductChange productchange;
			productchange.setItemIndex(itemIndex);
			productchange.printSrc();

			break;
		case DELETE:
			ProductDelete productdelete;
			productdelete.setItemIndex(itemIndex);
			productdelete.printSrc();
```

```
                        break;
                case RETURN:
                        return;
                }
        }
}
```

코드 설명

selectItems() 함수를 호출하여 상품 목록을 가져온다. 이 목록은 vector<ItemsDto> 형태로 저장된다. 상품 목록을 순회하면서 각 상품의 이름, 가격, 재고를 출력한다. 이를 위해 for문과 printf() 함수를 사용한다.
cursorControl() 함수를 호출하여 사용자로부터 선택한 상품의 Y 값, 즉 prodY를 반환받는다.
prodY를 기반으로 선택한 상품을 표시하기 위해 해당 위치에 "*"을 출력한다.
cursorControl() 함수를 호출하여 사용자로부터 CHANGE, DELETE, RETURN 중 하나의 값을 입력받는다.
switch 문을 통해 입력된 값에 따라 CHANGE일 경우, "ProductChange" 클래스의 객체를 생성하고 setItemIndex() 함수를 호출하여 선택한 상품의 인덱스를 설정한 후, printSrc() 함수를 호출하여 상품 변경 기능을 수행한다.
DELETE일 경우, "ProductDelete" 클래스의 객체를 생성하고 setItemIndex() 함수를 호출하여 선택한 상품의 인덱스를 설정한 후, printSrc() 함수를 호출하여 상품 삭제 기능을 수행한다.
RETURN일 경우, 함수를 종료하고 이전 메뉴로 돌아간다.

실행 결과

SaleMenu.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <vector>
#include "screenControl.h"
#include "SaleMenu.h"
#include "ManagerMenu.h"
#include "Dao.h"
using namespace std;
void SaleMenu::printSrc()
{
	clrscr();
	print_screen("./screen/sales_screen.txt");
	vector<OrdersDto> orders = selectOrders();
	int sale =0;
	for (OrdersDto order : orders) {
		ItemsDto item = findByItemId(order.item_id);
		if (strcmp(order.status, "확정") ==0) {
			sale += order.quantity * item.price;
		}
	}
	gotoxy(28, 14);
	printf("총 매출 : %d", sale);
	int y = cursorControl(29, 23, 23, 2);
	switch (y) {
	case BACK:
		break;
	}
}
```

코드 설명

selectOrders() 함수를 호출하여 주문 목록을 가져온다. 이 목록은 vector<OrdersDto> 형태로 저장된다.
매출을 저장할 변수인 sale을 0으로 초기화한다.
주문 목록을 순회하면서 각 주문의 상품 정보를 가져온다. 이를 위해 findByItemId() 함수를 호출하여 주문에 해당하는 상품을 찾는다.
주문의 상태가 "확정"인 경우에만 매출을 계산합니다. strcmp() 함수를 사용하여 주문의 상태를 확인한다.
매출 변수 sale에 주문 수량과 상품 가격을 곱한 값을 더한다.

화면의 특정 위치에 총 매출을 출력한다. 이를 위해 gotoxy() 함수와 printf() 함수를 사용한다.
cursorControl() 함수를 호출하여 사용자로부터 BACK 값을 입력받는다.
switch 문을 통해 입력된 값에 따라 BACK일 경우, 함수를 종료하고 이전 메뉴로 돌아갑니다.

실행 결과

ShoppingMenu.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <vector>
#include "ManagerMenu.h"
#include "ShoppingMenu.h"
#include "screenControl.h"
#include "LoginMenu.h"
#include "Dao.h"
#include "ProductDetails.h"
#include "TrackingOrder.h"
void ShoppingMenu::printSrc()
{
	while (1) {
		clrscr();
		print_screen("./screen/order_screen.txt");
		vector<ItemsDto> items = selectItems();  //아이템 리스트 받아옴
		int size = items.size();
		int i =0;
		for (ItemsDto item : items) {	//리스트 돌면서 하나씩 출력
			gotoxy(19, 10 + i++);
			printf("%s₩t%7d 원₩t%4d 개", item.name, item.price, item.stock);
		}
		int prodY = cursorControl(18, 10, 10 + (size -1), 1);	//선택한 아이템의 Y 값 반환
		int itemIndex = prodY -10;	//아이템 인덱스값 items.at(itemIndex)로 접근하면 선택한 아이템 접근 가능
		gotoxy(17, prodY);	//선택한 아이템 값에 마킹
		printf("*");
		int y = cursorControl(19, 22, 26, 2);
		switch (y) {
		case SEARCH:
			ProductDetails p;
			p.setIndex(itemIndex);
			p.setUser(this->user);
			p.printSrc();
			break;
		case VIEW:
			TrackingOrder t;
			t.setUser(user);
			t.printSrc();
```

```
                        break;
                case BACK:
                        return;
                }
        }
}
void ShoppingMenu::setUser(MembersDto u) {
        this->user = u;
}
```

코드 설명

selectOrder(user.member_id) 함수를 호출하여 해당 사용자의 주문 목록을 가져온다. 이 목록은 vector<OrdersDto> 형태로 저장된다.
주문 목록을 순회하면서 각 주문의 회원 정보와 상품 정보를 가져온다. 이를 위해 findByMemberId() 함수와 findByItemId() 함수를 호출하여 주문에 해당하는 회원과 상품을 찾는다.
화면의 특정 위치에 주문 정보를 출력한다. 이를 위해 gotoxy() 함수와 printf() 함수를 사용한다.
cursorControl() 함수를 호출하여 사용자로부터 EXIT 값을 입력받는다.
switch 문을 통해 입력된 값에 따라 EXIT일 경우, 함수를 종료하고 이전 메뉴로 돌아간다.
"setUser" 함수는 매개변수로 전달된 회원 정보를 클래스 내부의 멤버 변수인 "user"에 저장하는 역할을 한다.

실행 결과

TrackingOrderl.cpp

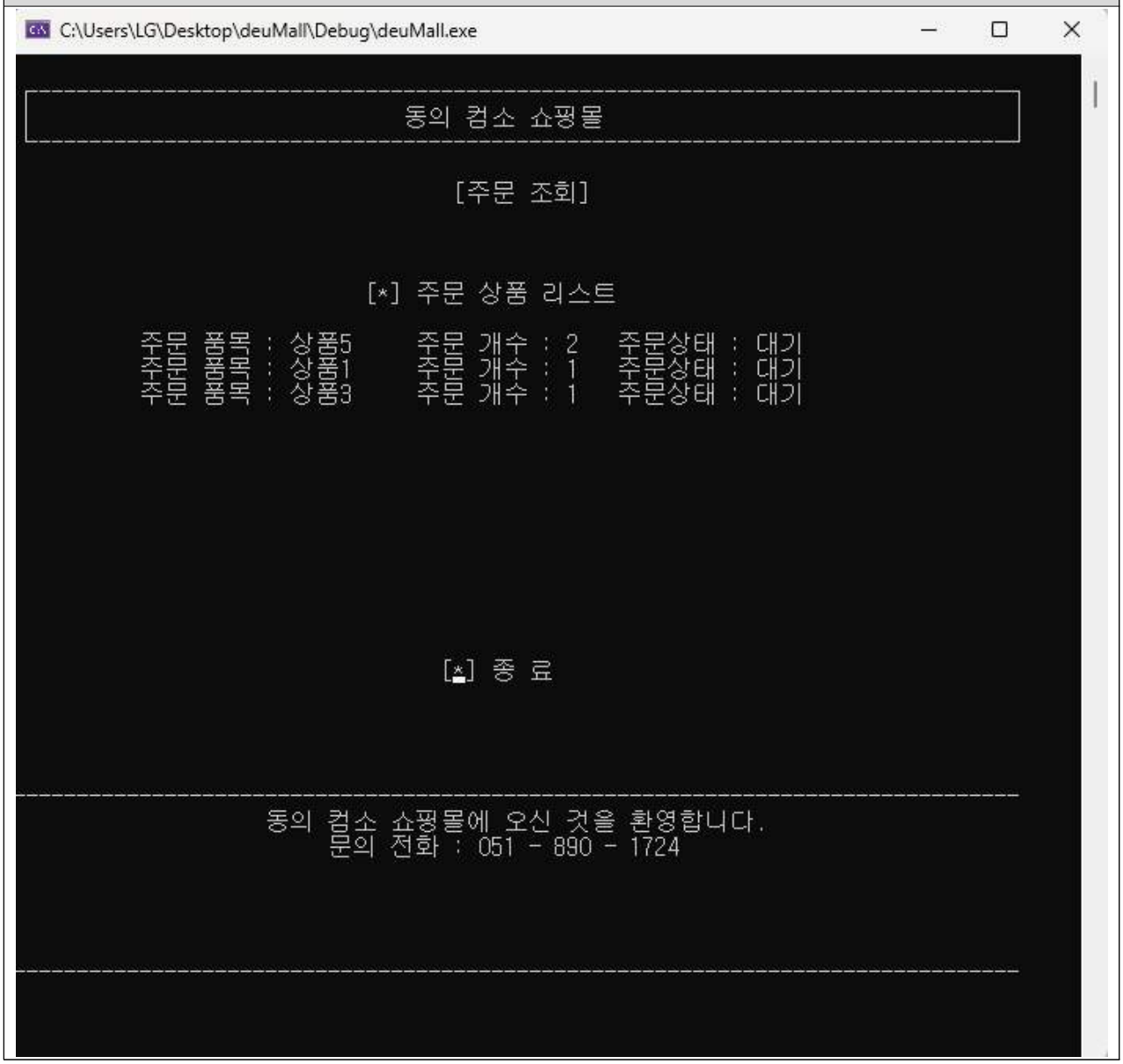
```
#include <stdlib.h>
#include <stdio.h>
```

```
#include <conio.h>
#include <vector>
#include "screenControl.h"
#include "TrackingOrder.h"
#include "MainMenu.h"
#include "Dao.h"
using namespace std;
void TrackingOrder::printSrc()
{
	clrscr();
	print_screen("./screen/ordertracking_screen.txt");
	vector<OrdersDto> orders = selectOrder(user.member_id);
	printf("test : %s", user.member_id);
	int i =0;
	for (OrdersDto order : orders) {
		MembersDto member = findByMemberId(order.member_id);
		ItemsDto item = findByItemId(order.item_id);
		gotoxy(10, 11 + i++);
		printf("주문 품목 : %s₩t주문 개수 : %d₩t주문상태 : %s", item.name,
order.quantity, order.status);
	}
	int y = cursorControl(35, 24, 24, 2);
	switch (y) {
	case EXIT:
		break;
	}
}
void TrackingOrder::setUser(MembersDto u) {
	this->user = u;
}
```

코드 설명

selectOrder(user.member_id) 함수를 호출하여 해당 사용자의 주문 목록을 가져온다. 이 목록은 vector<OrdersDto> 형태로 저장된다.
주문 목록을 순회하면서 각 주문의 회원 정보와 상품 정보를 가져온다. 이를 위해 findByMemberId() 함수와 findByItemId() 함수를 호출하여 주문에 해당하는 회원과 상품을 찾는다.
화면의 특정 위치에 주문 정보를 출력한다. 이를 위해 gotoxy() 함수와 printf() 함수를 사용한다.
cursorControl() 함수를 호출하여 사용자로부터 EXIT 값을 입력받는다.
switch 문을 통해 입력된 값에 따라 EXIT일 경우, 함수를 종료하고 이전 메뉴로 돌아간다.
"setUser" 함수는 매개변수로 전달된 회원 정보를 클래스 내부의 멤버 변수인 "user"에 저장하는 역할을 한다.

실행 결과


C:\Users\LG\Desktop\deuMall\Debug\deuMall.exe
동의 컴소 쇼핑몰
[주문 조회]
[*] 주문 상품 리스트
주문 품목 : 상품5 주문 개수 : 2 주문상태 : 대기
주문 품목 : 상품1 주문 개수 : 1 주문상태 : 대기
주문 품목 : 상품3 주문 개수 : 1 주문상태 : 대기
[*] 종 료
동의 컴소 쇼핑몰에 오신 것을 환영합니다.
문의 전화 : 051 - 890 - 1724

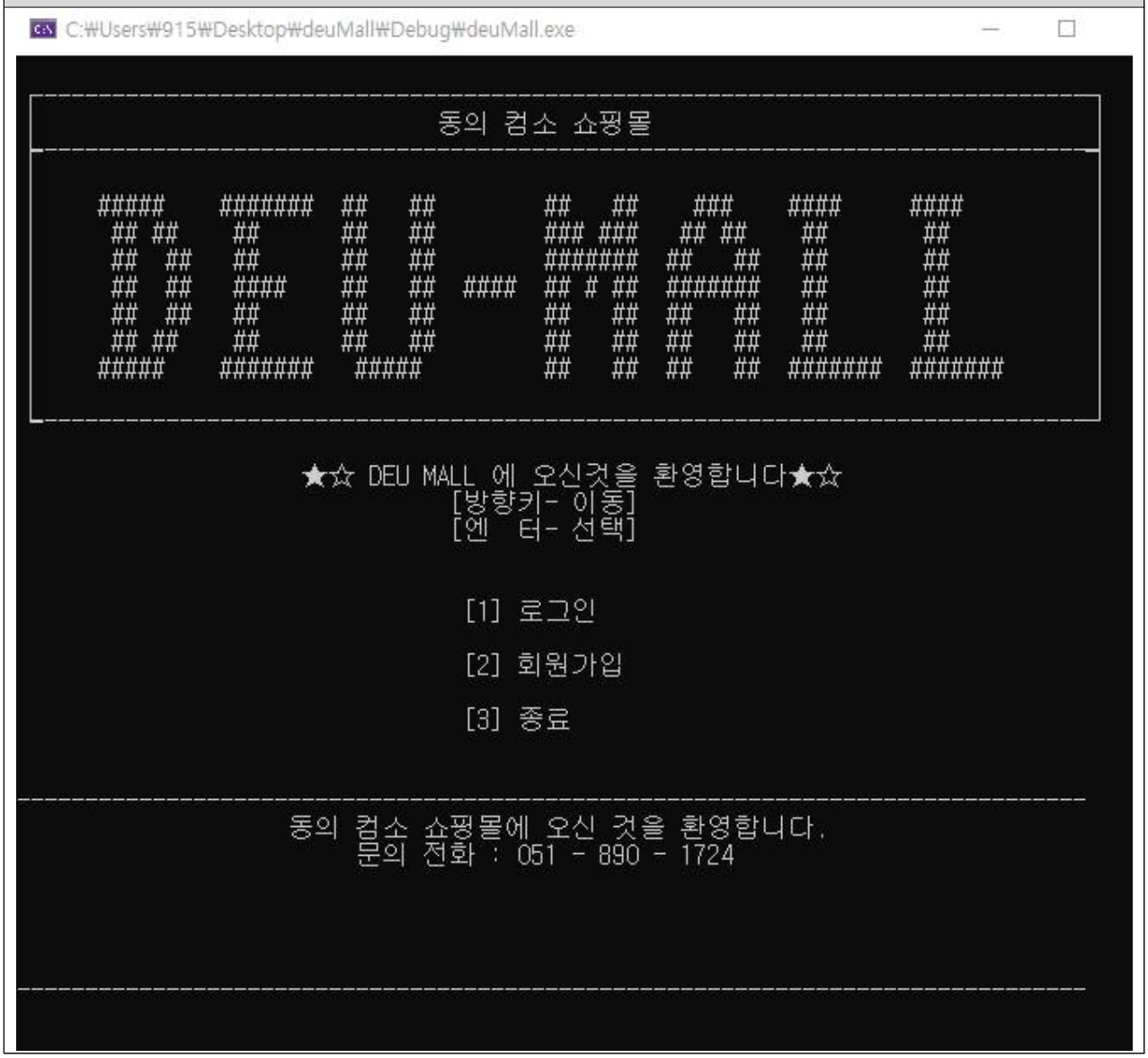
Main.cpp

```
void MainMenu::printSrc()
{
	while (1) {
		clrscr();
		print_screen("./screen/shopping_mall_screen.txt");

		int y = cursorControl(34, 20, 24, 2);
		switch (y) {
		case LOGIN:
			LoginMenu loginmenu;
			loginmenu.printSrc();
			break;
		case SIGNUP:
			SingupMenu singupmenu;
			singupmenu.printSrc();
			break;
		case EXIT:
			exit(0);
		}
	}
}
```

코드 설명

사용자가 시스템을 실행하면 쇼핑몰 메인화면을 출력하도록 한다.
사용자는 커서를 방향키로 이동가능하며, 엔터키로 메뉴를 선택할 수 있도록 한다.
사용자는 로그인, 회원가입, 종료 메뉴를 선택할 수 있다.
로그인 메뉴를 선택하면 로그인 화면을 출력하도록 한다.
회원가입 메뉴를 선택하면 회원가입 화면을 출력하
도록 한다.

실행 결과


C:₩Users₩915₩Desktop₩deuMall₩Debug₩deuMall.exe
동의 컴소 쇼핑몰
DEU MALL
★☆ DEU MALL 에 오신것을 환영합니다★☆
[방향키- 이동]
[엔 터- 선택]
[1] 로그인
[2] 회원가입
[3] 종료
동의 컴소 쇼핑몰에 오신 것을 환영합니다.
문의 전화 : 051 - 890 - 1724

LoginMenu.cpp

```
void LoginMenu::printSrc()
{
	while (1) {
		clrscr();
		print_screen("./screen/login_screen.txt");
		char id[20];
		char pw[20];
		gotoxy(35, 8);
		gets_s(id);
		gotoxy(35, 10);
		gets_s(pw);
		int y = cursorControl(34, 18, 20, 2);
		switch (y) {
		case SINGIN:
			if (strcmp(pw, (findByMemberId(id)).pw) ==0) {
				if (strcmp(id, "admin") ==0) {
					ManagerMenu managerMenu;
					managerMenu.printSrc();
					break;
				}
				ShoppingMenu shoppingmenu;
				shoppingmenu.setUser(findByMemberId(id));
				shoppingmenu.printSrc();
				break;
			}
			break;
		case BACK:
			return;
		}
	}
}
```

코드 설명

사용자에게 입력받은 ID와 PW를 회원 DB에 저장된 값과 비교하여 유효한 경우 로그인할 수 있도록 한다.
유효하지 않은 경우, ID와 PW를 다시 입력하도록 한다.
지정된 관리자 아이디로 로그인하면 관리자 화면을 출력하도록 한다.
일반 회원을 로그인하면 쇼핑몰 주문 화면으로 이동하게 된다.

실행 결과


C:₩Users₩915₩Desktop₩deuMall₩Debug₩deuMall.exe
동의 컴소 쇼핑몰
[로그인]
아 이 디 :
비밀번호 :
[*] 로 그 인
[*] 취 소
동의 컴소 쇼핑몰에 오신 것을 환영합니다.
문의 전화 : 051 - 890 - 1724

SignUpMenu.cpp

```
void SingupMenu::printSrc()
{
	clrscr();
	print_screen("./screen/signup_screen.txt");
	char inputId[20];
	char inputPw[20];
	char inputName[20];
	while (1) {
		gotoxy(35, 7);	//ID 입력
		gets_s(inputId);
		MembersDto member = findByMemberId(inputId);
		if (strcmp(member.member_id, inputId) ==0) {
			print_screen("./screen/signup_screen.txt");
			gotoxy(35, 9);
			printf("이미 사용중인 아이디입니다.");
		}
		else {
			gotoxy(35, 9);
			printf("사용 가능한 아이디입니다.  ");
			break;
		}
	}

	gotoxy(35, 11);	//PW 입력
	gets_s(inputPw);
	gotoxy(35, 15);	//이름
	gets_s(inputName);
	MembersDto member;
	strcpy(member.member_id, inputId);
	strcpy(member.pw, inputPw);
	strcpy(member.name, inputName);
	int y = cursorControl(34, 20, 22, 2);
	switch (y) {
	case COMPLETE:
		addMember(member);
		LoginMenu loginmenu;
		loginmenu.printSrc();
		break;
	case CANCLE:
		MainMenu mainmenu;
```

```
                mainmenu.printSrc();
                break;
        }
}
```

코드 설명

사용자에게 입력받은 ID와 PW와 이름을 입력받도록 한다.
입력받은 ID를 회원 DB에서 이미 존재하는지 ID중복확인을 진행한다.
이미 존재하는 ID인 경우 "이미 사용중인 아이디입니다" 라는 안내문을 출력하도록 한다.
입력한 ID가 존재하지 않는 경우 " 사용가능한 아이디 입니다" 라는 안내문을 출력하도록 한다.
방향키로 이동 가능하며, 엔터키로 선택할 수 있도록 한다.
사용자가 입력을 완료하고 완료 버튼을 선택할 시, 입력한 정보를 회원 DB에 추가한다.
뒤로가기 선택시, 이전 화면인 메인화면을 출력하도록 한다.

실행 결과

pc파일 설명

DB에서 정보를 받아와 처리하는 로직과 비지니스 로직을 분리하기 위하여 DB에서 정보를 처리하는 함수들을 execute_query 파일에 작성하고 받아온 정보를 비지니스 로직에 바로 사용이 가능하게 가공하는 함수를 dao 파일에 작성하여 로직을 분리하였다.

execute_query.pc
DB에 쿼리를 요청하는 부분의 모든 함수를 작성해둔 파일이다.
execute_query.h

```
#pragma once
struct MembersDto {
        char member_id[20];
        char pw[20];
        char name[20];
};
struct OrdersDto {
        int order_id;
        int item_id;
        char member_id[20];
        int quantity;
        char status[20];
};
struct ItemsDto {
        int item_id;
        char name[20];
        int price;
        int stock;
        char description[250];
};
struct MembersDto select_MemberId(char id[]);
void execute_update(char query[]);
struct ItemsDto *select_ItemList();
struct MembersDto* select_MemberList();
struct OrdersDto* select_OrderList();
struct OrdersDto* select_Order(char id[]);
struct OrdersDto select_OrderId(int id);
struct ItemsDto select_ItemId(int id);
```

함수 설명
- execute_update

```
void execute_update(char query[]){ //insert, delete, update
    DB_connect();
    EXEC SQL BEGIN DECLARE SECTION;
```

```
        char dynstmt[1000];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL WHENEVER SQLERROR DO sql_error("₩₩7ORACLE ERROR:₩₩n");

    sprintf(dynstmt, query);

    EXEC SQL EXECUTE IMMEDIATE :dynstmt ;

        EXEC SQL COMMIT WORK RELEASE ;

}
```

select문을 제외한 쿼리를 입력하여 보내면 해당 쿼리를 실행시키는 함수이다.
select_MemberId

```
struct MembersDto select_MemberId(char id[]){ //select
    DB_connect();
    EXEC SQL BEGIN DECLARE SECTION;
    varchar member_id[100];
    varchar pw[100];
    varchar name[100];
    char dynstmt[1000];
    EXEC SQL END DECLARE SECTION;
    EXEC SQL WHENEVER SQLERROR DO sql_error("₩₩7ORACLE ERROR:₩₩n");
    sprintf(dynstmt, "SELECT member_id, pw, name FROM members where member_id = '%s'",
id);
    EXEC SQL PREPARE S FROM :dynstmt ;
    EXEC SQL DECLARE c_cursor CURSOR FOR S ;
    EXEC SQL OPEN c_cursor ;
    //EXEC SQL WHENEVER NOT FOUND DO found();
    EXEC SQL FETCH c_cursor INTO :member_id, :pw, :name;
    if(sqlca.sqlcode !=0){
        struct MembersDto result = {NULL, NULL, NULL};
        return result;
    }
    member_id.arr[member_id.len] ='₩₩0';
    pw.arr[pw.len] ='₩₩0';
    name.arr[name.len] ='₩₩0';
    struct MembersDto result;
    strcpy(result.member_id, member_id.arr);
    strcpy(result.pw, pw.arr);
    strcpy(result.name, name.arr);
    return result;
}
```

사용자 아이디 값을 매개변수로 전달받아 해당 사용자의 정보를 DB에서 받아오는 함수이다.
select_ItemList

```
struct ItemsDto *select_ItemList(){
    DB_connect();
    EXEC SQL BEGIN DECLARE SECTION;
    varchar item_id[100];
    varchar name[100];
    varchar price[100];
    varchar stock[100];
    varchar description[250];
    char dynstmt[1000];
    EXEC SQL END DECLARE SECTION;

    struct ItemsDto* resultList = (struct ItemsDto*) malloc(sizeof(struct ItemsDto) *10); //구조체 배열 선언
    sprintf(dynstmt, "SELECT * FROM items");
    EXEC SQL PREPARE D FROM :dynstmt ;
    EXEC SQL DECLARE d_cursor CURSOR FOR D ;
    EXEC SQL OPEN d_cursor;
    for(int i=0;;i++){
        EXEC SQL WHENEVER NOT FOUND DO break;
        EXEC SQL FETCH d_cursor INTO :item_id, :name, :price, :stock, :description;
            item_id.arr[item_id.len] ='₩₩0';
            name.arr[name.len] ='₩₩0';
            price.arr[price.len] ='₩₩0';
            stock.arr[stock.len] ='₩₩0';
            description.arr[description.len] ='₩₩0';
            resultList[i].item_id =  atoi(item_id.arr);
            strcpy(resultList[i].name, name.arr);
            resultList[i].price = atoi(price.arr);
            resultList[i].stock = atoi(stock.arr);
            strcpy(resultList[i].description, description.arr);
    }
    int fetch_num = sqlca.sqlerrd[2]; //전송된 튜플의 수
    return resultList;
}
```

DB에 있는 모든 상품들의 정보를 받아오는 함수이다.
select_MemberList

```
struct MembersDto *select_MemberList(){
    DB_connect();
```

```
    EXEC SQL BEGIN DECLARE SECTION;
    varchar member_id[20];
    varchar pw[20];
    varchar name[20];
    char dynstmt[1000];
    EXEC SQL END DECLARE SECTION;
    struct MembersDto* resultList = (struct MembersDto*) malloc(sizeof(struct MembersDto) *10);
//구조체 배열 선언
    sprintf(dynstmt, "SELECT * FROM members");

    EXEC SQL PREPARE S FROM :dynstmt ;
    EXEC SQL DECLARE s_cursor CURSOR FOR S ;
    EXEC SQL OPEN s_cursor;
    for(int i=0;;i++){
        EXEC SQL WHENEVER NOT FOUND DO break;
        EXEC SQL FETCH s_cursor INTO :member_id, :pw, :name;
            member_id.arr[member_id.len] ='₩₩0';
            pw.arr[pw.len] ='₩₩0';
            name.arr[name.len] ='₩₩0';
            strcpy(resultList[i].member_id, member_id.arr);
            strcpy(resultList[i].pw, pw.arr);
            strcpy(resultList[i].name, name.arr);
    }
    int fetch_num = sqlca.sqlerrd[2]; //전송된 튜플의 수
    return resultList;
}
```

DB에 있는 모든 사용자의 정보를 받아오는 함수이다.
select_OrderList

```
struct OrdersDto *select_OrderList(){
    DB_connect();
    EXEC SQL BEGIN DECLARE SECTION;
    varchar order_id[20];
    varchar item_id[20];
    varchar member_id[20];
    varchar quantity[20];
    varchar status[20];
    char dynstmt[1000];
    EXEC SQL END DECLARE SECTION;
    struct OrdersDto* resultList = (struct OrdersDto*) malloc(sizeof(struct OrdersDto) *10); //구조
체 배열 선언
    sprintf(dynstmt, "SELECT * FROM orders");
```

```
    EXEC SQL PREPARE O FROM :dynstmt ;
    EXEC SQL DECLARE o_cursor CURSOR FOR O ;
    EXEC SQL OPEN o_cursor;
    for(int i=0;;i++){
        EXEC SQL WHENEVER NOT FOUND DO break;
        EXEC SQL FETCH o_cursor INTO :order_id, :item_id, :member_id, :quantity, :status;
            order_id.arr[order_id.len] ='₩₩0';
            item_id.arr[item_id.len] ='₩₩0';
            member_id.arr[member_id.len] ='₩₩0';
            quantity.arr[quantity.len] ='₩₩0';
            status.arr[status.len] ='₩₩0';
            resultList[i].order_id =  atoi(order_id.arr);
            resultList[i].item_id =  atoi(item_id.arr);
            strcpy(resultList[i].member_id, member_id.arr);
            resultList[i].quantity = atoi(quantity.arr);
            strcpy(resultList[i].status, status.arr);
    }
    int fetch_num = sqlca.sqlerrd[2]; //전송된 튜플의 수
    return resultList;
}
```

DB에 있는 모든 주문정보를 받아오는 함수이다.
select_Order

```
struct OrdersDto *select_Order(char id[]){
    DB_connect();
    EXEC SQL BEGIN DECLARE SECTION;
    varchar order_id[20];
    varchar item_id[20];
    varchar member_id[20];
    varchar quantity[20];
    varchar status[20];
    char dynstmt[1000];
    EXEC SQL END DECLARE SECTION;
    struct OrdersDto* resultList = (struct OrdersDto*) malloc(sizeof(struct OrdersDto) *10); //구조
체 배열 선언
    sprintf(dynstmt, "SELECT * FROM orders WHERE member_id = '%s'",id);
    EXEC SQL PREPARE OM FROM :dynstmt ;
    EXEC SQL DECLARE om_cursor CURSOR FOR OM ;
    EXEC SQL OPEN om_cursor;
    for(int i=0;;i++){
        EXEC SQL WHENEVER NOT FOUND DO break;
```

```
        EXEC SQL FETCH om_cursor INTO :order_id, :item_id, :member_id, :quantity, :status;
            order_id.arr[order_id.len] ='₩₩0';
            item_id.arr[item_id.len] ='₩₩0';
            member_id.arr[member_id.len] ='₩₩0';
            quantity.arr[quantity.len] ='₩₩0';
            status.arr[status.len] ='₩₩0';
            resultList[i].order_id =  atoi(order_id.arr);
            resultList[i].item_id =  atoi(item_id.arr);
            strcpy(resultList[i].member_id, member_id.arr);
            resultList[i].quantity = atoi(quantity.arr);
            strcpy(resultList[i].status, status.arr);
    }
    int fetch_num = sqlca.sqlerrd[2]; //전송된 튜플의 수
    return resultList;
}
```

사용자의 아이디를 매개변수로 전달받아 해당 사용자가 주문한 모든 내역을 받아오는 함수이다.
select_OrderId

```
struct OrdersDto select_OrderId(int id){ //select
    DB_connect();
    EXEC SQL BEGIN DECLARE SECTION;
    varchar order_id[20];
    varchar item_id[20];
    varchar member_id[20];
    varchar quantity[20];
    varchar status[20];
    char dynstmt[1000];
    EXEC SQL END DECLARE SECTION;
    EXEC SQL WHENEVER SQLERROR DO sql_error("₩₩7ORACLE ERROR:₩₩n");
    sprintf(dynstmt, "SELECT * FROM orders where order_id = %d", id);
    EXEC SQL PREPARE OI FROM :dynstmt ;
    EXEC SQL DECLARE oi_cursor CURSOR FOR OI ;
    EXEC SQL OPEN oi_cursor ;
    EXEC SQL WHENEVER NOT FOUND CONTINUE;

    EXEC SQL FETCH oi_cursor INTO :order_id, :item_id, :member_id, :quantity, :status;

    order_id.arr[order_id.len] ='₩₩0';
    item_id.arr[item_id.len] ='₩₩0';
    member_id.arr[member_id.len] ='₩₩0';
    quantity.arr[quantity.len] ='₩₩0';
    status.arr[status.len] ='₩₩0';
```

```
    struct OrdersDto result;
    result.order_id =  atoi(order_id.arr);
    result.item_id =  atoi(item_id.arr);
    strcpy(result.member_id, member_id.arr);
    result.quantity = atoi(quantity.arr);
    strcpy(result.status, status.arr);
    return result;
}
```

주문번호를 매개변수로 전달받아 해당 주문번호에 해당하는 주문정보를 받아오는 함수이다.
select_ItemId

```
struct ItemsDto select_ItemId(int id){ //select
    DB_connect();
    EXEC SQL BEGIN DECLARE SECTION;
    varchar item_id[100];
    varchar name[100];
    varchar price[100];
    varchar stock[100];
    varchar description[250];
    char dynstmt[1000];
    EXEC SQL END DECLARE SECTION;
    EXEC SQL WHENEVER SQLERROR DO sql_error("₩₩7ORACLE ERROR:₩₩n");
    sprintf(dynstmt, "SELECT * FROM items where item_id = %d", id);
    EXEC SQL PREPARE I FROM :dynstmt ;
    EXEC SQL DECLARE i_cursor CURSOR FOR I ;
    EXEC SQL OPEN i_cursor ;
    EXEC SQL WHENEVER NOT FOUND CONTINUE;


    EXEC SQL FETCH i_cursor INTO :item_id, :name, :price, :stock, :description;
    item_id.arr[item_id.len] ='₩₩0';
    name.arr[name.len] ='₩₩0';
    price.arr[price.len] ='₩₩0';
    stock.arr[stock.len] ='₩₩0';
    description.arr[description.len] ='₩₩0';

    struct ItemsDto result;
    result.item_id =  atoi(item_id.arr);
    strcpy(result.name, name.arr);
    result.price = atoi(price.arr);
    result.stock = atoi(stock.arr);
    strcpy(result.description, description.arr);
```

```
    return result;
}
```

상품번호를 매개변수로 전달받아 해당 상품번호에 해당하는 상품정보를 받아오는 함수이다.

Dao

execute_query의 함수들을 이용하여 DB를 조작하고 받아온 데이터를 가공하는 함수들의 집합인 파일이다.

dao.h

```
#pragma once
extern "C" {
#include "execute_query.h"
}
#include<vector>
using namespace std;
void addMember(MembersDto member);
void addItem(ItemsDto item);
void adddorder(OrdersDto order);
void deleteItem(int id);
void orderStatus(int id, const char status[]);
void updateItem(int id, ItemsDto item);
struct MembersDto findByMemberId(char id[]);
vector<ItemsDto> selectItems();
vector<MembersDto> selectMembers();
vector<OrdersDto> selectOrders();
vector<OrdersDto> selectOrder(char id[]);
struct OrdersDto findByOrderId(int id);
struct ItemsDto findByItemId(int id);
```

함수 설명

addMember

```
void addMember(MembersDto member) {
        sprintf(query, "insert into members values ('%s', '%s', '%s')", member.member_id, member.pw, member.name);
        execute_update(query);
}
```

사용자 정보를 담고있는 구조체를 매개변수로 전달받아 쿼리를 만들고 execute_query의 execute_update 함수를 실행시켜 DB에 해당 사용자의 정보를 추가하는 함수이다.

addItem

```
void addItem(ItemsDto item) {
        sprintf(query, "INSERT INTO Items (item_id, name, price, stock, description)
```

```
VALUES(item_id_seq.NEXTVAL, '%s', %d, %d, '%s')", item.name, item.price, item.stock, item.description);
        execute_update(query);
}
```

상품 정보를 담고있는 구조체를 매개변수로 전달받아 쿼리를 만들고 execute_query의 execute_update 함수를 실행시켜 DB에 해당 상품의 정보를 추가하는 함수이다.

adddorder

```
void adddorder(OrdersDto order) {
        sprintf(query, "insert into orders (order_id, item_id, member_id, quantity, status) values (item_id_seq.NEXTVAL, %d, '%s', %d, '%s')", order.item_id, order.member_id, order.quantity, order.status);
        execute_update(query);
}
```

주문 정보를 담고있는 구조체를 매개변수로 전달받아 쿼리를 만들고 execute_query의 execute_update 함수를 실행시켜 DB에 해당 주문의 정보를 추가하는 함수이다.

deleteItem

```
void deleteItem(int id) {
        sprintf(query, "delete from items where item_id = %d", id);
        execute_update(query);
}
```

상품 번호를 매개변수로 전달받아 쿼리를 만들고 execute_query의 execute_update 함수를 실행시켜 DB에 해당 상품의 정보를 삭제하는 함수이다.

orderStatus

```
void orderStatus(int id, const char status[]) {
        sprintf(query, "update orders set status = '%s' where order_id = %d", status, id);
        execute_update(query);
}
```

주문 번호와 주문 상태를 매개변수로 전달받아 쿼리를 만들고 execute_query의 execute_update 함수를 실행시켜 DB에 해당 주문의 상태를 수정하는 함수이다.

updateItem

```
void updateItem(int id, ItemsDto item) {
        sprintf(query, "update items set name = '%s', price = %d, stock = %d, description = '%s' where item_id = %d", item.name, item.price, item.stock, item.description, id);
        execute_update(query);
}
```

상품 번호와 상품의 정보를 담고있는 구조체를 매개변수로 전달받아 쿼리를 만들고 execute_query의 execute_update 함수를 실행시켜 DB에 해당 상품의 정보를 수정하는 함수이다.

findByMemberI

```
struct MembersDto findByMemberId(char id[]) {
```

```
        return select_MemberId(id);
}
```

사용자의 아이디를 매개변수로 전달받아 execute_query의 select_MemberId 함수를 실행시켜 받아온 상품의 정보를 반환하는 함수이다.

findByOrderId

```
struct OrdersDto findByOrderId(int id) {
        return select_OrderId(id);
}
```

주문번호를 매개변수로 전달받아 execute_query의 select_OrderId함수를 실행시켜 받아온 주문 정보를 반환하는 함수이다.

findByItemId

```
struct ItemsDto findByItemId(int id) {
        return select_ItemId(id);
}
```

상품 번호를 매개변수로 전달받아 execute_query의 select_ItemId함수를 실행시켜 받아온 상품 정보를 반환하는 함수이다.

selectItems

```
vector<ItemsDto> selectItems() {
        ItemsDto* items = select_ItemList();
        vector<ItemsDto> result;
        int count =0;
        for (int i =0; i <10; i++) {
                if (items[i].item_id >0 && items[i].item_id <=10) {
                        result.push_back(items[i]);
                }
        }
        free(items);
        return result;
}
```

execute_query의 select_ItemList 함수를 실행시켜 받아온 모든 상품의 정보를 유효한 값만 골라 반환하는 함수이다.

selectMembers

```
vector<MembersDto> selectMembers() {
        MembersDto* members = select_MemberList();
        vector<MembersDto> result;
        int count =0;
        for (int i =0; i <10; i++) {
                if (members[i].member_id[0] >=0 && members[i].member_id[0] <=127) {
                        result.push_back(members[i]);
```

```
                }
        }
        free(members);
        return result;
}
```

execute_query의 select_MemberList함수를 실행시켜 받아온 모든 사용자의 정보를 유효한 값만 골라 반환하는 함수이다.

selectOrders

```
vector<OrdersDto> selectOrders() {
        OrdersDto* orders = select_OrderList();
        vector<OrdersDto> result;
        int count =0;
        for (int i =0; i <10; i++) {
                if (orders[i].order_id >=0 && orders[i].order_id <=10) {
                        result.push_back(orders[i]);
                }
        }
        free(orders);
        return result;
}
```

execute_query의 select_OrderList함수를 실행시켜 받아온 모든 주문의 정보를 유효한 값만 골라 반환하는 함수이다.

selectOrder

```
vector<OrdersDto> selectOrder(char id[]) {
        OrdersDto* orders = select_Order(id);
        vector<OrdersDto> result;
        int count =0;
        for (int i =0; i <10; i++) {
                if (orders[i].order_id >=0 && orders[i].order_id <=10) {
                        result.push_back(orders[i]);
                }
        }
        free(orders);
        return result;
}
```

사용자 아이디를 매개변수로 전달받아 해당 사용자가 주문한 것에 해당하는 모든 주문 정보를 유효한 값만 골라 반환하는 함수이다.

5. 프로젝트 결과

5.1 프로젝트 완성도

기능	개발 완성도
고객은 자신의 아이디와 패스워드를 통해 로그인을 진행한다.	100%
고객은 상품에 대해 수량을 선택한 후 상품을 구매할 수 있다.	100%
고객은 주문한 상품의 리스트를 조회할 수 있고 배송상태를 알 수 있다.	100%
관리자는 상품의 상품명, 가격, 상품설명, 개수를 입력받고, 테이블에 반영할 수 있다.	100%
관리자는 모든 상품에 대해 조회가 가능하며 이를 수정 및 삭제 가능하다.	100%
관리자는 사용자의 정보를 조회할 수 있다.	100%
관리자는 총 매출액을 조회할 수있다.	100%
관리자는 주문 요청이 들어온 주문 리스트에 대해 상태를 처리할 수 있다.	100%

5.2 일정 계획

- 매주 회의를 통해 전 주의 계획에 피드백을 하였고, 금주의 할 일 또한 정하여 프로젝트를 진행함에있어 적절한 프로젝트 계획 범위를 설정하여 모든 작업을 완료하였음

5.3 역할 수행

	이름	역할
팀장	김남주	일정 조율, 프로젝트 관리, DB 스키마 설계, 시스템 구현
팀원	김민곤	화면 설계, DB 스키마 설계, 쿼리문 작성, 시스템 구현
	이종민	화면 설계, DB 스키마 설계, 쿼리문 작성, 시스템 구현
	오승헌	화면 설계, DB 스키마 설계, 문서작성, 쿼리문 작성, 시스템 구현

5.4 위험 처리

5.4.1. 프로젝트 구조 분리

- 동적 SQL 처리부 모듈화 및 기능별 분리

배경

PC 파일을 컴파일하는 과정에서 동적 SQL 작성 부분을 처리하는 것은 번거로운 작업이었습니다. 이로 인해 코드의 가독성과 유지보수성이 저하되는 문제가 발생하였습니다. 따라서, 이러한 문제를 해결하고자 동적 SQL 부분을 모듈화하여 분리하고, CPP 파일로만 기능을 구현하는 것이 필요하게 되었습니다.

위험처리

동적 SQL 부분을 모듈화하여 CPP 파일로 분리함으로써 다음과 같은 위험을 처리할 수 있었습니다:

코드 가독성 향상: 동적 SQL 작성 부분을 별도의 모듈로 분리하여 CPP 파일에 구현함으로써, 코드의 가독성이 향상되었습니다.

동적 SQL과 관련된 로직이 분리되어 있어 코드의 구조가 명확해지고, 개발자가 코드를 이해하고 유지보수할 때 효율성이 증가했습니다.

유지보수성 개선: 동적 SQL 모듈을 독립적으로 관리할 수 있으므로, 데이터베이스와의 상호작용을 담당하는 부분을 쉽게 수정하거나 업데이트할 수 있습니다.

이는 프로젝트 특성상 PC파일의 가독성이 떨어지므로, 해당 파일을 유지보수 과정에서 발생할 수 있는 위험을 줄여줍니다.

위험분석

동적 SQL 부분을 모듈화하여 CPP 파일로 분리함으로써 다음과 같은 위험 분석을 수행하였습니다.

구조 분리: 동적 SQL 모듈을 별도의 CPP 파일로 분리함으로써, 코드의 구조와 의도를 명확하게 분리할 수 있습니다. 이는 코드의 가독성을 향상시키고, 버그 발생 가능성을 줄여줍니다.오류 가능성 감소: 동적 SQL 작성에 따른 오류 가능성을 줄일 수 있습니다. 동적 SQL 부분을 별도의 모듈로 분리하여 관리함으로써, SQL 문장의 작성과 관련된 오류를 최소화할 수 있습니다.

장점

동적 SQL 부분을 모듈화하여 CPP 파일로만 기능을 구현함에 있어서 클래스를 사용한 점은 다음과 같은 장점을 갖도록 하였습니다.

코드 재사용성: 클래스를 사용하여 동적 SQL 모듈을 구현함으로써, 해당 기능을 다른 프로젝트에서도 쉽게 재사용할 수 있습니다. 클래스의 인스턴스를 생성하여 필요한 곳에서 호출하면 됩니다.

객체 지향적 설계: 클래스를 사용하여 동적 SQL 모듈을 구현하면, 객체 지향적인 설계 원칙을 따를 수 있습니다. 클래스의 멤버 변수와 멤버 함수를 적절하게 활용하여 코드의 구조와 의도를 명확하게 표현할 수 있습니다.

결론

동적 SQL 부분을 모듈화하여 CPP 파일로만 기능을 구현하는 결정은 프로젝트의 안정성과 효율성을 향상시키는 역할을 합니다. 위험 처리와 위험 분석을 고려한 이러한 결정은 코드의 가독성과 유지보수성을 향상시키고, 동적 SQL 작성에 따른 오류 가능성을 줄여줍니다. 또한, 클래스를 사용하여 구현함으로써 코드의 재사용성과 객체 지향적인 설계를 강조할 수 있습니다.

6. 소감

이름	소감
김남주	C언어를 오랜만에 다루면서 처음에는 어색하고 힘들었지만, 그 과정에서의 재미와 즐거움을 느낄 수 있었습니다. 특히 기존에는 C언어에서 뭘 할 수 있겠어 생각했었지만, C++로 들어가면서 객체지향 개념과 STL을 활용함으로써 더 다양하고 여러가지 기능들을 시도해볼 수 있었습니다. 또한, 데이터베이스를 C언어로 접속하는 것이 처음이었는데, Pro C를

	활용하여 DB를 사용하는 것은 좋은 경험이였습니다.
김민곤	처음에는 SQL과 C, C++ 언어를 결합하여 사용하는 것이 다소 어렵게 느껴졌다. 각 언어의 문법과 특성을 이해하고, 이를 통해 데이터베이스와 상호작용하는 방법을 배우는 과정은 복잡하고 도전적이었다. 그러나 이 과정에서 프로그래밍 언어와 데이터베이스 간의 인터페이스를 이해하는데 큰 도움이 되었다. 프로젝트를 진행하면서 가장 인상 깊었던 것은 데이터 모델링과 데이터베이스 설계 과정이었다. 쇼핑몰의 다양한 기능을 구현하기 위해 필요한 테이블과 관계를 신중하게 생각하고, 이를 효율적인 쿼리로 변환하는 과정은 매우 흥미로웠다. 이를 통해 실제 산업 환경에서 데이터베이스가 어떻게 활용되는지에 대한 깊은 이해를 얻을 수 있었다. 또한, 이 프로젝트를 통해 팀원들과 협업하는 능력도 향상되었다. 모든 팀원이 프로젝트의 다양한 부분에 참여하고, 문제를 해결하기 위해 함께 노력하는 과정에서 팀워크의 중요성을 체감할 수 있었다. 결국, 이 강의와 프로젝트는 저에게 데이터베이스응용의 실질적인 이해와 실제 산업 환경에서의 응용 능력을 키우는 데 매우 중요한 역할을 하였습니다. 이 경험은 저의 프로그래밍 역량을 향상시키고, 데이터 중심의 사고 방식을 갖추는 데 크게 기여하였습니다. 이를 바탕으로 앞으로도 더 복잡한 프로젝트에 도전하고, 새로운 기술을 배우는 데 자신감을 갖게 되었습니다.
이종민	동적 SQL을 사용한 프로젝트를 진행하면서 저는 많은 소감을 느꼈습니다. 첫째로, 동적 SQL을 사용하면 쿼리의 유연성이 크게 향상됩니다. 프로젝트 요구사항이 변경되거나 다양한 조건에 따라 쿼리를 동적으로 생성해야 할 때, 동적 SQL을 활용하면 훨씬 더 유연하게 쿼리를 구성할 수 있었습니다. 이는 프로젝트의 요구사항에 더욱 정확하게 대응할 수 있었고, 유지보수성도 높일 수 있었습니다. 둘째로, 동적 SQL을 사용하면 쿼리의 성능을 최적화할 수 있었습니다. 정적 SQL은 미리 작성된 쿼리를 실행하기 때문에 실행 계획을 최적화하기 어려웠습니다. 하지만 동적 SQL을 사용하면 실행 시점에 쿼리를 생성하므로, 실행 계획을 동적으로 최적화할 수 있었습니다. 이를 통해 프로젝트의 성능을 향상시키고 데이터베이스 리소스를 효율적으로 활용할 수 있었습니다. 셋째로, 동적 SQL을 사용하면 코드의 재사용성이 높아집니다. 정적 SQL은 미리 작성된 쿼리를 재사용하기 어려웠습니다. 하지만 동적 SQL을 사용하면 쿼리를 동적으로 생성하여 다양한 곳에서 재사용할 수 있었습니다. 이는 코드의 중복을 줄이고 개발 생산성을 높여주었습니다. 동적 SQL을 사용한 프로젝트를 진행하면서 이러한 장점들을 몸소 느꼈습니다. 하지만 동적 SQL을 사용할 때 주의해야 할 점도 있습니다. 쿼리의 보안 취약점을 방지하기 위해 쿼리 인젝션에 대비하고, 적절한 보안

	대책을 마련해야 한다는 것을 배웠습니다. 총평하자면, 동적 SQL을 사용한 프로젝트는 쿼리의 유연성과 성능 최적화, 코드의 재사용성을 향상시킬 수 있는 매우 유용한 방법이라고 생각합니다. 이를 통해 프로젝트의 효율성과 개발 생산성을 크게 향상시킬 수 있었습니다. 앞으로도 동적 SQL을 적극적으로 활용하여 프로젝트를 진행하고자 합니다
오승헌	프로젝트를 진행하며 데이터베이스의 중요성을 깨달았습니다. 쇼핑몰 프로젝트에서는 상품 정보, 주문 정보, 회원 정보 등 다양한 데이터를 관리해야 했습니다. 데이터베이스를 효율적으로 설계하고 적절한 쿼리를 작성하여 데이터를 검색하고 조작하는 것이 중요했습니다. 데이터베이스를 효과적으로 활용하면 데이터의 일관성과 정확성을 유지할 수 있고, 원하는 결과를 빠르게 얻을 수 있었습니다. PROC와 C 언어의 결합으로 프로젝트를 효율적으로 개발하였습니다. PROC는 C 코드 안에 SQL 문을 포함시켜 데이터베이스와의 상호작용을 용이하게 해주는 도구이며 이를 통해 데이터베이스에 접근하고 쿼리를 실행하는 과정을 간편하게 처리할 수 있었습니다. C 언어의 강력한 기능과 PRO*C의 편리함을 결합하여 프로젝트를 더욱 효율적으로 개발하였습니다. 저희 조의 주제로 쇼핑몰 프로젝트는 여러 모듈로 이루어져 있었기 때문에 팀원들과의 원활한 커뮤니케이션과 협업이 필요했습니다. 서로의 역할과 책임을 분담하고, 코드를 통합하고, 버그를 해결하는 과정에서 팀원들과의 협력을 통해 프로젝트를 성공적으로 마무리할 수 있었습니다. 이를 통해 팀워크와 커뮤니케이션 능력을 향상시킬 수 있었습니다. 결론적으로 데이터베이스 강의에서 PRO*C와 C 언어를 결합하여 진행한 쇼핑몰 프로젝트는 데이터베이스의 중요성을 깨닫게 해주었고, 프로젝트 개발에 효율성을 높여주었으며, 팀 협업 능력을 향상시켰습니다.