C++프로그래밍및실습

편의점 매대 관리 시스템

최종보고서

제출일자: 2024.12.22.
제출자명: 김태균
제출자학번: 213961

1. 프로젝트 목표 (16 pt)

1) 배경 및 필요성 (14 pt)

편의점 아르바이트를 하면서 매대 현황을 확인하여 진열된 상품이 판매되었을 경우 상품을 다시 채워야 한다. 이때 판매되는 물품을 일일이 다 기억하기 어려우며 특히 유통기한이 짧은 냉장 식품이 진열된 매대의 경우 진열되는 상품과 창고에 있는 상품이 매번 달라지기에 창고 상황과 매대 상황을 확인하여 진열해야 하는 번거로움이 있다. 매대의 상황을 실시간으로 알려주고 냉장 식품 매대 진열을 어떻게 해야 할지 제시해주는 프로그램이 필요하다

2) 프로젝트 목표

현재 매대의 진열된 상품의 수량을 보여주고 창고의 재고, 피크타임 여부등을 고려해 매대에 진열해야 하는 상품을 알려준다. 냉장 식품 매대의 경우 진열할 상품을 소비기한, 상품의 크기, 매대의 크기등을 고려하여 최적의 조합을 제시한다.

3) 차별점

기존의 프로그램은 단순히 매대에 진열된 상품의 수와 창고 내 재고 수를 비교하여 진열할 상품을 제시하는 기능만 존재한다. 이 프로그램의 경우 냉장 식품 매대의 크기에 맞게 들어갈 수 있는 최적의 조합을 계산해 실시간으로 제시해주는 기능을 제공해 사용자의 편의성을 높여준다

2. 기능 계획

1) 재고 관리 기능

- 현재 재고 상황을 관리하기 위해 재고 추가/제거, 수량 변경 등의 기능

2) 진열 필요 상품 제시 기능

- 현재 재고와 매대 상태를 확인하여 진열할 수 있는 상품이 있을 경우 해당 상품을 제시하는 기능

(1) 매대 상품 수량 변경 기능

- 상품이 판매되거나 상품을 진열했을 경우 상품 수량이 변경되도록 하는 기능

(2) 진열 필요 상품 제시 기능

- 재고 수량과 비교하여 현재 진열할 수 있는 상품들을 제시, 피크 타임을 고려한 우선도를 표시해 당장 진열이 필요한 상품, 매대에 수량이 어느정도 남은 상품으로 구분하여 제시한다

(3) 피크 타임 분류 기능

- 각 매장의 피크 타임을 분석하기 위해 현재 시간이 피크 타임인지 아닌지를 분류하는 기계학습 모델을 만들어 진열 상품의 우선도를 평가하는데 활용한다

3) 냉장 식품 매대 실시간 업데이트 기능

- 냉장 식품 매대에 진열 해야 하는 상품을 매대의 크기에 맞게 최적의 조합을 찾아 제시해준다

3. 프로그램 실행방법

- storage.csv, storage_ref.csv, customer_per_hour.csv, is_peak.csv 파일과 main.exe파일을 같은 폴더에 두고 exe폴더를 cmd로 실행

4. 기능 구현

A. 재고 관리 기능 – 데이터베이스 읽기

i. 코드 스크린샷

storage.h

```
// 상품 구조체
struct item
{
    string code; // 상품 코드
    string name; // 상품명
    string stand; // 상품이 진열될 매대
    int cur_stand_qnt; // 매대에 진열된 개수
    int max_stand_qnt; // 최대 진열 가능 개수
    int inventory; // 재고
    double item_length = 0; // 진열할 상품의 길이, 냉장 식품 매대 전용
    bool was_found; // 기본적으로 true, 상품이 검색되지 않았을 때 이 값이 false인 구조체 반환
} typedef Item;
```

```
// 데이터 저장공간 클래스
class Storage
{
protected:
    fstream fs;
    string storage_name;
    vector<Item> database; // 벡터를 활용한 동적 배열
public:
    Storage(){}
    Storage(const Storage& other); // database 멤버 변수를 깊은 복사하기 위한 복사생성자
    void Open(string filename); // 파일 여는 함수

    virtual Item GetItem(string code); // code에 해당하는 상품을 vector로 반환
    Item GetItem(int line) const; // line번째 상품을 벡터로 반환
    int GetSize() const; // database의 크기 반환, 목록 부분을 제외한 사이즈이다

    void AddLine(Item new_line); // database 행 추가
    virtual void RemoveItem(string code); // code에 해당하는 제품 삭제
    virtual void ItemSold(string code, int qnt); // code에 해당하는 제품 판매
    virtual void ItemToStand(string code, int qnt); // code에 해당하는 제품 qnt만큼 매대에 진열
    virtual void ItemStore(string code, int qnt); // code에 해당하는 제품 qnt만큼 재고수 추가
    virtual void PrintDatabase(); // database 출력

    void Close(); // 변경된 내용을 저장
};
```

storage.cpp

```cpp
void Storage::Open(string filename)
{
    string str_buf;
    this->storage_name = filename;
    this->fs.open(filename, ios::in);

    while(!fs.eof())
    {
        vector<string> tmp; // 파일을 읽기 위한 임시 벡터, 한 줄을 저장
        getline(this->fs, str_buf,'\n');

        // string 형태로 읽어진 csv파일 한 줄을 읽어 tmp에 저장한다
        int position;
        int cur_position = 0;
        while((position = str_buf.find(",", cur_position)) != string::npos)
        {
            int len = position - cur_position;
            tmp.push_back(str_buf.substr(cur_position, len));
            cur_position = position + 1;
        }
        tmp.push_back(str_buf.substr(cur_position));

        // tmp의 string 값을 자료형에 맞게 Item 구조체에 저장
        Item data_extracted;

        data_extracted.code = tmp[0];
        data_extracted.name = tmp[1];
        data_extracted.stand = tmp[2];
        data_extracted.cur_stand_qnt = stoi(tmp[3]);
        data_extracted.max_stand_qnt = stoi(tmp[4]);
        data_extracted.inventory = stoi(tmp[5]);
        data_extracted.was_found = true;

        // 추출한 Item을 database에 저장한다
        AddLine(data_extracted);
    }
    fs.close();
}
```

```cpp
Item Storage::GetItem(string code)
{
    // code로 해당 item 검색
    for(Item i : this->database)
    {
        if(i.code == code)
        {
            return i; // database에 해당 item이 있을 경우 반환
        }
    }
    // 검색되지 않을 경우에 반환하는 was_found 값이 false인 NULL 구조체
    Item null_item;
    null_item.was_found = false;
    return null_item;
}

Item Storage::GetItem(int line) const
{
    return this->database[line]; // line번째 Item 반환
}

int Storage::GetSize() const
{
    return this->database.size(); // database의 목록부분을 제외한 사이즈
}
```

```
Storage::Storage(const Storage &other)
{
    this->fs.open(this->storage_name);
    this->storage_name = other.storage_name;
    // 복사할 Storage 객체의 database에 기존의 database 복사
    for(Item i : other.database)
    {
        AddLine(i);
    }
}
```

main.cpp

```
Storage storage;
storage.Open("storage.csv");
```

ii. 입력

- string filename: csv파일의 파일명

iii. 반환값

- 없음

iv. 결과

- csv파일을 읽고 Item 구조체 동적 배열에 저장

v. 설명

- storage 객체 정의

- Item 구조체 정의

- storage 객체의 정보를 반환하는 get메서드 정의

- GetItem을 code로 검색했을 때 검색에 실패했을 경우 Item 구조체의 was_found값이 false인 Item 객체를 반환한다. 이를 통해 검색에 실패했음을 알 수 있다

- GetItem의 경우 상품의 code 또는 목록에서의 인덱스 값으로 검색할 수 있도록 오버로드 함

- Open 함수를 정의, Open 함수를 통해 파일명에 해당하는 csv파일을 읽고 멤버변수인 database에 값들을 저장

- 복사 생성자를 정의해 멤버변수 database를 복사할 때 깊은 복사를 할 수 있도록 설계함

- 헤더파일에서 선언된 virtual 함수들은 냉장 매대 클래스를 만들면서 재사용하기 위해 동적 바인딩 될 수 있도록 virtual로 구현

vi. 수업 연관 내용

- 7주차 function overload

- 10주차 vector

- 11주차 copy constructor

- 13주차 Polymorphism and Virtual

- 14주차 FileIO

B. 재고 관리 기능 – 재고 추가

i. 코드 스크린샷

storage.cpp

```
void Storage::ItemStore(string code, int qnt)
{
    // code로 해당 item 검색
    for(int i = 0; i < this->database.size(); i++)
    {
        if(this->database[i].code == code)
        {
            this->database[i].inventory += qnt; // 재고 수 증가
        }
    }
}
```

main.cpp

```
case 4: // Add item in storage
{
    // 상품 코드, 수량 입력
    string item_code;
    string qnt;

    cout << "Item Code: ";
    cin >> item_code;
    cout << "Quantity: ";
    cin >> qnt;

    // 재고 추가 처리
    for(auto *s : storages)
    {
        s->ItemStore(item_code, stoi(qnt));
    }

    // 품목 출력
    for(auto *s : storages)
    {
        s->PrintDatabase();
        cout << "--------------------------------------------------------------------------------" <<
        endl;
    }
    break;
}
```

ii. 입력

- string code: 재고 추가할 상품의 code

- int qnt: 추가할 재고량

iii. 반환값

- 없음

iv. 결과

- 상품의 재고량을 입력된 개수만큼 추가

v. 설명

- 입력된 code의 상품을 검색한 후 입력된 qnt만큼 재고를 추가해 저장

- 메뉴 선택 switch문 case4이다

vi. 수업 연관 내용

- 10주차 vector

c. 재고 관리 기능 – 상품 추가

i. 코드 스크린샷

storage.cpp

```
void Storage::AddLine(Item new_line)
{
    this->database.push_back(new_line); // database에 item 추가
}
```

main.cpp

```
case 2: // New item
{
    // 상품 정보 입력
    string item_code;
    string item_name;
    string stand;
    string cur_stand_qnt;
    string max_stand_qnt;
    string inventory;
    string item_length;

    cout << "Item Code: ";
    cin >> item_code;
    cout << "Item Name: ";
    cin >> item_name;
    cout << "Stand that Item belongs: ";
    cin >> stand;
    cout << "Current Qunatity in Stand: ";
    cin >> cur_stand_qnt;
    cout << "Max Qunatity of Stand: ";
    cin >> max_stand_qnt;
    cout << "Qunatity in Storage: ";
    cin >> inventory;
    cout << "Length of Item(Put 0 if it isn't Refrigerated Food): ";
    cin >> item_length;

    // 새로운 상품 생성
    Item new_item;
    new_item.code = item_code;
    new_item.name = item_name;
    new_item.stand = stand;
    new_item.cur_stand_qnt = stoi(cur_stand_qnt);
    new_item.max_stand_qnt = stoi(max_stand_qnt);
    new_item.inventory = stoi(inventory);
    new_item.item_length = stod(item_length);
    new_item.was_found = true;
```

```
    // storage 종류 입력받기
    int storage_type;
    cout << "Enter storge type: 0. Normal, 1. Refrigerated Food";
    cin >> storage_type;

    switch(storage_type)
    {
        case 0:
            // 상품 추가
            storage.AddLine(new_item);
            break;
        case 1:
        {
            // 매대 이름 입력 받기
            string stand_name;
            cout << "Enter stand name: ";
            cin >> stand_name;

            // 상품 추가
            storage_ref.AddLine(stand_name, new_item);
            break;
        }

    }

    // 품목 출력
    for(auto *s : storages)
    {
        s->PrintDatabase();
        cout << "--------------------------------------------------------------------------------" <<
        endl;
    }
    break;
}
```

ii. 입력

- Item new_line: 새로 추가할 상품의 정보가 담긴 Item 구조체

iii. 반환값

- 없음

iv. 결과

- 상품을 database에 추가한다

v. 설명

- 입력값인 new_line은 database의 목록 규격에 맞춰서 미리 생성되어 있어야 한다

- 메뉴 선택 switch문 case 2이다

- 코드 진행 도중 0을 입력하여 일반 매대에 상품 추가 진행

vi. 수업 연관 내용

- 10주차 vector

D. 재고 관리 기능 – 상품 제거

i. 코드 스크린샷

storage.cpp

```
void StorageRef::RemoveItem(string code)
{
    for(Stand &s : stands)
    {
        this->database = s.database; // 현재 database에 매대의 database 임시 할당
        Storage::RemoveItem(code); // 부모 함수 재사용
        s.database = this->database;
    }
}
```

main.cpp

```
case 5: // Remove item
{
    string item_code;
    cout << "Item Code: ";
    cin >> item_code;

    // 상품 삭제 처리
    for (auto *s : storages)
    {
        s->RemoveItem(item_code);
    }

    // 품목 출력
    for (auto *s : storages)
    {
        s->PrintDatabase();
        cout << "--------------------------------------------------------------------------------" << endl;
    }
    break;
}
```

ii. 입력

- string code: 상품의 code값

iii. 반환값

- 없음

iv. 결과

- code값에 해당하는 상품을 삭제한다

v. 설명

- code값으로 검색하여 해당 코드의 상품을 제거

- 메뉴 선택 switch문 case 5이다

vi. 수업 연관 내용

- 10주차 vector

E. 재고 관리 기능 – 관리 명령어 입력창

i. 코드 스크린샷

main.cpp

```
12      bool flag = true;
13      int menu;
14      while(flag)
15      {
```

```
// 메뉴 선택
cout << "1. Sell item  2. New item  3. Place item on stand  4. Add item in storage " << endl;
cout << "5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage" << endl;

cin >> menu;
switch (menu)
```

ii. 입력

- bool flag: 반복문 탈출 flag

- int menu: 명령어 입력 값

iii. 결과

- 입력 받은 메뉴에 따라 상품 판매, 상품 추가, 상품 진열, 재고 추가, 상품 삭제, 저장 및 종료 명령을 실행한다

iv. 설명

- switch문을 통해 선택한 메뉴를 실행한다

- 저장 및 종료 명령의 경우 flag값을 바꿔 반복문을 탈출, 프로그램이 종료된다.

v. 수업 연관 내용

- 4주차 condition

F. 재고 관리 기능 – 저장 및 종료

i. 코드 스크린샷

storage.cpp

```
void Storage::Close()
{
    cout << storage_name << " closed" << endl;
    ofstream outfile(this->storage_name);
    for(int i = 0; i < this->database.size(); i++)
    {
        outfile << this->database[i].code << ",";
        outfile << this->database[i].name << ",";
        outfile << this->database[i].stand << ",";
        outfile << this->database[i].cur_stand_qnt << ",";
        outfile << this->database[i].max_stand_qnt << ",";
        outfile << this->database[i].inventory;

        if(i != this->database.size() - 1)
            outfile << endl;
    }
    outfile.close();
}
```

main.cpp

```
case 6:
{
    // flag를 false로 바꿔 무한 반복문 탈출
    flag = false;
    break;
}
```

ii. 입력

- 없음

iii. 반환값

- 없음

iv. 결과

- 변경된 내용으로 동일한 파일명의 파일을 만들어 저장한다

v. 설명

- 반복문 종료 후 Close 함수를 호출

- Close 함수를 통해 데이터베이스의 내용을 기반으로 새로운 csv파일을 만들어 저장

- 메뉴 선택 switch문 case 6이다

vi. 수업 연관 내용

- 14주차 FileIO

G. 재고 관리 기능 – 데이터베이스 출력

i. 코드 스크린샷

storage.cpp

```
void Storage::PrintDatabase()
{
    cout << "         code                    name         stand" << endl;
    for(item i : this->database)
    {
        cout.width(13);
        cout << right << i.code << " ";
        cout.width(25);
        cout << right << i.name << " ";
        cout. width(13);
        cout << right << i.stand << "   " << i.cur_stand_qnt << "/"
             << i.max_stand_qnt << "/" << i.inventory << endl;
    }
}
```

ii. 입력

- 없음

iii. 반환값

- 없음

iv. 결과

- 데이터베이스 출력

v. 설명

- database의 각 입력들을 형식에 맞춰 출력할 수 있도록 cout.width()로 간격 설정 및 std::right로 간격에 맞춰 출력

H. 매대 상품 수량 변경 기능 – 상품 판매

i. 코드 스크린샷

storage.cpp

```
void Storage::ItemSold(string code, int qnt)
{
    // code로 해당 item 검색
    for(int i = 0; i < this->database.size(); i++)
    {
        // item이 검색 되었을 경우
        if(this->database[i].code == code)
        {
            int qnt_result = database[i].cur_stand_qnt - qnt; // 판매 후 진열된 수량
            // 진열된 수량이 판매 수량보다 적은 경우 에러 메시지
            if(qnt_result < 0)
            {
                cout << "Not enough items on the stand. Check quantity of the sold item" << endl;
                return;
            }
            // 아닐 경우 판매 수량만큼 진열된 수량 감소
            else
            {
                this->database[i].cur_stand_qnt  = qnt_result;
            }
        }
    }
}
```

main.cpp

```
case 1: // Sell Item
{
    // 판매할 상품 code와 수량 입력
    string sell_item_code;
    int sell_qnt;
    cout << "Item Code: ";
    cin >> sell_item_code;
    cout << "Quantity: ";
    cin >> sell_qnt;

    Item item;
    bool item_found = false; // 상품을 찾았는지 검사하는 변수
    int cnt = 0;
    for (auto *s : storages)
    {
        cnt++;
        // 해당 상품 code로 검색해서 없을 경우 NULL 구조체 저장된다
        item = s->GetItem(sell_item_code);
        // NULL 구조체인지 검사
        if (item.was_found == true) // NULL 구조체가 아닐경우, 상품을 찾은 것이므로 반복문 종료
        {
            item_found = true;
            break;
        }
        // NULL 구조체일 경우, 상품을 찾지 못한 것이므로 다른 storage에서 상품 찾기 위해 반복문 진행
    }
    cout << cnt << endl;
```

```
if (item_found == false) // 모든 storage를 검색해도 상품을 찾지 못한경우
{
    cout << "There is no item code " << sell_item_code << endl;
    break; // case 종료
}

// 상품 판매 처리
for (auto *s : storages)
{
    s->ItemSold(sell_item_code, sell_qnt);
}

// 품목 출력
for (auto *s : storages)
{
    s->PrintDatabase();
    cout << "--------------------------------------------------------------------------------" << endl;
}
break;
```

ii. 입력

- string code: 판매할 상품의 code

- int qnt: 판매할 수량

iii. 반환값

- 없음

iv. 결과

- 판매한 수량만큼 매대에서 수량감소

- 매대에 수량이 부족할 경우 잘못된 입력이라고 판단

v. 설명

- 상품의 code로 database를 검색하여 해당 상품의 매대 진열 수량을 변경한다

- GetItem 함수가 검색에 실패해 NULL 구조체를 반환하는 경우를 검사하여 오류 처리

- 메뉴 선택 switch문 case 1이다

I. 매대 상품 수량 변경 기능 – 매대 진열

i. 코드 스크린샷

storage.cpp

```
void Storage::ItemToStand(string code, int qnt)
{
    // code로 해당 item 검색
    for(int i = 0; i < this->database.size(); i++)
    {
        if(this->database[i].code == code)
        {
            // 재고가 부족할 경우 에러 메시지
            if(qnt > database[i].inventory)
            {
                cout << "Not enough items in the inventory. Check quantity of the item" << endl;
                return;
            }

            int qnt_result_cur = database[i].cur_stand_qnt + qnt; // 매대에 진열될 총 개수
            // 진열할 수 있는 최대치를 넘어서는 경우 에러 메시지
            if(qnt_result_cur > database[i].max_stand_qnt)
            {
                cout << "Not enough space for the item. Check quantity of the item" << endl;
                return;
            }
            this->database[i].cur_stand_qnt = qnt_result_cur; // 매대 진열 개수 변경
            this->database[i].inventory -= qnt; // 재고 수 변경
        }
    }
}
```

main.cpp

```
case 3: // Place item on stand
{
    // 상품 코드, 수량 입력
    string item_code;
    string qnt;

    cout << "Item Code: ";
    cin >> item_code;
    cout << "Quantity: ";
    cin >> qnt;

    // 상품 진열 처리
    for (auto *s : storages)
    {
        s->ItemToStand(item_code, stoi(qnt));
    }

    // 품목 출력
    for (auto *s : storages)
    {
        s->PrintDatabase();
        cout << "--------------------------------------------------------------------------------" << endl;
    }
    break;
}
```

ii. 입력

- string code: 판매할 상품의 code

- int qnt: 매대에 진열할 수량

iii. 반환값

- 없음

iv. 결과

- 재고 상품의 수량이 감소하고 매대에 진열된 수량이 증가한다

v. 설명

- 재고 수량이 부족하거나 매대에 공간이 부족한 상황에서 너무 많은 수량을 진열하려 한 경우 에러 메시지 출력

- 메뉴 선택 switch문 case3이다

J. 피크 타임에 따른 진열 필요 상품 제시

i. 코드 스크린샷

main.cpp

```
cout << "Item to display on the stand" << endl;
if (isPeak == true)
    cout << "Peaktime = true" << endl;
else
    cout << "Peaktime = false" << endl;

for (int i = 0; i < storage.GetSize(); i++)
{
    // line번째 아이템들의 값을 저장
    Item item = storage.GetItem(i);
    int cur_stand_qnt = item.cur_stand_qnt;
    int max_stand_qnt = item.max_stand_qnt;
    int storage_qnt = item.inventory;

    if (isPeak == true) // 피크타임일 경우 매대에 넣을 수 있는 공간이 있기만 하면 해당 상품 및 추가 진열 수량 제시
    {
        if (cur_stand_qnt < max_stand_qnt && storage_qnt >= (max_stand_qnt - cur_stand_qnt))
        {
            cout.width(13);
            cout << std::right << item.code;
            cout.width(25);
            cout << std::right << item.name << " ---> " << max_stand_qnt - cur_stand_qnt << endl;
        }
    }
    else // 피크타임이 아닐 경우
    {
        // 매대 최대 수량의 40%보다 적은 수량이 매대에 있을 경우 우선도가 높은 진열 품목으로 제시
        if (cur_stand_qnt < (int)(max_stand_qnt * 0.4) && storage_qnt >= ((int)(max_stand_qnt * 0.4) - cur_stand_qnt))
        {
            cout.width(13);
            cout << std::right << item.code;
            cout.width(25);
            cout << std::right << item.name << " ---> " << max_stand_qnt - cur_stand_qnt << "  Urgent" << endl;
        }
        // 매대 최대 수량의 40%이상 60% 미만의 수량이 매대에 있을 경우 우선도가 낮은 진열 품목으로 제시
        else if ((cur_stand_qnt >= (int)(max_stand_qnt * 0.4) && cur_stand_qnt < (int)(max_stand_qnt * 0.6))
            && storage_qnt >= ((int)(max_stand_qnt * 0.6) - cur_stand_qnt))
        {
            cout.width(13);
            cout << std::right << item.code;
            cout.width(25);
            cout << std::right << item.name << " ---> " << max_stand_qnt - cur_stand_qnt << "  Not Urgent" << endl;
        }
    }
}
```

ii. 입력

- bool isPeak: 피크타임 여부

- Item item: 상품 정보

- int cur_stand_qnt, max_stand_qnt, storage_qnt: 상품 정보로부터 추출한 현재 매대 진열 수량, 최대 진열 가능 수량, 재고 수

iii. 결과

- 매대 진열된 수량과 최대 진열 가능 주량,

iv. 설명

- 피크타임인 경우: 모든 상품을 최대한 빠르게 진열해야 하기 때문에 우선순위를 제시하지 않고 최대 진열 가능 수량보다 매대 진열 수량이 적을 경우 진열해야 하는 상품과 수량을 제시한다

- 피크타임이 아닌 경우: 매대 최대 수량의 40%보다 적게 진열된 경우 우선순위가 높은 상품, 60%보다 작고 40%이상 진열된 경우 우선순위가 낮은 상품, 두 카테고리로 나누어 진열해야 하는 상품과 수량을 제시한다

- 재고 수 보다 많은 수량을 진열해야 하는 경우는 제시하지 않는다

K. 로지스틱 회귀 분석을 통한 피크 타임 판별 – 시그모이드 함수

i. 코드

Peaktime.h

```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>
using namespace std;

const double LEARNING_RATE = 0.1;
const int EPOCH = 1000;
class PeakTime
{
private:
    double w; // weight 값
    double b; // bias 값
    double Sigmoid(double z); // 시그모이드 함수
    // 학습데이터의 X, Y값, weight, bias 값을 인자로 받아 경사하강법을 수행하는 함수
    void GradientDescent(vector<int>& X, vector<double>& Y, double& w, double& b, double learning_rate);
public:
    void ModelUpdate(double learning_rate, int epoch); // 회귀분석을 통해 w, b값을 업데이트
    bool Predict(int X); // 이전 시간대 판매량이 주어지면 이전에 피크타임이었는지 모델에 기반해 예측
};
```

Peaktime.cpp

```cpp
// 시그모이드 함수
double PeakTime::Sigmoid(double z)
{
    return (1 / (1 + exp(-z)));
}
```

ii. 입력

- double z: 시그모이드 함수의 인자

iii. 반환값

- 시그모이드 함수 값 반환

iv. 결과

- 시그모이드 함수 값을 계산한다

L. 로지스틱 회귀 분석을 통한 피크 타임 판별 – 경사하강법

i. 코드

```
// X에는 시간대별 고객 수가 저장된 1차원 벡터 대입
// Y에는 시간대 중 고객수가 많은 최상위 5개의 시간대를 피크타임이라고 판단해 1, 나머지는 0인 1차원 벡터 대입
void PeakTime::GradientDescent(vector<int>& X, vector<double>& Y, double & w, double& b, double learning_rate)
{
    const int m = X.size(); // 벡터 데이터의 사이즈, X, Y사이즈가 동일
    double loss = 0, dw = 0, db = 0;

    // cost function, dw, db값 계산
    for(int i = 0; i < m; i++)
    {
        double hyphthesis = Sigmoid(w * X[i] + b);

        dw += (hyphthesis - Y[i]) * X[i];
        db += hyphthesis - Y[i];
    }

    // weigth, bias 값 업데이트
    this->w -= learning_rate * dw / m;
    this->b -= learning_rate * db / m;
}
```

ii. 입력

vector<int>& X: 독립변수 벡터

vector<int>& Y: 종속변수 벡터

double& w: weight 값, 회귀식의 기울기

double& b: bias 값

double& learning_rate: 학습률

iii. 반환값

- 없음

iv. 결과

- X, Y를 기반으로 경사하강법을 통해 w, b값을 업데이트 한다

v. 설명

- X를 시그모이드 함수에 돌린 결과인 0~1사이의 값과 1과 0으로 구성된 Y와의 차를 바탕으로 w값과 b값을 업데이트 한다

M. 로지스틱 회귀 분석을 통한 피크 타임 판별 – 모델 업데이트

i. 코드

Peaktime.cpp

```
void PeakTime::ModelUpdate(double learning_rate, int epoch)
{
    // 파일 읽고 쓰기 위한 fstream 객체, 문자열 버퍼 생성
    ifstream ifs;
    string str_buf;

    vector<vector<int>> X; // 각 시간대별 판매량이 저장된 X 벡터
    vector<vector<double>> Y; // 각 시간대별 피크타임 여부가 저장될 12*n Y 행렬

    // 각 시간대별 판매 고객 수가 저장된 12*n 행렬이 저장된 csv 파일 읽기 모드로 오픈
    ifs.open("customer_per_hour.csv", ios::in);

    while(!ifs.eof())
    {
        vector<int> tmp; // X 행렬의 각 행을 임시 저장하기 위한 벡터
        getline(ifs, str_buf, '\n');

        // string 형태로 읽어진 csv파일 한 줄을 읽어 tmp에 저장한다
        int position;
        int cur_position = 0;
        while((position = str_buf.find(",", cur_position)) != string::npos)
        {
            int len = position - cur_position;
            tmp.push_back(stod(str_buf.substr(cur_position, len)));
            cur_position = position + 1;
        }
        tmp.push_back(stoi(str_buf.substr(cur_position)));

        X.push_back(tmp); // tmp행을 X 행렬에 추가
    }
    ifs.close(); // 파일 종료

    // 각 시간대별 피크타임 여부가 저장된 12*n 행렬이 저장된 csv 파일 읽기 모드로 오픈
    ifs.open("is_peak.csv", ios::in);
```

```
while(!ifs.eof())
{
    vector<double> tmp; // Y 행렬의 각 행을 임시 저장하기 위한 벡터
    getline(ifs, str_buf, '\n');

    // string 형태로 읽어진 csv파일 한 줄을 읽어 tmp에 저장한다
    int position;
    int cur_position = 0;
    while((position = str_buf.find(",", cur_position)) != string::npos)
    {
        int len = position - cur_position;
        tmp.push_back(stod(str_buf.substr(cur_position, len)));
        cur_position = position + 1;
    }
    tmp.push_back(stod(str_buf.substr(cur_position)));

    Y.push_back(tmp); // tmp행을 Y 행렬에 추가
}
ifs.close(); // 파일 종료

// 로지스틱 선형 회귀 모델 학습, epoch만큼 학습한다
for(int k = 0; k < epoch; k++)
{
    for(int i = 0; i < X.size(); i++)
    {
        GradientDescent(X[i], Y[i], w, b, learning_rate);
    }
}


cout << w << " " << b << endl;
```

main.cpp

```
PeakTime pt;
pt.ModelUpdate(LEARNING_RATE, EPOCH); // 프로그램 시작할 때 피크타임 계산 모델 업데이트
```

ii. 입력

- double learning_rate: GradientDescent 함수에 전달할 학습률

- int epoch: 학습 반복 횟수

iii. 반환값

- 없음

iv. 결과

- 모델을 학습시킨다

- w값과 b값을 출력한다

v. 설명

- customer_per_hour.csv 파일을 읽어 X벡터에, is_peak.csv 파일을 읽어 Y벡터에 저장, epoch만큼 GradientDescent 함수를 호출한다

- is_peak.csv파일은 하루를 기준으로 판매량이 가장 많은 5개의 시간대를 피크타임, 나머지를 피크타임X로 분류한 라벨 데이터

N. 로지스틱 회귀 분석을 통한 피크 타임 판별 – 예측

i. 코드

```
bool PeakTime::Predict(int prev_sales_vol)
{
    // 이전 시간대가 피크 타임이었을 경우 true 아닐 경우 false 반환
    return (Sigmoid(this->w * prev_sales_vol + this->b) < 0.5) ? false : true;
}
```

ii. 입력

- int prev_sales_vol: 이전 시간대 판매량

iii. 반환값

- 피크 타임 여부에 대한 bool값

iv. 결과

- 이전 시간대가 피크타임이라고 예측될 경우 true 아닐 경우 false를 반환한다

O. 이전 판매량 업데이트

i. 코드

main.cpp

```
    // 현재 시각 표시해주기
    time_t raw_time = time(NULL); // 현재 시간정보를 가지고 있는 time_info 구조체 생성
    struct tm *time_info = localtime(&raw_time);
    cout << "Current Time: " << time_info->tm_hour << "h " << time_info->tm_min << "m" << endl; // 현재 시각 출력
    cout << "After " << time_info->tm_hour + 1 << ":00, Update Peaktime" << endl; // 수동 업데이트 해야 할 시각 알려주기
```

```
case 7: // Update Peaktime
{
    int customer_num; // 이전 시간대 손님 수
    cout << "Number of customers in previous time: ";
    cin >> customer_num;              // 손님 수 입력
    isPeak = pt.Predict(customer_num); // 손님 수를 바탕으로 모델을 통해 이전 시간대가 피크타임이었는지 예측
}
```

ii. 입력

- customer_num: 이전 시간대 손님수

iii. 결과

- 현재 시각과 이전 시간대 손님 수를 업데이트 해야 할 시각을 출력한다

- 이전 시간대 피크타임 여부를 업데이트 한다

iv. 설명

- 무한 반복문이 실행될 때 마다 현재 시각과 이전 시간대 손님 수를 업데이트 해야 할 시각을 출력한다

- 업데이트 해야 할 시각에 손님 수를 수동으로 업데이트 하는 방식으로 설계

- 7번 메뉴선택 입력된 이전 시간대 손님 수를 모델에 대입해 피크타임 여부를 판별한다

- 메뉴 선택 switch문 case 7이다

P. 냉장 매대 조합 업데이트 기능 – 헤더파일

i. 코드

storage.h

```cpp
enum StandType
{
    Fix,
    Free,
    Oneline
}; // 냉장 매대 타입
```

```cpp
// 냉장 매대 구조체
struct stand
{
    string name; // 매대 이름
    int max_stand_qnt; // 세로로 최대 진열 개수
    double stand_length; // 매대의 길이
    StandType stand_type; // 매대의 타입, Fix, Free, Oneline이 있다
    string item_type; // 진열될 상품의 타입, Free 매대의 경우 Free
    vector<Item> database; // 매대에 등록된 상품들의 동적 배열
    bool was_found; // 기본적으로 true, 매대가 검색되지 않았을 때 이 값이 false인 구조체 반환
} typedef Stand;
```

```cpp
// 냉장 식품 매대 코너
class StorageRef : public Storage
{
protected:
    vector<Stand> stands; // 매대를 저장하는 동적 배열
    // nCr계산 함수, n크기의 피추출 집단, r크기의 임시 벡터, 결과 저장 벡터, r값,
    // 재귀함수의 초기값(index=0,depth=0으로 세팅)
    void Combination(vector<Item> arr, vector<Item> comb, vector<vector<Item>>& result,
        int r, int index, int depth);
public:
    void Open(string filename); // 파일 여는 함수 재정의

    // 각 매대에 저장된 database에 대해 작동할 수 있도록 함수 재정의
    Item GetItem(string code);

    void AddLine(string name, Item new_line);
    void RemoveItem(string code);
    void ItemSold(string code, int qnt);
    void ItemToStand(string code, int qnt);
    void ItemStore(string code, int qnt);
    void PrintDatabase();

    void NewStand(Stand new_stand); // stnad 추가
    void RemoveStand(string name); // stand 삭제
    vector<Item> UpdateItemList(); // 냉장 매대에 채워 넣어야 할 상품 목록 생성 및 반환

    void Close(); // 파일 저장 함수 재정의
};
```

ii. 설명

- enum StandType: 냉장 매대의 종류

Fix: 한 종류의 음식을 횡방향으로 진열가능(예: 삼각김밥만)

Free: 여러 종류의 음식 진열가능

Oneline: 한 종류의 음식을 종방향으로 진열가능

- struct Stand: 냉장 매대 구조체

- StorageRef: 냉장 매대 클래스, Storage 클래스 상속

Q. 냉장 매대 조합 업데이트 기능 – 함수 오버라이드

i. void Open(string filename)

```
// 냉장식품 Storage는 기본 Storage와 자료의 형태가 다르므로 함수를 재정의하여 형식에 맞게 읽을 수 있도록 한다
void StorageRef::Open(string filename)
{
    string str_buf;
    this->storage_name = filename;
    this->fs.open(filename, ios::in);
    int stand_idx = -1;

    while(!fs.eof())
    {
        int position;
        int cur_position = 0;

        vector<string> tmp; // 파일을 읽기 위한 임시 벡터, 한 줄을 저장
        getline(this->fs, str_buf,'\n');

        if(str_buf == "/") // 매대 정보 구분자를 읽을 경우
        {
            getline(this->fs, str_buf, '\n'); // 매대 정보 한 줄을 저장
            // string 형태로 읽어진 csv파일 한 줄을 읽어 tmp에 저장한다
            while((position = str_buf.find(",", cur_position)) != string::npos)
            {
                int len = position - cur_position;
                tmp.push_back(str_buf.substr(cur_position, len));
                cur_position = position + 1;
            }
            tmp.push_back(str_buf.substr(cur_position));

            // tmp의 string 값을 자료형에 맞게 Stand 구조체에 저장
            Stand data_extracted;
            data_extracted.name = tmp[0];
            data_extracted.max_stand_qnt = stoi(tmp[1]);
            data_extracted.stand_length = stod(tmp[2]);
            if(tmp[3] == "Fix")
            {
                data_extracted.stand_type = Fix;
            }
```

```
        else if(tmp[3] == "Oneline")
        {
            data_extracted.stand_type = Oneline;
        }
        else
        {
            data_extracted.stand_type = Free;
        }
        data_extracted.was_found = true;

        this->stands.push_back(data_extracted); // Stand 구조체 동적 배열에 저장
        stand_idx++; // 다음 읽어지는 상품 정보들이 현재 읽은 매대에 등록되도록 현재 매대의 인덱스 값 증가
        continue;
    }

    // string 형태로 읽어진 csv파일 한 줄을 읽어 tmp에 저장한다
    while((position = str_buf.find(",", cur_position)) != string::npos)
    {
        int len = position - cur_position;
        tmp.push_back(str_buf.substr(cur_position, len));
        cur_position = position + 1;
    }
    tmp.push_back(str_buf.substr(cur_position));

    // tmp의 string 값을 자료형에 맞게 Item 구조체에 저장
    Item data_extracted;

    data_extracted.code = tmp[0];
    data_extracted.name = tmp[1];
    data_extracted.stand = tmp[2];
    data_extracted.cur_stand_qnt = stoi(tmp[3]);
    data_extracted.max_stand_qnt = stoi(tmp[4]);
    data_extracted.inventory = stoi(tmp[5]);
    data_extracted.item_length = stod(tmp[6]); // 냉장식품의 경우 자료의 종류가 하나 더 있기 때문에 추가로 처리해준다
    data_extracted.was_found = true;

    // 추출한 Item을 특정 stand의 database에 저장한다
    this->stands[stand_idx].database.push_back(data_extracted);
}
fs.close();
}
```

ii. Item GetItem(string code)

```
Item StorageRef::GetItem(string code)
{

    Item tmp;
    for(Stand s : stands)
    {
        this->database = s.database; // 현재 database에 매대의 database 임시 할당
        tmp = Storage::GetItem(code); // 부모 함수 재사용
        if(tmp.was_found == true)
        cout << "item found" << endl;
            break;
    }
    return tmp;
}
```

iii. void RemoveItem(string code)

```
void StorageRef::RemoveItem(string code)
{
    for(Stand &s : stands)
    {
        this->database = s.database; // 현재 database에 매대의 database 임시 할당
        Storage::RemoveItem(code); // 부모 함수 재사용
        s.database = this->database;
    }
}
```

iv. void ItemSold(string code, int qnt)

```
void StorageRef::ItemSold(string code, int qnt)
{
    for(Stand &s : stands)
    {
        this->database = s.database; // 현재 database에 매대의 database 임시 할당
        Storage::ItemSold(code, qnt); // 부모 함수 재사용
        s.database = this->database;
    }
}
```

v. void ItemToStand(string code, int qnt)

```
void StorageRef::ItemToStand(string code, int qnt)
{
    for(Stand &s : stands)
    {
        this->database = s.database; // 현재 database에 매대의 database 임시 할당
        Storage::ItemToStand(code, qnt); // 부모 함수 재사용
        s.database = this->database;
    }
}
```

vi. void ItemStore(string code, int qnt)

```
void StorageRef::ItemStore(string code, int qnt)
{
    for(Stand &s : stands)
    {
        this->database = s.database; // 현재 database에 매대의 database 임시 할당
        Storage::ItemStore(code, qnt); // 부모 함수 재사용
        s.database = this->database;
    }
}
```

vii. void PrintDatabase()

```cpp
void StorageRef::PrintDatabase()
{
    for(Stand s : stands)
    {
        cout << "Stand: " << s.name << endl;
        cout << "Type: ";
            if(s.stand_type == Fix)
                cout << "Fix" << endl;
            else if(s.stand_type == Free)
                cout << "Free" << endl;
            else
                cout << "Oneline" << endl;
        this->database = s.database; // 현재 database에 매대의 database 임시 할당
        Storage::PrintDatabase(); // 부모 함수 재사용
    }
}
```

viii. void Close()

```cpp
void StorageRef::Close()
{
    cout << storage_name << " closed" << endl;
    ofstream outfile(this->storage_name);
    for(int i = 0; i < this->stands.size(); i++)
    {
        outfile << "/" << endl;
        outfile << this->stands[i].name << ",";
        outfile << this->stands[i].max_stand_qnt << ",";
        outfile << this->stands[i].stand_length << ",";
        outfile << this->stands[i].stand_type << endl;

        for(int k = 0; k < this->stands[i].database.size(); k++)
        {
            outfile << this->stands[i].database[k].code << ",";
            outfile << this->stands[i].database[k].name << ",";
            outfile << this->stands[i].database[k].stand << ",";
            outfile << this->stands[i].database[k].cur_stand_qnt << ",";
            outfile << this->stands[i].database[k].max_stand_qnt << ",";
            outfile << this->stands[i].database[k].inventory << "," ;
            outfile << this->stands[i].database[k].item_length;
            if(i != this->stands.size() - 1 || k != this->stands[i].database.size() - 1)
                outfile << endl;
        }
    }
    outfile.close();
}
```

ix. 설명

main.cpp

```cpp
// 이후 다른 종류의 storage가 추가됐을 경우의 확장성 고려한 코드
Storage *storages[] = {&storage, &storage_ref}; // 다른 storage가 추가되어도 배열에 추가만 하면 된다
```

- virtual로 선언된 함수들의 다형성을 통해 main 함수에서 database를 검색하는 모든 과정을 오버라이드된 형태로 실행할 수 있다

x. 수업 연관 내용

- 13주차 Polymorphism

R. 냉장 매대 조합 업데이트 기능 – 함수 오버로드

i. void Addline(string name, Item new_line)

```
void StorageRef::AddLine(string name, Item new_line)
{
    Stand* tmp;
    bool was_found = false;

    // name으로 stand 검색
    for(Stand &s : this->stands)
    {
        if(s.name == name)
        {
            tmp = &s; // 해당 stand가 있을 경우 반환
            was_found = true;
        }
    }
    if(was_found == false) // 찾지 못 한 경우 함수 종료
        return;

    // 정해진 종류에 맞게 등록해야 하는데 그러지 않은 경우 에러 발생
    if(tmp->stand_type != Free && !tmp->item_type.compare(new_line.stand))
    {
        cout << "Type is not matching!" << endl;
        return;
    }
    tmp->database.push_back(new_line);
}
```

- string name: 추가된 입력, 상품을 추가할 매대 이름을 추가로 받은 뒤 해당 매대에 상품 추가

S. 조합 구현 함수

i. 코드

```cpp
void StorageRef::Combination(vector<Item> arr, vector<Item> comb, vector<vector<Item>> &result, int r, int index, int depth)
{
    if (r == 0)
    {
        vector<Item> tmp;
        for(Item i : comb)
        {
            tmp.push_back(i);
        }
        result.push_back(tmp);
    }
    else if (depth == arr.size())  // depth == n // 계속 안뽑다가 r 개를 채우지 못한 경우는 이 곳에 걸려야 한다.
    {
        return;
    }
    else
    {
        // arr[depth] 를 뽑은 경우
        comb[index] = arr[depth];
        Combination(arr, comb, result, r - 1, index + 1, depth + 1);

        // arr[depth] 를 뽑지 않은 경우
        Combination(arr, comb, result, r, index, depth + 1);
    }
}
```

ii. 입력

- vector<Item> arr: 조합을 추출할 리스트

- vector<Item> comb: 길이가 r인 임시 배열, 재귀구조를 위해 필요한 인자

- vector<vector<Item>> &result: 조합의 결과를 저장할 2차원 동적 배열

- int r: nCr의 r값

- int index, depth: 재귀구조를 위해 필요한 인자

iii. 반환값

iv. 결과

- nCr의 모든 조합이 저장된 2차원 동적 배열을 result 인자로 전달된 배열을 참조해 저장한다

v. 설명

- index, depth는 초기에 0으로 세팅하고 함수를 호출해야 한다

- 최종 결과가 인자로 전달되는 result에 저장된다

T. 냉장 매대 추가

i. 코드

```
void StorageRef::NewStand(Stand new_stand)
{
    this->stands.push_back(new_stand);
}
```

ii. 입력

- Stand new_stand: 리스트에 추가할 새로운 Stand 구조체

iii. 반환값

- 없음

iv. 결과

- 새로운 냉장 매대 구조체가 객체의 stands 벡터에 추가

v. 설명

- main 함수에서 함수를 불러올 때 구조체가 만들어져 있어야 한다

U. 냉장 매대 삭제

i. 코드

```
void StorageRef::RemoveStand(string name)
{
    for(int i = 0; i < this->stands.size(); i++)
    {
        // 검색되었을 경우 해당 stand 삭제
        if(this->stands[i].name == name)
        {
            this->stands.erase(this->stands.begin() + i);
            return;
        }
    }
}
```

ii. 입력

- string name: 삭제할 냉장 매대 이름값

iii. 반환값

- 없음

iv. 결과

- 검색된 냉장 매대가 삭제된다

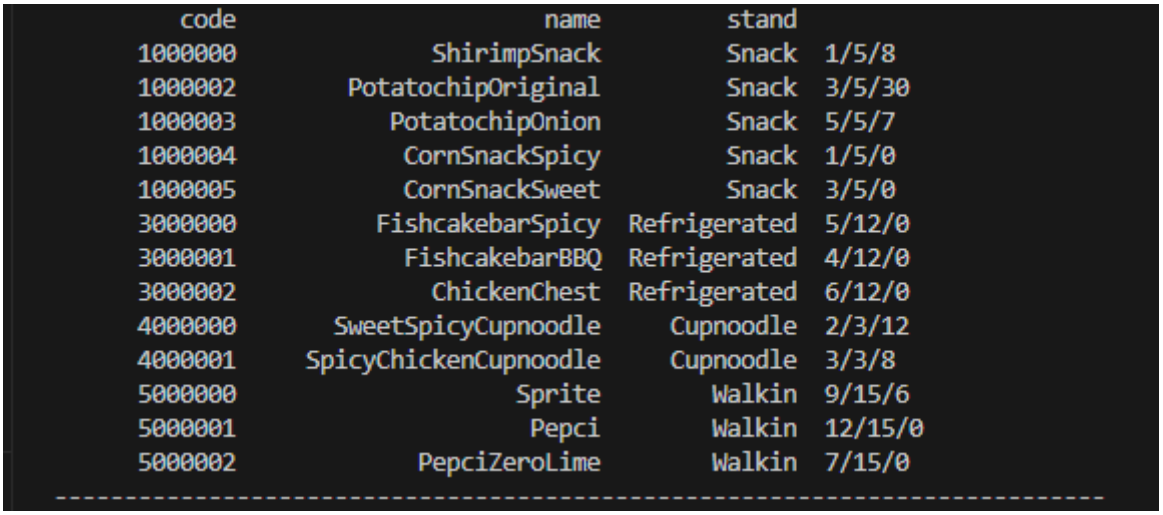
v. 설명

- 검색되지 않을 경우 삭제되지 않는다

5. 테스트결과

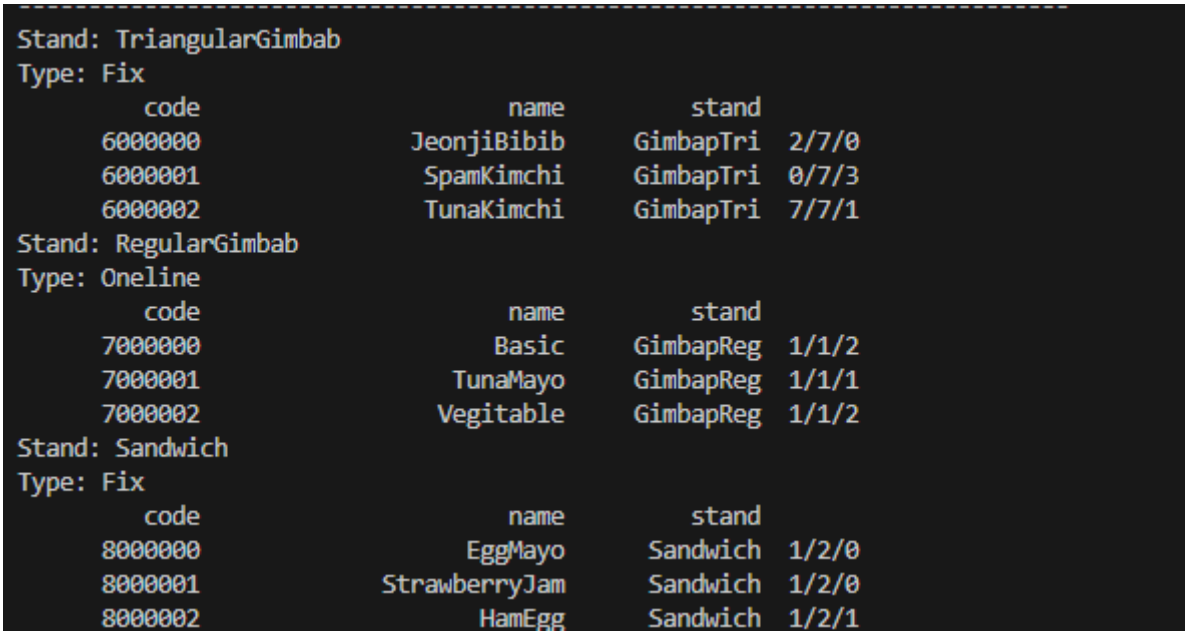
A. database 출력

- 일반 매대



code	name	stand	
1000000	ShirimpSnack	Snack	1/5/8
1000002	PotatochipOriginal	Snack	3/5/30
1000003	PotatochipOnion	Snack	5/5/7
1000004	CornSnackSpicy	Snack	1/5/0
1000005	CornSnackSweet	Snack	3/5/0
3000000	FishcakebarSpicy	Refrigerated	5/12/0
3000001	FishcakebarBBQ	Refrigerated	4/12/0
3000002	ChickenChest	Refrigerated	6/12/0
4000000	SweetSpicyCupnoodle	Cupnoodle	2/3/12
4000001	SpicyChickenCupnoodle	Cupnoodle	3/3/8
5000000	Sprite	Walkin	9/15/6
5000001	Pepci	Walkin	12/15/0
5000002	PepciZeroLime	Walkin	7/15/0

- 냉장 매대



Stand: TriangularGimbab
Type: Fix

code	name	stand	
6000000	JeonjiBibib	GimbapTri	2/7/0
6000001	SpamKimchi	GimbapTri	0/7/3
6000002	TunaKimchi	GimbapTri	7/7/1

Stand: RegularGimbab
Type: Oneline

code	name	stand	
7000000	Basic	GimbapReg	1/1/2
7000001	TunaMayo	GimbapReg	1/1/1
7000002	Vegitable	GimbapReg	1/1/2

Stand: Sandwich
Type: Fix

code	name	stand	
8000000	EggMayo	Sandwich	1/2/0
8000001	StrawberryJam	Sandwich	1/2/0
8000002	HamEgg	Sandwich	1/2/1

```
Stand: Lunchbox
Type: Fix
        code                    name        stand
     9000000                 Bulgogi     Lunchbox  1/1/0
     9000001           CompactDietBox     Lunchbox  1/1/0
     9000002           SpicyFriedPork     Lunchbox  1/1/0
     9000003         CompactFriedRice     Lunchbox  0/1/1
     9000004              PorkCutlet     Lunchbox  0/1/1
     9000005             Combination     Lunchbox  0/1/2
Stand: Free1
Type: Free
        code                    name        stand
    10000000                Rollcake       Dairy  1/2/0
    10000001             ChickenSalad       Salad  1/1/0
    10000002           FineCheeseSet       Diary  2/3/0
    10000006                 Pudding       Diary  0/3/3
    10000007                 EggSalad       Salad  0/1/1
Stand: Free2
Type: Free
        code                    name        stand
    10000003          MilkCreamBread       Diary  1/3/0
    10000004       ChocolateCreamBread       Diary  2/3/0
    10000005      StrawberryCreamBread       Diary  0/3/1
```

B. 상품 추가

- 일반 매대에 추가

```
1. Sell item  2. New item  3. Place item on stand  4. Add item in storage
5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage
2
Item Code: 0000
Item Name: test
Stand that Item belongs: Cupnoodle
Current Qunatity in Stand: 2
Max Qunatity of Stand: 3
Qunatity in Storage: 15
Length of Item(Put 0 if it isn't Refrigerated Food): 0
Enter storge type: 0. Normal, 1. Refrigerated Food0
        code                    name        stand
     1000000            ShirimpSnack       Snack  1/5/8
     1000002       PotatochipOriginal       Snack  3/5/30
     1000003          PotatochipOnion       Snack  5/5/7
     1000004           CornSnackSpicy       Snack  1/5/0
     1000005           CornSnackSweet       Snack  3/5/0
     3000000         FishcakebarSpicy  Refrigerated  5/12/0
     3000001           FishcakebarBBQ  Refrigerated  4/12/0
     3000002             ChickenChest  Refrigerated  6/12/0
     4000000      SweetSpicyCupnoodle    Cupnoodle  2/3/12
     4000001    SpicyChickenCupnoodle    Cupnoodle  3/3/8
     5000000                   Sprite       Walkin  9/15/6
     5000001                    Pepci       Walkin  12/15/0
     5000002           PepciZeroLime       Walkin  7/15/0
        0000                    test    Cupnoodle  2/3/15
```

- 냉장 매대에 추가

```
1. Sell item  2. New item  3. Place item on stand  4. Add item in storage
5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage
2
Item Code: test
Item Name: test
Stand that Item belongs: Lunchbox
Current Qunatity in Stand: 0
Max Qunatity of Stand: 1
Qunatity in Storage: 1
Length of Item(Put 0 if it isn't Refrigerated Food): 13.5
Enter storge type: 0. Normal, 1. Refrigerated Food1
Enter stand name: Lunchbox
```

C. 상품 판매

```
1. Sell item  2. New item  3. Place item on stand  4. Add item in storage
5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage
1
Item Code: 1000003
Quantity: 2
1000003          PotatochipOnion          Snack  5/5/7
1000003          PotatochipOnion          Snack  3/5/7
```

D. 진열 필요 상품 제시(피크타임 아닌 경우)

```
Item to display on the stand
Peaktime = false
      1000000            ShirimpSnack ---> 4  Urgent
      6000001              SpamKimchi ---> 3
      6000001              SpamKimchi ---> 3
      9000003         CompactFriedRice ---> 1
      9000003         CompactFriedRice ---> 1
      9000004               PorkCutlet ---> 1
      9000005              Combination ---> 1
         test                     test ---> 1
     10000006                  Pudding ---> 3
     10000006                  Pudding ---> 3
     10000007                 EggSalad ---> 1
     10000005     StrawberryCreamBread ---> 1
```

E. 진열 필요 상품 제시(피크 타임인 경우)

```
Item to display on the stand
Peaktime = true
      1000000            ShirimpSnack ---> 4
      1000002       PotatochipOriginal ---> 2
      4000000      SweetSpicyCupnoodle ---> 1
      5000000                  Sprite ---> 6
      6000001             SpamKimchi ---> 3
      6000001             SpamKimchi ---> 3
      9000003        CompactFriedRice ---> 1
      9000003        CompactFriedRice ---> 1
      9000004             PorkCutlet ---> 1
      9000005            Combination ---> 1
     10000006                Pudding ---> 3
     10000006                Pudding ---> 3
     10000007               EggSalad ---> 1
     10000005     StrawberryCreamBread ---> 1
--------------------------------------------------------------------------------
```

F. 상품 진열

```
 1. Sell item  2. New item  3. Place item on stand  4. Add item in storage
 5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage
 3
 Item Code: 10000005
 Quantity: 1
```

```
10000005      StrawberryCreamBread         Diary  0/3/1
```

```
10000005      StrawberryCreamBread         Diary  1/3/0
```

```
Item to display on the stand
Peaktime = true
      1000000            ShirimpSnack ---> 4
      1000002       PotatochipOriginal ---> 2
      4000000      SweetSpicyCupnoodle ---> 1
      5000000                  Sprite ---> 6
      6000001             SpamKimchi ---> 3
      6000001             SpamKimchi ---> 3
      9000003        CompactFriedRice ---> 1
      9000003        CompactFriedRice ---> 1
      9000004             PorkCutlet ---> 1
      9000005            Combination ---> 1
     10000006                Pudding ---> 3
     10000006                Pudding ---> 3
     10000007               EggSalad ---> 1
```

- 진열 후 진열해야 할 목록에서 없어진 것을 확인

G. 재고 추가

```
1. Sell item  2. New item  3. Place item on stand  4. Add item in storage
5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage
4
Item Code: 4000000
Quantity: 3
```

```
4000000       SweetSpicyCupnoodle      Cupnoodle  2/3/12
```

```
4000000       SweetSpicyCupnoodle      Cupnoodle  2/3/15
```

H. 제품 삭제

```
1. Sell item  2. New item  3. Place item on stand  4. Add item in storage
5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage
5
Item Code: 10000004
```

```
Type: Free
        code                    name         stand
    10000003          MilkCreamBread         Diary  1/3/0
    10000004       ChocolateCreamBread       Diary  2/3/0
    10000005     StrawberryCreamBread        Diary  1/3/0
```

```
Stand: Free2
Type: Free
        code                    name         stand
    10000003          MilkCreamBread         Diary  1/3/0
    10000005     StrawberryCreamBread        Diary  1/3/0
```

I. 피크 타임 업데이트

```
Item to display on the stand
Peaktime = false
      1000000            ShirimpSnack ---> 4  Urgent
      6000001              SpamKimchi ---> 3
      6000001              SpamKimchi ---> 3
      9000003         CompactFriedRice ---> 1
      9000003         CompactFriedRice ---> 1
      9000004               PorkCutlet ---> 1
      9000005              Combination ---> 1
     10000006                  Pudding ---> 3
     10000006                  Pudding ---> 3
     10000007                 EggSalad ---> 1
     10000005     StrawberryCreamBread ---> 1
--------------------------------------------------------------------------------
Current Time: 15h 24m
After 16:00, Update Peaktime
1. Sell item  2. New item  3. Place item on stand  4. Add item in storage
5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage
7
Number of customers in previous time: 60
Item to display on the stand
Peaktime = true
```

```
1.32952 -25.0154
```
<- 프로그램 시작 시 학습된 모델의 파라미터

J. 냉장 매대 추가

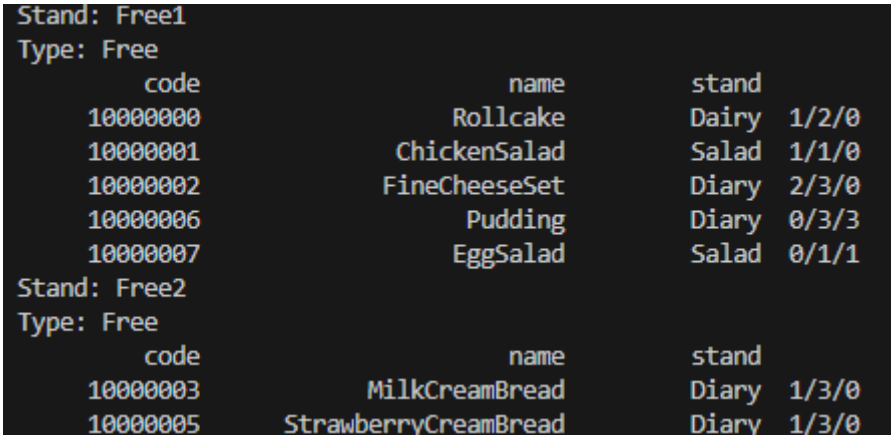
```
1. Sell item  2. New item  3. Place item on stand  4. Add item in storage
5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage
8
1. Add stand 2. Remove Stand 3. Previous menu
1
Stand name: test
Max Qunatity of Stand: 3
Length of Stand: 7
Type of Stand: Fix
Type of Item belongs to Stand: GimbapTri
```

```
Stand: test
Type: Fix
```

K. 냉장 매대 제거

```
1. Sell item  2. New item  3. Place item on stand  4. Add item in storage
5. Remove item 6. Save and Exit 7. Update Peaktime 8. Refrigerated food Storage
8
1. Add stand 2. Remove Stand 3. Previous menu
2
Enter name of stand to removetest
```

```
Stand: TriangularGimbab
Type: Fix
        code                 name        stand
     6000000          JeonjiBibib    GimbapTri  2/7/0
     6000001           SpamKimchi    GimbapTri  0/7/3
     6000002           TunaKimchi    GimbapTri  7/7/1
Stand: RegularGimbab
Type: Oneline
        code                 name        stand
     7000000                Basic    GimbapReg  1/1/2
     7000001             TunaMayo    GimbapReg  1/1/1
     7000002            Vegitable    GimbapReg  1/1/2
Stand: Sandwich
Type: Fix
        code                 name        stand
     8000000              EggMayo     Sandwich  1/2/0
     8000001         StrawberryJam     Sandwich  1/2/0
     8000002               HamEgg     Sandwich  1/2/1
Stand: Lunchbox
Type: Fix
        code                 name        stand
     9000000               Bulgogi     Lunchbox  1/1/0
     9000001        CompactDietBox     Lunchbox  1/1/0
     9000002         SpicyFriedPork     Lunchbox  1/1/0
     9000003       CompactFriedRice     Lunchbox  0/1/1
     9000004             PorkCutlet     Lunchbox  0/1/1
     9000005            Combination     Lunchbox  1/1/1
```



5. 계획 대비 변경 사항

1) 피크타임 판별 기능

- 이전: 현재 시간이 몇 시인지를 대입하면 학습한 모델이 피크타임인지 아닌지 판별

- 이후: 이전 시간대 판매량을 대입하면 학습한 모델이 피크타임인지 아닌지 판별

- 사유: 시간대별 판매량이 기존의 추세와 많이 달라지는 상황(예: 발렌타인 데이)에 대응하기 위해 피크타임 반별 기준 수정

6. 느낀점

- C++을 통해 객체지향 프로그래밍을 할 수 있었다

- git hub를 통해서 지속적으로 프로젝트를 진행하며 프로그램을 업데이트 할 수 있었다

- StorageRef 클래스를 Storage 클래스를 상속해 만드는 과정에서 다형성을 구현하는 과정을 경험하고 관련되어 학습했던 내용을 더 잘 이해할 수 있었다

- 상속한 클래스의 함수를 재정의하는 과정에서 기존의 함수를 재사용하는 것이 코드를 효율적으로 작성하는데 매우 중요하다는 것을 직접 체험할 수 있었다

- C++로 회귀분석, 러신머닝을 구현하는 것은 python에 비해 비효율적인 측면이 많다는 것을 알 수 있었다