프로그래밍언어론 디지털 꾸러미 만들기 과제

<product information print system>

20205602 박정현

본 보고서에서는 프로그래밍언어론 수업에서 배운 object oriented concept을 활용해 만든 디지털 꾸러미에 관한 설계와 구현에 관한 설명을 다루고자 한다. 디지털 꾸러미란 일반적으로 프로그래밍에서 객체(object)라 불리어진다. 데이터는 kaggle.com에서 공개된 data set을 사용하였으며 데이터를 처리할 수 있는 방법, 절차, 함수를 포함하는 class를 설계하여 c++로 구현하였다.

해당 프로젝트에서 <여러 종류의 제품을 하나의 공간에 저장하고, 제품의 id를 입력하면 해당 제품의 상세 정보가 출력>되는 system을 설계 및 제작하였다. 이를 "*product information print system*"이라고 명명하였으며 지금부터 프로젝트 진행 과정에서 고안되었던 아이디어 및 내용과 결과에 대한 설명을 하고자 한다.

1. 주제 선정 이유

앞서 프로젝트의 주제를 "*product information print system*"이라고 밝혔다. 여러 데이터셋으로 흩어져있는 다양한 제품을 한 공간에 모아 관리하고, 제품의 id를 입력 받으면 해당 제품의 상세 정보가 출력되는 프로그램이다.

인터넷에는 정말 많고 다양한 제품에 대한 정보가 존재한다. 인터넷에 하나의 글자만 입력해도 그와 관련된 수많은 제품과 상세 정보가 제공된다. 그러나 각 사이트마다 제공되는 정보의 종류가 다를 뿐 더러 성질이 매우 다른 제품 군은 보통 각기 다른 사이트에서 관리한다. 하나의 사이트에서 다양한 종류의 제품 군의 정보를 제공하는 경우도 있기는 하지만 사용자가 원하는 정보만을 선택적으로 추출 하기란 쉽지 않다.

따라서 시스템 이용자가 현재 관심있는 데이터셋만을 한 공간에 모아 이를 관리하고 상세 정보를 출력할 수 있는 시스템을 설계 및 제작하였다. 본인만의 데이터셋을 구축해 그것의 수정이 가능하며 지속적으로 시스템을 유지 및 관리할 수 있는 이용자에게 특히나 매우 유용하게 사용될 것이다.

2. 선택한 data 및 해당 data 선택 배경

제품들을 공통된 특성을 추상화 하여 data 상에서 관리할 것이기 때문에 각 제품의 추상화된 공통 속성과 대비하여 제품간의 차이가 강조 될 수 있도록 각 제품의 차별성이 뚜렷이 느껴질 수 있는 데이터 셋들을 선택하고자 노력했다.

최종적으로 kaggle.com에서 선택한 data는 총 3가지로 **IKEA Furniture**, **DMart Products, Mobile Phones Data**이다. 각 데이터에 대한 정보는 다음과 같다.

- **IKEA Furniture**: IKEA에서 판매하는 상품 중 가구(Furniture)에 속하는 제품의 정보를 담고있다. (IKEA Furniture | Kaggle)
- **DMart Products**: 온라인 식료품점 빅 바스켓의 웹사이트에서 판매하는 식료품의 정보를 담고있다. (DMart Products | Kaggle)
- **Mobile Phones Data**: 우크라이나에서 구입할 수 있는 지난 4년 동안 출시된 휴대폰의 정보를 담고있다. (Mobile Phones Data | Kaggle)

각각의 데이터들은 가구(furniture), 식료품(grocery), 휴대폰(phone)에 관한 정보를 담고 있으며 모두 차별화 할 수 있는 특징이 있는 제품 군이라고 판단이 되어 해당 데이터들을 선택했다. 더 다양하고 많은 데이터 셋을 선택해 프로젝트 구현에 이용할 수도 있었으나, 시간적 제약 사항으로 인해 데이터셋을 3가지로 제한했다. 하지만 다른 제품 군이 추가되어도 코드 상의 큰 수정 없이 충분히 프로젝트의 확장이 가능하며 추가할 수 있는 데이터가 많다는 점에서 프로그램의 활용도를 높이 평가할 수 있다.

설계 및 구현에 들어가기에 앞서 kaggle.com에서 제공되는 데이터에 대해 EDA(Exploratory Data Analysis)를 진행하며 데이터의 특징 및 분포를 파악하였다. 해당 데이터가 웹크롤링으로 수집이 되어 정제가 전혀 되어있지 않아 결측치 및 이상치가 많이 존재한다. 이와 관련하여 data에 대한 모든 전처리를 c++로 하기에는 무리가 있다고 판단 되어 python으로 간단한 1차 전처리를 진행했다. (이와 관련된 python 코드 또한 제출 파일에 함께 첨부하였다. - 제출물 중 3. data 및 전처리 코드) 데이터분석 결과 정보가 중복되는 칼럼이 존재하며 불필요한 정보를 담은 칼럼 또한 존재하였다. 예를 들면 제품 id의 경우 프로그램 내부에서 새로 생성하여 할당할 것이기에 불필요하다. 또한 null값이 존재하는 행의 경우 c++에서 파일 입력 처리할 때 꽤 까다로운 문제가 다수 발생할 것으로 예상되어 제거하였다.

결과적으로 불필요한 칼럼 제거 및 null값 제거의 등의 간단한 전처리만을 진행하여 깨끗한 데이터셋으로 새롭게 재구성해 이를 프로그램에 이용하였다. 해당 과정을 제외하고는 data에 대한 전처리는 모두 c++로 진행하였다. 전처리 전/후 각 데이터셋의 칼럼 변화는 다음과 같다.

- **IKEA Furniture:**

  ['Name', 'Brand', 'Price', 'DiscountedPrice', 'Category', 'SubCategory', 'Quantity', 'Description', 'BreadCrumbs']

  ⇨ ['Name', 'Price', 'DiscountedPrice', 'SubCategory', 'Brand', 'Quantity', 'Description']

- **DMart Products:**

  [Unnamed: 0, item_id, name, category, price, old_price, sellable_online, link, other_colors, short_description, designer, depth, height, width]

  ⇨ ['name', 'price', 'category', 'designer', 'sellable_online', 'depth', 'height', 'width', 'short_description']

- **Mobile Phones Data:**

  ['Unnamed: 0', 'brand_name', 'model_name', 'os', 'popularity', 'best_price', 'lowest_price', 'highest_price', 'sellers_amount', 'screen_size', 'memory_size', 'battery_size', 'release_date']

  ⇨ ['model_name', 'best_price', 'lowest_price', 'highest_price', 'brand_name', 'os', 'popularity', 'screen_size', 'memory_size', 'battery_size', 'release_date']

3. 설계 내용

위의 데이터 셋들을 활용해 구현한 프로젝트의 설계 내용에 대해 설명하고자 한다. 설계 과정에서 UML을 작성하였으며 해당 UML을 통해 프로그램의 전체적인 동작 방식 및 내용 이해에 도움이 될 것으로 여겨져 첨부하였다. (제출물 중 2. data 및 결과 내용 설명) 해당 uml을 참고하며 보고서를 읽는다면 해당 프로젝트에 대해 더 깊은 이해를 할 수 있을 것으로 기대된다.

각 클래스는 클래스 명, 멤버 변수, 메서드로 구성되어 있다. 각 이름 앞에 -, #, +기호들이 함께 작성되어 있으며 이는 각각 순서대로 private, protected, public을 의미한다. 클래스 간의 관계에 따라 연관 관계(association), 일반화 관계(generalization), 집약 관계(aggregation), 합성 관계(composition), 의존 관계(dependency), 실체화 관계(realization) 등을 갖는다. 해당 관계들은 각각의 서로 다른 종류의 선으로 나타나며 선의 종류에 대한 자세한 설명은 생략하도록 한다.

이제 위의 uml을 바탕으로 각각의 class의 구조 및 관계를 자세히 살펴보며 설계 내용 및 방법에 대한 설명을 이어가고자 한다. 각 class의 멤버 변수 및 메서드를 중심으로 설명하고자 하며 자세한 설명이 필요 없는 메서드의 경우 설명을 생략하였다. 헤더 파일과 아래의 설명을 함께 읽으면 이해에 더 많은 도움이 될 것으로 예상된다.

- PRODUCT class

  PRODUCT class 및 이를 상속받는 자식 class들에서 사용되는 변수들의 타입을 typedef를 사용해 기존 데이터형의 이름을 새롭게 부여하였다. 이를 통해 코드의 가독성을 높이고 형변환의 가능성에 유연하게 대처할 수 있다.

  `virtual ~PRODUCT()`: 추상 class의 경우 소멸자를 virtual로 선언하지 않으면 자식 class의 객체 소멸 시 해당 부모 클래스의 소멸자만 실행된다. 소멸자를 virtual로 선언함으로써 자식 class의 소멸자까지 정상적으로 실행될 수 있도록 하였다.

  `ProductName get_name()`, `ProductId get_id()`, `ProductPrice get_price()`: 멤버 변수가 protected로 선언되어 있으므로 외부에서 PRODUCT class의 멤버 변수에 접근하기 위해 getter 함수를 선언하였다. 따라서 해당 class는 자신이 보여주고 싶은 방식으로 본인의 정보를 외부에 공개할 수 있다. 이를 통해 캡슐화가 잘 지켜질 수 있다.

  멤버 변수: PRODUCT class의 경우 서로 다른 상품 군의 공통적인 정보만을 추출해 추상화 시켜놓은 abstract class이다. 모든 상품이 공통으로 지녀야 할 정보는 상품 식별 id(pid), 상

품명(pname), 상품가격(pprice), 상품 타입(ptype)으로 해당 정보들을 추상화된 PRODUCT class의 멤버 변수로 정의하였다. 자식 class에서도 해당 변수들에 접근할 수 있게 하기 위해 protected로 선언하였다.

`virtual ProductType get_type() = 0`: PRODUCT class를 상속받는 자식 클래스들의 경우 상품 타입(ptype)에 의해 class의 종류가 결정된다. 각 자식 클래스마다 자신의 타입을 알맞게 나타낼 수 있어야 한다. 따라서 각 자식 클래스에 맞는 타입을 반환하게 하기 위해서 get_type()을 virtual로 선언하였으며 해당 함수를 통해 각각의 자식 클래스들은 자신의 type을 외부로 드러낼 수 있다.

`virtual void info_print() const = 0`: PRODUCT class를 상속받는 자식 클래스들의 경우 자신들의 정보를 자신들만의 방 식대로 출력할 수 있어야 한다. 따라서 자식들이 자신의 정보를 출력하는 행동을 강제하기 위해 info_print()를 virtual로 선언하였고 자식 class들은 모두 해당 함수를 선언 및 정의하고 있어야 한다. 그 결과 PRODUCT class를 상속받는 자식 클래스들은 모두 자신의 정보를 출력할 수 있으며, 출력의 내용 및 방법은 각각의 클래스마다 다르다.

`virtual void initialize_properties(InitialVlaues) = 0`: PRODUCT class를 상속받는 자식 클래스들의 경우 자신의 멤버 변수를 초기화 할 수 있어야 한다. 따라서 initialize_properties()를 virtual로 선언하였고 자식 class들은 모두 해당 함수를 선언 및 정의하고 있어야 한다. 그 결과 PRODUCT class를 상속받는 자식 클래스들은 모두 자신의 멤버 변수를 초기화할 수 있으며, 초기화 내용 및 방법은 각각의 클래스마다 다르다. initialize_properties()는 생성자 내부에서 유용하게 사용될 수 있다. 생성자의 오버로딩이 필요한 경우 해당 함수를 각각의 생성자 내부에 사용함으로써 코드 상의 반복되는 내용을 줄일 수 있다.

해당 프로젝트 내의 가장 중요하며 설계의 중심이라고 할 수 있는 상품의 클래스인 PRODUCT class의 내부 구조에 대해 자세히 살펴 보았다. 이제 해당 class를 상속받는 자식 class인 FURNITURE class, GROCERY class, PHONE class를 순차적으로 살펴보며 각 class의 특징에 대해 파악해보고자 한다.

- FURNITURE class

  멤버 변수: FURNITURE class의 경우 앞서 설명한 **IKEA Furniture** 데이터셋의 정보를 담는 객체이다. 따라서 추상화된 변수 이외에도 해당 데이터의 고유의 특징을 담기 위한 변수들이 존재하며 ft_category, ft_designer, ft_description, ft_sellable_online, ft_volume이 이에 해당된다. 여기서 ft_volume의 type은 VOLUME class다. VOLUME class는 **IKEA Furniture**의 depth, width, height 칼럼 정보를 받아와 각각을 곱하여 volume을 계산 한 후, 해당 데이터들을 하나로 묶어 저장 및 관리한다. VOLUME class에 대한 설명은 후술하도록 한다.

  `static ProductIntId ft_last_id`: ft_last_id는 static으로 선언되어 있어 해당 클래스 자체

의 변수이다. 즉 해당 클래스로 생성된 모든 인스턴스들이 공유하는 변수임을 의미한다. 생성되는 인스턴스들 각각을 구분할 수 있게 해주는 id로, 인스턴스가 생성됨에 따라 순차적으로 증가한다.

`static ProductType ptype`: FURNITURE class의 경우 ptype은 항상 furniture이기 때문에 이는 초기화 값이 선언되어 있으며 바뀌지 않는다.

`FURNITURE()`, `FURNITURE(InitialVlaues)`: 생성자가 FURNITURE(), FURNITURE(InitialVlaues) 두 가지로 오버로딩 되어있다. 전자의 경우 모든 멤버 변수가 0또는 ""값 등으로 초기화 된다. 후자의 경우 파일에서 읽어온 정보를 바탕으로 멤버 변수들을 초기화 시킨다. 넘겨주는 인자를 다르게 함으로써 필요한 생성자를 상황에 맞게 호출하여 사용할 수 있다. 소멸자의 경우 virtual로 선언되어 소멸시 자식 class 자신의 소멸자까지 정상적으로 호출 되도록 한다.

PRODUCT class에서 virtual로 선언된 함수들은 구현이 강제되기 때문에 info_print(), initialize_properties(), get_type() 함수들이 선언 및 정의되어 있다.

각 멤버 변수에 대한 setter함수가 선언되어 있다. 외부 파일에서 값을 읽어와 해당 데이터를 바탕으로 객체를 생성해야 하므로 본 class의 멤버 변수에 접근할 수 있는 setter함수가 필요하며 이는 initialize_properties() 내부에서 사용된다. set_ft_volume()은 VOLUME class를 생성하여 이를 멤버 변수에 할당하는 함수이다. 특히 `set_ft_description(Value description)`의 경우 파일에서 읽어온 정보를 furniture class 내부 구조에 맞게 변형시켜서 정보를 받아온다.

- GROCERY class

  멤버 변수: GROCERY class의 경우 앞서 설명한 **DMart Products** 데이터셋의 정보를 담는 객체이다. 따라서 추상화된 변수 이외에도 해당 데이터의 고유의 특징을 담기 위한 변수들이 존재하며 gr_discounted_price, gr_discount_rate, gr_category, gr_brand, gr_quantity, gr_description이 이에 해당된다. gr_discount_rate는 파일에서 받아오는 정보가 아닌 파일에서 받아온 price와 discounted 칼럼 정보를 바탕으로 계산된 할인율을 저장하는 변수이다. 이는 calculate_gr_discount_rate()함수를 사용하여 계산된다.

  `static ProductIntId gr_last_id`: gr_last_id는 static으로 선언되어 있어 해당 클래스 자체의 변수이다. 즉 해당 클래스로 생성된 모든 인스턴스들이 공유하는 변수임을 의미한다. 생성되는 인스턴스들 각각을 구분할 수 있게 해주는 id로, 인스턴스가 생성됨에 따라 순차적으로 증가한다.

  `static ProductType ptype`: GROCERY class의 경우 ptype은 항상 grocery이기 때문에 이는 초기화 값이 선언되어 있으며 바뀌지 않는다.

  `GROCERY()`, `GROCERY(InitialVlaues)`: 생성자가 GROCERY(), GROCERY (InitialVlaues) 두 가지

로 오버로딩 되어있다. 전자의 경우 모든 멤버 변수가 0또는 ""값 등으로 초기화 된다. 후자의 경우 파일에서 읽어온 정보를 바탕으로 멤버 변수들을 초기화 시킨다. 넘겨주는 인자를 다르게 함으로써 필요한 생성자를 상황에 맞게 호출하여 사용할 수 있다. 소멸자의 경우 virtual로 선언되어 소멸시 자식 class 자신의 소멸자까지 정상적으로 호출 되도록 한다.

PRODUCT class에서 virtual로 선언된 함수들은 구현이 강제되기 때문에 info_print(), initialize_properties(), get_type() 함수들이 선언 및 정의되어 있다.

각 멤버 변수에 대한 setter함수가 선언되어 있다. 외부 파일에서 값을 읽어와 해당 데이터를 바탕으로 객체를 생성해야 하므로 본 class의 멤버 변수에 접근할 수 있는 setter함수가 필요하며 이는 initialize_properties() 내부에서 사용된다.

- PHONE class

  멤버 변수: PHONE class의 경우 앞서 설명한 **Mobile Phones Data** 데이터셋의 정보를 담는 객체이다. 따라서 추상화된 변수 이외에도 해당 데이터의 고유의 특징을 담기 위한 변수들이 존재하며 ph_lowest_price, ph_highest_price, ph_brand, ph_os, ph_popularity, ph_size, release_date, release_period가 이에 해당된다. 여기서 ph_size의 type은 SIZE class로 **Mobile Phones Data**의 screen_size, memory_size, battery_size 칼럼 정보를 받아와 해당 데이터들을 하나로 묶어 저장 및 관리한다. SIZE class에 대한 설명은 후술하도록 한다. 또한 release_period는 파일에서 받아오는 정보가 아닌 파일에서 받아온 release_date칼럼 정보를 바탕으로 오늘날짜로부터 출시된 날짜까지의 기간을 저장하는 변수이다. 이는 calculate_release_period()함수를 사용하여 계산된다.

  `static ProductIntId ph_last_id`: ph_last_id는 static으로 선언되어 있어 해당 클래스 자체의 변수이다. 즉 해당 클래스로 생성된 모든 인스턴스들이 공유하는 변수임을 의미한다. 생성되는 인스턴스들 각각을 구분할 수 있게 해주는 id로, 인스턴스가 생성됨에 따라 순차적으로 증가한다.

  `static ProductType ptype`: PHONE class의 경우 ptype은 항상 phone이기 때문에 이는 초기화 값이 선언되어 있으며 바뀌지 않는다.

  `PHONE()`, `PHONE(InitialVlaues)`: 생성자가 PHONE(), PHONE(InitialVlaues) 두 가지로 오버로딩 되어있다. 전자의 경우 모든 멤버 변수가 0또는 ""값 등으로 초기화 된다. 후자의 경우 파일에서 읽어온 정보를 바탕으로 멤버 변수들을 초기화 시킨다. 넘겨주는 인자를 다르게 함으로써 필요한 생성자를 상황에 맞게 호출하여 사용할 수 있다. 소멸자의 경우 virtual로 선언되어 소멸시 자식 class 자신의 소멸자까지 정상적으로 호출 되도록 한다.
  PHONE class에서 virtual로 선언된 함수들은 구현이 강제되기 때문에 info_print(), initialize_properties(), get_type() 함수들이 선언 및 정의되어 있다.

각 멤버 변수에 대한 setter함수가 선언되어 있다. 외부 파일에서 값을 읽어와 해당 데이터를 바탕으로 객체를 생성해야 하므로 본 class의 멤버 변수에 접근할 수 있는 setter함수가 필요하며 이는 initialize_properties() 내부에서 사용된다. set_ph_size()는 SIZE class를 생성하여 이를 멤버 변수에 할당하는 함수이다.

지금까지 PRODUCT class를 상속받는 3개의 자식 class들에 대해 살펴보았다. 각각의 구조는 대체적으로 비슷하며 고유의 특징을 담고있는 변수 및 그와 관련된 함수에서 차이가 나는 것을 알 수 있다. 이제 해당 자식 class들 내부에서 사용되었던 VOLUME class와 SIZE class에 대해 살펴보도록 한다

- VOLUME class

  VOLUME class는 FURNITURE class 내부에서 생성 및 사용된다. depth, width, height 정보를 바탕으로 volume을 계산하고 이를 저장한다. volume의 계산은 calculate_volume()을 사용한다.

  `VOLUME()`, `VOLUME(Volume, Volume, Volume)`: 생성자가 VOLUME(), VOLUME(Volume, Volume, Volume) 두 가지로 오버로딩 되어있다. 전자의 경우 모든 멤버 변수가 0또는 ""값 등으로 초기화 된다. 후자의 depth, width, height 정보를 받아 멤버 변수들을 초기화 시킨다. 넘겨주는 인자를 다르게 함으로써 필요한 생성자를 상황에 맞게 호출하여 사용할 수 있다.
  `void info_print() const`: info_print()를 사용하여 volume에 관한 정보를 해당 class가 정의한 방법대로 출력할 수 있다.

- SIZE class

  SIZE class는 PHONE class 내부에서 생성 및 사용된다. screen_size, memory_size, battery_size 정보를 받아와 이를 저장한다.

  `SIZE()`, `SIZE(Size, Size, Size)`: 생성자가 SIZE(), SIZE(Size, Size, Size) 두 가지로 오버로딩 되어있다. 전자의 경우 모든 멤버 변수가 0또는 ""값 등으로 초기화 된다. 후자의 경우 screen_size, memory_size, battery_size 정보를 받아 멤버 변수들을 초기화 시킨다. 넘겨주는 인자를 다르게 함으로써 필요한 생성자를 상황에 맞게 호출하여 사용할 수 있다.
  `void info_print() const`: info_print()를 사용하여 size에 관한 정보를 해당 class가 정의한 방법대로 출력할 수 있다.

위에서 제품과 관련된 class 및 그 내부에서 사용되는 class의 구조에 대해 상세히 살펴보았다. 그러나 해당 class로 생성된 수 많은 객체들이 프로그램 내부에서 흩어져 있으면 프로그램의 작동 및 동작 관리가 매우 힘들기에 제품들이 한 공간에 모여 관리될 필요가 있다. 따라서 제품들을 저장 및 관리할 수 있는 class가 필요하고 판단이 되었다. 다라서 해당 기능을 수행하는 CLASS를 설계하였고, 지금부터 프로그램 내부의 제품들을 모두 모아 관리하는

PRODUCT_MENU class에 대해 살펴보도록 한다.

- PRODUCT_MENU class

  ProductMenu pmenu: PRODUCT_MENU class 내부에 vector<PRODUCT*>형의 pmenu라는 멤버 변수가 있다. Vector(Queue)는 동적 배열 구조를 C++로 구현한 것으로 맨 끝에서만 삽입 및 삭제가 일어나는 구조이다. 해당 프로젝트 내에서 상품의 삭제는 일어나지 않으며 컨테이너 맨 끝에만 상품 객체가 계속 추가되므로 vector의 사용이 효율적일 것이라 판단했다. 또한 추후에 삭제가 가능하도록 변경이 된다고 하더라도 객체의 추가 및 탐색이 훨씬 빈번하게 일어나기 때문에 삭제에 대한 효율을 가장 낮게 고려하였다.

  또한 동적으로 크기가 변하고 메모리가 연속적이기 때문에 자동으로 배열의 크기를 조절할 수 있고 유연하게 객체의 추가 및 삭제가 가능하다. 메모리가 연속적이기 때문에 다른 STL구조인 deque, list에 비해 개별 원소에 대한 접근 속도가 빠르다. 해당 프로젝트 내에서 vector에 담기는 product* 객체의 수가 매우 많으며 vector에서의 search가 매우 빈번히 일어나기에 같은 STL구조 중 vector가 최적의 컨테이너라고 판단했다.

  PRODUCT_MENU(): 생성자에서는 pmenu를 사용하기 전에 내부 원소들을 초기화해주었다. 소멸자의 경우 pmenu내부에 담기는 자료형이 PRODUCT*형이며 이는 모두 new를 사용하여 동적으로 할당된 메모리이기 때문에 내부 PRODUCT객체들을 모두 delete해주어야 메모리 누수가 발생하지 않는다. vector의 iterator를 사용하여 모든 내부 원소들에 접근해서 delete를 완료한 후 소멸할 수 있도록 설계했다.

  void add_menu(PRODUCT*): add_menu()의 경우 동적으로 할당된 PRODUCT*형의 product객체를 pmenu의 맨 뒤에 추가한다.

  void show_menu(): show_menu()의 경우 랜덤으로 pmenu에 담긴 제품을 랜덤으로 20개 추출하여 화면에 출력한다. 현실에서는 모든 메뉴를 이용자가 모두 볼 수 있어야 하겠지만, 본 프로젝트에서는 시각적인 편의성을 위해 프로그램 이용자가 원하는 상품의 개수를 입력 받아 그 개수만큼만 상품 정보가 출력되는 것으로 제한을 두었다. 추후에 더 많은 상품이 화면에 보여야한다고 판단이 되거나 혹은 전체 상품이 출력되는게 더 좋다고 판단이 된다면 해당 함수의 수정을 통해 얼마든지 해결할 수 있다.

  void print_info(ProductId): print_info()의 경우 pmenu내부에 전달받은 ProductId와 일치하는 상품이 존재하면 해당 상품의 정보를 출력한다. 상품이 존재하지 않을 경우 존재하지 않는 상품이라는 메시지를 출력한다.

  bool search_product(ProductId): search_product()의 경우 pmenu내부에 전달받은 ProductId와 일치하는 상품이 존재하면 true, 존재하지 않으면 false를 반환한다. 이는 제품의 정보를 출력할 때 해당 제품의 존재 여부를 판단할 때 사용 된다.

`PRODUCT* get_product(ProductId)`: get_product()의 경우 search_product()로 해당 제품이 존재한다고 판단 되었을 때만 사용되며, ProductId와 일치하는 제품이 존재하고 있을 시에만 해당 함수가 동작하도록 설계했다.

`unsigned int get_total_count()`: get_total_count()의 경우 현재 pmenu에 담겨있는 제품의 전체 수량을 반환한다. pmenu는 private으로 해당 클래스의 내부에서만 접근가능하도록 설계했기에 외부에서 pmenu에 직접 접근할 수 없다. 따라서 pmenu의 size를 반환하는 함수를 만들어 우회적으로 접근할 수 있도록 했다.

`void run()`: 마지막으로 run()함수는 프로그램의 동작을 실행시키는 함수이다. 이를 main()에 구현할 수도 있겠지만 외부에서 해당 class의 동작을 관리하기 보다 class내부에 이를 구현하여 동작을 관리하면 추후에 코드 수정 및 관리가 용이하다고 판단이 되어 멤버 함수로 구현했다. 상품 조회 및 프로그램 종료 기능을 가지고 있다.

위에서 PRODUCT_MENU class의 멤버 변수 및 함수에 대해 상세하게 다루었다. 프로그램의 전체 동작을 관리하는 중요한 class이며 해당 class의 구조 변화에 따라 프로그램의 성능 및 전체적인 기능이 크게 좌우될 수 있다. 따라서 효율성 및 변경가능성을 최대한 고려하여 설계할 수 있도록 했다.

이제 해당 class를 멤버 변수로 가지고 있으며 외부 파일의 내용을 바탕으로 상품들을 생성하여 pmenu에 넣어주는 class인 READ_FILE class에 대해 알아보고자 한다.

- READ_FILE class

  READ_FILE class는 외부에서 파일의 내용을 한 줄씩 읽어 해당 내용을 바탕으로 적절한 type의 객체를 생성하며, 해당 객체를 프로젝트내의 제품이 모여있는 컨테이너에 할당한다. 이는 main()함수에서 구현이 되어도 이상할 것 없는 기능이지만 코드의 유지보수 및 관리를 편하게 하고 파일의 추가 라는 확장 가능성에 대해 고려한 결과 해당 class가 존재하는 것의 이점을 더 높이 평가 하여 설계에 반영했다.

  외부에서 READ_FILE class를 사용할 때에는 Product_create()라는 멤버 함수 만을 사용한다. 파일의 추가 라는 확장 가능성을 고려하여 캡슐화가 잘 이루어지게 하였다. 만약 다른 파일이 추가되어 새로운 타입의 객체의 생성이 필요하더라도 내부에 멤버 함수를 추가하여 Product_create()함수 내의 코드만 수정해주면 된다. 이 덕분에 기존 코드를 변경하지 않으면서(Close) 기능을 추가(Open)할 수 있도록 설계가 되어야 한다는 oop의 기본 설계 원칙인 개방 폐쇄 원칙이 잘 지켜지고 있음을 판단할 수 있다.

지금까지 해당 프로그램에서 설계한 모든 class의 내부 구조에 대해 살펴보았다. 최대한 OOP의 기본 컨셉을 지키고자 노력하였으며 OOP의 5대 설계 원칙인 SOLID 원칙 또한 고려하며 설계하였다.

이제 위에서 살펴본 class들간의 상호작용이 어떻게 일어나며 상호작용의 효과에 대해 순차적으로 알아보도록 한다.

4. 프로그램 동작 및 결과

지금부터는 프로그램의 동작 방식을 중심으로 및 결과 화면에 대해 살펴보고자 한다. 해당 프로그램은 메뉴판에 나타나있는 상품 정보를 바탕으로 상품을 검색해 상세정보를 출력한다.

우선, main함수에서 READ_FILE class를 통해 만들어진 read_file 객체를 통해 3가지 데이터셋을 읽는다. 읽은 파일 내용을 바탕으로 함수 내부에서 각각의 타입에 맞는 객체를 생성한다. 이때 main() 함수 내에서는 파일 이름과 파일의 유형만 read_file 객체에게 넘겨주기만 하면 된다. 외부에서 함수를 구별 할 필요 없다. reae_file내부에서 파일 타입을 구분하고 그에 맞는 행동을 한다. 생성된 객체들은 동적 할당된 메모리 영역인 heap에 존재하며 product_menu에 저장된다.

프로그램이 실행되면 시작되었다는 메시지와 함께 총 몇 개의 상품이 존재하는지 알려준다. 데이터셋의 크기가 크고, 상품의 종류가 많기 때문에 화면에 전체 메뉴를 다 출력할 수 없다고 판단하였다. 따라서 사용자의 선택에 따라 메뉴판에 보여질 상품의 개수를 입력 받는다. 당연히 총 상품의 개수보다는 작은 숫자가 입력되어야 한다.

입력된 숫자만큼 메뉴가 랜덤으로 중복없이 메뉴판에 출력된다. 해당 프로그램으로 실행할 수 있는 동작은 '1. 상품 조회', '2. 프로그램 종료' 두 가지 이다.

'1. 상품 조회'를 선택하면 조회할 상품의 id를 입력하라는 메시지가 출력된다. 메뉴판을 참고하여 자세한 정보를 조회하고 싶은 상품의 id를 입력하면, id, type, name, price, category 등 FURNITURE, GROCERY, PHONE 각각의 객체에 알맞은 정보가 출력된다.

1) 존재하지 않는 상품의 id를 입력 할 경우 다시 상품의 id를 입력한다.

2) 0을 입력할 경우 새로운 메뉴판이 제공된다. 새롭게 제공된 메뉴판의 정보를 바탕으로 새로운 상품의 id를 검색할 수 있다.

동작 선택 화면에서 '2. 프로그램 종료'를 선택하면 프로그램이 종료되었다는 메시지와 함께 정상적으로 프로그램이 종료된다.

각 상품 객체들은 동적 할당된 메모리 공간에 위치하기 때문에 프로그램이 종료될 때 PRODUCT_MENU의 소멸자 내부에서 해당된 메모리를 delete시켜준다. 조금 더 확실한 확인을 위해 Visual Studio 디버거와 CRT(C 런타임 라이브러리)를 사용하여 메모리 누수를 감지하고 있다. 메모리의 누수 없이 프로그램이 정상적으로 종료되는 것을 확인할 수 있다.

5. 프로그램 해석 및 발전 방향

지금까지 <product information print system> 프로그램에 사용된 data set, class, 프로그램 동작 방식 등에 상세하게 살펴보았다. 이제 프로그램에 대한 전체적인 평가를 하며 본 프로그램에 대한 논의를 마무리 해보고자 한다. 우선 프로그램의 OOP의 설계적 관점을 기반으로 확장 가능성과 개선 방향에 대해 논의해보도록 한다.

프로그램의 목적이자 핵심 기능은 다양한 상품을 한 공간에 모아 관리하는 것이다. 이를 염두에 두고 프로그램의 확장 가능성에 대해 살펴보았을 때, 충분히 높을 확장가능성을 지녔다고 할 수 있다. 현재 존재하는 상품 종류인 furniture, grocery, phone 뿐만 아니라 음료수, 전자기기, 의복 등 다양한 타입의 상품이 추가되더라도 다른 코드 상의 변동은 전혀 없이 PRODUCT 추상 class를 상속받는 class만 추가되면 된다. 따라서 얼마든지 여러가지 종류의 상품을 한 공간에 모아 관리할 수 있으며 확장 가능성 또한 충분하다고 볼 수 있다. 이를 OOP 설계적 관점에서 살펴보면 개방-폐쇄 원칙이 매우 잘 치켜지고 있음을 알 수 있다. 확장에는 열려있고 수정에는 닫혀 있기 때문이다.

FIRNITURE 객체 내부에 VOLUME 이라는 객체가 존재한다. FURNITURE 객체가 할 수 있는 행동을 VOLUME class를 생성함으로써 volume에 관련된 책임을 전가했다고 볼 수 있다. PHONE 객체와 SIZE 객체도 마찬가지로 SIZE class를 생성함으로써 size와 관련된 책임을 모두 SIZE class에게 전가했다고 볼 수 있다. 이를 OOP 설계적 관점에서 살펴보면 각각의 객체는 하나의 책임만 가지고 있어야 한다는 단일 책임 원칙이 잘 지켜지고 있음을 알 수 있다. 추후에 volume이나 size와 관련된 행동을 해야하는 class가 추가된다면 해당 class를 사용할 수도 있을 것이다.

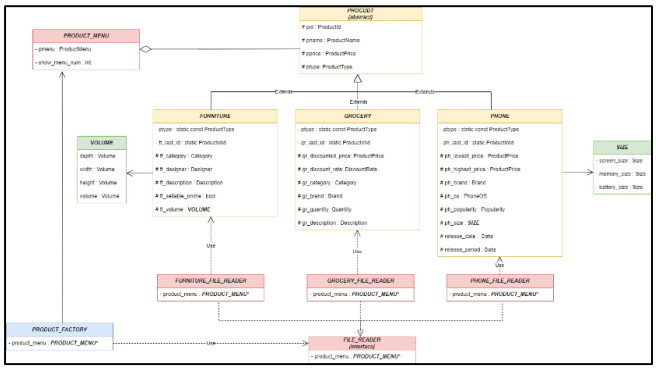
PRODUCT_MENU class는 PRODUCT*를 담고 있는 벡터로 상품을 관리한다. 이때, 추상 클래스 내부에서 virtual로 선언되어 있는 함수와 그냥 선언 되어있는 함수의 작동 방식이 다를 것이다. virtual 함수는 자식 class들에게 구현을 강제하였기에 각각의 자식들의 행동의 방법은 다르더라도 행동의 결과는 같을 것이다. 따라서 PRODUT의 종류가 무엇인지 PROCUDT_MENU class는 전혀 알 필요 없다. 추상클래스를 상속받은 객체들이 가지고 있는 행동들을 실행만 시켜주면 각 class들이 알아서 기대되는 행동을 할 것이다. 이를 OOP 설계적 관점에서 보면 부모 객체와 이를 상속한 자식 객체가 있을 때 부모 객체를 호출하는 동작에서 자식 객체가 부모 객체를 완전히 대체할 수 있다는 원칙인 리스코프치환원칙이 잘 지켜지고 있음을 알 수 있다. info_print(), show_short_info() 등 부모 class에게 기대되는 행동을 자식 class인 furniture, grocery, phone 객체들이 완전히 대체할 수 있기 때문이다.

또한 PRODUT_MENU class는 내부에 run()이라는 함수를 가지고 있다. 이는 사실 main()함수에서 작동되어도 전혀 이상할 것 없는 함수이다. 그러나 class 내부에 해당 함수를 정의함으로써 객체의 속성과 행위를 하나로 묶어 실제 구현 내용 일부를 감춰 내부에 은닉할 수 있다. class 내부에 관련 있는 정보들을 하나로 묶어 관리할 수 있다. 즉, 캡슐화가 매우 잘 이루어

지고 있으며 이를 통한 정보 은닉이 잘 작동되고 있음을 의미한다.

READ_FILE class는 파일데이터를 읽어와 해당 내용을 바탕으로 적절한 타입의 객체를 생성한다. 파일데이터를 읽는 행동을 main()함수 내에서 하지 않고 해당 행동을 담당하는 하나의 class를 만든 것이다. 이렇게 함으로써 file이 추가되거나 file의 수정이 필요할 때 더 쉽고 유연한 유지 보수가 가능하다는 장점이 있다.

그러나 READ_FILE class를 지금처럼 하나의 class가 아닌 인터페이스로 만들어 파일의 종류에 따라 해당 인터페이스를 상속 받는 설계 방법도 좋을 것 같다고 생각이 들었다. 각 class는 단 하나의 파일을 만드는 책임만 지게 되고, 이는 단일 책임 원칙을 고려하였을 때에 더 적합한 방법이라고 생각된다. 해당 방식으로 설계 방향을 바꿔 uml diagram을 간략하게 다시 그려 보았다. 멤버 변수와 함수의 자세한 메서드의 작성은 제외 하였고 객체간의 관계를 중심으로 참고하면 좋을 것 같다.



6. 결론

프로그램의 성능을 더 개발하고 많은 기능이 확장된다면 훨씬 복잡한 설계 구조를 갖게 될 것이다. 새로운 상품이 추가되거나 원하는 성능이 확장되어도 최대한 코드의 수정을 최소화 할 수 있도록 설계했다.

그 과정에서 데이터를 추상화(*abstraction*)하여 추상 클래스 및 그것을 상속받는 자식 class들을 구현하였다. 상품id, 상품name, 상품price 등 새로운 상품 종류가 생성 되더라도 상품이라면 응당 지녀야 할 공통된 특성들을 모아 추상화 하였고 차별화 되는 특성들을 각각의 class에게 할당했다. 이것을 상속(*inheritance*)이라 하며 상속을 통해 더 많은 상품 타입이 해당 프로그램에 입력되어도 코드 상의 큰 변동 없이 추가가 가능하게 만든다.

변수와 해당 변수의 operation은 항상 같이 정의되어 있어야 한다. 이것을 *encapsulation*이라 하며 class라는 구조를 통해 이를 코드 상으로 표현하였다.

하나의 class에서 정의된 함수나 변수의 경우 상속받은 자식 class 혹은 자신의 class 내부에서 오버라이딩 및 오버리딩의등 다형성(*polymorphism*)을 구현하여 외부에서 바라봤을 때는 아무런 차이가 없어 보이는 이름이 동일한 함수나 변수라도 상황과 타입에 맞게 적절한 연산을 수행하거나 적절한 변수를 저장할 수 있게 하였다.

이와 같이 oop의 기본 컵셉과 설계 원칙을 최대한 고려하며 코드를 구현하였으며 이는 프로

그램의 효율성 증대 및 확장 가능성의 증대라는 큰 이점을 제공한다.

지금까지 해당 보고서에서 데이터 읽기에서 출력까지 가능한 디지털 꾸러미를 만드는 과정에 대한 소개를 했다. <product information print system>을 개발을 통해 원하는 상품의 정보를 시스템 상에 저장하고 관리할 수 있게 되었으며 저장된 상품의 정보를 간단하고 빠르게 검색할 수 있다. 기능의 추가가 조금 더 이루어 진다면 훨씬 완성도 있는 프로그램이 될 것으로 예상된다.