**2022 Spring OOP Assignment Report**

과제 번호 : 2

학번 : 20210054

이름 : 정하우

Povis ID : howru0321

**명예서약 (Honor Code)**

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

프로그램을 하다 보면 결정해야 할 세부 사항이 많은데, 이러한 세부 사항을 처리한 방법과 이유를 보고서에 쓰십시오.

문제가 여러 개인 경우, 각 문제별로 정리해서 작성합니다.

각 문항별 설명은 편의를 위한 것으로, 삭제하고 제출한다.

1. **프로그램 개요**
   * 문제 1에서 구현한 Vector class의 형식을 이용해서 실제 피자 배달 상황을 구현하는 class를 구현하였다. 문제 1에서는 float 형의 배열에 적용되는 class라면, 여기서 사용되는 Vector\_Customer과 Vector\_Rider class는 우리가 임의로 만든 class 인 Rider Customer class 배열을 대상으로 만든 class이다. 사용한 class 로는 Pizza class, Customer class, Rider class, Vector\_Customer class, Vector\_Rider class가 있다.
2. **프로그램의 구조 및 알고리즘**

<Vector\_Customer>

private:

* + - Customer\* arr : 배열
    - int len : 데이터 개수
    - int msize : 할당된 메모리 사이즈

public:

* + - typedef Customer\* iterator : Customer\*를 iterator 형으로 선언.
    - typedef const Customer\* const\_iterator : const Customer\*를 const\_iterator 형으로 선언.
    - Vector\_Customer() : 기본 생성자. 모든 멤버 변수를 0 혹은 NULL 로 초기화한다.
    - Vector\_Customer(int size) : 데이터 크기와 메모리 할당량은 size 로 초기화. size 개수만큼 0 으로 초기화된 데이터 저장.
    - Vector\_Customer(int size, const Customer& init) : Customer init 으로 초기화된 데이터를 저장. 데이터 크기와 메모리 할당량은 size 로 초기화.
    - Vector\_Customer(int size, const Customer\* init) : Customer 배열 init 으로 초기화된 데이터 저장. 데이터 크기와 메모리 할당량은 size 로 초기화.
    - Vector\_Customer(const Vector\_Customer& v) : 복사 생성자.
    - ~Vector\_Customer() : 소멸자. 할당된 메모리를 삭제한다.
    - int capacity() const : 데이터에 할당된 메모리 사이즈를 리턴하는 함수.
    - int size() const : 데이터 개수를 리턴하는 함수.
    - iterator begin() : 데이터의 맨 처음 주소를 리턴하는 함수.
    - const\_iterator begin() const : 데이터의 맨 처음 주소를 리턴하는 함수. (const 버전)
    - iterator end() : 데이터의 맨 마지막 주소를 리턴하는 함수.
    - const\_iterator end() const : 데이터의 맨 마지막 주소를 리턴하는 함수. (const 버전)
    - Customer& front() : 데이터의 맨 처음 데이터를 리턴하는 함수.
    - const Customer& front() const : 데이터의 맨 처음 데이터를 리턴하는 함수.(const 버전)
    - Customer& back() : 데이터의 맨 마지막 데이터를 리턴함수.
    - const Customer& back() const : 데이터의 맨 마지막 데이터를 리턴하는 함수. (const 버전)
    - void allocate(int capacity) : 데이터에 할당된 메모리를 capacity 만큼 동적 할당한다. 재할당할때는 사용하지 않으며, 모든 데이터를 0으로 초기화한다.
    - void resize(int size) : size 만큼 데이터에 할당된 메모리를 동적 할당하며, 데이터 개수는 size 만큼 갖는다. 재할당할때 사용하며, 기존 데이터를 가지고 나머지 빈 부분은 0으로 초기화한다. 데이터 개수 보다 작은 크기로 재할당하면 범위 밖 부분은 날라간다. 데이터 개수를 size로 맞추었지만, 이는 실제로 사용했을 때 데이터 개수는 데이터 개수보다 적은 범위로 재할당하는 경우를 제외하고 기존 데이터 개수와 동일하므로, 이 함수를 사용할 때 데이터 개수는 다시 돌려놓는 알고리즘을 항상 적용했다.
    - void push\_back(const Customer& value) : 데이터의 맨 뒤에 새로운 데이터 value 를 추가한다. 2.문제 세부 조건 규칙에 따라 메모리를 동적 할당하는 함수
    - void pop\_back() : 맨 마지막 데이터를 삭제하는 함수. 이때 메모리 할당은 유지한다.
    - void insert(iterator position, const Customer& value) : 데이터의 특정 위치(position)에 value 를 삽입하는 함수. 2.문제 세부 조건 규칙에 따라 메모리를 동적 할당한다.
    - void insert(iterator position, const Customer\* first, const Customer\* last) : : 데이터에서 특정 위치(position)에 Customer array 를 삽입하는 함수. 이때 first, last 는 각각 Customer array 의 맨 처음 주소, 맨 마지막 주소이다. 2.문제 세부 조건 규칙에 따라 메모리를 동적 할당한다.
    - void erase(iterator position) : 데이터에서 특정 위치(position)의 값을 삭제하는 함수. 이때 메모리 할당은 유지한다.
    - void clear() : 할당된 메모리 사이즈, 데이터 개수를 0 으로 설정하며, 데이터를 NULL 로 설정하는 함수. 이때 재할당하지 않는다. 실제 Vector\_Customer class에 있는 clean()함수 역시 재할당하지 않는다.
    - Customer& operator[](int index) : 연산자 ‘[]’를 오버로딩한다. 데이터에서 index 위치에 해당하는 값을 리턴한다.
    - const Customer& operator[](int index) const : 연산자 ‘[]’를 오버로딩한다. 데이터에서 index 위치에 해당하는 값을 리턴한다. (const 버전)
    - Vector\_Customer& operator=(const Vector\_Customer& v) : 연산자 ‘=’를 오버로딩한다. 현재 멤버변수를 입력 Vector\_Customer 의 멤버변수로 치환한다.

<Vector\_Rider>

private:

* + - Rider\* arr : 배열
    - int len : 데이터 개수
    - int msize : 할당된 메모리 사이즈

public:

* + - typedef Rider\* iterator : Rider\*를 iterator 형으로 선언.
    - typedef const Rider\* const\_iterator : const Rider\*를 const\_iterator 형으로 선언.
    - Vector\_Rider() : 기본 생성자. 모든 멤버 변수를 0 혹은 NULL 로 초기화한다.
    - Vector\_Rider(int size) : 데이터 크기와 메모리 할당량은 size 로 초기화. size 개수만큼 0 으로 초기화된 데이터 저장.
    - Vector\_Rider(int size, const Rider& init) : Rider init 으로 초기화된 데이터를 저장. 데이터 크기와 메모리 할당량은 size 로 초기화.
    - Vector\_Rider(int size, const Rider\* init) : Rider 배열 init 으로 초기화된 데이터 저장. 데이터 크기와 메모리 할당량은 size 로 초기화.
    - Vector\_Rider(const Vector\_Rider& v) : 복사 생성자.
    - ~Vector\_Rider() : 소멸자. 할당된 메모리를 삭제한다.
    - int capacity() const : 데이터에 할당된 메모리 사이즈를 리턴하는 함수.
    - int size() const : 데이터 개수를 리턴하는 함수.
    - iterator begin() : 데이터의 맨 처음 주소를 리턴하는 함수.
    - const\_iterator begin() const : 데이터의 맨 처음 주소를 리턴하는 함수. (const 버전)
    - iterator end() : 데이터의 맨 마지막 주소를 리턴하는 함수.
    - const\_iterator end() const : 데이터의 맨 마지막 주소를 리턴하는 함수. (const 버전)
    - Rider& front() : 데이터의 맨 처음 데이터를 리턴하는 함수.
    - const Rider& front() const : 데이터의 맨 처음 데이터를 리턴하는 함수.(const 버전)
    - Rider& back() : 데이터의 맨 마지막 데이터를 리턴함수.
    - const Rider& back() const : 데이터의 맨 마지막 데이터를 리턴하는 함수. (const 버전)
    - void allocate(int capacity) : 데이터에 할당된 메모리를 capacity 만큼 동적 할당한다. 재할당할때는 사용하지 않으며, 모든 데이터를 0으로 초기화한다.
    - void resize(int size) : size 만큼 데이터에 할당된 메모리를 동적 할당하며, 데이터 개수는 size 만큼 갖는다. 재할당할때 사용하며, 기존 데이터를 가지고 나머지 빈 부분은 0으로 초기화한다. 데이터 개수 보다 작은 크기로 재할당하면 범위 밖 부분은 날라간다. 데이터 개수를 size로 맞추었지만, 이는 실제로 사용했을 때 데이터 개수는 데이터 개수보다 적은 범위로 재할당하는 경우를 제외하고 기존 데이터 개수와 동일하므로, 이 함수를 사용할 때 데이터 개수는 다시 돌려놓는 알고리즘을 항상 적용했다.
    - void push\_back(const Rider& value) : 데이터의 맨 뒤에 새로운 데이터 value 를 추가한다. 2.문제 세부 조건 규칙에 따라 메모리를 동적 할당하는 함수
    - void pop\_back() : 맨 마지막 데이터를 삭제하는 함수. 이때 메모리 할당은 유지한다.
    - void insert(iterator position, const Rider& value) : 데이터의 특정 위치(position)에 value 를 삽입하는 함수. 2.문제 세부 조건 규칙에 따라 메모리를 동적 할당한다.
    - void insert(iterator position, const Rider\* first, const Rider\* last) : : 데이터에서 특정 위치(position)에 Rider array 를 삽입하는 함수. 이때 first, last 는 각각 Rider array 의 맨 처음 주소, 맨 마지막 주소이다. 2.문제 세부 조건 규칙에 따라 메모리를 동적 할당한다.
    - void erase(iterator position) : 데이터에서 특정 위치(position)의 값을 삭제하는 함수. 이때 메모리 할당은 유지한다.
    - void clear() : 할당된 메모리 사이즈, 데이터 개수를 0 으로 설정하며, 데이터를 NULL 로 설정하는 함수. 이때 재할당하지 않는다. 실제 Vector\_Rider class에 있는 clean()함수 역시 재할당하지 않는다.
    - Rider& operator[](int index) : 연산자 ‘[]’를 오버로딩한다. 데이터에서 index 위치에 해당하는 값을 리턴한다.
    - const Rider& operator[](int index) const : 연산자 ‘[]’를 오버로딩한다. 데이터에서 index 위치에 해당하는 값을 리턴한다. (const 버전)
    - Vector\_Rider& operator=(const Vector\_Rider& v) : 연산자 ‘=’를 오버로딩한다. 현재 멤버변수를 입력 Vector\_Rider 의 멤버변수로 치환한다.

<Rider>

private:

float money : 배달 기사가 가진 돈

float earnMoney : 배달 기사가 번 돈의 총 합산

float time : 피자 배달하는데 걸린 총 시간

friend class Pizza;

public:

Rider() : 기본 생성자.money, earnMoney, time 모두 0 으로 초기화한다.

Rider(float money, float earnMoney, float time) : 멤버 변수 money, earnMomey, time 을 입력 값으로 설정한다.

Rider(const Rider& rider) : 복사 생성자.

~Rider() : 소멸자.함수 내에 별다른 구현은 필요 없습니다.

float reportMoney() const : 배달 기사가 현재까지 번 돈의 총 합산을 리턴한다.

float reportTime() const : 배달 기사가 현재까지 피자를 배달하는데 걸린 총 시간을 리턴한다.

void deliver(float price, float time) : 정해진 피자 가격만큼 돈을 받으며, 정해진 배달 시간만큼 총 배달 시간에 더해준다.

friend ostream& operator <<(ostream& os, const Rider& x) : << 오버로딩

<Customer>

private:

float money : 고객이 현재 가진 돈

float paidMoney : 고객이 지불한 돈의 총 합산

float time : 고객이 피자 배달 받는데 걸린 총 시간

friend class Pizza;

public:

Customer() : 기본 생성자.money, paidMoney, time 각각 1000, 0, 0 으로 초기화한다.

Customer(float money, float paidMoney, float time) : 멤버 변수 money, paidMoney, time 을 입력 값으로 설정한다.

Customer(const Customer& customer) : 복사 생성자.

~Customer() : 소멸자.

float reportMoney() const : 고객이 현재까지 지불한 돈의 총 합을 리턴하는 함수.

float reportTime() const : 고객이 현재까지 피자를 배달 받는데 걸린 총 시간을 리턴하는 함수.

void pay(float price, float time) : 피자 가게에 정해진 피자 가격만큼 돈을 지불하며, 정해진 배달 시간만큼 총 배달 시간에 더해주는 함수.

friend ostream& operator <<(ostream& os, const Customer& x) : << 오버로딩

<Pizza>

private:

float money : 가게의 돈

float priceA : A 구역의 피자 가격

float priceB : B 구역의 피자 가격

float feeA : A 구역 배달 수수료

float feeB : B 구역 배달 수수료

float timeA : A 구역 배달 소요 시간

float timeB : B 구역 배달 소요 시간

Vector\_Customer A : A 구역 고객들.Vector\_float class 사용.

Vector\_Customer B : B 구역 고객들.Vector\_float class 사용.

Vector\_Rider R : 배달 기사.Vector\_Rider class 사용.

public:

Pizza() : 기본 생성자

Pizza(float money) : 멤버 변수 money 를 입력으로 받아 설정하며, 나머지는 기본 생성자와 동일하다.

Pizza(const Pizza& pizza) : 복사 생성자. 모든 멤버 변수를 입력의 멤버 변수로 대체한다.

~Pizza() : 소멸자.

void setPrice() : A, B 구역의 피자 가격과 배달 수수료를 책정하는 함수.

void setPrice(float a, float b) : a, b 가 주어졌을 때 A, B 구역의 피자 가격과 배달 수수료를 책정하는 함수.

void hire(const Rider& rider) : 배달 기사 채용 기능을 구현하는 함수.

void hire(const Rider\* riders, int size) : 배달 기사 여러 명을 한 번에 채용하는 기능을 구현하는 함수.

void enroll(const Customer& customer, bool a) : 특정 지역의 고객을 추가하는 함수.

void enroll(const Customer\* customers, int size, bool a) : 특정 지역의 다수 고객들을 추가하는 기능을 구현하는 함수.

float reportMoney() const : 피자 가게가 현재 가지고 있는 돈을 리턴하는 함수.

int numRiders() const : 배달 기사의 현재 인원을 리턴한다.

int numCustomers(bool a) const : 고객수를 리턴하는 함수. Bool a 가 true 일 경우, A 지역 고객 수를 리턴한다. Bool a 가 false 일 경우, B 지역 고객수를 리턴한다.

float meanFee() const : 현재 고용 중인 라이더가 배달을 통해 번 돈의 평균을 리턴하는 함수.

float meanTimeRiders() const : 현재 고용 중인 라이더가 배달하는데 걸린 시간의 평균을 리턴하는 함수.

float meanMoneyCustomers() const : 현재 관리 중인 고객들이 피자를 구매하는데 든 비용의 평균을 리턴하는 함수.

float meanTimeCustomers() const : 현재 관리 중인 고객들이 배달 받는데 걸린 시간의 평균을 리턴하는 함수.

void deliver() : 각 구역의 고객들에게 일괄적으로 피자를 배달하는 함수.

friend ostream& operator <<(ostream& os, const Pizza& x) : 피자가가게가 보유한 돈, 구역별 고객 수, 피자 가격, 배달 수수료, 배달기사수를 콘솔에 출력하는 함수.

1. **토론 및 개선**
   * + 이번 문제는 문제1에서 잡힌 class개념을 응용해보면서 class개념을 확실하게 잡은 것 같다. 구현을 하는 도중에 c3646 알 수 없는 재정의 지정자입니다 라는 오류가 떴는데, 이는 Pizza.h에서 사용되는 Vector\_Rider.h 에 include “Pizza.h”를 선언해서 재정의가 되어서 이런 오류가 뜬 것 같다.
     + 전체적으로 class를 만드는 방법, class 맴버함수를 사용하는 방법이 많이 익숙해진 것 같다
     + Friend 라는 개념을 통해 private 값도 참조 가능하다는 사실을 알게 되었다.
     + 연산자 오버로드 개념이 확실히 잡힌 것 같다. 특히 입출력 연산자의 경우에는 맴버변수로는 할 수 없고 전역 변수로만 할 수 있다는 사실이 매우 흥미로웠다.
     + Template를 이용하면 Rider과 Customer class 자료형을 하나로 만들 수 있기 때문에 굳히 Vector\_Customer과 Vector\_Rider class 둘다 만들지 않고 하나의 class로 해결할 수 있어 구현하는데 훨신 편했을것같다.
2. **참고 문헌**

* <https://ansohxxn.github.io/cpp/chapter9-3/> - 입출력 연산자 오버로딩
* <https://choryeonworkshop.tistory.com/130> - 연산자 오버로딩과 프랜드
* <https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=kks227&logNo=60205596757> – 입출력 연산자 오버로딩