

CSED 232 Object-Oriented Programming (Spring 2022)

Programming Assignment # 2

- Classes & objects -

Due date : 4 월 8 일 23 시 59 분

담당 조교 : 진원준

문제 공통 조건

- **cstdio, iostream, fstream** 을 제외한 라이브러리의 사용은 허용하지 않습니다. 위 라이브러리에 포함된 모든 함수는 사용 가능합니다.
- 각 클래스마다 **header file** 과 **cpp file** 을 나누어 구현하여야 합니다. 추가적으로 필요한 **멤버 함수(method)**나 **멤버 변수(member variable)**가 있을 경우, 추가해도 무방합니다.
- 문제에 명시되어 있지 않더라도 각 클래스마다 생성자(Constructor), 소멸자(Destructor)는 필수입니다. 기본 생성자의 경우, 별다른 언급이 없다면 멤버 변수에 int, float 은 0, string 은 "", pointer 는 NULL 로 초기화해 주십시오.
- 각 클래스마다 **클래스명과 파일명을 통일**하여 주시기 바랍니다. (대소문자 유의)
- **Template 제외**, C 언어 및 C++언어의 **모든 문법 사용** 가능합니다.
- 모든 클래스의 멤버 변수는 별다른 언급이 없을 경우 **private** 으로 선언하여야 합니다.
- 각 문제 별 추가적인 세부 조건 또한 만족하여야 합니다.
- **채점은 문제의 각 기능이 잘 동작하는지 확인하는 방식으로 진행하며, 어떤 기능을 확인할지는 공개하지 않습니다.** 수강생은 각 문제에 대해 구현한 클래스의 **header file** 과 **cpp file** 을 제출하시면 됩니다.
- 문제 조건이 복잡합니다. 질문 전, 모든 문제의 세부 조건을 꼼꼼히 읽어 보시기 바랍니다.

감점

- 제출 기한에서 하루(24 시간) 늦을 때마다 20%씩 감점
1 일 이내: 20% 감점, 2 일 이내: 40% 감점, 4 일 이상 지연: 0 점
- 컴파일이 정상적으로 이루어지지 않을 경우 0 점

제출방식

채점은 **Windows Visual Studio 2022 환경에서 이루어집니다.** Linux, macOS 사용자에게 양해를 구합니다. 파일을 업로드하실 때, 작업하신 환경이 있는 **프로젝트 폴더에서 디버그 폴더를 삭제한 후**

그대로 압축해서 올려 주시기 바랍니다. 폴더명은 문제#_학번으로 만들어 주십시오. 또한 문제 폴더 안에 각 문제에 해당하는 Report 도 같이 넣어서 zip 파일로 만든 후 제출해 주시기 바랍니다. 이때, 문제마다 따로 프로젝트를 생성하고, 따로 압축하여 제출해 주시기 바랍니다. 즉, **총 2 개의 파일을 제출**하셔야 합니다. 제출은 반드시 PLMS 를 통해 제출해주시기 바랍니다. 이메일 제출은 인정되지 않습니다. 4 일이 지날 경우 0 점이므로 과제 제출 마감일로부터 4 일 후인 4 월 12 일 23 시 59 분 이후엔 PLMS 를 통해 제출하실 수 없습니다.

예) prob1_20219999.zip, prob2_20219999.zip

공통 채점 기준

1. 프로그램 기능

- 프로그램이 요구 사항을 모두 만족하면서 올바르게 실행되는가?

2. 프로그램 설계 및 구현

- 각 클래스가 header file 과 cpp file 로 나누어 졌는가?
- 요구 사항을 만족하기 위한 변수 및 알고리즘 설계가 잘 되었는가?
- 문제에서 제시된 세부 조건을 모두 만족하였는가?
- 설계된 내용이 요구된 언어를 이용하여 적절히 구현되었는가?

3. 프로그램 가독성

- 프로그램이 읽기 쉽고 이해하기 쉽게 작성되었는가?
- 변수 명이 무엇을 의미하는지 이해하기 쉬운가?
- 프로그램의 소스 코드를 이해하기 쉽도록 주석을 잘 붙였는가?

4. 보고서 구성 및 내용, 양식

- 보고서는 적절한 내용으로 이해하기 쉽고 보기 좋게 잘 작성되었는가?
- 보고서의 양식을 잘 따랐는가?

다른 사람의 프로그램이나 인터넷에 있는 프로그램을 복사(copy)하거나 간단히 수정해서 제출하면 학점은 무조건 'F'가 됩니다. 이러한 부정행위가 발견되면 학과에서 정한 기준에 따라 추가적인 불이익이 있을 수 있습니다.

문제 1 번 (배점 : 50 점)

C++의 대표적인 STL 중 하나인 **vector** 의 **float 버전**을 구현한다. 아래 설명을 읽고 float 데이터를 저장하는 **Vector class** 를 구현하라. ("**V**ector"에서 **빨간색은 대문자**이다.)

- 참고: STL vector document (<https://www.cplusplus.com/reference/vector/vector/>)

1. Vector class

1) 멤버 변수

Private:

- 기능 구현에 필요한 private 멤버 변수는 직접 설계한다.

Public:

- typedef float* iterator : float*를 iterator 형으로 선언.
- typedef const float* const_iterator : const float*를 const_iterator 형으로 선언.

2) 멤버 함수

- Vector()

: 기본 생성자. 모든 멤버 변수를 0 혹은 NULL 로 초기화한다.

- Vector(int size)

: 데이터 크기와 메모리 할당량은 size 로 초기화. size 개수만큼 0 으로 초기화된 데이터 저장.

- Vector(int size, const float& init)

: float init 으로 초기화된 데이터를 저장. 데이터 크기와 메모리 할당량은 size 로 초기화.

- Vector(int size, const float* init)

: float 배열 init 으로 초기화된 데이터 저장. 데이터 크기와 메모리 할당량은 size 로 초기화.

- Vector(const Vector& v)

: 복사 생성자.

- ~Vector()

: 소멸자. 할당된 메모리를 삭제한다.

- int capacity() const

: 데이터에 할당된 메모리 사이즈를 리턴한다.

- int size() const

: 데이터 개수를 리턴한다.

- iterator begin()

: 데이터의 맨 처음 주소를 리턴한다.

- const_iterator begin() const

: 데이터의 맨 처음 주소를 리턴한다. (const 버전)

- iterator end()

: 데이터의 맨 마지막 주소를 리턴한다.

- const_iterator end() const

: 데이터의 맨 마지막 주소를 리턴한다. (const 버전)

- float& front()

: 데이터의 맨 처음 데이터를 리턴한다.

- const float& front() const

: 데이터의 맨 처음 데이터를 리턴한다. (const 버전)

- float& back()

: 데이터의 맨 마지막 데이터를 리턴한다.

- const float& back() const

: 데이터의 맨 마지막 데이터를 리턴한다. (const 버전)

- void allocate(int capacity)

: 데이터에 할당된 메모리를 capacity 만큼 동적 할당한다.

- void resize(int size)

: size 만큼 데이터에 할당된 메모리를 동적 할당하며, 데이터 개수는 size 만큼 갖는다.

- void push_back(const float& value)

: 데이터의 맨 뒤에 새로운 데이터 value 를 추가한다. **2.문제 세부 조건** 규칙에 따라 메모리를 동적 할당한다.

- void pop_back()

: 맨 마지막 데이터를 삭제한다. 이때 **메모리 할당은 유지**한다.

- void insert(iterator position, const float& value)

: 데이터의 특정 위치(position)에 value 를 삽입한다. **2.문제 세부 조건** 규칙에 따라 메모리를 동적 할당한다.

- void insert(iterator position, const float* first, const float* last)

: 데이터에서 특정 위치(position)에 float array 를 삽입한다. 이때 first, last 는 각각 float 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 삭제한다. 이때 **메모리 할당은 유지**한다.

- void clear()

: 할당된 메모리 사이즈, 데이터 개수를 0 으로 설정하며, 데이터를 NULL 로 설정한다.

- float& operator[](int index)

: 연산자 '['를 오버로딩한다. 데이터에서 index 위치에 해당하는 값을 리턴한다.

- const float& operator[](int index) const

: 연산자 '['를 오버로딩한다. 데이터에서 index 위치에 해당하는 값을 리턴한다. (const 버전)

- Vector& operator=(const Vector& v)

: 연산자 '='를 오버로딩한다. 현재 멤버변수를 입력 Vector 의 멤버변수로 치환한다.

2. 문제 세부 조건

- 1) 참고 자료 STL vector 에는 존재하나, 과제의 Vector class 에는 존재하지 않는 멤버변수 및 멤버함수도 있다.
- 2) 메모리 할당 외 모든 기능은 STL vector 와 동일하다. **메모리 할당 규칙은 과제의 설명을 따른다.**
- 3) **“메모리 할당” 규칙은 다음과 같다.**
 - **“최종 데이터 개수”**는 기존에 저장된 데이터 개수와 추가되는 데이터 개수의 합산이다.
 - **“할당된 메모리 사이즈”**는 저장할 수 있는 데이터의 개수이다.
 - 데이터 추가 시, 최종 데이터 개수가 현재 할당된 메모리 사이즈보다 크거나 같을 경우, 메모리를 다시 할당한 후 데이터를 추가한다. 메모리 할당 규칙은 아래와 같다.
 - 최종 데이터 개수가 5 보다 작을 경우, 최종 데이터 개수만큼의 메모리를 할당한다.
 - 최종 데이터 개수가 5 보다 크거나 같을 경우, 최종 데이터 개수의 2 배만큼의 메모리를 할당한다.
 - **아래는 메모리 할당 규칙에 대한 예시이다.**
 - (현재 데이터 개수)=0, (할당된 메모리 사이즈)=0 일때, push_back 을 통해 1 개의 float 을 추가할 경우, 메모리를 1 만큼 할당해야 한다.
 - (현재 데이터 개수)=0, (할당된 메모리 사이즈)=1 일때, push_back 을 통해 1 개의 float 을 추가할 경우, 메모리를 추가로 할당할 필요 없다.
 - (현재 데이터 개수)=5, (할당된 메모리 사이즈)=7 일때, insert 를 통해 2 개의 float 이 저장된 array 를 삽입할 경우, 메모리를 7 만큼 추가 할당해야 한다. (총 14 만큼 할당. $14 = (5+2) \times 7$)
 - (현재 데이터 개수)=5, (할당된 메모리 사이즈)=9 일때, insert 를 통해 3 개의 float 이 저장된 array 를 삽입할 경우, 메모리를 추가 할당할 필요 없다.
- 4) 수강생이 구현한 Vector class 의 멤버 변수명과 멤버 함수명은 PDF 와 동일해야 한다. 다를 경우 해당 기능 및 구현에 대한 평가는 0 점 처리한다. (대소문자 유의)
- 5) 기능 확인을 위한 main.cpp 를 첨부파일로 제공한다. 수강생이 직접 main.cpp 에 코딩하여 구현한 class 의 추가 기능을 확인해도 된다. (제공한 main.cpp 는 채점에 사용되지 않음)
- 6) 그 외 Vector class 구현에 필요한 조건은 추가로 구현하여도 된다. 하지만 이에 대한 추가 점수는 부여하지 않는다.
- 7) 콘솔 출력 시 공백, 반올림 등 출력 조건은 고려하지 않아도 된다.

3. 채점 세부 기준

- 세부 조건 따로 언급 없는 경우, page 2 의 채점 기준과 동일함.

1) 프로그램 기능 – 40%

- 구현한 Vector class 의 모든 기능 확인

2) 프로그램 설계 및 구현 – 40%

- 생성자 및 소멸자 – (5%)
- 메모리 할당 – (10%)
- 데이터 접근 및 관리 – (20%)
- 연산자 오버로딩 – (5%)

3) 프로그램 가독성 – 10%

4) 보고서 구성 및 내용, 양식 – 10%

4. main.cpp

```
#include "Vector.h"
#define num 33

void print_vec(const Vector& vec)
{
    cout << "-----" << endl;
    cout << "Print (Data size, Allocation size)" << endl;
    cout << "(" << vec.size() << ", " << vec.capacity() << ")WnWn";
    cout << "Print Data" << endl;
    for (auto it = begin(vec); it != end(vec); ++it) cout << *it << "Wt";
    cout << endl;
}

int main()
{
    cout << "-----" << endl;
    cout << "1. init vector" << endl;
    float tmp[] = { 3.2, 2.5, 2.11 };
    Vector vec1;
    Vector vec2(sizeof(tmp) / sizeof(*tmp), tmp);
    Vector vec3(vec2);
    print_vec(vec1);
    print_vec(vec2);
    print_vec(vec3);
    cout << endl;

    cout << "-----" << endl;
    cout << "2. push_back" << endl;
    float push = 3.763;
    vec1.push_back(push);
    vec2.push_back(push);

    print_vec(vec1);
    print_vec(vec2);
    cout << endl;

    cout << "-----" << endl;
    cout << "3. insert" << endl;
    float insrt = -5.8775;
    vec1.insert(vec1.begin(), insrt);

    float insrt_arr[] = { 3.2105, 2.51, -3.70, 3.14 };
    vec1.insert(vec1.begin(), insrt_arr, insrt_arr + sizeof(insrt_arr) /
sizeof(*insrt_arr));
    print_vec(vec1);
    cout << endl;

    cout << "-----" << endl;
    cout << "4. pop_back" << endl;
    vec1.pop_back();
    print_vec(vec1);
    cout << endl;
```



```

cout << "===== " << endl;
cout << "5. erase" << endl;
vec1.erase(vec1.begin() + 3);
print_vec(vec1);
cout << endl;

cout << "===== " << endl;
cout << "6. resize" << endl;
vec1.resize(3);
print_vec(vec1);
cout << endl;

cout << "===== " << endl;
cout << "7. operator []" << endl;
for (int i = 0; i < 3; i++)
    cout << vec1[i] << "Wt ";
cout << endl;

cout << "===== " << endl;
cout << "8. operator =" << endl;
Vector vec5;
vec5 = vec1;
print_vec(vec1);
print_vec(vec5);
cout << endl;

cout << "===== " << endl;
cout << "9. clear" << endl;
vec1.clear();
print_vec(vec1);
cout << endl;

cout << "===== " << endl;
cout << "10. allocate and insert" << endl;
float pi[num] = { 0, };
for (int i = 0; i < num; i++)
    pi[i] = 3.14;
vec1.allocate(int(num * 1.5));
vec1.insert(vec1.begin(), pi, pi + num);
print_vec(vec1);
cout << endl;

return 0;

```

```

}

```

문제 2 번 (배점 50 점)

효자동에 피자 가게를 새로 개업하려고 한다. 가게는 배달만 취급하며 배달 지역은 A 구역과 B 구역으로 나뉜다. 아래 설명을 읽고 **Pizza class, Customer class, Rider class, Vector_Customer class, Vector_Rider class** 를 구현하라.

■ 문제 공통 조건: 아래 조건은 문제 2 의 모든 상황에 적용되니 유의하시기 바랍니다.

- A 구역과 B 구역 고객의 합은 항상 10 명으로 유지하며, 배달기사의 수는 항상 5 명으로 유지한다.
- 각 구역 고객의 수는 0 명 이하 혹은 10 명 이상이 될 수 없다.
- 각 구역 고객 및 배달 기사의 추가, 제거는 항상 index 기준 선입선출의 원칙을 따른다. 즉, 데이터를 제거해야 할 경우, index 상에서 앞에 있는 것을 먼저 삭제한다.
- 가격에 대한 언급이 없다면, **Pizza class** 에서 배달 지역 별 가격 책정은 각 구역의 고객 수가 달라질 때마다 밑의 규칙을 따른다.
 - **priceA** : $[100 - 3 \times \text{A 구역 고객 수}]$
 - **priceB** : $[100 - 6 \times \text{B 구역 고객 수}]$
 - **feeA** : $[10 - \text{A 구역 고객 수}]$
 - **feeB** : $[10 - \text{B 구역 고객 수}]$

1. Pizza class

1) 멤버 변수

- 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_Customer** class 사용.
- Vector_Customer B : B 구역 고객들. **Vector_Customer** class 사용.
- Vector_Rider R : 배달 기사. **Vector_Rider** class 사용.

2) 멤버 함수

- Pizza() : 기본 생성자.

Money, timeA, timeB 는 각각 10000, 10, 20 으로 설정한다.

A 와 **B** 는 각각 기본 생성자로 생성된 **Customer** 객체를 5 개씩 갖는다.

R 은 기본 생성자로 생성된 **Rider** 객체를 5 개 갖는다.

priceA, priceB, feeA, feeB 는 객체의 **setPrice()** 멤버 함수를 통해 일괄 설정한다.

(입력 인자가 없는 setPrice() 함수)

- 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. 함수 내에 별다른 구현은 필요 없습니다.

- void setPrice() :

A, B 구역의 피자 가격과 배달 수수료를 **문제 공통 조건**과 같이 책정한다.

- void setPrice(float a, float b) :

a, b 가 주어졌을 때 A, B 구역의 피자 가격과 배달 수수료를 아래와 같이 책정한다.

priceA [100 - a x A 구역 고객 수] (A 구역에 7 명 고객이 있으면 100 - a * 7),

priceB [100 - b x B 구역 고객 수] (B 구역에 3 명 고객이 있으면 100 - b * 3),

feeA [10 - A 구역 고객 수],

feeB [10 - B 구역 고객 수]

- void hire(const Rider& rider) :

배달 기사 채용 기능을 구현한다. 배달 기사 관리 원칙은 선입선출을 따른다.

- void hire(const Rider* riders, int size) :

배달 기사 여러 명을 한 번에 채용하는 기능을 구현한다. Rider 객체 배열을 입력으로 받으며, size 는 배열의 객체 개수이다. 배달 기사 관리 원칙은 선입선출을 따른다. 즉 새로운 3 명의 배달 기사를 채용할 경우 가장 먼저 고용된 3 명의 배달 기사를 해고한다.

- void enroll(const Customer& customer, bool a) :

특정 지역의 고객을 추가한다. 외부에서 선언된 Customer class 를 입력으로 받는다. Bool a 가 true 이면 A 지역에 추가하며, false 이면 B 지역에 추가한다. 고객 관리 원칙은 **문제 공통 조건**을 따른다.

- void enroll(const Customer* customers, int size, bool a) :

특정 지역의 다수 고객들을 추가하는 기능을 구현한다. Customer 객체 배열을 입력으로 받으며, size 는 배열의 객체 개수이다. Bool a 가 true 이면 A 지역에 추가하며, false 이면 B 지역에 추가한다. 고객 관리 원칙은 **문제 공통 조건**을 따른다.

- float reportMoney() const :

피자 가게가 현재 가지고 있는 돈을 리턴한다.

- int numRiders() const :

배달 기사들의 현재 인원을 리턴한다.

- int numCustomers(bool a) const :

Bool a 가 true 일 경우, A 지역 고객 수를 리턴한다. Bool a 가 false 일 경우, B 지역 고객 수를 리턴한다.

- float meanFee() const :

현재 고용 중인 라이더가 배달을 통해 번 돈의 평균을 리턴한다.

- float meanTimeRiders() const :

현재 고용 중인 라이더가 배달하는데 걸린 시간의 평균을 리턴한다.

- float meanMoneyCustomers() const :

현재 관리 중인 고객들이 피자를 구매하는데 든 비용의 평균을 리턴한다.

- float meanTimeCustomers() const :

현재 관리 중인 고객들이 배달 받는데 걸린 시간의 평균을 리턴한다.

- void deliver() :

각 구역의 고객들에게 일괄적으로 피자를 배달한다. (배달 시각, 라이더 배치 시간 겹침 등은 고려 X)
고객은 가게에 돈을 지불한다. 고객의 돈은 소속된 구역의 피자 가격만큼 빠져나가며, 가게의 돈은 그만큼 증가한다. 또한, **고객의 총 배달시간은 소속된 구역의 배달 시간만큼 증가**한다. (Customer class 참고)

가게는 배달 기사에게 수수료를 지불한다. 가게의 돈은 각 배달하는 구역의 배달 수수료만큼 빠져나가며, 배달 기사의 돈은 그만큼 증가한다. 또한, 배달 기사의 총 배달시간은 배달하는 구역의 배달 시간만큼 증가한다. (Rider class 참고)

피자 배달 및 라이더 배치 원칙은 다음과 같다.

피자 배달은 A 구역을 먼저 끝낸 후, B 구역에 배달하는 것을 원칙으로 한다. 각 구역 내 고객 사이에서 피자 배달 우선순위는 Vector_Customer 에서 index 가 낮을수록 먼저 배달하는 것을 원칙으로 한다. 라이더 배치 우선순위 또한 Vector_Rider 에서 index 가 낮을수록 먼저 배치한다.

피자 배달 예시

A, B 구역에 각각 3, 7 명의 고객이 있을 경우, R[0], R[1], R[2]가 A 구역에 배달을 한 번씩 진행하고, R[3], R[4]가 B 구역에 배달을 한 번씩 진행한다. B 구역에 남은 5 명의 고객들에겐 모든 Rider 가 각각 한 번씩 배달하면 피자 배달이 완료된다. (R 은 Vector_Rider class 를 의미)

A, B 구역에 각각 7, 3 명의 고객이 있을 경우, 모든 Rider 가 A 구역에 배달을 한 번씩 진행하고, R[0], R[1]이 A 구역에 한 번 더 배달을 진행한다. 나머지 Rider 들은 B 구역의 3 명에 고객에게 배달을 진행한다.

- friend ostream& operator <<(ostream& os, const Pizza& x) :

콘솔 창에 아래와 같이 출력되도록 오버로딩을 통해 구현한다. 꺾쇠 괄호([])는 변수를 의미한다. (꺾쇠 '[', ']' 는 출력할 필요 없음)

- | |
|---|
| <ul style="list-style-type: none">1) Money : [피자가게가 보유한 돈]2) (Customers, Price, Fee)<ul style="list-style-type: none">- A : ([A 구역 고객 수], [A 구역 피자 가격], [A 구역 배달 수수료])- B : ([B 구역 고객 수], [B 구역 피자 가격], [B 구역 배달 수수료])3) Riders: [배달 기사 수] |
|---|

2. Customer class

1) 멤버 변수

- float money : 고객이 현재 가진 돈
- float paidMoney : 고객이 지불한 돈의 총 합산
- float time: 고객이 피자 배달 받는데 걸린 총 시간

2) 멤버 함수

- 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. 함수 내에 별다른 구현은 필요 없습니다.

- float reportMoney() const :

고객이 현재까지 지불한 돈의 총 합을 리턴한다.

- float reportTime() const :

고객이 현재까지 피자를 배달 받는데 걸린 총 시간을 리턴한다.

- void pay(float price, float time) :

피자 가게에 정해진 피자 가격만큼 돈을 지불하며, 정해진 배달 시간만큼 총 배달 시간에 더해준다.

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

콘솔 창에 아래와 같이 출력되도록 오버로딩을 통해 구현한다. 꺾쇠 괄호([])는 변수를 의미한다. (꺾쇠 '[', ']' 는 출력할 필요 없음)

(money, paidMoney, time) : ([money], [paidMoney], [time])

3. Rider class

1) 멤버 변수 (3 개)

- float money : 배달 기사가 가진 돈
- float earnMoney : 배달 기사가 번 돈의 총 합산
- float time: 피자 배달하는데 걸린 총 시간

2) 멤버 함수 (7 개)

- Rider() :

기본 생성자. money, earnMoney, time 모두 0 으로 초기화한다.

- Rider(float money, float earnMoney, float time) :

멤버 변수 money, earnMoney, 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 가격**만큼 돈을 받으며, 정해진 배달 시간만큼 총 배달 시간에 더해준다.

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

콘솔 창에 아래와 같이 출력되도록 오버로딩을 통해 구현한다. 꺾쇠 괄호([])는 변수를 의미한다. (꺾쇠 '[', ']' 는 출력할 필요 없음)

(money, earnMoney, time) : ([money], [earnMoney], [time])

4. Vector_Customer & Vector_Rider class

- 아래 설명에서 **[class]** 는 Customer class 혹은 Rider class 를 의미한다.
(Vector_**[class]**는 Vector_Customer 혹은 Vector_Rider class 를 의미)
- **Vector_C**ustomer, **Vector_R**ider 에서 **빨간색은 대문자**이다.
- Vector_Customer.h, Vector_Customer.cpp, Vector_Rider.h, Vector_Rider.cpp 네 개의 파일로 구현한다.
- 아래에서 설명할 class 는 문제 1 번의 Vector class 와 모든 기능이 동일하다. 자세한 설명은 문제 1 번을 참고한다. **차이점은 저장할 자료형이 float 에서 각 [class]로 변경된 점이다.**
- Template 에 대한 설명을 검색해 본 후, Template 을 사용해 문제 2 번을 구현한다면 어떤 점이 좋을지 Report Form 에 자신의 의견을 작성하시오.
([https://en.wikipedia.org/wiki/Template_\(C%2B%2B\)](https://en.wikipedia.org/wiki/Template_(C%2B%2B)))

구현해야 하는 Vector_**[class]**의 멤버 변수 및 멤버 함수는 아래와 같다.

1) 멤버 변수

Private:

- 기능 구현에 필요한 private 멤버 변수는 직접 설계한다.

Public:

- typedef **[class]*** iterator
- typedef const **[class]*** const_iterator

2) 멤버 함수

- Vector_**[class]**()
- Vector_**[class]**(int size)
- Vector_**[class]**(int size, const **[class]**& init)
- Vector_**[class]**(int size, const **[class]*** init)
- Vector_**[class]**(const Vector_**[class]**& v)
- ~Vector_**[class]**()

- int capacity() const
- int size() const
- iterator begin()
- const_iterator begin() const
- iterator end()
- const_iterator end() const
- **[class]&** front()
- const **[class]&** front() const
- **[class]&** back()
- const **[class]&** back() const
- void allocate(int capacity)
- void resize(int size)
- void push_back(const **[class]&** value)
- void pop_back()
- void insert(iterator position, const **[class]&** value)
- void insert(iterator position, const **[class]*** first, const **[class]*** last)
- void erase(iterator position)
- void clear()
- **[class]&** operator[](int index)
- const **[class]&** operator[](int index) const
- Vector_**[class]&** operator=(const Vector_**[class]&** v)

5. 문제 세부 조건

- 1) 수강생이 구현한 Pizza, Rider, Customer, Vector_Customer, Vector_Rider class 의 멤버 변수명과 멤버 함수명은 PDF 와 동일해야 한다. 다를 경우 해당 기능 및 구현에 대한 평가는 0 점 처리한다. (대소문자 유의)
- 2) 문제 2 번에 대해 각 class 별로 header file 과 cpp file 을 작성하여 총 10 개의 코드 파일을 구현한다.
- 3) 기능 확인을 위한 main.cpp 를 첨부파일로 제공한다. 수강생이 직접 main.cpp 에 코딩하여 구현한 class 의 추가 기능을 확인해도 된다. (제공한 main.cpp 는 채점에 사용되지 않음)
- 4) 그 외 문제 2 번에 필요한 조건은 추가로 구현하여도 된다. 하지만 이에 대한 추가 점수는 부여하지 않는다.
- 5) 콘솔 출력 시 공백, 반올림 등의 조건은 고려하지 않아도 된다.

6. 채점 세부 기준

세부 조건 따로 언급 없는 경우, page 2 의 채점 기준과 동일함.

1) 프로그램 기능 – 40% (main.cpp 콘솔 출력 기준으로 채점)

- 구현한 class 의 모든 기능 확인

2) 프로그램 설계 및 구현 – 40% (코드 기준 채점)

- Pizza class – (20%)
- Rider class – (5%)
- Customer class – (5%)
- Vector_Rider class – (5%)
- Vector_Customer class – (5%)

3) 프로그램 가독성 – 10%

4) 보고서 구성 및 내용, 양식 – 10%

7. main.cpp

```
#include <iostream>
#include "Pizza.h"
#include "Customer.h"
#include "Rider.h"

using namespace std;

int main()
{
    cout << "===== " << endl;
    cout << "1. Start Pizza Store" << endl;
    Pizza pizza1;
    Pizza pizza2(50000);
    Pizza pizza3(pizza2);
    cout << pizza1 << endl;
    cout << pizza2 << endl;
    cout << pizza3 << endl;

    cout << "===== " << endl;
    cout << "2. Hire a rider & Enroll a Customer in A place " << endl;
    Rider rider1;
    Customer customer1;
    pizza1.hire(rider1);
    pizza1.enroll(customer1, true);
    cout << pizza1 << endl;

    cout << "===== " << endl;
    cout << "3. Deliver pizza to All Customers " << endl;
    pizza1.deliver();
    cout << pizza1 << endl;

    cout << "===== " << endl;
    cout << "4. Hire riders & Enroll customers in B place " << endl;
    Rider riders[] = { rider1, rider1, rider1 };
    Customer customers[] = { customer1, customer1 };
    pizza1.hire(riders, 3);
    pizza1.enroll(customers, 2, false);
    pizza1.deliver();
    cout << pizza1 << endl;

    cout << "===== " << endl;
    cout << "5. Set Price" << endl;
    pizza1.setPrice(4, 7);
    pizza1.deliver();
    cout << pizza1 << endl;

    cout << "===== " << endl;
    cout << "6. Check Riders" << endl;
    cout << "Mean money : " << pizza1.meanFee() << endl;
    cout << "Mean time : " << pizza1.meanTimeRiders() << endl << endl;

    cout << "===== " << endl;
    cout << "7. Check Customers" << endl;
```

```
    cout << "Mean money : " << pizza1.meanMoneyCustomers() << endl;  
    cout << "Mean time : " << pizza1.meanTimeCustomers() << endl << endl;  
  
    return 0;  
}
```