LAB 4

실습 질문 하는 곳

https://padlet.com/dcslabcp/0405-dzhnbgt9gp1wjvvn

오늘 실습할 내용

- 클래스
  - 멤버 변수, 멤버 함수
  - 접근 제어자 (Access modifier)
  - 생성자
  - 두 가지 방식의 인스턴스 생성
- typedef
- static 멤버 변수, static 멤버 함수
- .h 와 .cpp

Class

- 클래스는 객체 지향 프로그래밍의 가장 중요한 단위입니다.
- 객체(object) 는 클래스에 의해 구현된 대상을 의미합니다.

```
class class_name {
  access_specifier :
    member1;
  access_specifier:
    member2;
  ...
};
```

```
class Rectangle {
    int width = 5;
    int height = 5;
  public:
    void set_values (int a,int b) {
      width = a;
      height = b;
    };
    int area(){
      return width * height;
    };
};
```

Class – 멤버 변수와 멤버 함수

- 클래스 내부에서 선언된 변수 또는 함수를 멤버(member) 라고 합니다.

```
class DateClass {
  public:
    int m_year;
    int m_month;
    int m_day;
    void print() {
      std::cout << m_year << "/" << m_month << "/" << m_day;
    }
}
```

- 멤버 변수 : m_year, m_month, m_day
- 멤버 함수 : print()

Class – 멤버 함수의 정의

- 멤버 함수의 선언은 클래스 내부에서 하지만, 정의는 클래스 밖에서 할 수도 있습니다. 이때 :: 를 사용합니다.

```
class Rectangle {
    int width = 0, height = 0;
  public:
    void set_values (int x,int y) {
      width = x;
      height = y;
    };
    int area(){
      return width * height;
    };
};
```

```
class Rectangle {
  int width = 0, height = 0;
  public:
    void set_values(int,int);
    int area(void);
};

void Rectangle::set_values(int x, int y){
  width = x;
  height = y;
}

int Rectangle::area(){
  return width * height;
}
```

Class – 접근 제어자

- 접근 제어자는 클래스 멤버들의 접근 권한을 정의합니다.
- 기본 접근 제어자는 private 입니다.(아무것도 설정하지 않는 경우)
- Public : 어디에서나 접근할 수 있는 멤버입니다
- Protected : 같은 클래스의 다른 멤버, 혹은 하위 클래스의 멤버에서 접근이 가능합니다.
- Private : 해당 클래스 혹은 friend 클래스에서만 접근이 가능합니다.

```cpp
#include <iostream>
#include <string>

class Base {
  protected:
    std::string s = "Base";
}

class Derived : Base {
  public:
    void print() {
      std::cout << s << std::endl;
    }
}

int main() {
    Derived d;
    d.print() //Base
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

class PeopleA{
private:
    string name;
    int height;
    friend class PeopleB; //friend 클래스 선언
public:
    //생성자
    PeopleA(string name, int height) {
        this->name = name;
        this->height = height;
    }
};
```

```cpp
class PeopleB {
public:
    //friend 선언으로 인해 PeopleA의 private 멤버에 접근가능
    void info_A(PeopleA a) {
        cout << "이름 : " << a.name << endl;
        cout << "신장 : " << a.height << endl;
    }
};
```

Class – 생성자

- 일반적으로 생성자는 public에 선언!!

- Constructor body에서 = 연산자를 통하여 멤버 변수를 초기화할 수 있습니다.

```
class class_name {
  public:
    class_name(): member_initializer_list {
      // Constructor Body
    }
}
```

```
#include <iostream>

class Rectangle{
    int width, height;
  public:
    Rectangle(int x, int y) { width = x; height = y; }
    int area() { return width * height; }
};

int main() {
    Rectangle rect(3,5);
    std::cout << rect.area() << std::endl; //15
}
```

Class – 생성자 : Default Constructor

- 생성자를 따로 정의하지 않으면, 디폴트 생성자가 정의됩니다.

```cpp
#include <iostream>

class Rectangle{
    public :
    int width=5;
    int height=10;

   // Rectangle() {}  Default 생성자 생성
};

int main(){
    Rectangle rect;
     Rectangle rect2();
    std::cout<<rect.width<<std::endl;
    std::cout<<rect.height<<std::endl;
    std::cout<<rect2.width<<std::endl;
    std::cout<<rect2.height<<std::endl;
}
```

```
PS C:\vscodepractice\test_C++\helloworldcpp> ./rectangle.exe
5
10
PS C:\vscodepractice\test_C++\helloworldcpp> g++ rectangle.cpp -o rectangle

rectangle.cpp: In function 'int main()':
rectangle.cpp:16:22: error: request for member 'width' in 'rect2', which is
 of non-class type 'Rectangle()'
     std::cout<<rect2.width<<std::endl;
                      ^~~~~
rectangle.cpp:17:22: error: request for member 'height' in 'rect2', which i
s of non-class type 'Rectangle()'
     std::cout<<rect2.height<<std::endl;
                      ^~~~~~
PS C:\vscodepractice\test_C++\helloworldcpp>
```

Class – 생성자 : Member Initializer List

- 컴마로 구분된 리스트입니다.
- Constructor body 보다 먼저 처리됩니다.

```
class class_name {
  public:
    class_name(): member_initializer_list {
      // Constructor Body
    }
}
```

```
#include <iostream>

class Rectangle{
  public:
    int width, height, area;
    Rectangle(int x, int y) : width(x), height(y) {
      area = x * y;
    }
};

int main() {
    Rectangle rect(4,9);
    std::cout << rect.width << std::endl; //4
    std::cout << rect.height << std::endl; //9
    std::cout << rect.area << std::endl; //36
}
```

Width=x height=y

Class – 생성자 오버로딩

- 오버로딩 (Overloading)
  : 메서드의 이름은 같지만 매개변수의 개수 또는 타입이 다른 메서드를 정의하는 것

- 즉, 매개변수가 서로 다른 생성자를 두 개 이상 구현할 수 있다는 뜻입니다.

```cpp
#include <iostream>

class Rectangle{
  public:
    int width, height;
    Rectangle() {
      width = 5; height = 5;
    }
    Rectangle(int x, int y) {
      width = x; height = y;
    }
};

int main() {
    Rectangle rect1(4,9);
    std::cout << rect1.width << std::endl; //4
    std::cout << rect1.height << std::endl; //9
    Rectangle rect2;
    std::cout << rect2.width << std::endl; //5
    std::cout << rect2.height << std::endl; //5
}
```

Class – 인스턴스 생성

- Object Instantiation 인스턴스 생성은 두 가지 방식으로 가능 합니다.
  - 변수로 생성하기
  - 포인터로 생성하기
- 생성자를 사용하여 변수로 생성할 수 있습니다.
  - 변수로 생성하는 경우 . 을 사용하여 멤버에 접근합니다.

```cpp
#include <iostream>

class Rectangle{
  public:
    int width, height;
    int area() { return width * height; }
    Rectangle(int x, int y) {
      width = x; height = y;
    }
};

int main() {
    Rectangle rect(3,11);
    std::cout << rect.width << std::endl; //3
    std::cout << rect.height << std::endl; //11
    std::cout << rect.area() << std::endl; //33
}
```

Class – 인스턴스 생성

- new 키워드를 사용하면 포인터로 인스턴스를 생성할 수 있습니다.
- 포인터로 생성하는 경우 `->` 을 사용하여 멤버에 접근합니다.

```cpp
#include <iostream>

class Rectangle{
  public:
    int width, height;
    int area() { return width * height; }
    Rectangle(int x, int y) {
      width = x; height = y;
    }
};

int main() {
    Rectangle* rect;
    rect = new Rectangle(2,7);
    std::cout << rect->width << std::endl; //2
    std::cout << rect->height << std::endl; //7
    std::cout << rect->area() << std::endl; //14
}
```

typedef

- typedef 키워드를 사용하여 새로운 데이터 타입 이름을 정의합니다
- 실제로 새로운 데이터 클래스를 만드는 것이 아니라, 이미 존재하는 데이터 타입에 별명을 붙이는 것입니다

```
typedef type name;
```

```
typedef unsigned char Alphabet;

int main() {
    Alphabet b1, b2;
    b1 = 'c';
    b2 = 'o';
    std::cout << b1 << " " << b2 << std::endl;
    // c o
}
```

Static Members

- static 키워드는 클래스의 멤버가 클래스 인스턴스 하나하나가 아닌, 클래스 자체에 종속되게끔 합니다
- Static member 는 non-static member 함수 내부에서 초기화 될 수 없습니다.

```
class Worker{
  public:
    Worker(){
      std::cout << "Worker Id : " << total << std::endl;
      total++;
    }
    static int total;
};

int Worker::total = 0;

int main() {
    Worker w1,w2,w3;
    //Worker ID : 0
    //Worker ID : 1
    //Worker ID : 2
}
```

Static Member Functions

- static 함수는 static 변수와 마찬가지로 인스턴스 하나에 속하는 함수가 아니라 클래스에 속하는 함수입니다.
- 정적 함수 내부에서는 오직 정적 변수만 사용할 수 있습니다.
- 인스턴스에서 호출하거나, 클래스 이름에 :: 를 사용하여 호출할 수 있습니다.

```
class Counter {
  private:
    static int count;
  public:
    static void inc() {
      count++;
      std::cout << "Count: " << count << std::endl;
    }
};

int Counter::count = 0;

int main() {
    Counter c1,c2,c3;
    c1.inc(); //Count: 1
    c2.inc(); //Count: 2
    c3.inc(); //Count: 3
    Counter::inc(); //Count: 4
}
```

Header (.h) & Body (.cpp)

- 변수, 함수, 클래스에 대하여 선언(declaration) 과 구현(implementation) 을 분리합니다
- Header (.h) 파일에서 선언하고, Body (.cpp) 에서 구현합니다
- Cpp 파일에서 헤더 파일을 include하면, 컴파일러가 include를 처리할 때 해당 파일의 내용이 그 위치에 대신 들어갑니다
  - 프로그래머 대신 복사-붙여넣기를 해주는 것과 같습니다
- #ifndef 를 사용하여 헤더 파일의 중복 선언을 방지합니다

C_H 가 선언되어 있지 않다면
C_H를 선언
C_H가 있다면 define C_H 무시

header_practice > h C.h

```h
#ifndef C_H
#define C_H

#include <iostream>

void fncC();

#endif
```

header_practice > h B.h

```h
#ifndef B_H
#define B_H

#include "C.h"

void fncB();

#endif
```

header_practice > C++ C.cpp

```cpp
#include "C.h"

void fncC(){
    std::cout<<"functionC" << std::endl;
}
```

header_practice > C++ B.cpp

```cpp
#include "B.h"

void fncB(){
    std::cout << "functionB" << std::endl;
    fncC();
}
```

header_practice > C++ A.cpp

```cpp
#include "B.h"

int main(){
    std::cout << "functionA" << std::endl;
    fncB();
    return 0;
}
```

실습 exercise– 아주 간단한 포켓몬 클래스

- 포켓몬 클래스 Monster는 다음과 같은 멤버를 갖습니다
  - 멤버 변수
    - 체력 hp
    - 평타 공격력 damage
    - 치명타 공격력 critical_damage
    - 치명타 횟수 critical_left
      - 초기에는 3회로 설정됩니다
      - 치명타를 한번 사용할 때마다 1회씩 차감됩니다
    - 이름 name

  - 멤버 함수
    - 체력 감소 void decrease_hp(int)
    - 평타 공격 void attack(Monster*)
    - 치명타 공격 void critical_attack(Monster*)
    - 체력 반환 int get_hp()
    - 정보 출력 void get_info()
  - 정적 멤버 변수
    - 총 몬스터의 수 num_monsters
  - 정적 멤버 함수
    - 총 몬스터의 수 반환 int get_num()

실습 – 접근 제어자

- 포켓몬의 체력은 attack과 critical_attack에 의해서만 깎입니다.
- 총 포켓몬의 수는 생성자가 호출될 때에만 1씩 증가합니다.
- 치명타의 횟수는 critical_attack이 호출될 때에만 1씩 감소합니다.

- 멤버 변수
  - 체력 hp
  - 평타 공격력 damage
  - 치명타 공격력 critical_damage
  - 치명타 횟수 critical_left
  - 이름 name

- 멤버 함수
  - 체력 감소 void decrease_hp(int)
  - 평타 공격 void attack(Monster*)
  - 치명타 공격 void critical_attack(Monster*)
  - 체력 반환 int get_hp()
  - 정보 출력 void get_info()
- 정적 멤버 변수
  - 총 몬스터의 수 num_monsters
- 정적 멤버 함수
  - 총 몬스터의 수 반환 int get_num()

- private, protected, public

실습 – namespace, typedef 활용

- attack 또는 critical_attack에서 공격이 끝나면, 아래와 같은 메세지를 출력한다
  - 꼬부기가 피카츄를 공격했다 !
  - 피카츄의 남은 체력은 192 or 피카츄는 쓰러졌다
- 따라서 std::cout , std::endl 등을 편하게 사용하기 위하여 namespace를 사용한다
- 체력에 해당하는 int에는 hp_t 라는 별명을 붙인다
- 포켓몬의 이름에 해당하는 std::string에는 mon_name이라는 별명을 붙인다

```
using namespace std;
typedef int hp_t;
typedef string mon_name;
```

monster.h

실습 Exercise

- 해당 내용을 print하도록 나머지 monster.cpp의 함수를 작성해보세요.

```cpp
#include "monster.h"
#include "monster.cpp"
#include <string>
#include <stdio.h>

int main()
{
    Monster m1 = Monster(100, 8, 10, "꼬북");
    cout << "현재 몬스터 수는" << Monster::get_num() << endl;
    Monster *m2;
    m2 = new Monster(200, 5, 7, "피카츄");
    cout << "현재 몬스터 수는" << Monster::get_num() << endl;
    Monster *m3;
    m3 = new Monster(150, 3, 20, "이상해씨");
    cout << "현재 몬스터 수는" << Monster::get_num() << endl;
    m1.attack(m2);
    m2->critical_attck(&m1);
    m2->critical_attck(&m1);
    m2->critical_attck(m3);
    m2->critical_attck(&m1);
    m1.critical_attck(m2);
    m1.get_info();
    m2->get_info();
    m3->get_info();
}
```

Main.cpp

```
현재 몬스터 수는1
현재 몬스터 수는2
현재 몬스터 수는3
***
꼬북이(가)피카츄을(를) 공격했다!
피카츄의 남은 체력은192
*****
***
피카츄이(가)꼬북에게 치명타를 가했다!
꼬북의 남은 체력은93
******
***
피카츄이(가)꼬북에게 치명타를 가했다!
꼬북의 남은 체력은86
******
***
피카츄이(가)이상해씨에게 치명타를 가했다!
이상해씨의 남은 체력은143
******
피카츄은(는)치명타를 모두 사용했다
***
꼬북이(가)피카츄에게 치명타를 가했다!
피카츄의 남은 체력은182
******
꼬북의 현재 체력은: 86, 남은 치명타는:2
피카츄의 현재 체력은: 182, 남은 치명타는:0
이상해씨의 현재 체력은: 143, 남은 치명타는:3
```

output

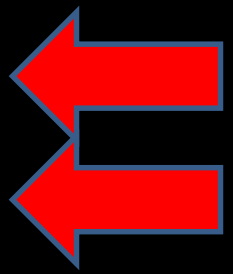
실습 Exercise– monster.h, monster.cpp

```h
#ifndef POKEMON_MONSTER_H
#define POKEMON_MONSTER_H
#include <iostream>

using namespace std;
typedef int hp_t;
typedef string mon_name;

class Monster
{
private:
    hp_t hp;
    hp_t damage;
    hp_t critical_damage;
    int critical_left = 3;
    static int num_monsters;

protected:
    mon_name name;
    void decrease_hp(hp_t attack_damage);

public:
    Monster(hp_t hp, hp_t damage, hp_t
criticla_damage, mon_name name);

    void attack(Monster *attacked_monster);
    void critical_attck(Monster *attacked_monster);
    hp_t get_hp();
    void get_info();
    static int get_num();
};
#endif
```

Monster.h

```cpp
#include "monster.h"
int Monster::num_monsters = 0;

Monster::Monster(hp_t hp, hp_t damage, hp_t critical_damage, mon_name name)
    : hp(hp), damage(damage), critical_damage(critical_damage), name(name)
{
    num_monsters++;
};

void Monster::decrease_hp(hp_t attack_damage)
{
    hp -= attack_damage;
    if (hp < 0)
        hp = 0;
};
hp_t Monster::get_hp()
{
    return hp;
};
void Monster::get_info()
{
    if (hp > 0)
    {
        std::cout << name << "의 현재 체력은: " << hp << ", 남은 치명타는:"
<< critical_left << std::endl;
    }
    else
    {
        std::cout << name << "는 쓰러졌다" << std::endl;
    }
}

int Monster::get_num()
{
    return num_monsters;
}
void Monster::attack(Monster *attacked_monster)
{
                //fill in
}

void Monster::critical_attck(Monster *attacked_monster)
{
                //fill in
};
```

Monster.cpp



실습 – main.cpp

- Pointer로 만든 몬스터와 value로 만든 몬스터 사이에 싸움을 붙여보세요.
- 두 몬스터 사용법에는 어떤 차이가 있나요 ?
- 현재까지 만들어진 몬스터의 수를 출력합니다.
  - get_num()

```
#include "monster.h"
#include "monster.cpp"
#include <string>
#include <stdio.h>

int main()
{
    Monster m1 = Monster(100, 8, 10, "꼬북");
    cout << "현재 몬스터 수는" << Monster::get_num() << endl;
    Monster *m2;
    m2 = new Monster(200, 5, 7, "피카츄");
    cout << "현재 몬스터 수는" << Monster::get_num() << endl;
    Monster *m3;
    m3 = new Monster(150, 3, 20, "이상해씨");
    cout << "현재 몬스터 수는" << Monster::get_num() << endl;
    m1.attack(m2);
    m2->critical_attck(&m1);
    m2->critical_attck(&m1);
    m2->critical_attck(m3);
    m2->critical_attck(&m1);
    m1.critical_attck(m2);
    m1.get_info();
    m2->get_info();
    m3->get_info();
}
```

실습 – 생성자

- 포켓몬을 만들때에 체력, 평타 데미지, 치명타 데미지, 이름을 직접 초기화하려고 한다.
  - Member Initializer List 사용
  - Constructor Body 에서 몬스터 수를 1증가 시켜준다
- 생성자 선언을 헤더 파일에 추가한다 (21번째 줄).

```
20  public:
21      Monster(hp_t hp, hp_t damage, hp_t critical_damage, mon_name name);
22      void attack(Monster *attacked_monster);
23      void critical_attack(Monster *attacked_monster);
24      hp_t get_hp();
25      void get_info();
26      static int get_num();
27  };
```

monster.h

실습 – monster.cpp

- Static 변수 num_monsters 를 0으로 초기화 한다.
- 생성자 구현
  - member initializer list 로 멤버 변수 값 초기화
  - 생성자가 한번 호출 될 때마다 num_monsters 1씩증가

```cpp
#include "monster.h"

int Monster::num_monsters = 0;

Monster::Monster(hp_t hp, hp_t damage, hp_t critical_damage, mon_name name)
    : hp(hp), damage(damage), critical_damage(critical_damage), name(name) {
      num_monsters++;
    };

```

monster.cpp

실습 – monster.cpp

- void decrease_hp(hp_t attack_damage)
  - 공격받은 만큼 체력이 감소
  - 체력이 0보다 작아지면 체력을 0으로 설정
- hp_t get_hp()
  - 체력을 반환
- void get_info()
  - 체력이 0보다 크면, '피카츄의 현재 체력은 : 120, 남은 치명타 횟수는 : 3' 과 같이 출력
  - 체력이 0보다 작으면 '피카츄는 쓰러졌다'와 같이 출력
- int get_num()
  - 전체 몬스터 수 반환

실습 – monster.cpp

```
10  void Monster::decrease_hp(hp_t attack_damage){
11    hp -= attack_damage;
12    if(hp < 0) hp = 0;
13  };
14
15  hp_t Monster::get_hp() {
16    return hp;
17  };
18
19  void Monster::get_info() {
20    if(hp > 0)
21      std::cout << name << "의 현재 체력은 : " << hp << ", 남은 치명타는 : " << critical_left << std::endl;
22    else
23      std::cout << name << "는 쓰러졌다" << std::endl;
24  }
25
26  int Monster::get_num() {
27    return num_monsters;
28  }
29
```

실습 – monster.cpp

- void attack(Monster *attacked_monster)
  - attacked_monster의 체력을 나의 평타 데미지 만큼 감소시킨다
    - 어떤 함수에 어떻게 접근하면 좋을까요 ?
  - 공격이 끝나면 아래와 같은 메세지 출력
    - ~~*~~
    - 피카츄가 이상해씨를 공격했다 !
    - 이상해씨의 남은 체력은 192 or 이상해씨가 쓰러졌다 !
    - ~~****~~

```
30  void Monster::attack(Monster *attacked_monster)
31  {
32      //fill in
33  };
34
```

실습 – monster.cpp

- void critical_attack(Monster *attacked_monster)
  - 나의 치명타 횟수가 남아 있는 경우,
    - attacked_monster의 체력을 나의 치명타 데미지 만큼 감소시킨다
    - 공격이 끝나면 아래와 같은 메세지 출력
      - **
      - 피카츄가 이상해씨에게 치명타를 가했다 !
      - 이상해씨의 남은 체력은 10or 이상해씨가 쓰러졌다 !
      - ****
  - 치명타 횟수를 다 쓴 경우 아래와 같은 메세지 출력
    - 피카츄는 치명타를 모두 사용했다

```
37  void Monster::critical_attack(Monster *attcked_monster)
38  {
39      if(critical_left>0)
40      {
41          //fill in
42      }
43
44      else
45      {
46          //fill in
47      }
48  };
```

실습 – monster.cpp

```
30  void Monster::attack(Monster *attacked_monster){
31      attacked_monster->decrease_hp(damage);
32      std::cout << "***" << std::endl;
33      std::cout << this->name << "이(가) " << attacked_monster->name << "을(를) 공격했다 !" << std::endl;
34      if(attacked_monster->get_hp() > 0){
35          std::cout << attacked_monster->name << "의 남은 체력은 " << attacked_monster->get_hp() << std::endl;
36      } else{
37          std::cout << attacked_monster->name << "가 쓰러졌다 !" << std::endl;
38      }
39      std::cout << "******" << std::endl;
40  };
```

실습 – monster.cpp

```
42  void Monster::critical_attack(Monster *attacked_monster){
43      if(critical_left > 0){
44          attacked_monster->decrease_hp(critical_damage);
45          critical_left -= 1;
46          std::cout << "***" << std::endl;
47          std::cout << this->name << "이(가) " << attacked_monster->name << "에게 치명타를 가했다 !" << std::endl;
48          if(attacked_monster->get_hp() > 0){
49              std::cout << attacked_monster->name << "의 남은 체력은 " << attacked_monster->get_hp() << std::endl;
50          } else{
51              std::cout << attacked_monster->name << "가 쓰러졌다 !" << std::endl;
52          }
53          std::cout << "******" << std::endl;
54      } else {
55          std::cout << this->name << "은(는) " << "치명타를 모두 사용했다" << std::endl;
56      }
57  };
```

추가 실습 – main.cpp

- While문을 사용하여 둘 중 하 나가 쓰러질 때까지 싸우도록 싸움을 붙여보세요.
- 싸움이 끝난 후에 두 몬스터의 상태를 출력해보세요.

```
51    while(1)
52    {
53     // break로 while문 나옴
54    }
55    m2->get_info();
56    m3->get_info();
```

실습 – main.cpp

- While문을 사용하여 둘 중 하나가 쓰러질 때까지 싸우도록 싸움을 붙여보세요.
- 싸움이 끝난 후에 두 몬스터의 상태를 출력해보세요.

```
22      while(1){
23          m2->attack(m3);
24          if(m3->get_hp() == 0) break;
25          m3->attack(m2);
26          if(m2->get_hp() == 0) break;
27      }
28      m2->get_info();
29      m3->get_info();
```

Exercise 1

Problem 1. 이차 방정식 덧셈 (Overloading 연습)

두 개의 이차 방정식을 더하는 프로그램을 작성

Ex) 3x^2+2x+1 + 2x^2+3x+1 =5x^2+5x+2

Equation 클래스와 EquationUtility 클래스를 작성

```
int main() {
    Equation e1(2);
    Equation e2(4, -5);
    EquationUtility a;
    Equation result = a.add(e1, e2);
    cout << a.output(result) << endl;
    Equation e3(3, 0, 5);
    result = a.add(e1, e3);
    cout << a.output(result) << endl;
    return 0;
}
```

Main 함수

```
4x-3
3x^2+7
```

출력 결과

Exercies code

```cpp
#include <iostream>
using namespace std;

class Equation {
public:
    int a, b, c;

    Equation(int _c) {
        a = 0;
        b = 0;
        c = _c;
    }
    Equation(int _b, int _c) {
        a = 0;
        b = _b;
        c = _c;
    }
    Equation(int _a, int _b, int _c) {
        a = _a;
        b = _b;
        c = _c;
    }
};
```

Equation 클래스

```cpp
class EquationUtility{
public:
    Equation add(Equation e1, Equation e2) {
        return Equation(e1.a + e2.a, e1.b + e2.b, e1.c + e2.c);
    }
    string out(Equation e) {
        string str;
        if (e.a != 0) {
            str.append(to_string(e.a));
            str.append("x^2");
            if (e.b > 0) {
                str.append("+");
            }
            if (e.b == 0 & e.c > 0){
                str.append("+");
            }
        }
        if (e.b != 0) {
            str.append(to_string(e.b));
            str.append("x");
            if (e.c > 0) {
                str.append("+");
            }
        }
        if (e.c != 0) {
            str.append(to_string(e.c));
        }
        return str;
    }
};
```

EquationUtility 클래스

Exercise1

Problem 2. 샌드위치 가게 메뉴 가격 계산 (Overriding 연습)

샌드위치 가게에서 메뉴 가격을 계산하는 프로그램을 작성합니다.

- 메뉴: 샐러드 그리고 샌드위치
  - 샐러드와 샌드위치는 고기 종류를 치킨과 터키 중에서 고를 수 있습니다.
  - 샌드위치는 길이를 30cm 단위로 주문할 수 있습니다.
  - ex) 30cm, 60cm 주문 O / 29cm, 31cm 주문 X

토핑 추가는 샐러드와 샌드위치 둘 다 가능

- 치킨 샐러드 : 8,500원, 터키 샐러드 : 9,000원
- 치킨 샌드위치(30cm 당) : 7,500원
- 터키 샌드위치(30cm 당) : 8,000원
- 토핑 추가
  - 아보카도 2,000원
  - 치즈 1,000원

Exercise1

Salad 클래스와 Sandwich 클래스를 작성합니다.

- Salad 클래스는 고기 종류를 입력으로 받습니다.
- Sandwich 클래스는 샌드위치 길이와 고기 종류를 입력으로 받습니다.

- 다음은 코드 작성 간 제한 사항

1. Sandwich 클래스는 Salad 클래스를 상속받습니다.
Sandwich : 자식 클래스, Salad : 부모 클래스
2. addSomething 함수와 showPrice 함수는 Salad 클래스에서만 작성합니다.
3. addSomething 함수의 파라미터는 1개이며, 함수를 여러 번 실행해도 모두 금액에 추가됩니다.
4. 메뉴 가격을 계산하는 함수는 오버라이딩해서 작성합니다.

아래 스크린샷은 main 함수입니다.

```
int main()
{
    int num;
    cout << "샐러드 주문은 1, 샌드위치 주문은 2" << endl;
    cin >> num;
    if(num == 1){
        Salad salad1("chicken");
        salad1.addSomething("cheese");
        salad1.showPrice();
    }
    else if(num == 2){
        Sandwich sandwich1(30, "turkey");
        sandwich1.addSomething("avocado");
        sandwich1.showPrice();
    }
}
```

아래 스크린샷은 출력 결과입니다.

```
샐러드 주문은 1, 샌드위치 주문은 2
1
price : 9500원
```

```
샐러드 주문은 1, 샌드위치 주문은 2
2
price : 10000원
```

1을 입력하면, 치킨 샐러드 8500원에
치즈 토핑 1000원을 추가한
9500원이 출력되고
2를 입력하면, 터키 샌드위치 8000원에
아보카도 토핑 2000원을 추가한
10000원이 출력됩니다.

```cpp
#include <iostream>
using namespace std;
class Salad{
private:
    string meat;
    string menu;

    void calcPrice()
    {
        if (meat == "chicken"){
            price = 8500;
        }
        else if(meat == "turkey"){
            price = 9000;
        }
    }

public:
    int price;

    Salad(string _meat)
    {
        price = 0;
        meat = _meat;
        calcPrice();
    }

    void addSomething(string _menu){
        menu = _menu;
        if(menu == "avocado"){
            price = price + 2000;
        }
        else if(menu == "cheese"){
            price = price + 1000;
        }
    }
    void showPrice()
    {
        cout << "price :" << price << endl;
    }
};
```

Salad 클래스

```cpp
class Sandwich : public Salad{
private:
    int len;
    string meat;

    void calcPrice()
    {
        if (meat == "chicken"){
            price = 7500 * (len/30);
        }
        else if(meat == "turkey"){
            price = 8000 * (len/30);
        }
    }

public:
    Sandwich(int _len, string _meat) : Salad(_meat){
        price = 0;
        len = _len;
        meat = _meat;
        calcPrice();
    }
};
```

Sandwich 클래스