클래스 배열, friend 함수

[문제] main 함수의 실행 결과가 아래와 같이 될 수 있도록 Point 클래스와 GetSumX, GetSumY 함수를 작성하시오.
[조건] Point 클래스의 member 변수 m_x, m_y 는 private으로 선언하고 friend 함수를 활용하시오.

[결과]
(9, 12)

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

int main() {
    Point ary[5] = {Point(1,2), Point(3,4), Point(5,6)};
    Point sum;
    sum.SetXY(GetSumX(ary, 5), GetSumY(ary, 5));
    sum.Print();
    return 0;
}
```

friend 선언

- friend 선언의 의미
  - class 멤버가 아니면서도 클래스의 private member까지 접근 가능하게 해 줌.

- friend 전역 함수 선언
- friend 클래스 선언
- friend 맴버 함수 선언

```
class ClassA {
private:
    friend void Func( ClassA &a, int a);
    friend class Controller;
    friend void ClassB::Func();
};
```

[Note!]
friend 선언은 private 영역이든 public 영역이든 상관 없음.

클래스 배열, friend 함수

```
class Point {
public:
    Point(int x, int y) { m_x = x, m_y = y; }
    Point() { m_x = m_y = 0; }
    void Print() { cout << "(" << m_x << ", " << m_y << ")“ << endl; }
    void SetXY(int x, int y) { m_x = x; m_y = y; }
private:
    int m_x, m_y;
    friend int GetSumX(Point *pPoint, int count);
    friend int GetSumY(Point *pPoint, int count);
};
int GetSumX(Point *pPoint, int count) {
    int i, sum = 0;
    for (i = 0; i < count; i++)
        sum += pPoint[i].m_x;
    return sum;
}
int GetSumY(Point *pPoint, int count) {
    int i, sum = 0;
    for (i = 0; i < count; i++)
        sum += pPoint[i].m_y;
    return sum;
}
```

this 포인터 활용.
멤버함수 overloading.

[문제] 앞의 예제를 수정한 main 함수의 실행 결과가 아래와 같이 될 수 있도록
Point 클래스에 Move 멤버 함수 2개를 overloading하여 구현하시오.

[결과]
(9, 12)
(16, 20)

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

int main() {
    Point ary[5] = {Point(1,2), Point(3,4), Point(5,6)};
    Point sum;
    sum.SetXY(GetSumX(ary, 5), GetSumY(ary, 5));
    sum.Print();
    sum.Move(3).Move(4, 5);
    sum.Print();
    return 0;
}
```

this 포인터 활용.
멤버함수 overloading.

```
class Point {
public:
    Point(int x, int y) { m_x = x, m_y = y; }
    Point() { m_x = m_y = 0; }

    void Print() { cout << "(" << m_x << ", " << m_y << ")" << endl; }
    void SetXY(int x, int y) { m_x = x; m_y = y; }
    Point &Move(int inc) { m_x += inc; m_y += inc; return *this; }
    Point &Move(int x, int y) { m_x += x; m_y += y; return *this; }
private:
    int m_x, m_y;
    friend int GetSumX(Point *pPoint, int count);
    friend int GetSumY(Point *pPoint, int count);
};
```

static 멤버 변수, static 멤버 함수

[문제] 앞의 예제를 수정한 main 함수의 실행 결과가 아래와 같이 될 수 있도록 static 멤버 변수 int s_count; 와 static 멤버 함수 PrintHowMany()를 구현하시오.

[결과]
(9, 12)
(16, 20)
총 6개

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

int main() {
    Point ary[5] = {Point(1,2), Point(3,4), Point(5,6)};
    Point sum;
    sum.SetXY(GetSumX(ary, 5), GetSumY(ary, 5));
    sum.Print();
    sum.Move(3).Move(4, 5);
    sum.Print();
    Point::PrintHowMany();
    return 0;
}
```

static 멤버 변수, static 멤버 함수

- **static 멤버 변수**
  - 각 개체별로 생성되는 것이 아니고 **클래스당 하나만 생성**됨.
  - 특정 개체를 통해서도 접근 가능하지만 클래스 이름을 통해서도 접근 가능.
    ```
    sum.PrintHowMany();
    Point::PrintHowMany();
    ```
  - 명시적으로 선언하는 문장 필수.
- **static 멤버 함수**
  - **static 멤버에만 접근 가능한 함수**.
  - 특정 개체를 통해서도 접근 가능하지만 클래스 이름을 통해서도 접근 가능.

static 멤버 변수, static 멤버 함수

```
class Point {
public:
    Point(int x, int y) { m_x = x, m_y = y; s_count++; }
    Point() { m_x = m_y = 0; s_count++;}
    ~Point() { s_count--; }
    void Print() { cout << "(" << m_x << ", " << m_y << ")" << endl; }
    void SetXY(int x, int y) { m_x = x; m_y = y; }
    Point &Move(int inc) { m_x += inc; m_y += inc; return *this; }
    Point &Move(int x, int y) { m_x += x; m_y += y; return *this; }
    static void PrintHowMany() { cout << "총 " << s_count << "개" << endl; };
private:
    static int s_count;
    int m_x, m_y;
    friend int GetSumX(Point *pPoint, int count);
    friend int GetSumY(Point *pPoint, int count);
};
int Point::s_count = 0;
```

상속 (inheritance)

- 상속이란
  - 새로운 클래스를 만들 때 기존 클래스의 특징을 모두 상속 받는 것.
  - 코드의 재활용
  - 클래스 library와 선언(*.h 파일)부만 있으면 상속 받아 자신만의 클래스를 만들 수 있다.

- 상속의 문법

```
class Truck : public Car
{
    ......
};
```

base class

derived class

Access 지정자 : (public, protected, private)
Base class에서 정한 접근 방법을 확장하지는 못함.
유지 또는 더욱 한정하는 것만 가능.

상속 : 생성자와 소멸자

- Derived class 객체 생성/소멸시의 생성자.소멸자 수행 순서
  - Base class 생성자
  - Derived class 생성자
  - Derived class 소멸자
  - Base class 소멸자

- Derived class 생성자에서 base class 생성자 호출.

```
Sphere::Sphere( int x, int y, int z, double radius ) :
   Circle(x, y, radius) {
    ......
}
```

명시적으로 base class 생성자 호출.
이 부분이 생략되면 default constructor 사용됨.