



Chapter 11. Various applications of object

목차

1. Object pointer
2. Passing parameters to methods of the object
3. Static member variables and static member functions
4. An array of objects
5. Friend function
6. Functions for manipulating the object
7. Operator Overloading

학습목표

- After declaring a pointer to the class object using indirect reference to it.
- Study for this to refer to an instance of itself.
- The learning function for a friend to use a private member outside the class
- Defines the various functions that the object as a parameter.
- Students learn how a user overloading the operator to deal with a defined class.

01 Object pointer

- The basic format for declaring an object pointer variable is as follows.

```
Class name * object pointer variable;
```

객체 포인터 변수 기본 형식

- Object pointer variable and stores the address of the specific object parameters.

```
Complex x(10, 20);
```

```
Complex *pCom;
```

```
pCom = &x;
```

- . Operator is used to access the object member, and -> operator is used to refer to a member as an object pointer.

```
pCom->ShowComplex();
```

Example 11-1. Using object pointer (11_01.cpp)

```
#include <iostream>
using namespace std;
class Complex {
private:
    int real;
    int image;
public:
    Complex(int r = 0, int i = 0);
    void ShowComplex() const;
};
Complex::Complex(int r, int i) : real(r), image(i)
{
}
void Complex::ShowComplex() const
{
    cout << "( " << real << " + " << image << "i )"
        << endl;
}
```

```
void main() {
    Complex x(10, 20);
    Complex y;

    cout << " Object x => ";
    x.ShowComplex();
    cout << " Object y => ";
    y.ShowComplex();

    Complex *pCom;    // 객체 포인터 선언
    pCom = &x;         // 객체 x의 주소 할당

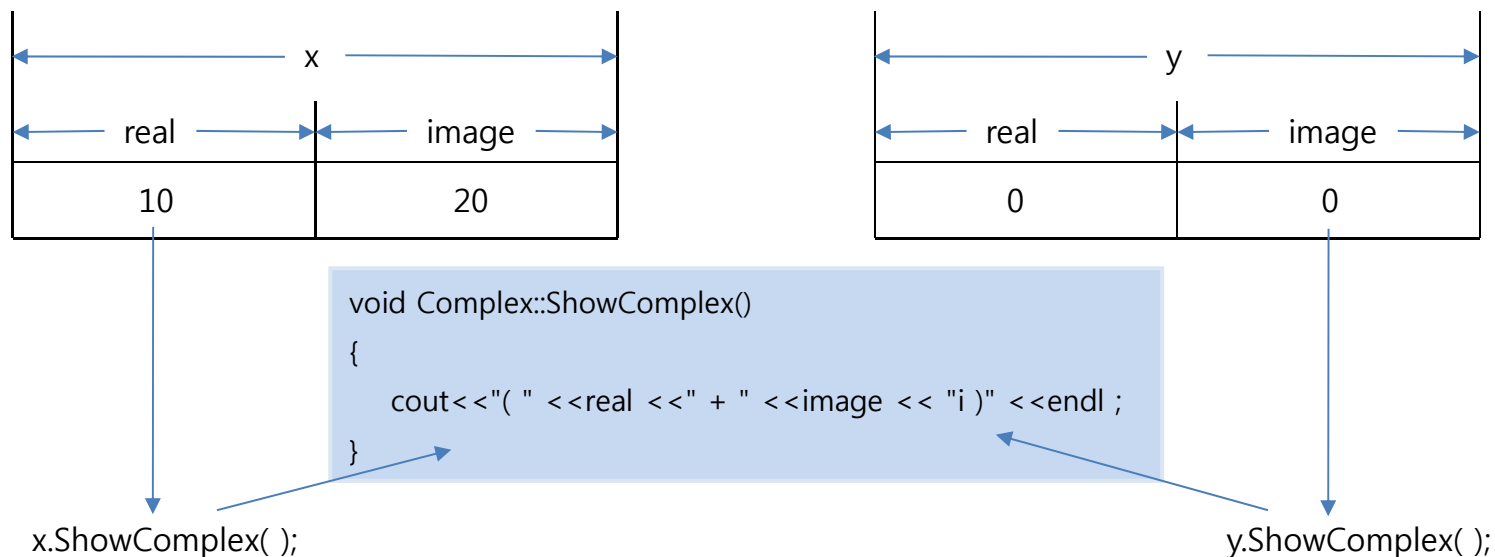
    cout << "\n pCom->ShowComplex() => ";
    pCom->ShowComplex();    // 객체 x의 멤버함수 호출

    pCom = &y;          // 객체 y의 주소 할당
    cout << " pCom->ShowComplex() => ";
    pCom->ShowComplex();    // 객체 y의 멤버함수 호출
}
```

01 Object pointer

■ The structure of the object member variables and member functions

- Calling the member function is a member function, member variables in the variables of a particular object.
- Member variables are managed separately received allocation of storage space whenever you declare an object.
- For calls x.Show Complex () member function in the Show Complex is a member of the object x,
- For calls y.Show Complex () member function in the Show Complex is a member of an object y.

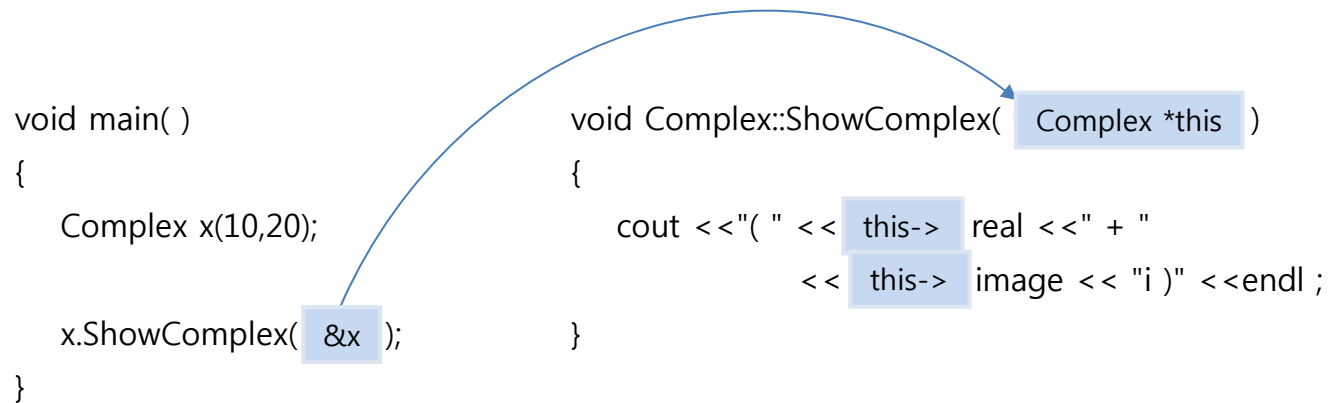


[Picture 11-3] The operation principle of the member function

01 Object pointer

■ this is internal pointer

- this is a pointer that is generated by the compiler, pointing to the object of calling the member function can be used only within member functions.



[Picture 11-4] Identity of the object referred to by the member variables of the member function.

- Member variable is a member (members), so the "object name the members 'or' object pointer -> member, you need to specify who, as a member, but until now has been used in the member function, member variables without any doubt. This is because the compiler implicitly gave paste `this->` in front of all members.

01 Object pointer

- **Summarizing the this pointer as follows:**
 - ① this pointer is a pointer for storing the address of the called member function in the object.
 - ② this pointer is always present within the member functions do not need to separately declared by the programmer is provided by the compiler.
 - ③ **When a member function is invoked by the object compiler will store the addresses of the calling object to the this pointer in the member function.**
 - ④ Refer to the member variables within the member functions or access to this pointer implicitly when you call another member function.
 - ⑤ Refer to the member variables within the member functions or access to this pointer implicitly when you call another member function.
- **If you must use this is when you need to distinguish them by the same parameters, member variables of the function name.**

Example 11-2. Using the this pointer explicitly (11_02.cpp)

```
#include <iostream>
using namespace std;
class Complex {
private:
    int real;
    int image;
public:
    Complex(int real = 0, int image = 0);
    void ShowComplex() const;
};

/*
// If the same name as a member variable of the
function of the parameter
Complex::Complex(int real, int image)
{
    real=real;
    image=image;
}

*/
```

```
// Separate the parameters using this function and member
variable of a pointer
Complex::Complex(int real, int image)
{
    this->real = real;
    this->image = image;
}

// 호출한 객체의 주소를 가지는 포인터 this
void Complex::ShowComplex() const
{
    cout << "( " << this->real << " + " << this->image << "i )"
        << endl;
}

void main()
{
    Complex x(10, 20);
    x.ShowComplex();
}
```

02 Passing parameters to methods of the object

① Transfer scheme according to the value of the object.

- 'Propagation scheme according to the value "it is transmitted to the side of the type parameter of a value function when calling the function described yarn parameters.

That is, the format parameters are actual parameters and is assigned a separate memory area and copied values here.

The return of the object

- Use the return statement can return the object.

② Transfer scheme according to the address of the object

- By passing the address of the object to a function it can be an indirect reference pointer within the function.

※ Each other and the object type declared in the same class can be replaced with the value of a member variable assignment.

That is, similarly to the case of a structure substituted with an object unit, the values of all member variables in the object is copied.

Example 11-4. Creating a function of the delivery method based on object value (11_04.cpp)

```
#include <iostream>
using namespace std;
class Complex {
private :
    int real;
    int image;
public :
    Complex( int r=0, int i=0);
    void ShowComplex() const;
    void SetComplex(int r=0, int i=0);
};

Complex::Complex(int r, int i) {
    real=r;
    image=i;
}

void Complex::ShowComplex() const {
    cout<<"( " <<real <<" + " <<image << "i )" <<endl ;
}
```

Example 11-4. Creating a function of the delivery method based on object value (11_04.cpp)

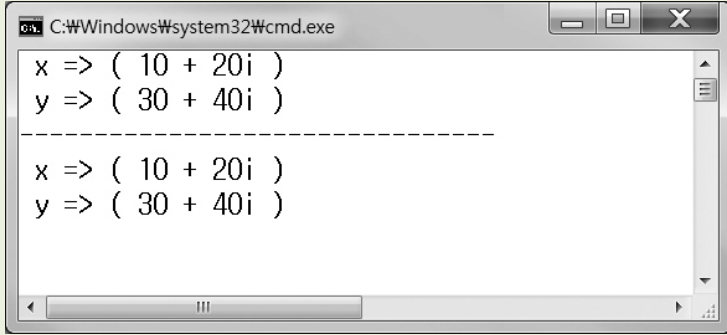
```
void Complex::SetComplex(int r, int i) {
    real=r;
    image=i;
}

void CopyComplex(Complex des, Complex src) {    // 객체의 값에 의한 전달방식을 사용하는 객체 복사 함수
    des=src;
}

void main() {
    Complex x(10, 20);
    Complex y(30, 40);

    cout<<" x => " ;
    x.ShowComplex();
    cout<<" y => " ;
    y.ShowComplex();

    cout<<"----- \n" ;
    CopyComplex(y, x);                                // 객체의 값에 의한 전달방식이므로 함수종료 후 y의 멤버변수 값들은 변화가 없음
    cout<<" x => " ;
    x.ShowComplex();
    cout<<" y => " ;
    y.ShowComplex();
}
```



```
C:\Windows\system32\cmd.exe
x => ( 10 + 20i )
y => ( 30 + 40i )
-----
x => ( 10 + 20i )
y => ( 30 + 40i )
```

Example 11-6. Creating a function of the delivery method based on the address of the object (11_06.cpp)

```
#include <iostream>
using namespace std;
class Complex
{
private :
    int real;
    int image;
public :
    Complex( int r=0, int i=0);
    void ShowComplex() const;
    void SetComplex(int r=0, int i=0);
};
Complex::Complex(int r, int i) {
    real=r;
    image=i;
}
void Complex::ShowComplex() const {
    cout<<"( " <<real <<" + " <<image <<"i )" <<endl ;
}
void Complex::SetComplex(int r, int i) {
    real=r;
    image=i;
}
```

Example 11-6. Creating a function of the delivery method based on the address of the object (11_06.cpp)

```
void CopyComplex(Complex *pDes, Complex src) { // 객체의 주소에 의한 전달방식을 사용하는 객체 복사 함수
    *pDes=src;
}
```

```
void main() {
    Complex x(10, 20);
    Complex y(30, 40);
```

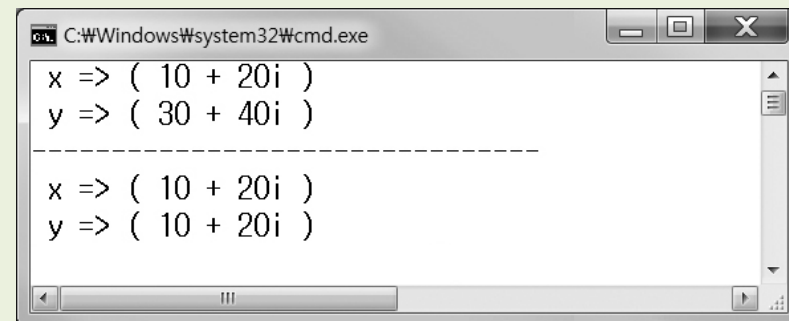
```
    cout<<" x => " ;
    x.ShowComplex();
    cout<<" y => " ;
    y.ShowComplex();
```

```
    cout<<"----- \n" ;
```

```
    CopyComplex(&y, x);
```

// 객체 y를 주소에 의해 전달하여 참조함

```
    cout<<" x => " ;
    x.ShowComplex();
    cout<<" y => " ;
    y.ShowComplex();
}
```



02 Passing parameters to methods of the object

③ Transfer scheme according to the object of reference.

- With reference variable can change the value of the physical parameter.
- Let's change the CopyComplex function as follows:

```
void CopyComplex(Complex &des, const Complex &src)
{
    des=src;
}
```

Example 11-7. Creating a function of the transmission method according to the object reference (11_07.cpp)

```
#include <iostream>
using namespace std;
class Complex
{
private :
    int real;
    int image;
public :
    Complex( int r=0, int i=0);
    void ShowComplex() const;
    void SetComplex(int r=0, int i=0);
};
Complex::Complex(int r, int i) {
    real=r;
    image=i;
}
void Complex::ShowComplex() const {
    cout<<"( " <<real <<" + " <<image <<"i )" <<endl ;
}
void Complex::SetComplex(int r, int i) {
    real=r;
    image=i;
}
```


Example 11-7. Creating a function of the transmission method according to the object reference (11_07.cpp)

```
void CopyComplex(Complex &des, const Complex &src) { // 객체 참조에 의한 전달방식
```

```
    des = src;
```

```
}
```

```
void main() {
```

```
    Complex x(10, 20);
```

```
    Complex y(30, 40);
```

```
    cout << " x => ";
```

```
    x.ShowComplex();
```

```
    cout << " y => ";
```

```
    y.ShowComplex();
```

```
    CopyComplex(y, x);
```

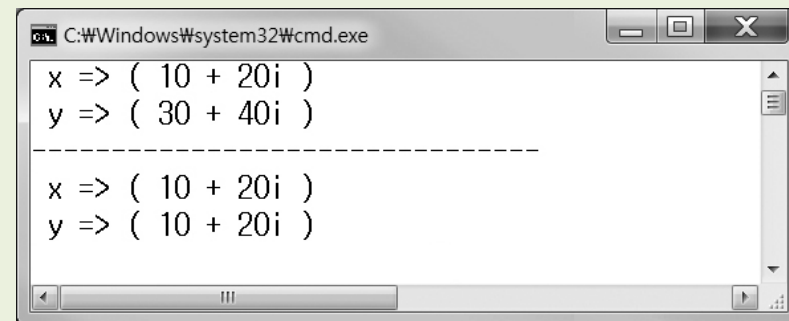
```
    cout << " x => ";
```

```
    x.ShowComplex();
```

```
    cout << " y => ";
```

```
    y.ShowComplex();
```

```
}
```



03 Static member variables and static member functions.

■ Static member variables.

- Features of the static member variables.
 - ① A static member variable is shared by all instances (objects) of the class.
 - ② The principle of a static member variable is the same as the global variable. However, static member variables should be approached with the appropriate class name.
 - Two conditions for using a static member variable.
 - ① Static member variables must be declared inside a class.

When you declare an object, the object is created as a member variable unit. If all objects are instances of a single class must share one static member variables using a reserved word as ① creates a static member variables.
 - ② Static member variables must be initialized separately, outside of class.

Static member variables are variables that are generated simultaneously with the start of the program, regardless of the class instance.

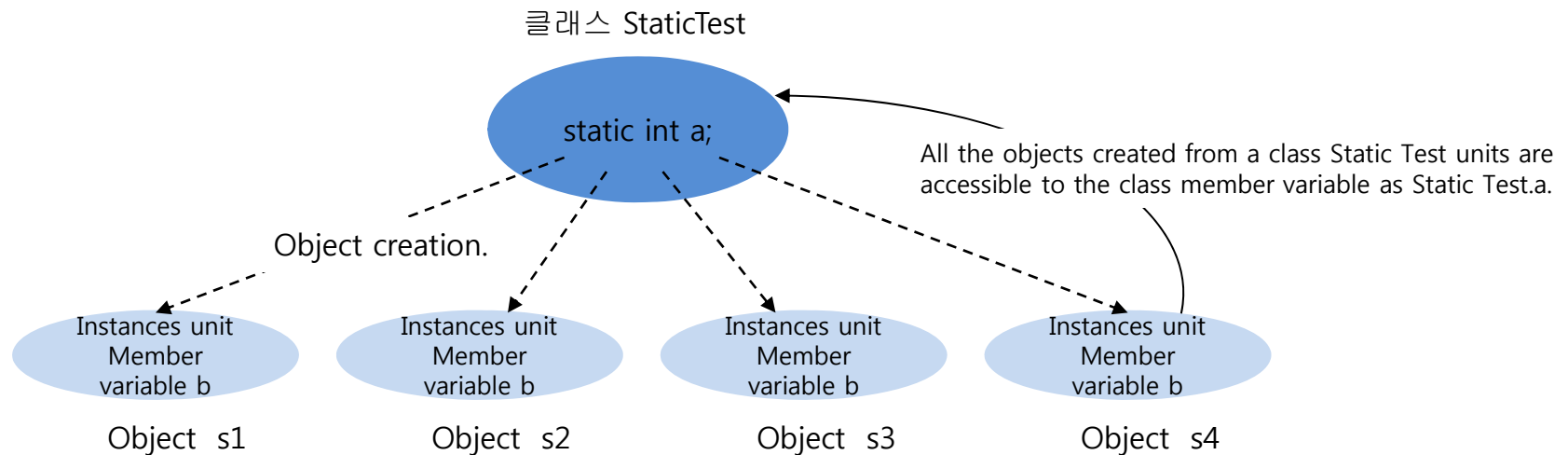
 - ① Outside of class, as must be initialized separately.
- ① class StaticTest {
 public:
 static int a;
 int b;
 }

② int StaticTest::a=10;

03 Static member variables and static member functions.

- Let's see an example of creating an object 4 to Static Text. Even creating an object 4 members are declared as static, only one generation per the class.

```
StaticTest s1, s2, s3, s4;
```



[Picture 11-6] Static member variables, classes, objects,

- static member variable is shared by multiple objects can access the objects without generation members with class names as follows.

```
StaticTest::a;
```

Example 11-8. Learn the difference between class and instance variables Unit member units member variable (11_08.cpp).

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

class StaticTest {
public:
    static int a; // Static member variables.
    int b; // Regular member variable.

    StaticTest(); // 생성자
};

StaticTest::StaticTest()
{
    b = 20;
}

int StaticTest::a = 10; // Initialize static member variables.
```

```
void main() {
    cout << " StaticTest::a : " << StaticTest::a << "\n\n";

    StaticTest s1, s2;

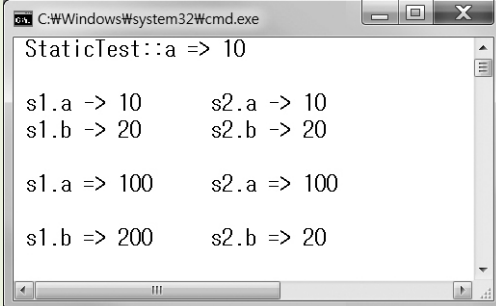
    cout << " s1.a : " << s1.a << "\t s2.a : " << s2.a << "\n";
    cout << " s1.b : " << s1.b << "\t s2.b : " << s2.b << "\n\n";

    s1.a = 100; // Static member variables value changes that are
shared by Static Test class.

    cout << " s1.a : " << s1.a << "\t s2.a : " << s2.a << "\n";

    s1.b = 200; // The regular member variable value changes belong
to each object instance.

    cout << " s1.b : " << s1.b << "\t s2.b : " << s2.b << "\n";
}
```



```
C:\Windows\system32\cmd.exe
StaticTest::a => 10

s1.a -> 10      s2.a -> 10
s1.b -> 20      s2.b -> 20

s1.a => 100     s2.a => 100
s1.b => 200     s2.b => 20
```

03 Static member variables and static member functions.

■ Static member function.

- If you declare a static member variables to private member functions to use static member variables it should be arranged separately. Member functions for handling static member variables are supposed to be designed to be used in class rather than a per-instance basis, and it is a static member functions provided for this purpose.

```
static int GetA();
```

- Precautions When Using a static member function.
 - Features of the static member function.
 - ❶ In the static member function can not refer to this point.
 - ❷ In the static member function you can not use an instance member variables.
 - ❸ Static member function is not of overriding (after additional explanation).

Example 11-9. Defining a static member function (11_09.cpp)

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

class StaticTest {
private:
    static int a;          // private 정적 멤버변수
    int b;
public:
    StaticTest();          // 생성자
    static void PrintA();  // 정적 멤버함수
    void PrintB();        // 일반 멤버함수
};

int StaticTest::a = 10;   // 정적 멤버변수 초기화

StaticTest::StaticTest()
{
    b = 20;
}
```

```
// 정적 멤버함수는 특정 인스턴스와 관계되지 않으므로,
// 즉, 해당 클래스의 모든 인스턴스와 공유되므로
// this 포인터를 사용할 수 없음.
// 또한, 특정 인스턴스의 멤버변수인 b는 클래스 전체적 단위로
// 호출되는 정적 멤버함수에서는 사용할 수 없다.
void StaticTest::PrintA() {
    cout << " a : " << a << endl;
    //cout <<" this->a : " << this->a << endl;
    //cout <<" b : " << b << endl;
}

void StaticTest::PrintB() {
    cout << " this->b : " << this->b << endl;
}

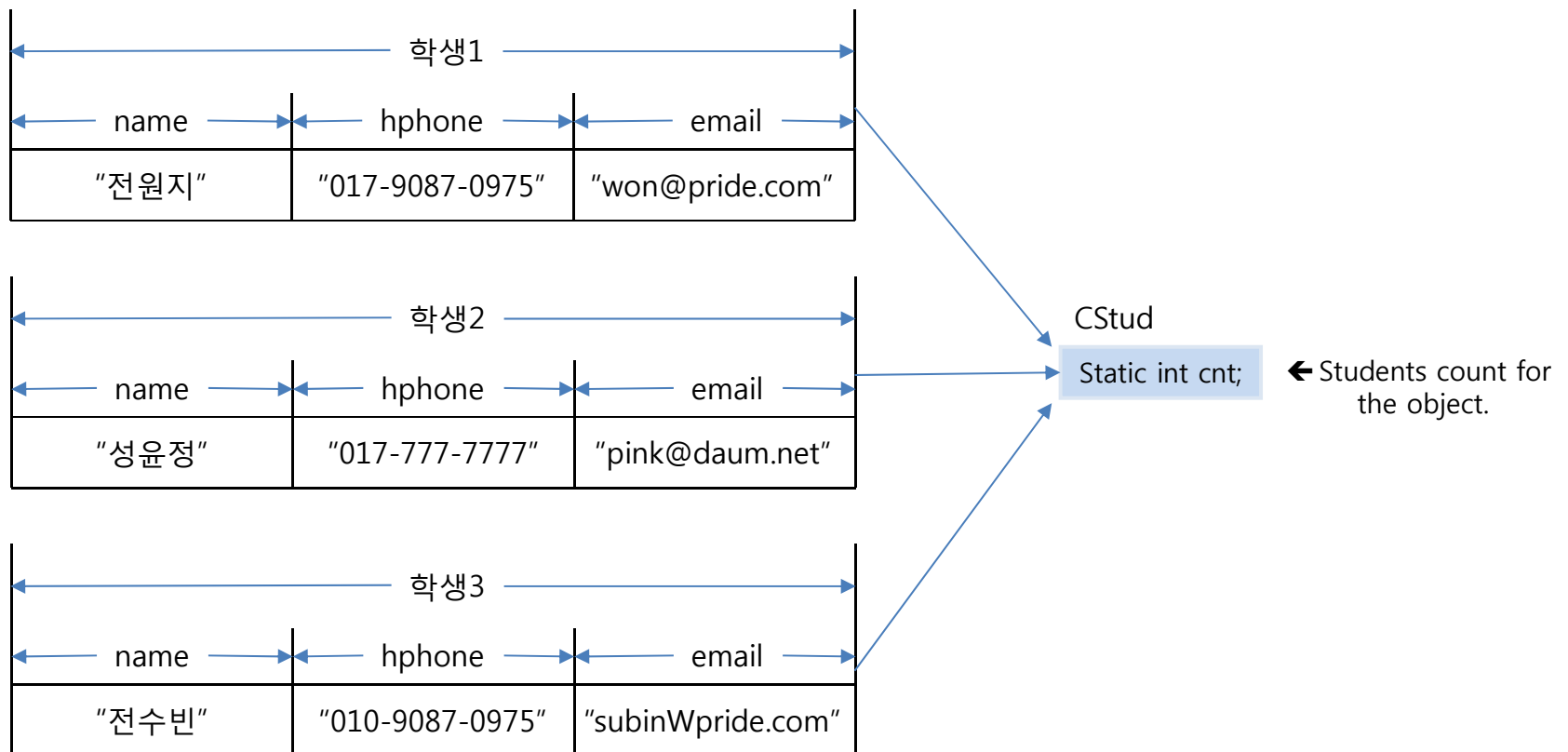
void main() {
    StaticTest s1;

    s1.PrintA();
    s1.PrintB();
}
```

03 Static member variables and static member functions.

■ Useful use of static member variables.

- As you design a class there are instances when you need to share all the member variables and member functions are generated, such as when you must count the total number of instances, a case like this, you should use static reserved.



[Picture 11-7] A static member variable is shared by multiple objects.

Example 11-11. Using a static member variable that counts the number of instances (11_11.cpp)

```
#include <iostream>
#include<string>    // 문자열 복사함수 사용을 위해
using namespace std;

class CStud {
private:
    char name[30];
    char hphone[20];
    char email[30];
    static int cnt;    // 정적 멤버변수
public:
    CStud(char *n = "성윤정", char *h = "017-777-7777", char *e = "pink@daum.net");
    ~CStud();
    void prn();
    static void prn_cnt(); // 정적 멤버함수
};

int CStud::cnt = 0;    // 정적 멤버변수 초기화
```


Example 11-11. Using a static member variable that counts the number of instances (11_11.cpp)

```
CStud::CStud(char *n, char *h, char *e) {    // 생성자
    strcpy_s(name, n);    // 문자열 복사함수
    strcpy_s(hphone, h);
    strcpy_s(email, e);
    cnt++;                // 생성자 호출 시 객체 카운트 1개 증가
}

CStud::~CStud() {        // 소멸자
    cnt--;                // 소멸자 호출 시 객체 카운트 1개 감소
}

void CStud::prn() {
    cout << "이름 : " << name << endl;
    cout << "핸드폰 : " << hphone << endl;
    cout << "이메일 : " << email << endl << endl;
}

void CStud::prn_cnt() { // 정적 멤버변수를 사용하는 정적 멤버함수
    cout << "현재까지 등록된 인원수 : " << cnt << "WnWn";
}
```

Example 11-11. Using a static member variable that counts the number of instances (11_11.cpp)

```
void main() {  
    // 객체가 선언되어 있지 않은 경우 클래스명으로 정적 멤버함수 호출  
    CStud::prn_cnt();  
  
    CStud man1("전수빈", "0110-9087-0975", "subin@pride.com");  
    man1.prn();  
    CStud man2("전원지", "017-9087-0975", "won@pride.com");  
    man2.prn();  
  
    cout << "Wn# 중간에 인원수를 파악합니다.";  
    man2.prn_cnt();    // 객체명으로도 정적 멤버함수를 호출할 수 있음  
  
    CStud man3;  
    man3.prn();  
  
    CStud::prn_cnt();    // 클래스명으로 정적 멤버함수 호출  
}
```

04 An array of objects.

- When declaring an array of objects and then declare a class is declared only when the array form when declaring the object.

```
Array class people people [One can introduce;
```

객체 배열 선언 형식

- Complex elements into four classes, let's create a personal object array.

```
Complex arr[4];
```

- When you see an array of objects is a mix of a subscript to an array of objects and then specify the name of the element to the desired position.

```
Array name [subscript] member variables;  
Array name [subscript] member function;
```

객체 배열 참조 형식

- If the second parameter is defined in the Private Constructors Complex class you can explicitly call the constructor to initialize the elements of an array of objects.

```
Complex arr[4] = {  
    Complex(2, 4);  
    Complex(4, 8);  
    Complex(8, 16);  
    Complex(16, 32);  
};
```

Example 11-12. Using an array of objects (11_12.cpp)

```
#include <iostream>
using namespace std;
class Complex {
private :
    int real;
    int image;
public :
    Complex( int r=0, int i=0);
    void ShowComplex() const;
};

Complex::Complex( int r, int i) : real(r), image(i)
{
}

void Complex::ShowComplex() const
{
    cout<<"( " <<real <<" + " <<image <<"i )" <<endl ;
}
```

```
void main() {
    Complex arr[4] = {
        Complex( 2, 4),
        Complex( 4, 8),
        Complex( 8, 16),
        Complex(16, 32),
    };

    for(int i = 0; i<4; i++)
        arr[i].ShowComplex();
}
```

04 An array of objects.

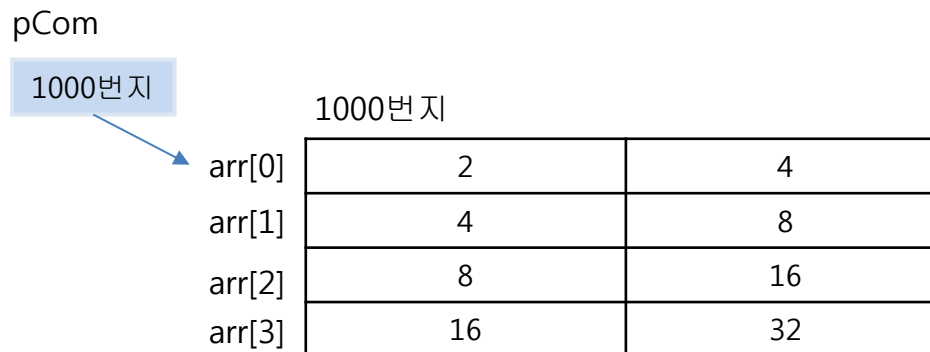
① Object arrays and pointers.

- When you save the object pointer to an array of objects and people will point to the first element of the array of objects.

```
Complex arr[4] = {  
    Complex( 2, 4),  
    Complex( 4, 8),  
    Complex( 8, 16),  
    Complex(16, 32),  
};
```

```
Complex *pCom;  
pCom = arr; // pX = &arr[0];
```

- Let's represent an object pointer to an array of objects as a picture.



04 An array of objects.

- PCom object pointer points to the beginning of the array. Therefore, Com ShowComplex call the member function output is a $(2 + 4i)$ of the arr [0] first element of the array.

```
pCom->ShowComplex();
```

- By increasing the pointer object can calculate the address of the object array element to the desired position.

```
(pCom+1)->ShowComplex(); // arr[1]의 멤버함수가 호출되어  $(4 + 8i)$ 가 출력됨
```

② Using Array object as a parameter of the function.

- By passing the starting address of the array of objects to the function pointer to an object through a pointer approaches the operation element of the object array.

Example 11-13. Indirect reference to an array of objects with object pointer (11_13.cpp)

```
#include <iostream>
using namespace std;
class Complex {
private :
    int real;
    int image;
public :
    Complex( int r=0, int i=0);
    void ShowComplex() const;
};

Complex::Complex( int r, int i) : real(r), image(i)
{
}

void Complex::ShowComplex() const
{
    cout<<"( " <<real <<" + " <<image <<"i )" <<endl ;
}
```

```
void main() {
    Complex arr[4] = {
        Complex( 2, 4),
        Complex( 4, 8),
        Complex( 8, 16),
        Complex(16, 32),
    };

    Complex *pCom = arr;

    pCom->ShowComplex();
    (pCom+1)->ShowComplex();
}
```

05 Friend functions.

- The general function can not use the private members belonging to a particular class directly. However, it is a function that allows it to friends (friend) function. Friend function is used only in exceptional cases because it is in violation of data hiding.

- To create a generic function when a friend function of a particular class is declared as follows:

friend function name (parameter list);

프렌드 함수 선언 형식

- Conditions to be friends function is as follows.
 - ① Class with a private member declares a friend who wants to access an internal function.
 - ② When you declare a friend function friend tags reserved in front of the function name.

Example 11-15. Defining friend function (11_15.cpp)

```
#include <iostream>
using namespace std;
class Complex {
private :
    int real ;
    int image;
public :
    Complex( int r=0, int i=0);
    void ShowComplex() const;

    friend void prn(Complex *pCom); // friend 함수
};

Complex::Complex( int r, int i) : real(r), image(i)
{ }

void Complex::ShowComplex() const {
    cout<<"( " <<real <<" + " <<image <<"i )" <<endl ;
}
```

```
void prn(Complex *pCom ) {
    for(int i=0; i<4; i++)
        cout<<"( " <<pCom[i].real <<" + "
            <<pCom[i].image<<"i )" <<endl ;
        // friend 일반함수에서 private 멤버를 직접 접근
    }

void main() {
    Complex arr[4] = {
        Complex( 2, 4),
        Complex( 4, 8),
        Complex( 8, 16),
        Complex(16, 32),
    };

    prn(arr); // friend 함수 호출 (일반함수로 호출)
}
```

06 Functions for manipulating the object.

① To implement a function to obtain the sum of the two objects.

- This result plus the object Complex Complex objects (20, 40) to (10, 20) so that the Define Complex objects (30, 60).
- Define the member function Sum to obtain the sum of two objects Complex looks a call as follows:

```
Complex x(10, 20), y(20, 40), z;  
z=x.Sum(y);
```

- Sum function by adding parameters of Complex Objects Complex objects x to y and returns the result. And the return value is stored in the object Complex z. Since the function name preceded by a class member function of Complex Sum Tags Complex ::

```
Complex Complex::Sum(Complex rightHand);
```

- X prepended information on the object name may have a pointer to this, information on the yarn y is passed to the parameter type parameter rightHand. Inside this function points to obtain a sum between the member variable of the real object (x) and the type parameters rightHand object (y), the sum obtained between the member variable image and stores the result in the res.

```
Complex Complex::Sum(Complex rightHand)  
{  
    Complex res;  
    res.real = this->real + rightHand.real;  
    res.image = this->image + rightHand.image;  
    return res;  
}
```

Example 11-16. Create member function adds two complex numbers (11_16.cpp)

```
#include <iostream>
using namespace std;
class Complex {
private :
    int real;
    int image;
public :
    Complex(int r=0, int i=0);
    void ShowComplex();
    Complex Sum(Complex rightHand); // 객체 덧셈 함수
};

void Complex::ShowComplex() {
    cout<<"( " <<real <<" + " <<image <<"i )" <<endl ;
}

Complex::Complex( int r, int i) {
    real=r;
    image=i;
}
```

```
Complex Complex::Sum(Complex rightHand) {
    Complex res;
    res.real = this->real + rightHand.real; // x.real + y.real
    res.image = this->image + rightHand.image;
    return res;
}

void main() {
    Complex x(10,20), y(20, 40);
    Complex z;

    z = x.Sum( y );

    x.ShowComplex();
    y.ShowComplex();
    z.ShowComplex();
}
```

06 Functions for manipulating the object.

- Using the normal function Let's save the sum of the two objects.
- General function uses the friends function because you can not call a private member variable.
- Because of the general function, not only in this member function pointer as a parameter it should be both two operands. Since the result of adding together the complex is a complex number Complex should be a class type.

```
Complex Sum(Complex leftHand, Complex rightHand)
{
    Complex res;
    res.real = leftHand.real + rightHand.real;
    res.image = leftHand.image + rightHand.image;
    return res;
}
```

Example 11-17. Creating a Friend function adds two complex numbers (11_17.cpp)

```
#include <iostream>
using namespace std;
class Complex {
private :
    int real;
    int image;
public :
    Complex(int r=0, int i=0);
    void ShowComplex();
    friend Complex Sum(Complex leftHand, Complex
rightHand); // friend 함수 선언
};
void Complex::ShowComplex() {
    cout<<"( " <<real <<" + " <<image << "i )" <<endl ;
}
Complex::Complex( int r, int i) {
    real=r;
    image=i;
}
```

```
// friend로 지정된 일반함수를 이용한 객체 덧셈
Complex Sum(Complex leftHand, Complex rightHand) {
    Complex res;
    res.real = leftHand.real + rightHand.real;
    res.image = leftHand.image + rightHand.image;
    return res;
}

void main() {
    Complex x(10,20), y(20, 40);
    Complex z;

    // 두 객체를 매개변수로 가지는 일반함수 호출
    z = Sum( x, y );

    x.ShowComplex();
    y.ShowComplex();
    z.ShowComplex();
}
```

06 Functions for manipulating the object.

② Implementing a function to increase their value by one.

- Let's define the AddOn Prefix to increase by 1 to operand prior to handling member functions.

```
Complex x(10,20), y(20, 40);  
Complex z;  
z=x.AddOnePrefix();
```

- When implemented, the operator ++ member function as AddOn Prefix function is not required because there is a pointer to specify this parameter.

```
Complex Complex::AddOnePrefix()  
{  
    ++this->real;  
    ++this->image;  
    return *this;  
}
```

- Information used as an operand of x in the function inde AddOnePrefix only this, this is a pointer because it must pass the full value of the object is not stored in the address pointed to this when you returned as a result.

```
return *this;
```

06 Functions for manipulating the object.

- Let's define a function that `AddOnePostfix` the same behavior as the `++` operator to post processing.

```
z=y.AddOnePostfix();
```

- Such as post-processing operator `++`, increases the current value of the member variables after the object you have stored on the first object and the current temp, and finally return the object before it increases in value.

```
// c=b++; ← Examples of post-processing

Complex Complex::AddOnePostfix()
{
    Complex temp;
    temp=*this;
    ++this->real;
    ++this->image;
    return temp;    // Values of the member variables increases,
                   // the object returned will return the object
                   // with the value prior to increasing
}
```

Example 11-18. Create member function to increase their value by 1 (11_18.cpp) _1

```
#include <iostream>
using namespace std;
class Complex {
private :
    int real;
    int image;
public :
    Complex(int r=0, int i=0);
    void ShowComplex();

    Complex AddOnePrefix();    // 선행처리
    Complex AddOnePostfix();  // 후행처리
};

Complex::Complex( int r, int i) {
    real=r;
    image=i;
}
```

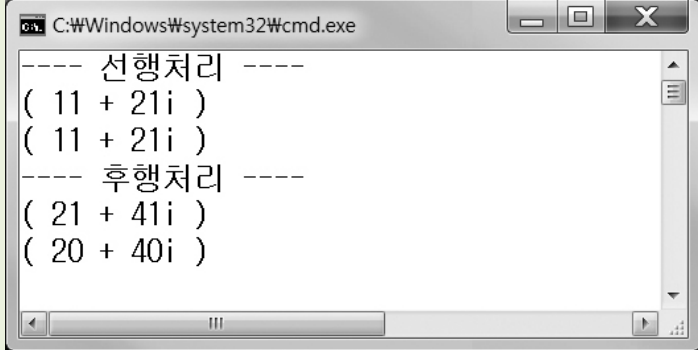
```
void Complex::ShowComplex() {
    cout<<"( " <<real <<" + " <<image <<"i )" <<endl ;
}

Complex Complex::AddOnePrefix() {
    ++this->real;
    ++this->image;
    return *this;
}

Complex Complex::AddOnePostfix() {
    Complex temp;
    temp=*this;
    ++this->real;
    ++this->image;
    return temp;
}
```


Example 11-18. Create member function to increase their value by 1 (11_18.cpp) _2

```
void main() {  
    Complex x(10,20), y(20, 40);  
    Complex z;  
  
    cout<<"---- 선행처리----\n";  
    z=x.AddOnePrefix(); // Save the value after the  
                        // increment in z  
  
    x.ShowComplex();  
    z.ShowComplex();  
  
    cout<<"---- 후행처리----\n";  
    z=y.AddOnePostfix(); // Save the value before the  
                       // increment in z  
  
    y.ShowComplex();  
    z.ShowComplex();  
}
```



```
C:\Windows\system32\cmd.exe  
---- 선행처리 ----  
( 11 + 21i )  
( 11 + 21i )  
---- 후행처리 ----  
( 21 + 41i )  
( 20 + 40i )
```

07 Operator overloading.

① Meaning of Operator Overloading

- Summarized as follows for operator overloading.
 - ① Operator overloading is to override the default operator used in C ++ data types.
 - ② In C ++ to handle even because operator functions may overload the operator with the method for defining the function.
 - ③ The overloaded operator because the operator overloading operators in the form of a function, also called function.
 - ④ When defining the operator to the operand involved in the operation is implemented as a parameter.
 - ⑤ This data type that can be used by the operator to the data type of the parameter is determined to define the operator.
- Operator function name represents the operators that were used by the default data type of an arithmetic operator with a symbol reserved.

연산자 정의 기본 형식

```
Returns operator operator (parameter 1, parameter 2, ...)  
{  
    The body of the function  
}
```

07 Operator overloading.

② Complex objects as operands to the operator overloading.

- The + operator used for numeric data, let's operator overloading to Complex operations on objects.

```
Complex x(10,20), y(20, 40), z;  
z = x + y;
```

- Two further object to describe the "z = x + y;" In other words, the form that is called by the compiler:

```
z = x.operator+(y);
```

- Is the same as the structure that calls the member function Sum to obtain the addition of a complex number.

```
z = x.Sum(y);
```

- + Definition of operator functions are also the same as the Sum function.

Implemented as a general example, the member function.	For the implementation of operator functions.
Complex Complex::Sum(Complex rightHand) { ... }	Complex Complex::operator+(Complex rightHand) { ... }

07 Operator overloading.

- Let overloading the "operator", which for the Complex class can be defined in two purposes.

```
Complex Complex::operator-(const Complex &rightHand) const
{
    Complex res;
    res.real = this->real - rightHand.real;
    res.image = this->image - rightHand.image;
    return res;
}
```

- Subtraction should be implemented with care, it is important in order for the operand location. Subtract the left side of the operands y object in the right operand of the object x, so note the position should be filled by the contents of the member function.

```
z = x - y ; // z = x.operator-(y);
```

07 Operator overloading.

- Let's define the "operator as a negative sign" rather than subtraction. In front of the variable - When the operator marks only the sign changed but parameters do not change. Complex class also to change the sign of the object-overloading should the operator.

```
Complex Complex::operator-() const
{
    Complex res;
    res.real = -real ;
    res.image = -image ;
    return res;
}
```

Example 11-20. Complex classes of operator overloading to (11_20.cpp)

_1

```
#include <iostream>
using namespace std;
class Complex
{
private :
    int real;
    int image;
public :
    Complex(int r=0, int i=0);
    void ShowComplex();
    Complex operator+(Complex rightHand);
    Complex operator-(const Complex &rightHand) const;
    Complex operator-() const;
};
```

```
void Complex::ShowComplex()
{
    if(image>0)
        cout<<"(" <<real <<"+" <<image << "i)" <<endl ;
    else if(image<0)
        cout<<"(" <<real << image << "i)" <<endl ;
    else
        cout<<real<<endl ;
}

Complex::Complex( int r, int i)
{
    real=r;
    image=i;
}
```

Example 11-20. Complex classes of operator overloading to (11_20.cpp)

_2

```
Complex Complex::operator+(Complex rightHand) {
    Complex res;
    res.real = this->real + rightHand.real;
    res.image = this->image + rightHand.image;
    return res;
}

Complex Complex::operator-(const Complex &rightHand)
const {
    Complex res;
    res.real = this->real - rightHand.real;
    res.image = this->image - rightHand.image;
    return res;
}

Complex Complex::operator-() const {
    Complex res;
    res.real = -real ;
    res.image = -image ;
    return res;
}
```

```
void main() {
    Complex x(10,20), y(20, 40), z;
    cout<<"-- 두Complex 객체에대한덧셈--\n";
    z = x + y;
    x.ShowComplex();
    y.ShowComplex();
    z.ShowComplex();

    cout<<"\n-- 두Complex 객체에대한뺄셈--\n";
    z = x - y;
    x.ShowComplex();
    y.ShowComplex();
    z.ShowComplex();

    cout<<"\n-- Complex 객체의부호변경--\n";
    z=-x;
    x.ShowComplex();
    z.ShowComplex();
}
```

07 Operator overloading.

④ When you overload the attention of the operator details.

- ① Only operators who already use in C ++ can be overloaded.
- ② Binary operator is as a binary operator, unary operators can be overloaded unary operators only. For example, such as 10%4, % operator has to form a binary operator overloading. The following is an example of the wrong.

```
int a;
```

```
% a; // The operator can not be used to obtain the rest of the unary operators.
```

- ③ Among operators used in C ++, the following operators can not be overloaded.

.(See members operator): :(scope operators)? :(Conditional operator), sizeof (sizeof operator), * (a pointer operator).

- ④ To overload an operator must be at least one operand types are user-defined.

The following is an example of the wrong.

```
double operator+(double x, double y) // Invalid operator overloading
```

- ⑤ Most operators can be overloaded as a member function or a friend function. However, the following operators can be overloaded member functions only.
= (Assignment) () (function call operator), [] (subscript assignment operator) -> (Member of the object pointer reference operator)

Homework

- Chapter 11 Exercise: 1, 2, 3, 4, 5, 6 ,7 ,8, 9