

# 객체지향언어



## 6장 클래스의 기초

# 학습목표

- 객체지향 프로그램의 특징을 설명할 수 있다.
- 클래스와 객체의 개념을 설명할 수 있다.
- 생성자와 소멸자의 개념, 특징을 설명하고 생성자, 소멸자를 사용할 수 있다.
- 클래스 객체의 대입에 대한 개념을 설명하고, 복사 생성자를 사용할 수 있다.
- 객체의 포인터를 사용할 수 있다.

# 객체지향 프로그래밍의 특징

## ■ C 언어

- 절차적 언어(Procedural Language)
- 데이터를 가공하여 결과를 구하는 함수들의 집합으로 프로그래밍
- 함수를 중심으로 프로그램을 설계하고 함수에 필요한 데이터를 정의
- 문제를 해결하기 위한 함수들의 호출로 기술되고 기술된 순서대로 수행됨

## ■ C++ 언어

- 객체지향프로그래밍(Object Oriented Programming)
- 객체로 이루어진 프로그램에서 그 객체에 일어나는 사건을 처리하는 식으로 프로그램
- 객체(데이터)를 먼저 생각하여 객체를 정의하고, 그 객체를 사용하기 위한 함수(메서드(method))를 객체 내부에 정의

# 객체지향 프로그래밍의 특징

## ■ 객체지향 프로그래밍의 특징

### ■ 캡슐화(encapsulation)

- 데이터와 데이터의 동작을 결정하는 함수(메서드)를 하나로 묶는 구조
- 데이터 은닉(data hiding) 가능
  - 캡슐화에 의해 외부의 접근을 막고 내부 정보도 숨길 수 있음
- 객체 안의 비공개(private) 멤버는 객체 외부 프로그램의 다른 부분들이 접근할 수 없고 그 객체 안에서만 접근할 수 있으므로 정보가 보호됨

### ■ 상속성(inheritance)

- 하나의 객체가 다른 객체의 특성을 이어받는 성질
- 이미 존재하는 클래스를 재활용하여 새로운 프로그램을 개발할 수 있고 개발 기간도 단축할 수 있음

### ■ 다형성(polymorphism)

- 하나의 이름으로 여러 동작을 사용할 수 있도록 해주는 것으로 함수나 연산자를 자료에 따라 다르게 동작하도록 할 수 있게 하는 성질
- 함수의 다중 정의(function overloading)
- 연산자 다중 정의(operator overloading)

# 클래스와 객체

## ■ 클래스

- C++ 언어의 중심이 되는 요소
- 어떤 목적을 위해 함수(메서드)와 변수들이 하나로 캡슐화 되어 있는 집합체

## ■ 클래스의 선언

- 메서드의 선언과 정의(구현)을 분리시킬 수 있다.

```
class 클래스_이름 {                                // 클래스의 정의
    접근_지정자 :
        자료형 멤버_변수;
    ...
    접근_지정자 :
        반환_자료형 멤버_함수();
    ...
};                                                  // 마지막에 ;을 넣어야 한다.

반환_자료형   클래스명::멤버_함수()                // 멤버 함수 정의
{
    ...
}
```

# 클래스와 객체

## ■ 접근 지정자

접근 지정자	접근 권한
public	동일한 클래스 내의 메서드, 다른 외부 클래스, 외부 메서드
private	동일한 클래스 내의 메서드
protected	동일한 클래스 내의 메서드, 그 클래스로부터 상속받은 메서드

## ■ 클래스

- 일종의 사용자 정의 자료형
- 기억공간을 배정받지 않는 자료의 구조와 동작 방법을 정해 둔 자료형

## ■ 객체(object, instance)

- 클래스를 이용하여 만들어진 클래스형 자료
- 기억공간을 배정받는 실체로,
- 멤버 변수만이 기억공간을 배정받고,
- 멤버 함수(메서드)는 객체의 동작을 정하는 것으로 클래스에서 공유

# 클래스와 객체

## ■ 캡슐화(encapsulation)

- 데이터와 함수를 클래스 안에 넣어서 묶는 것을 의미
- 프로그래머가 사용하기 편하고 재사용하기도 쉬움
- 멤버 변수
  - 보통 **private**로 지정하여 외부에서 함부로 접근 할 수 없게 만들어 정보를 은닉하여 보호
- 멤버 함수
  - 보통 **public**으로 지정하여 외부에서 정보를 주고받는 통로로 사용

## ■ 객체 멤버의 접근 방법

객체명.멤버_변수;	// . 멤버 참조 연산자 사용
객체명.멤버_함수;	// . 멤버 참조 연산자 사용
객체_포인터->멤버_변수;	// -> 멤버 참조 연산자 사용
객체_포인터->멤버_함수;	// -> 멤버 참조 연산자 사용

# 클래스와 객체

## ■ ex6\_1.cpp (1) (클래스의 예)

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

class Rectangle                                // 클래스 선언
{
private :                                     // 멤버변수의 접근 지정자: private
    int width;
    int height;
public :                                     // 멤버함수의 접근 지정자: public
    void GetRect(int w, int h);
    void ShowRect();
};

void Rectangle::GetRect(int w, int h)          // 메서드를 분리하여 정의
{
    width = w;
    height = h;
}

void Rectangle::ShowRect()                    // 메서드를 분리하여 정의
{
    cout << "width = " << width << ", height = " << height << endl;
}
```



# 클래스와 객체

## ■ ex6\_1.cpp (2) (클래스의 예)

```
void main()
{
    Rectangle r;           // 객체 선언
    // r.width = 5;        // 컴파일 오류
    // r.height = 4;       // 컴파일 오류
                           // 멤버변수는 private이므로 직접 접근은 안 된다.
    r.GetRect(5, 4);       // 메서드를 사용
    r.ShowRect();          // 메서드를 사용
}
```

```
width = 5, height = 4
계속하려면 아무 키나 누르십시오 . . .
```

# 클래스와 객체

## ■ inline 멤버 함수

- 인라인 함수를 사용하면 함수의 복사본이 직접 삽입, 실행 속도가 개선
- 함수 선언 앞에 inline 키워드를 사용

## ■ const 멤버 함수

- 클래스의 멤버 함수를 정의할 때 함수 이름 뒤에 const를 붙이면, 멤버 함수 내에서 어떠한 멤버 변수 값도 변경할 수 없음
- 멤버 변수 값이 변하지 않는다는 것을 보장하므로 함수(메서드) 내에서 값이 변경되는 것을 예방할 수 있음

```
class Vehicle
{
    ...
    inline void ShowNumber( ) {                // inline 멤버 함수
        cout << number << endl;
    }
    void ShowYear( ) const {                    // const 멤버 함수
        cout << year << endl;
    }
};
```

## ■ ex6\_2.cpp (1) (inline과 const 멤버 함수를 사용한 예)

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

class Vehicle
{
    int number;           // 접근 지정자를 지정하지 않으면 private가 된다.
    int year;
public :
    void Register(int num, int yr) ;
    inline void ShowNumber( ) const {
        cout << "Number : " << number << endl;
    };
    inline void ShowYear( ) const {
        cout << "Year : " << year << endl;
    };
};

void Vehicle::Register(int num, int yr) {
    number = num;
    year = yr;
}
```

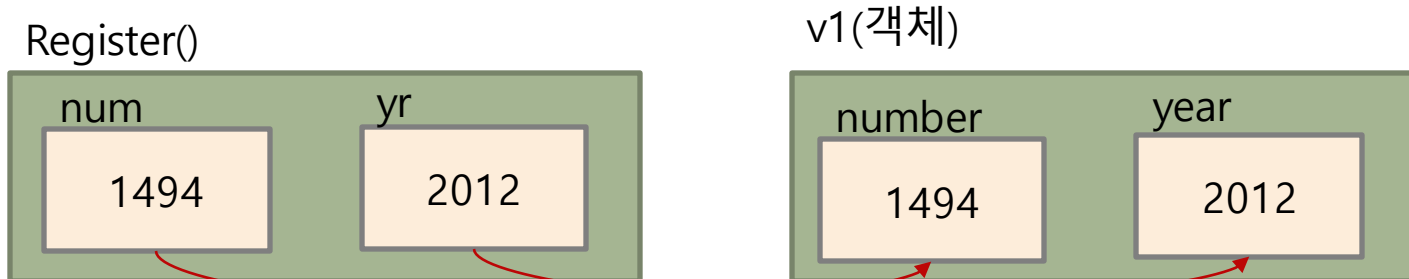
# 클래스와 객체

## ■ ex6\_2.cpp (2) (inline과 const 멤버 함수를 사용한 예)

```
void main()
{
    Vehicle v1;
    int n, y;
    cout << "차량번호와 연식을 입력하세요: " ;
    cin >> n >> y;

    v1.Register(n, y);           //일반 멤버 함수
    v1.ShowNumber();             //inline과 const 멤버 함수
    v1.ShowYear();               //inline과 const 멤버 함수
}
```

```
차량번호와 연식을 입력하세요 : 1494 2012
Number : 1494
Year : 2012
계속하려면 아무 키나 누르십시오 . . .
```



## ■ 생성자(constructor)

- 생성자는 객체를 생성하고 초기화하기 위한 멤버함수
- 객체가 선언될 때 **컴파일러에 의해 자동으로 호출**되어 객체를 생성
- 프로그래머가 명시적으로 생성자를 만들지 않으면 C++ 컴파일러가 매개변수가 없는 생성자, 즉 **기본 생성자(default constructor)**를 만들어 줌
- 생성자의 이름은 클래스 이름과 동일하고, 반환 자료형을 지정하지 않음
- 생성자는 매개변수를 달리하여 **다중 정의(overloading)가 가능**

## ■ 소멸자(destructor)

- 객체가 소멸될 때 자동으로 호출되어 객체의 마지막을 마무리해 줌
- 프로그래머가 명시적으로 소멸자를 만들지 않으면 C++ **컴파일러가 디폴트로 만들어 줌**
- 소멸자의 이름은 클래스 이름과 동일하지만 앞에 **틸드(~)**기호를 붙이고, 반환 자료형을 지정하지 않음
- 소멸자는 **전달 매개변수를 지정할 수 없고, 다중 정의도 할 수 없음**

# 생성자와 소멸자

## ■ 생성자와 소멸자 사용

```
class Vehicle
{
private :
    int  number;
    int  year;
public :
    Vehicle( ) { }                // 기본 생성자
    Vehicle(int num, int yr) { number = num; year = yr; } // 일반 생성자
    ~Vehicle( ) { }              // 소멸자
};

void main()
{
    Vehicle v1;                  // 기본 생성자 호출
    Vehicle v2(1234, 2012);      // 일반 생성자 호출
    ...
}
```

## ■ ex6\_3.cpp (1) (생성자와 소멸자 사용 예)

```
#include <iostream>
using namespace std;
class Vehicle
{
private :
    int number;
    int year;

public :
    Vehicle();                // 기본 생성자
    Vehicle(int num, int yr); // 일반 생성자
    ~Vehicle();               // 소멸자
    void ShowVehicle() const ;

};

Vehicle::Vehicle()
{
    cout << "Vehicle constructor" << endl;
    number = 0, year = 0;
}
```



## ■ ex6\_3.cpp (2) (생성자와 소멸자 사용 예)

```
Vehicle::Vehicle(int num, int yr) {  
    cout << "Vehicle constructor with number and year" << endl;  
    number = num;  
    year = yr;  
}  
Vehicle::~~Vehicle() {  
    cout << "Vehicle destructor of " << number << endl;  
}  
void Vehicle::ShowVehicle() const {  
    cout << "Number : " << number << ", Year : " << year << endl;  
}  
void main() {  
    // 기본 생성자 호출시에는 v1()과 같이 괄호를 사용해서는 안 된다.  
    Vehicle v1;  
    Vehicle v2(1234, 2012);  
  
    v1.ShowVehicle();  
    v2.ShowVehicle();  
}
```

```
Vehicle constructor  
Vehicle constructor with number and year  
Number : 0, Year : 0  
Number : 1234, Year : 2012  
Vehicle destructor of 1234  
Vehicle destructor of 0  
계속하려면 아무 키나 누르십시오 . . .
```

## ■ 생성자의 디폴트 매개변수 값 지정하기

- 생성자 선언에서 디폴트 매개변수 값을 지정하여 기술하는 경우, 실매개변수 없이 호출하면 디폴트 매개변수 값으로 초기화되어 객체가 생성
- 함수에 디폴트 매개변수를 지정하는 인수는 다른 인수들보다 뒤에 와야 함

```
Vehicle(int n=0, int y=0);    // 디폴트 매개변수 값을 지정
Vehicle v1(1234, 2012);      // number=1234, year=2012
Vehicle v2(1234);            // number=1234, year=0
Vehicle v3;                   // number=0, year=0
```

# 생성자와 소멸자

## ■ 생성자의 초기화 섹션

- 생성자 코드에서 콜론(:)부터 중괄호({} 전까지를 초기화 섹션이라고 함
- 콜론 뒤에 초기화할 변수들을 쉼표로 나열하여 초기화할 수 있음

```
클래스명::생성자(인수, 인수) : 멤버_변수(인수), 멤버_변수(인수)
{
    ...
}
```

[예]

```
Vehicle::Vehicle(int n, int y) : number(n), year(y) { }
```

## ■ ex6\_4.cpp (1) (생성자의 디폴트 매개변수 값 지정과 초기화 섹션 사용)

```
#include <iostream>
using namespace std;
class Vehicle
{
private :
    int number;
    int year;
public :
    Vehicle(int num, int yr);      // 생성자
    ~Vehicle();                   // 소멸자
    void ShowVehicle() const ;
};

//생성자에 디폴트 매개변수 지정, 초기화 섹션 사용
Vehicle::Vehicle(int n=0, int y=0) : number(n), year(y) {

}
```

## ■ ex6\_4.cpp (2) (생성자의 디폴트 매개변수 값 지정과 초기화 섹션 사용)

```
Vehicle::~Vehicle(){

}

void Vehicle::ShowVehicle() const{
    cout << "Number : " << number << ", Year : " << year << endl ;
}

void main()
{
    Vehicle v1;
    Vehicle v2(1234);
    Vehicle v3(1234, 2012);

    v1.ShowVehicle();
    v2.ShowVehicle();
    v3.ShowVehicle();
}
```

```
Number : 0, Year : 0
Number : 1234, Year : 0
Number : 1234, Year : 2012
계속하려면 아무 키나 누르십시오 . . .
```

# 클래스 객체간의 대입

## ■ 클래스 객체의 비트 단위 치환(대입)

```
int x = 10, y;
```

```
y = x;           // 변수 x의 값이 y에 대입된다 .
```

```
                // y의 값은 x의 값으로 치환된다.
```

- 하나의 객체의 값이 다른 객체의 값으로 치환될 때 기본적으로 모든 멤버 변수들은 비트 단위로 복사
- 단순히 두 객체의 데이터는 동일하게 됨
- 따라서 치환된 후에 원하는 데로 동작을 하지 않을 수가 있으므로 주의해야 함

# 클래스 객체간의 대입

## ■ ex6\_5.cpp (1) (클래스 객체의 비트 단위 대입)

```
#include <iostream>
#include <cstring>          // 문자열 처리함수 사용을 위한 헤더파일
using namespace std;

class Vehicle
{
private :
    int number;
    char *owner;            // 문자열 저장을 위한 포인터 변수 선언
public :
    Vehicle() {             // 기본 생성자
        number = 0;
        owner = NULL;
    }
```

# 클래스 객체간의 대입

## ■ ex6\_5.cpp (2) (클래스 객체의 비트 단위 대입)

```
Vehicle(int n, char* m) {           // 일반 생성자
    number = n;
    int c = strlen(m);
    owner = new char[c+1];          // 동적 메모리 공간 할당
    strcpy_s(owner, c+1, m);        // 문자열 복사 함수로 문자열 복사
}

void set(int n, char *m) {
    number = n;
    strcpy_s(owner, strlen(m)+1, m); // 문자열 복사 함수로 문자열 복사
}

void show() {
    cout << "number : " << number << ", owner : " << owner ;
    cout << endl;
}

};
```



# 클래스 객체간의 대입

## ■ ex6\_5.cpp (3) (클래스 객체의 비트 단위 대입)

```
void main()
{
    Vehicle x(1234, "Kim");
    cout << "x ==> " ;
    x.show() ;

    Vehicle y;
    y = x;
    cout << "y ==> " ;
    y.show() ;

    x.set(2345, "Hong");
    cout << " -----x.set( )----- " << endl;
    cout << "x ==> " ;
    x.show();
    cout << "y ==> " ;
    y.show() ;
}
```

```
x ==> number : 1234, owner : Kim
y ==> number : 1234, owner : Kim
-----x.set( )-----
x ==> number : 2345, owner : Hong
y ==> number : 1234, owner : Hong
계속하려면 아무 키나 누르십시오 . . .
```

# 클래스 객체간의 대입

## ■ 클래스 객체의 비트 단위 대입

- 클래스 객체의 치환에서 비트 단위로 복사의 문제점
  - Vehicle y; y = x; 에 의해 멤버의 값이 그대로 복사되므로 owner가 가지는 주소가 같게 되고 따라서 같은 메모리를 가리키게 되어 같은 값을 가짐
  - X.set(2345, "Hong");에 의해 x의 값이 변경되지만, y.owner도 x.owner와 같은 메모리를 가리키고 있으므로 y.owner도 바뀜

Vehicle x(1234, "Kim");

x 객체	
number	1234
owner	char *

Vehicle y; y = x;

y 객체	
number	1234
owner	char *

"Kim"

x.set(2345, "Hong");

x 객체	
number	2345
owner	char *

y 객체	
number	1234
owner	char *

"Hong"

# 클래스 객체간의 대입

## ■ 클래스 객체의 값에 의한 대입

- 객체를 같은 클래스의 다른 객체에 대입할 때, 값에 의한 전달이 되도록 멤버 함수를 만들어야 함
- `owner = new char[strlen(src.owner)+1];` 에 의해
  - 객체 y의 owner가 참조할 기억장소를 새로 동적으로 생성함으로써, 객체 x와 객체 y의 owner가 참조하는 기억장소를 별개로 함
- `Strcpy_s(owner, strlen(src.owner)+1, src.owner);` 에 의해
  - 객체 x의 문자열 데이터를 객체 x의 문자열 데이터로 복사해 넣음

y.CopyVehicle(x);

x 객체	
number	2345
owner	char *

“Kim”

y 객체	
number	1234
owner	char *

“Kim”

# 클래스 객체간의 대입

## ■ ex6\_6.cpp (1) (클래스 객체의 값에 의한 치환의 예)

```
#include <iostream>
#include <cstring>           // 문자열 처리 함수 사용을 위한 헤더 파일
using namespace std;
class Vehicle
{
private :
    int number;
    char *owner;             // 문자열 저장을 위한 포인터 변수
public :
    Vehicle() {              // 기본 생성자
        number = 0;
        owner = NULL;
    }
    Vehicle(int n, char* na) { // 일반 생성자
        number = n;
        owner = new char[strlen(na)+1]; // 동적 메모리 공간 할당
        strcpy_s(owner, strlen(na)+1, na); // 문자열 복사
    }
}
```

# 클래스 객체간의 대입

## ■ ex6\_6.cpp (2) (클래스 객체의 값에 의한 치환의 예)

```
~Vehicle() {      delete [] owner;      }      // 소멸자
void CopyVehicle(const Vehicle &src) {
    number = src.number;
    delete [] owner;      // 동적메모리 공간 해제
    owner = new char[strlen(src.owner)+1] ; // 동적 메모리 공간 할당
    strcpy_s(owner, strlen(src.owner)+1, src.owner);      //문자열 복사
}
void set(int n, char *na) {
    number = n;
    delete [] owner;      // 동적메모리 공간 해제
    owner = new char[strlen(na)+1];      // 동적 메모리 공간 할당
    strcpy_s(owner, strlen(na)+1, na);      //문자열 복사
}
void show() {
    cout << "number : " << number << ", owner : " << owner ;
    cout << endl;
}
};
```

# 클래스 객체간의 대입

## ■ ex6\_6.cpp (3) (클래스 객체의 값에 의한 치환의 예)

```
void main()
{
    Vehicle x(1234, "Kim");
    cout << "x ==> " ;
    x.show();

    Vehicle y(5678, "Park");
    y.CopyVehicle(x);
    cout << "y ==> " ;
    y.show();

    cout << " -----x.set( )----- " << endl ;
    x.set(2345, "Hong");
    cout << "x ==> " ;
    x.show();
    cout << "y ==> " ;
    y.show();
}
```

```
x ==> number : 1234, owner : Kim
y ==> number : 1234, owner : Kim
-----x.set( )-----
x ==> number : 2345, owner : Hong
y ==> number : 1234, owner : Kim
계속하려면 아무 키나 누르십시오 . . .
```

# 클래스 객체간의 대입

## ■ 얕은 복사와 깊은 복사

### ■ 얕은 복사 (shallow copy)

- 포인터 멤버 변수를 복사할 때 비트 그대로 복사하여, 주소가 그대로 복사되어 결국 같은 메모리를 가리키게 되는 복사

### ■ 깊은 복사(deep copy)

- 별개의 기억공간을 생성하고, 참조하는 값을 복사하므로 별개의 기억공간을 갖는 복사 방법

## ■ 복사 생성자 (1)

### ■ Vehicle x(1234, "Kim"); 와 같이 선언한다면 일반 생성자를 호출

- Vehicle y = x; 와 같이 선언한다면 기본 생성자를 호출하는 것이 아니고 복사 생성자 (copy constructor)를 호출한

### ■ 디폴트 복사 생성자

- 프로그래머가 복사 생성자를 만들어 주지 않으면 C++ 컴파일러가 디폴트로 복사 생성자를 제공
- 디폴트로 제공되는 복사 생성자는 모든 멤버 변수들이 **비트 단위로 복사되어**, 단순히 두 객체의 데이터는 동일하게 됨

# 클래스 객체간의 대입

## ■ ex6\_7.cpp (1) (디폴트 복사 생성자의 동작)

```
#include <iostream>
#include <cstring>
using namespace std;
class Vehicle
{
private :
    int number;      char *owner;
public :
    Vehicle() {                                // 기본 생성자
        cout << "기본 생성자 호출" << endl;
        number = 0;
        owner = NULL;
    }
    Vehicle(int n, char* m) {                  // 일반 생성자
        cout << "일반 생성자 호출" << endl;
        number = n;
        owner = new char[strlen(m)+1];
        strcpy_s(owner, strlen(m)+1, m);
    }
}
```



# 클래스 객체간의 대입

## ■ ex6\_7.cpp (2) (디폴트 복사 생성자의 동작)

```
void set(int n, char *m) {
    number = n;
    strcpy_s(owner, strlen(m)+1, m);
}

void show() {
    cout << "number : " << number << ", owner : " << owner ;
    cout << endl;
}

};

void main() {
    Vehicle x(1234, "Kim");           // 일반 생성자 호출
    Vehicle y = x;                   // 복사 생성자 호출
    cout << " -----x.set( )----- " << endl;
    x.set(2345, "Hong");
    cout << "x ==> " ;    x.show();
    cout << "y ==> " ;    y.show();
}
```

```
일반 생성자 호출
-----x.set( )-----
x ==> number : 2345, owner : Hong
y ==> number : 1234, owner : Hong
계속하려면 아무 키나 누르십시오 . . .
```

# 클래스 객체간의 대입

## ■ 복사 생성자 (2)

### ■ 복사 생성자 선언

- 값에 의한 복사가 되도록, C++ 컴파일러가 디폴트로 만들어 주는 복사 생성자가 호출되지 않도록 **프로그래머가 명시적으로 복사 생성자를 작성해** 주어야 함

```
클래스_이름(const 클래스_이름 &소스_객체);
```

### ■ [예]

```
Vehicle(const Vehicle &src);
```

# 클래스 객체간의 대입

## ■ ex6\_8.cpp (1) (복사 생성자의 사용 예)

```
#include <iostream>
#include <cstring>
using namespace std;
class Vehicle
{
private :
    int number;      char *owner;
public :
    Vehicle() {                                // 기본 생성자
        cout << "기본 생성자 호출" << endl;
        number = 0;
        owner = NULL;
    }
    Vehicle(int n, char* na) {                  // 일반 생성자
        cout << "일반 생성자 호출" << endl;
        number = n;
        owner = new char[strlen(na)+1];
        strcpy_s(owner, strlen(na)+1, na);
    }
}
```

# 클래스 객체간의 대입

## ■ ex6\_8.cpp (2) (복사 생성자의 사용 예)

```
Vehicle(const Vehicle &src) {                // 복사 생성자
    cout << "복사 생성자 호출" << endl;
    number = src.number;
    owner = new char[strlen(src.owner)+1];
    strcpy_s(owner, strlen(src.owner)+1, src.owner);
}
~Vehicle() {                                // 소멸자
    cout << "소멸자 호출" << endl;
    delete[] owner;
}
void set(int n, char *na) {
    number = n;
    delete [] owner;
    owner = new char[strlen(na)+1];
    strcpy_s(owner, strlen(na)+1, na);
}
```

## ■ ex6\_8.cpp (3) (복사 생성자의 사용 예)

```
void show() {  
    cout << "number : " << number << ", owner : " << owner ;  
    cout << endl;  
}  
};    //클래스 끝  
void main()  
{  
    Vehicle x(1234, "Kim");           // 일반 생성자 호출  
    Vehicle y = x;                   // 복사 생성자 호출  
    Vehicle z(x);                   // 복사 생성자 호출  
  
    cout << "x ==> " ;    x.show();  
    cout << "y ==> " ;    y.show();  
    cout << "z ==> " ;    z.show();  
}
```

# 클래스 객체간의 대입

## ■ ex6\_8.cpp (4) (복사 생성자의 사용 예)

```
cout << " -----x.set( )----- " << endl;
x.set(2345, "Hong");
cout << "x ==> " ;    x.show();
cout << "y ==> " ;    y.show();
cout << "z ==> " ;    z.show();

}
```

- new 연산자에 의해 기억장소를 동적으로 생성하면 소멸자를 추가하여, 동적 생성한 기억장소를 delete 연산자로 해제하여 메모리의 누수를 방지해야 한다.
- 실행 결과

```
일반 생성자 호출
복사 생성자 호출
복사 생성자 호출
x ==> number : 1234, owner : Kim
y ==> number : 1234, owner : Kim
z ==> number : 1234, owner : Kim
-----x.set( )-----
x ==> number : 2345, owner : Hong
y ==> number : 1234, owner : Kim
z ==> number : 1234, owner : Kim
소멸자 호출
소멸자 호출
소멸자 호출
계속하려면 아무 키나 누르십시오 . . .
```

# 객체 포인터

## ■ 객체에서 멤버참조

- 점(.) 연산자를 사용

## ■ 객체 포인터에서 멤버참조

- 화살표(->) 연산자를 사용

[예]

```
void main()
{
    Vehicle v1(1234, 2012);
    Vehicle v2;
    v1.ShowVehicle();      // 객체의 멤버 참조
    v2.ShowVehicle();      // 객체의 멤버 참조

    Vehicle *pv;
    pv = &v1;
    pv->ShowVehicle() ;    // 객체 포인터의 멤버 참조
    pv = &v2;
    pv->ShowVehicle() ;    // 객체 포인터의 멤버 참조
}
```

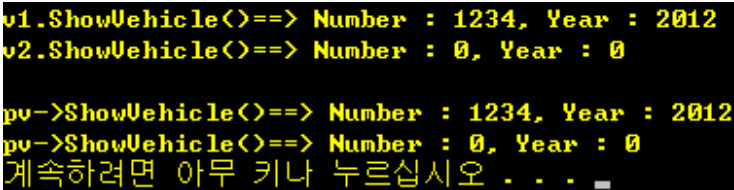
## ■ ex6\_9.cpp (1) (객체와 객체 포인터의 멤버 참조 예)

```
#include <iostream>
using namespace std;
class Vehicle
{
private :
    int number;
    int year;
public :
    Vehicle(int num=0, int yr=0);           //일반생성자
    ~Vehicle();                             //소멸자
    void ShowVehicle() const ;
};
Vehicle::Vehicle(int n, int y):number(n), year(y) {
}
Vehicle::~~Vehicle(){
}
```



## ■ ex6\_9.cpp (2) (객체와 객체 포인터의 멤버 참조 예)

```
void Vehicle::ShowVehicle() const {  
    cout << "Number : " << number << ", Year : " << year << endl;  
}  
void main()  
{  
    Vehicle v1(1234, 2012);  
    Vehicle v2;  
    cout << "v1.ShowVehicle()" << "==" << " ;    v1.ShowVehicle();  
    cout << "v2.ShowVehicle()" << "==" << " ;    v2.ShowVehicle();  
    cout << endl;  
  
    Vehicle *pv;  
    pv = &v1;  
    cout << "pv->ShowVehicle()" << "==" << " ;    pv->ShowVehicle() ;  
    pv = &v2;  
    cout << "pv->ShowVehicle()" << "==" << " ;    pv->ShowVehicle() ;  
}
```



```
v1.ShowVehicle()=> Number : 1234, Year : 2012  
v2.ShowVehicle()=> Number : 0, Year : 0  
  
pv->ShowVehicle()=> Number : 1234, Year : 2012  
pv->ShowVehicle()=> Number : 0, Year : 0  
계속하려면 아무 키나 누르십시오 . . .
```

## ■ this 포인터

- 객체가 멤버 함수를 호출하여 함수 내에서 멤버 변수를 사용할 때, 멤버 변수는 멤버이므로 '**객체명.멤버**' 또는 '**객체포인터->멤버**'와 같이 누구 (어느 객체)의 멤버인지를 명시해야 함
- this 포인터는 멤버 함수를 호출한 객체를 가리킴
- this 포인터는 객체가 멤버 함수를 호출할 때, 그 멤버 함수 내에서 호출 객체의 주소를 저장하는 포인터로 멤버 함수를 호출한 객체를 가리키는 데 사용됨
- this 포인터는 프로그래머가 명시적으로 선언하지 않으면 컴파일러가 묵시적으로 this 포인터를 제공함

## ■ ex6\_10.cpp (1) (this 포인터 사용 예)

```
#include <iostream>
using namespace std;
class Vehicle
{
private :
    int number;
    int year;
public :
    Vehicle(int number=0, int year=0);
    ~Vehicle();
    void ShowVehicle() const ;
};

Vehicle::Vehicle(int number, int year) {
    this->number = number; // this->number의 number는 멤버변수 number이고,
                           // 뒤의 number는 객체 생성할 때 주는 값이다.
    this->year = year;     // this->year의 year는 멤버변수 year이고,
                           // 뒤의 year는 객체 생성할 때 주는 값이다.
}
```

## ■ ex6\_10.cpp (2) (this 포인터 사용 예)

```
Vehicle::~Vehicle() {  
}  
  
void Vehicle::ShowVehicle() const {  
    cout << "Number : " << this->number << ", Year : " << this->year << endl;  
}  
  
void main()  
{  
    Vehicle v1(1234, 2012);  
    cout << "v1.ShowVehicle()" << "==> " ;  
    v1.ShowVehicle() ;  
    cout << endl;  
  
    Vehicle *pv;  
    pv = &v1;  
    cout << "pv->ShowVehicle()" << "==> " ;  
    pv->ShowVehicle() ;  
}
```

```
v1.ShowVehicle<>==> Number : 1234, Year : 2012
```

```
pv->ShowVehicle<>==> Number : 1234, Year : 2012
```

```
계속하려면 아무 키나 누르십시오 . . .
```



Thank You

---