객체지향언어

7장 클래스의 활용

학습목표

- 객체 배열을 사용할 수 있다.
- 클래스 객체를 함수의 인수로 전달할 수 있다.
- 정적 멤버변수와 정적 멤버함수를 설명하고 사용할 수 있다.
- 프렌드 함수를 설명하고 사용할 수 있다.
- 객체의 동적 할당 방법을 설명할 수 있다.
- 객체를 클래스의 멤버로 사용할 수 있다.

객체의 배열

객체 배열의 선언

```
클래스_이름  객체_배열명[원소의_개수];
```

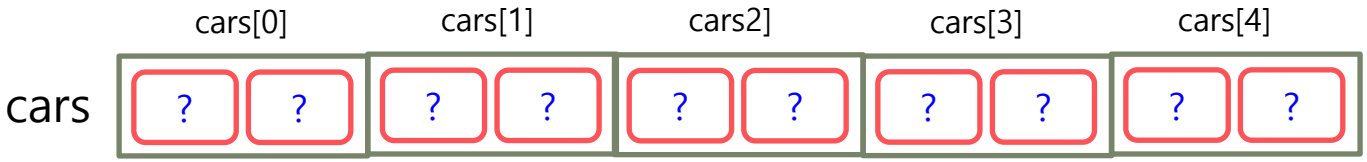
- 첨자는 일반 배열과 같이 0부터 시작한다.
- 객체 배열의 원소를 초기화하기 위하여 생성자를 명시적으로 호출하고 생성자를 호출할 때 인수 값을 준다.
- 각 배열의 원소마다 서로 다른 생성자를 호출할 수 있다.

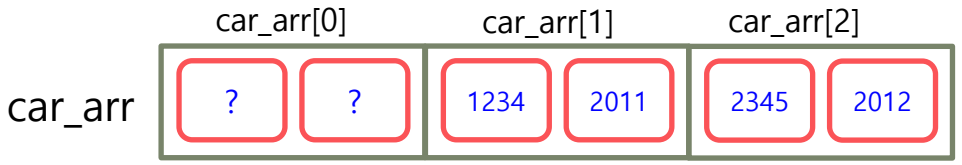
- 배열 이름이 **배열의 주소**가 된다.
- 객체 포인터에 객체 배열의 이름을 대입하면 객체 포인터는 객체 배열의 첫 번째 원소를 가리키게 된다.
- 객체 포인터로 멤버변수에 접근하려면 **화살표 연산자(->)를 사용**한다.

객체의 배열

객체 배열의 선언 예

```
class Car {
        int number;
        int year;
public:
        Car();
         Car(int n, int y) : number(n), year(y) {  };
};
void  main()
{
        Car  cars[5];                          // 초기화하지 않고 배열만 선언
        Car  car_arr[3] = { Car(), Car(1234, 2011), Car(2345, 2012) };
                                               // 배열 선언과 동시에 초기화
        Car *pcar = car_arr ;
}
```





객체의 배열

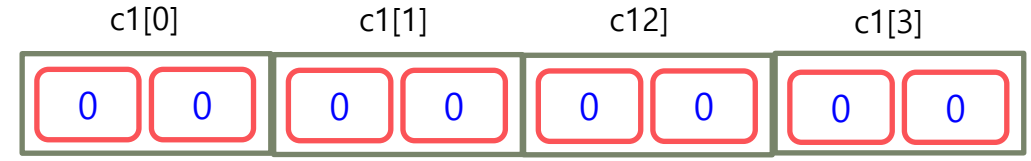
- **ex7_1.cpp (1) (객체 배열의 사용 예)**

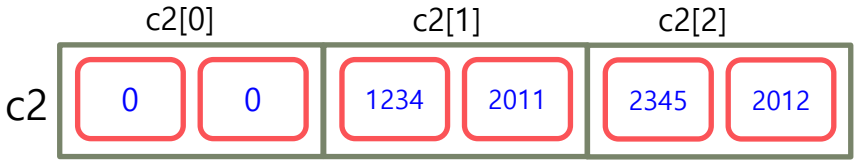
```
#include <iostream>
using namespace std;

class Car
{
private:
        int number;
        int year;
public:
        Car() : number(0), year(0) { };                    //기본생성자
        Car(int n, int y) : number(n), year(y) {  };       //일반생성자
        void showCar();
};
void Car::showCar()
{
        cout << "번호: " << number << ", 년도: " << year << endl;
}
```

객체의 배열

- **ex7_1.cpp (2) (객체 배열의 사용 예)**

```
void main()
{
        int i;
        Car c1[4];
        Car c2[3] = { Car(), Car(1234, 2011), Car(2345, 2012)};

        for (i=0; i<4; i++) {
                cout << "c1[" << i << "]==> " ;
                c1[i].showCar();
        }
        cout << endl;
        for (i=0; i<3; i++) {
                cout << "c2[" << i << "]==> " ;
                c2[i].showCar();
        }
}
```






```
c1[0]==> 번호: 0, 년도: 0
c1[1]==> 번호: 0, 년도: 0
c1[2]==> 번호: 0, 년도: 0
c1[3]==> 번호: 0, 년도: 0

c2[0]==> 번호: 0, 년도: 0
c2[1]==> 번호: 1234, 년도: 2011
c2[2]==> 번호: 2345, 년도: 2012
계속하려면 아무 키나 누르십시오 . . .
```

객체의 배열

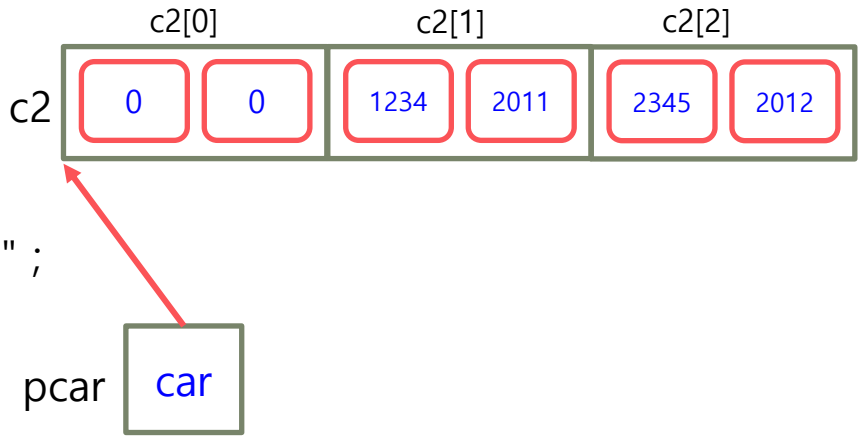
- **ex7_2.cpp (1) (객체 포인터의 사용 예)**

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

class Car
{
private:
        int number;
        int year;
public:
        Car() : number(0), year(0) { };
        Car(int n, int y) : number(n), year(y) {    };
        void showCar();
};
void Car::showCar()
{
        cout << "번호: " << number << ", 년도: " << year << endl;
}
```

객체의 배열

- **ex7_2.cpp (2) (객체 포인터의 사용 예)**

```
void main()
{
        int i;
        Car c2[3] = { Car(), Car(1234, 2011), Car(2345, 2012)};
        Car *pcar = car;

        for (i=0; i<3; i++) {
                cout << "(pcar + " << i << ") ==> " ;
                (pcar + i)->showCar();
        }
}
```




```
(pcar + 0) ==> 번호: 0, 년도: 0
(pcar + 1) ==> 번호: 1234, 년도: 2011
(pcar + 2) ==> 번호: 2345, 년도: 2012
계속하려면 아무 키나 누르십시오 . . .
```

객체의 배열

■ 함수의 인수(매개변수)로 객체 배열 전달

- 객체 배열을 함수의 인수로 전달하려면 **객체 배열의 이름을 실인수로 전달**

- 호출되는 함수에서 가인수로 객체 배열을 전달받는 방법

(1) 가인수를 전달받는 객체 배열과 같은 자료형의 배열로 선언하는 방법

- 호출함수에서는 객체 배열로 처리
- 가인수에서 객체 배열의 크기는 지정할 필요가 없음

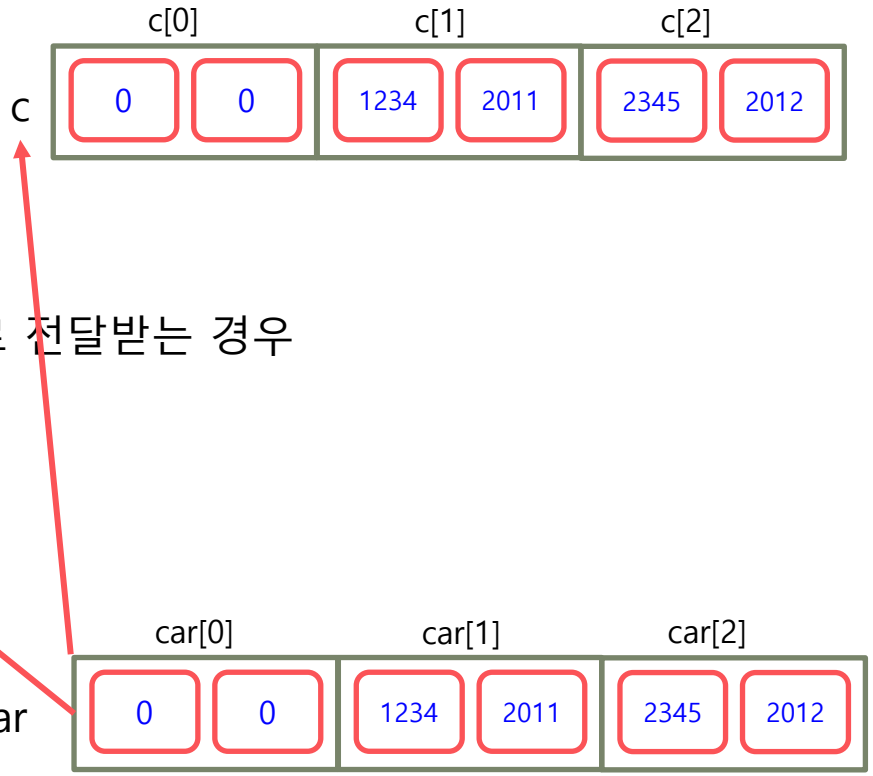
(2) 호출되는 함수의 가인수를 객체 포인터로 전달받는 방법

- 전달되는 객체 배열의 시작주소가 객체 포인터 가인수에 저장
- 가인수를 객체 포인터로 하여 객체 배열의 원소를 처리

객체의 배열

- **함수의 인수로 객체 배열 전달 예**

```
void print1 (Car  c[] )          // 객체배열로 전달받는 경우
{
        int  i;
        for (i=0; i<3; i++)
                c[i].showCar();
}
void print2( Car  *c )           // 객체포인터로 전달받는 경우
{
        int  i;
        for (i=0; i<3; i++)
           (c + i)->showCar();
}
void main()
{
        Car car[3] = { Car(), Car(1234, 2011), Car(2345, 2012)};
        print1( car );                  // 객체배열의 이름을 전달
        print2( car );                  // 객체배열의 이름을 전달
}
```



객체의 배열

- **ex7_3.cpp (1) (함수의 인수로 객체 배열 전달 예)**

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

class Car
{
private:
        int number;
        int year;
public:
        Car() : number(0), year(0) { };
        Car(int n, int y) : number(n), year(y) { };
        void showCar();
};
void Car::showCar()
{
        cout << "번호: " << number << ", 년도: " << year << endl;
}
```

객체의 배열

- **ex7_3.cpp (2) (함수의 인수로 객체 배열 전달 예)**

```
void print1( Car c[] )          // 객체 배열로 전달받는 경우
{
        int i;
        for (i=0; i<3; i++) {
                cout << "car[" << i << "].showCar() ==> " ;
                c[i].showCar();
        }
}
void print2( Car *c )           // 객체 포인터로 전달받는 경우
{
        int i;
        for (i=0; i<3; i++) {
                cout << "(car + " << i << ")->showCar() ==> " ;
                (c + i)->showCar();
        }
}
```

객체의 배열

- **ex7_3.cpp (3) (함수의 인수로 객체 배열 전달 예)**

```
void main()
{
        Car car[3] = { Car(), Car(1234, 2011), Car(2345, 2012)};
        print1( car );          // 객체 배열의 이름을 전달
        cout << endl;
        print2( car );          // 객체 배열의 이름을 전달
}
```

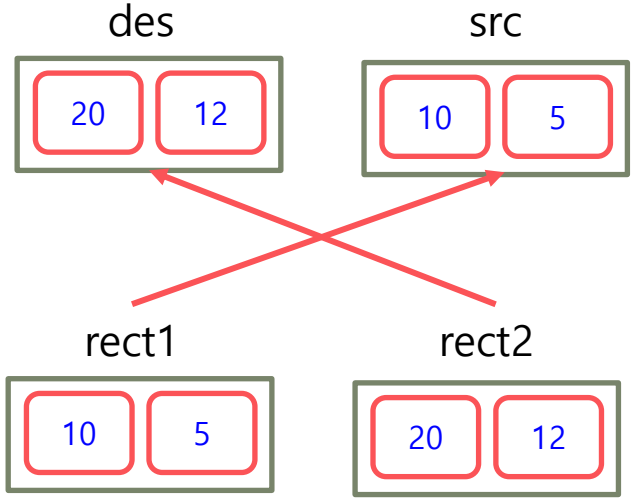
```
car[0].showCar() ==> 번호: 0, 년도: 0
car[1].showCar() ==> 번호: 1234, 년도: 2011
car[2].showCar() ==> 번호: 2345, 년도: 2012

(car + 0)->showCar() ==> 번호: 0, 년도: 0
(car + 1)->showCar() ==> 번호: 1234, 년도: 2011
(car + 2)->showCar() ==> 번호: 2345, 년도: 2012
계속하려면 아무 키나 누르십시오 . . .
```

함수의 인수로 전달하는 클래스 객체

객체의 값에 의한 전달 방식(call by value)

- 함수를 호출할 때 클래스의 객체를 인수로 사용하면, 기본적으로 값에 의한 전달 방식으로 전달
- 객체의 값에 의한 전달 방식 예

```
class Rectangle {
        int   width;
        int   height;
};
void copyRectangle(Rectangle des, Rectangle src) {
        des = src;
 }
int main( ) {
        Rectangle   rect1(10, 5), rect2(20, 12);
        copyRectangle(rect2, rect1);
}
```



함수의 인수로 전달하는 클래스 객체

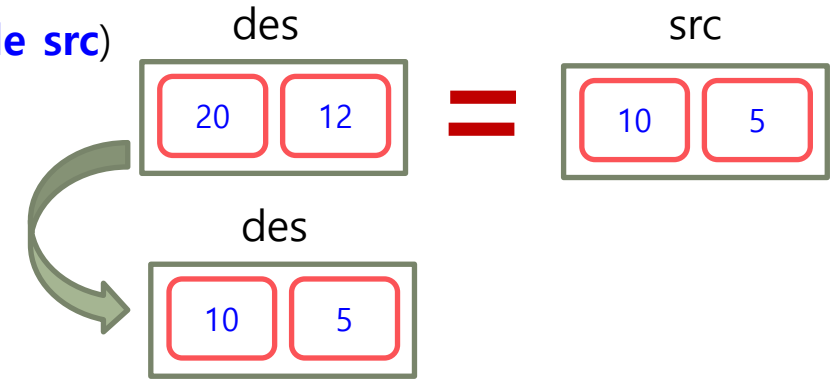
- **ex7_4.cpp (1) (객체의 값에 의한 전달 방식 예)**

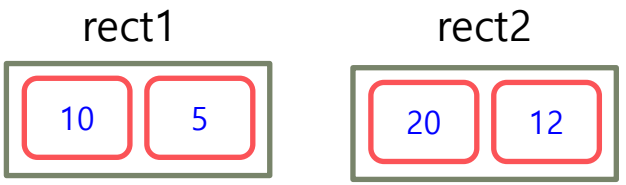
```
#include <iostream>
using namespace std;
class Rectangle
{
private :
        int width;
        int height;
public :
        Rectangle() : width(0), height(0) { };
        Rectangle(int w, int h) : width(w), height(h) { };
        void showRectangle();
};
void Rectangle::showRectangle()
{
        cout << "width : " << width << ", height : " << height << endl;
}
```

함수의 인수로 전달하는 클래스 객체

- **ex7_4.cpp (2) (객체의 값에 의한 전달 방식 예)**

```
void copyRectangle(Rectangle des, Rectangle src)
{
        des = src;
}
int main()
{
        Rectangle rect1(10, 5), rect2(20, 12);
        cout << "(before)rect1 ==> " ;
        rect1.showRectangle();
        cout << "(before)rect2 ==> " ;
        rect2.showRectangle();
        cout << endl;

        copyRectangle(rect2, rect1);
        cout << "(after)rect2 ==> " ;
        rect2.showRectangle();
        return 0;
}
```






```
(before)rect1 ==> width : 10, height : 5
(before)rect2 ==> width : 20, height : 12

(after)rect2 ==> width : 20, height : 12
계속하려면 아무 키나 누르십시오 . . .
```

함수의 인수로 전달하는 클래스 객체

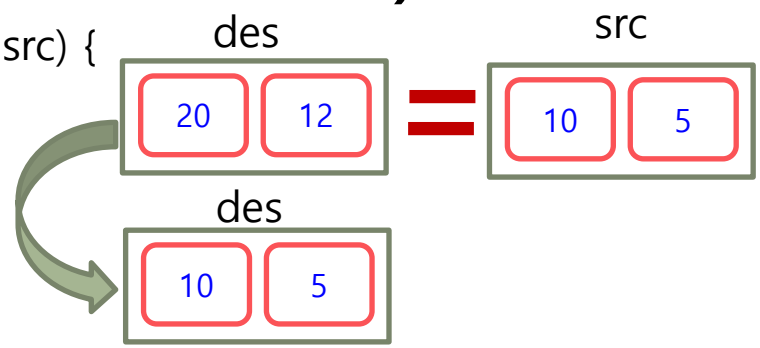
- **ex7_5.cpp (1) (함수의 반환 값이 객체인 경우의 예)**

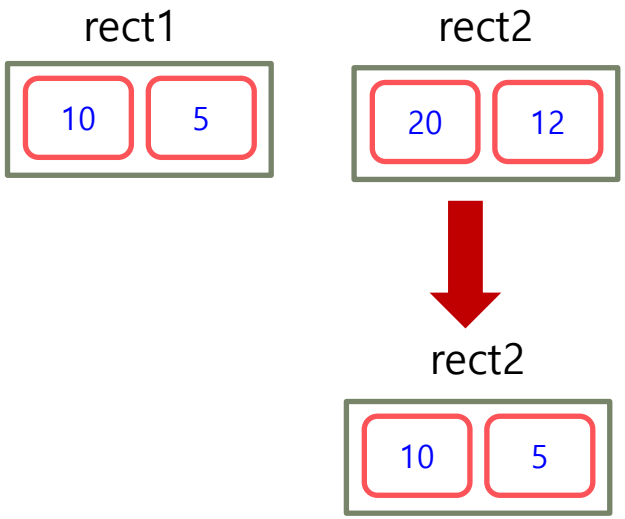
```
#include <iostream>
using namespace std;
class Rectangle
{
private :
        int width;
        int height;
public :
        Rectangle() : width(0), height(0) { };
        Rectangle(int w, int h) : width(w), height(h) { };
        void showRectangle();
};
void Rectangle::showRectangle()  {
        cout << "width : " << width << ", height : " << height << endl;
}
```

함수의 인수로 전달하는 클래스 객체

- **ex7_5.cpp (2) (함수의 반환값이 객체인 경우의 예)**

```
Rectangle copyRectangle(Rectangle des, Rectangle src) {
        des = src;
        return des;
}
int main()
{
        Rectangle rect1(10, 5),  rect2(20, 12);
        cout << "(before)rect1 ==> " ;
        rect1.showRectangle();
        cout << "(before)rect2 ==> " ;
        rect2.showRectangle();
        cout << endl;

        rect2 = copyRectangle(rect2, rect1);
        cout << "(after)rect2 ==> " ;
        rect2.showRectangle();
        return 0;
}
```



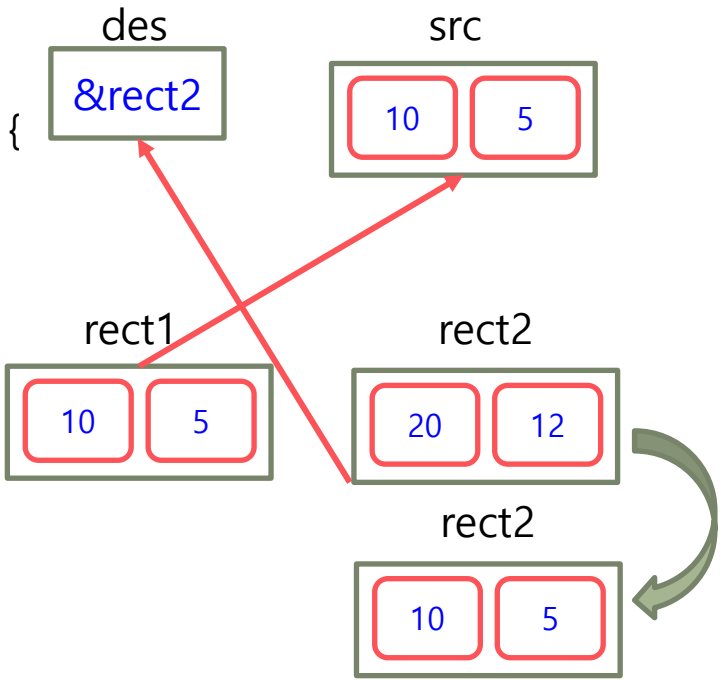



```
(copyRectangle()함수 호출 이전)rect1 ==> width : 10, height : 5
(copyRectangle()함수 호출 이전)rect2 ==> width : 20, height : 12

(copyRectangle()함수 호출 이후)rect2 ==> width : 10, height : 5
계속하려면 아무 키나 누르십시오 . . .
```

함수의 인수로 전달하는 클래스 객체

- **객체의 주소에 의한 전달 방식 (call by address)**
  - 함수를 호출할 때 클래스 객체의 주소를 인수로 사용
  - 호출 함수에서 객체의 값이 변경되면 그대로 호출하는 함수에도 영향을 받게 됨
  - 객체의 주소에 의한 전달 방식 예

```
class Rectangle {
        int   width;
        int   height;
};
void copyRectangle(Rectangle *des, Rectangle src) {
        *des = src;
}
int main(){
        Rectangle rect1(10, 5), rect2(20, 12);
        copyRectangle(&rect2, rect1);
}
```



함수의 인수로 전달하는 클래스 객체

- **ex7_6.cpp (1) (객체의 주소에 의한 전달로 객체 복사)**

```
#include <iostream>
using namespace std;
class Rectangle
{
private :
        int width;
        int height;
public :
        Rectangle() : width(0), height(0) { };
        Rectangle(int w, int h) : width(w), height(h) { };
        void showRectangle();
};
void Rectangle::showRectangle()
{
        cout << "width : " << width << ", height : " << height << endl;
}
```

함수의 인수로 전달하는 클래스 객체

- **ex7_6.cpp (2) (객체의 주소에 의한 전달로 객체 복사)**

```
void copyRectangle(Rectangle *des, Rectangle src)  {
        *des = src;
}
int main()
{
        Rectangle rect1(10, 5), rect2(20, 12);
        cout << "(before)rect1 ==> " ;
        rect1.showRectangle();
        cout << "(before)rect2 ==> " ;
        rect2.showRectangle();
        cout << endl;

        copyRectangle(&rect2, rect1);
        cout << "(after)rect1 ==> " ;
        rect1.showRectangle();
        cout << "(after)rect2 ==> " ;
        rect2.showRectangle();
        return 0;
}
```

```
<before>rect1 ==> width : 10, height : 5
<before>rect2 ==> width : 20, height : 12

<after>rect1 ==> width : 10, height : 5
<after>rect2 ==> width : 10, height : 5
계속하려면 아무 키나 누르십시오 . . .
```

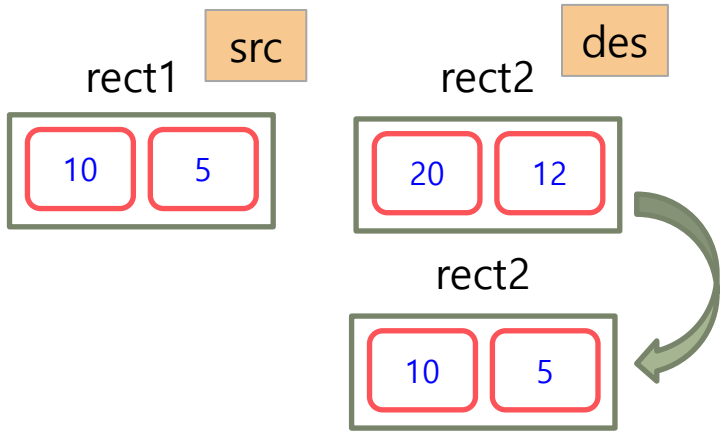
함수의 인수로 전달하는 클래스 객체

- **객체의 참조에 의한 전달 방식(call by reference) (1)**
  - 객체의 참조는 **기억공간을 별도로 할당 받지 않고** 별도의 이름(별칭)을 부여 받는 것이므로 호출함수에서 객체의 값이 변경되면 그대로 **호출하는 함수에도 영향을 받음**
  - 실인수의 값이 변경되지 않기를 바란다면, **const 키워드를 참조 객체 앞에 사용**

함수의 인수로 전달하는 클래스 객체

- 객체의 참조에 의한 전달 방식(call by reference) 예

```
class Rectangle
{
        int   width;
        int   height;
};
Rectangle & copyRectangle(Rectangle &des, const Rectangle &src)
{
        des = src;
        return des;
}
int main()
{
        Rectangle   rect1(10, 5), rect2(20, 12);
        rect2 = copyRectangle(rect2, rect1);
}
```



함수의 인수로 전달하는 클래스 객체

- **ex7_7.cpp (1) (객체의 참조에 의한 전달로 객체의 복사)**

```
#include <iostream>
using namespace std;
class Rectangle
{
private :
        int width;
        int height;
public :
        Rectangle() : width(0), height(0) { };
        Rectangle(int w, int h) : width(w), height(h) { };
        void showRectangle();
};
void Rectangle::showRectangle()
{
        cout << "width : " << width << ", height : " << height << endl;
}
```

함수의 인수로 전달하는 클래스 객체

- **ex7_7.cpp (2) (객체의 참조에 의한 전달로 객체의 복사)**

```
void copyRectangle(Rectangle &des, const Rectangle &src) {
        des = src;
}
int main() {
        Rectangle rect1(10, 5), rect2(20, 12);
        cout << "(copyRectangle()함수 호출 이전)rect1 ==> " ;
        rect1.showRectangle();
        cout << "(copyRectangle()함수 호출 이전)rect2 ==> " ;
        rect2.showRectangle();    cout << endl;

        copyRectangl( rect2, rect1 );
        cout << "(copyRectangle()함수 호출 이후)rect1 ==> " ;
        rect1.showRectangle();
        cout << "(copyRectangle()함수 호출 이후)rect2 ==> " ;
        rect2.showRectangle();
        return 0;
}
```

```
(copyRectangle()함수 호출 이전)rect1 ==> width : 10, height : 5
(copyRectangle()함수 호출 이전)rect2 ==> width : 20, height : 12

(copyRectangle()함수 호출 이후)rect1 ==> width : 10, height : 5
(copyRectangle()함수 호출 이후)rect2 ==> width : 10, height : 5
계속하려면 아무 키나 누르십시오 . . .
```

정적 멤버변수와 정적 멤버함수

정적 멤버변수 (1)

- **static 키워드 사용** → 정적 멤버 변수
- 클래스의 **모든 객체들이 멤버변수를 공유**

- 일반 멤버변수와 멤버 함수 : 객체의 멤버 변수
  - 객체 단위의 멤버 → 객체의 이름으로 멤버에 접근
  - 객체를 생성할 때 개별적으로 기억공간 할당

- 정적 멤버 변수: **클래스의 멤버 변수**
  - **클래스 단위의 멤버** → 객체의 생성 없이도 클래스의 이름으로 멤버에 접근 가능
  - **프로그램이 시작할 때 단 한번만 기억공간 할당 받아 생성, 프로그램이 끝날 때가지 지속**
  - 해당 클래스 내부에서 정적 멤버변수를 선언
  - **초기화는 클래스 밖에서 별도**로 설정해야 함

정적 멤버변수와 정적 멤버함수

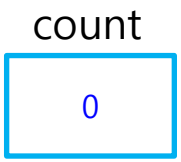
정적 멤버변수 (2)

- 정적 멤버변수의 선언

```
static  자료형  정적_변수명;
```

- 정적 멤버변수 사용 예

```
class Car
{
private:
        int number;                 // 일반 멤버변수
        static int count;           // 정적 멤버변수
        …
};
int Car::count = 0;                 // 정적 멤버변수의 초기화
        …
```



정적 멤버변수와 정적 멤버함수

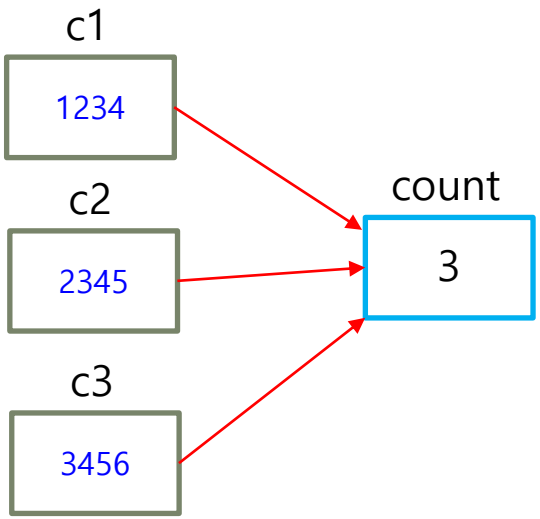
- **ex7_8.cpp (1) (클래스의 정적 멤버변수의 사용 예)**

```
#include <iostream>
using namespace std;
class Car
{
private:
        int number;             // 일반 멤버변수
public:
        static int count;       // 정적 멤버변수
        Car(int n);
        void showCar();
};
int Car::count = 0;             // 정적 멤버변수 counter의 초기화
Car::Car(int n)
{
        number = n;
        count++;
}
```

정적 멤버변수와 정적 멤버함수

- **ex7_8.cpp (2) (클래스의 정적 멤버변수의 사용 예)**

```
void Car::showCar() {
        cout << "번호: " << number << endl;
}
void main() {
        Car c1(1234);
        c1.showCar();
        cout << "등록대수: " << c1.count << endl;

        Car c2(2345);
        c2.showCar();
        cout << "등록대수: " << c2.count << endl;

        Car c3(3456);
        c3.showCar();
        cout << "등록대수: " << c3.count << endl;
}
```




```
번호: 1234
등록대수: 1
번호: 2345
등록대수: 2
번호: 3456
등록대수: 3
계속하려면 아무 키나 누르십시오 . . .
```

정적 멤버변수와 정적 멤버함수

정적 멤버함수

- **정적 멤버변수의 접근 제한자가 private으로 선언되어** 있을 때, 정적 멤버변수를 사용하기 위하여, **클래스 차원에서 접근 가능하도록 선언된 멤버함수**
- 객체의 생성과 관계없이 **클래스 이름으로 멤버에 접근 가능**
- 정적 멤버함수는 객체에 의한 호출이 아니고 **클래스에 의한 호출**이므로 객체 멤버변수나 객체를 구분하는 **this 포인터를 사용할 수 없음**
- 정적 멤버함수는 **멤버함수의 재정의(function overriding)가 되지 않음**

- 정적 멤버함수 예

```
static   int   GetCount();
Car::GetCount();
```

정적 멤버변수와 정적 멤버함수

- **ex7_9.cpp (1) (정적 멤버변수와 정적 멤버함수의 사용 예)**

```
#include <iostream>
using namespace std;
class Car
{
private:
        int number;                 // 일반 멤버변수
        static int count;           // 정적 멤버변수
public:
        Car(int n);
        void showCar();             // 일반 멤버함수
        static int GetCount();      // 정적 멤버함수
};
int Car::count = 0;                 // 정적 멤버변수 counter의 초기화
Car::Car(int n) {
        number = n;
        count++;
}
```

정적 멤버변수와 정적 멤버함수

- **ex7_9.cpp (2) (정적 멤버변수와 정적 멤버함수의 사용 예)**

```
void Car::showCar() {
        cout << "번호: " << number << endl;
}
int Car::GetCount()  {       // 정적 멤버함수 정의
        return count;
}
void main() {
        Car c1(1234);
        c1.showCar();
        cout << "등록대수: " << Car::GetCount() << endl ;
        Car c2(2345);
        c2.showCar();
        cout << "등록대수: " << Car::GetCount() << endl ;
        Car c3(3456);
        c3.showCar();
        cout << "등록대수: " << Car::GetCount() << endl ;
}
```

```
번호: 1234
등록대수: 1
번호: 2345
등록대수: 2
번호: 3456
등록대수: 3
계속하려면 아무 키나 누르십시오 . . .
```

프렌드 함수

프렌드 함수(friend function)

- **클래스의 멤버는 아니지만 클래스의 비공개(private) 멤버에 접근할 수 있음**
- 자료 은닉에 위배되므로 꼭 필요한 경우에만 사용
- **클래스의 멤버가 아니므로 상속되지 않음**

> **friend** 리턴 데이터형 프렌드_함수명(매개변수, ...);

- 프렌드 함수 예

```
class Car {
private:
        int   number;
        char *owner;
public:
        Car();                                  // 생성자, 멤버함수
        void  showCar() const;                  // 멤버함수
        friend  void  printCar(Car car[]);      // 프렌드 함수 – 비멤버함수
};
void printCar(Car car[]) {                      // 프렌드 함수의 정의
        ...
}
```

프렌드 함수

- **ex7_10.cpp (1) (프렌드 함수 예)**

```
#include <iostream>
#include <cstring>
using namespace std;
class Car {
private:
        int number;
        char* owner;
public:
        Car(int n, char* m);                        // 생성자, 멤버함수
        void showCar() const ;                      // 멤버함수
        friend void printCar(Car car[], int n);     // 프렌드 함수
};
Car::Car(int n, char* m) {
        number = n;
        owner = new char[strlen(m) + 1];
        strcpy_s(owner, strlen(m)+1, m);
}
```

프렌드 함수

- **ex7_10.cpp (2) (프렌드 함수 예)**

```
void Car::showCar() const {
        cout << "번호: " << number << ", 소유주: " << owner << endl;
}
void printCar(Car car[], int n) {          // 객체 배열로 전달받는다
        int i;
        for (i=0; i<n; i++)
                cout << "car[" << i << "] ==> " << car[i].number ₩
                        << ", " << car[i].owner << endl;
}
void main()  {
        Car car[3] = { Car(1234, "Kim"), Car(2345, "Hong"), Car(3456, "Lee") };
        car[0].showCar();          // 멤버함수이므로 객체와 함께 사용
        printCar(car, 3);          // 멤버함수가 아니므로 객체의 이름과 함께 사용할 수 없다.
                                   // 일반함수처럼 호출하여야 한다. 객체 배열의 이름을 전달
        cout << endl;
}
```

```
번호: 1234, 소유주: Kim
car[0] ==> 1234, Kim
car[1] ==> 2345, Hong
car[2] ==> 3456, Lee

계속하려면 아무 키나 누르십시오 . . .
```

객체의 동적 관리

동적 객체 생성 및 제거

- 객체를 동적으로 할당하는 경우 인수가 없는 기본 생성자만이 할당이 가능하므로 기본 생성자를 반드시 정의해 두어야 한다.
- 동적으로 기억공간을 할당한 객체는 더 이상 사용할 필요가 없으면 해제하여야 한다.
- 동적 객체 생성 및 제어 예

```
Car  *pcar = new Car;          // 동적 객체 생성
delete  pcar;                  // 동적 객체 제거
```

객체의 동적 할당

- **ex8_11.cpp (1) (객체를 동적으로 생성하고 제거하는 예)**

```
#include <iostream>
#include <cstring>
using namespace std;
class Car
{
private:
        int number;
        char *owner;
public:
        Car();
        ~Car();
        Car(int n, char *m);
        void showCar() const ;
        void getCar();
};
```

객체의 동적 할당

- **ex8_11.cpp (2) (객체를 동적으로 생성하고 제거하는 예)**

```
Car::Car(){
        number = 0;
        owner = NULL;
        cout << "기본 생성자 호출" << endl;
}
Car::~Car() {
        cout << "소멸자 호출" << endl;
        delete [] owner;                    // 멤버변수의 문자열 기억공간 제거
}
Car::Car(int n, char *m) {
        number = n;
        owner = new char[strlen(m) +1];
        strcpy_s(owner, strlen(m)+1, m);
        cout << "일반 생성자 호출" << endl;
}
void Car::showCar() const  {
        cout << "번호: " << number << ", 소유주: " << owner << endl;
}
```

객체의 동적 할당

- **ex8_11.cpp (3) (객체를 동적으로 생성하고 제거하는 예)**

```
void Car::getCar() {
        char name[20];
        cout << "차량번호를 입력 하세요: " ;
        cin >> number;
        cin.clear();   cin.ignore(INT_MAX, '₩n');     // 입력 버퍼 비우기, fflush(stdin); 대체
        cout << "소유주를 입력 하세요: " ;
        cin.getline(name, 20);
        delete [] owner;                              // 멤버변수의 문자열 기억공간 제거
        owner = new char[strlen(name)+1];   // 멤버변수의 문자열 기억공간을 동적생성
        strcpy_s(owner, strlen(name)+1, name);   //문자열 복사
}
int main() {
        Car *pcar = new Car;  // 객체의 동적생성
        pcar->getCar();
        pcar->showCar();
        delete pcar;                  // 객체의 제거
        return 0;
}
```

```
기본 생성자 호출
차량번호를 입력 하세요: 1234
소유주를 입력 하세요: Hong Gildong
번호: 1234, 소유주: Hong Gildong
소멸자 호출
계속하려면 아무 키나 누르십시오 . . .
```

객체를 멤버로 갖는 클래스

- **has-a 관계 (1)**
  - Is-a 관계 예
    - 개는 동물이다(Dog is a Animal.)
    - Dog와 Animal 사이의 관계는 is-a 관계로 두 개의 클래스는 상속으로 처리할 수 있다.
  - Has-a 관계 예
    - 컴퓨터는 CPU를 가지고 있다(Computer has a CPU.)
    - Computer와 CPU 사이의 관계는 has-a 관계로 포함으로 처리할 수 있다.
    - 두 개의 클래스 사이에 has-a 관계일 때 한 클래스의 객체는 다른 클래스의 멤버 변수로 포함될 수 있다.
  - Has-a 관계 예에서 CPU 객체는 Computer 클래스의 멤버 객체가 될 수 있음
  - Computer 클래스의 객체를 생성하려면 CPU 클래스의 객체도 생성해야 함
  - 만약 Computer 클래스와 CPU 클래스에서 생성자를 정의하지 않으면 컴파일러가 디폴트 생성자를 제공하고 이 경우에는 클래스의 멤버 변수들은 쓰레기 값을 갖게 됨

객체를 멤버로 갖는 클래스

- **has-a 관계 예**

```
class CPU {                          // CPU 클래스
private:
        string   cpuname;
        float    speed;
        .......
};

class Computer{
private:
        CPU   c_cpu;                 // 멤버 객체 : CPU 클래스의 객체
        int     m_size;
        .......
};
```

객체를 멤버로 갖는 클래스

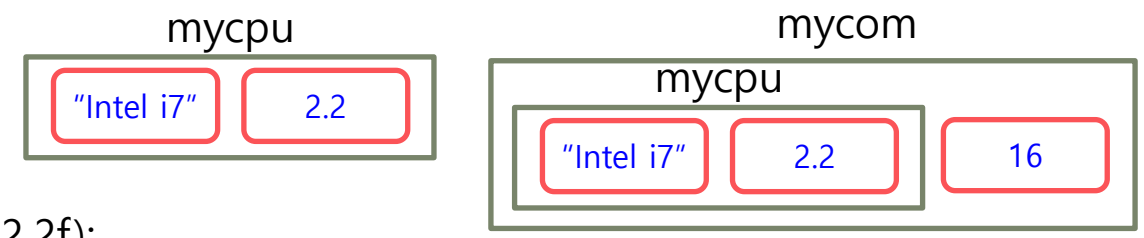
- **ex7_12.cpp (1) (멤버 객체를 가지는 클래스)**

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

class CPU
{
private:
        string cpuname;
        float speed;
public:
        CPU(string cn, float sp) : cpuname(cn), speed(sp) {  }
        void show() const {
        cout<<"processor name: "<<cpuname<<", speed :"<<speed<<" GHz"<< endl;
        }
};
```

객체를 멤버로 갖는 클래스

- **ex7_12.cpp (2) (멤버 객체를 가지는 클래스)**

```
class Computer   {
private:
        CPU c_cpu;        // 멤버 객체
        int m_size;
public:
        Computer(CPU cpu1, int size): c_cpu(cpu1), m_size(size) { }
        ~Computer() { };
        void show() {
                c_cpu.show();
                cout << "memory : " << m_size << " GB" << endl;
        }
};
void main()
{
        CPU mycpu("Intel i7", 2.2f);
        Computer mycom(mycpu, 16);
        mycom.show();
}
```




```
processor name: Intel i7, speed :2.2 GHz
memory : 16 GB
계속하려면 아무 키나 누르십시오 . . .
```

객체를 멤버로 갖는 클래스

- **클래스가 다른 클래스의 friend 클래스로 되는 경우**
  - A클래스가 B클래스의 멤버로 활용될 때 B클래스가 A클래스의 **private 멤버**에 접근하기 위해서는 **friend 클래스로 선언해야 접근할 수 있음**
    - Point 클래스의 객체가 Circle 클래스의 멤버 객체로 되어 있는 경우 Circle 클래스의 멤버가 Point 클래스의 private 멤버에 접근할 수 없는 문제가 발생하며, Point 클래스의 private 멤버에 접근하기 위해서는 Circle 클래스를 Point 클래스의 friend 클래스로 선언해 주어야 함
  - 클래스가 다른 클래스의 friend 클래스로 되는 경우 예

```
class Point {                    // 클래스
private:
        int px;
        int py;
        .......
        friend class Circle;     // Circle 클래스를 Point 클래스의 friend 클래스로 선언
};
class Circle {
        .......
        Point center;            // 멤버 객체
};
```

객체를 멤버로 갖는 클래스

- **ex7_13.cpp (1) (멤버 객체를 가지는 클래스의 예)**

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

class Point
{
private:
        int   px;               // private 멤버
        int   py;               // private 멤버
public:
        Point(int x=0, int y=0):px(x), py(y) {  }
        void showPoint() const
        {
                cout << "점의 위치: x = " << px << ", y = " << py << endl;
        }
        friend class Circle;      // Circle 클래스에서 px, py를 사용하기 위해
};
```

객체를 멤버로 갖는 클래스

- **ex7_13.cpp (2) (멤버 객체를 가지는 클래스의 예)**

```
class Circle
{
private:
        Point center;                   // 멤버 객체
        int radius;
public:
        Circle(Point cen, int rad=0):center(cen), radius(rad)  {   }
        ~Circle() { };
        void showCircle() {
                // Point 클래스의 private 멤버 변수 px와 py를 사용하기 위하여
                // Point 클래스에서 Circle 클래스를 friend 클래스로 선언하였다.

                cout << "중심점: (x = " << center.px << ", y = " << center.py << ")";
                cout << " 반지름: " << radius << endl;
        }
};
```

객체를 멤버로 갖는 클래스

- **ex7_13.cpp (3) (멤버 객체를 가지는 클래스의 예)**

```
void main()
{
        Point cen(5, 7);
        cen.showPoint();

        Circle cir1(cen, 10);
        cir1.showCircle();
}
```

```
점의 위치: x = 5, y = 7
중심점: (x = 5, y = 7)  반지름: 10
계속하려면 아무 키나 누르십시오 . . .
```

Thank You