

## 5장 클래스의 활용


- # 클래스와 배열
- # 객체 포인터
- # this 포인터
- # 멤버 함수 오버로딩
- # 디폴트 매개변수의 사용
- # friend (전역 함수, 클래스, 멤버 함수)
- # 내포 클래스
- # 지역 클래스
- # static 멤버
- # const 멤버와 const 객체
- # explicit 생성자

# 1. 클래스와 배열

## # int형 배열 선언 및 초기화

```
int main(void)
{
    int ary[5] = { 1, 2, 3, 4, 5 };
    for (int i = 0; i < 5; i++)
        cout << "ary[" << i << "] = " << ary[i] << endl;

    return 0;
}
```



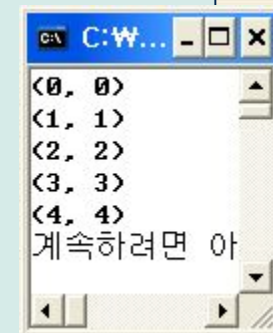
```
C:\WWI...
ary[0] = 1
ary[1] = 2
ary[2] = 3
ary[3] = 4
ary[4] = 5
계속하려면 아무
```

# 1. 클래스와 배열

## # 5개의 CPoint형 객체 원소를 갖는 배열 선언 및 사용

```
class CPoint {  
private :  
    int x, y;  
  
public :  
    void SetXY(int a, int b) { x = a; y = b; }  
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }  
};  
  
int main(void)  
{  
    CPoint pt[5];           // 5개 원소를 갖는 CPoint형 객체 배열  
    int i;  
  
    for (i = 0; i < 5; i++)  
        pt[i].SetXY(i, i);  
  
    for (i = 0; i < 5; i++)  
        pt[i].Print();  
  
    return 0;  
}
```

기본적으로는 일반 변수의 배열 사용 방법과 동일!



# 1. 클래스와 배열

※ 객체 배열 선언 시 각 객체의 값을 (0, 0), (1, 1) ... (4, 4)로 초기화

■ 생성자에 주의

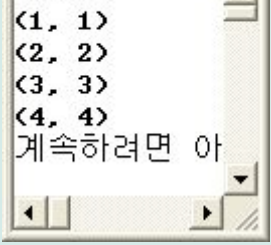
```
class CPoint {
private :
    int x, y;

public :
    CPoint(int a, int b) : x(a), y(b) { }
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }
};

int main(void)
{
    CPoint pt[5] = { CPoint(0, 0), CPoint(1, 1), CPoint(2, 2),
                    CPoint(3, 3), CPoint(4, 4) };    // 객체 배열 선언 & 초기화

    for (int i = 0; i < 5; i++)
        pt[i].Print();

    return 0;
}
```



각 객체의 초기화 방법에 주의  
**CPoint P1 = CPoint(1, 1);** 과 같은 형태  
**CPoint(int, int)** 생성자가 존재해야 함

# 1. 클래스와 배열

## ■ 객체 배열 생성 및 초기화와 생성자의 사용 예

- `CPoint pt[3] = { CPoint(), CPoint(1), CPoint(2, 3) };`
  - 매개변수 0개, int형 1개, int형 2개인 생성자가 존재해야 함
- `CPoint pt[3] = { CPoint(2, 3) };`
  - 매개변수 2개인 생성자
  - 매개변수 0개인 생성자 필요 : 두 번째 원소부터 적용
- `CPoint pt[3] = { 1, 2, CPoint(3, 4) };`
  - 매개변수 int형 1개, int형 2개인 생성자 필요
  - 매개변수가 1개인 경우 클래스명 생략 가능
    - ✓ `CPoint P1 = 5;` 와 같은 형태임

# 1. 클래스와 배열

## ※ 2차원 객체 배열 : 3행 2열

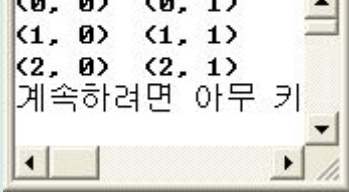
```
class CPoint {
private :
    int x, y;

public :
    CPoint(int a, int b) : x(a), y(b) { }
    void Print() { cout << "(" << x << ", " << y << ")"; }
};

int main(void)
{
    CPoint pt[3][2] = { { CPoint(0, 0), CPoint(0, 1) },
                        { CPoint(1, 0), CPoint(1, 1) },
                        { CPoint(2, 0), CPoint(2, 1) } }; // 2차원 배열 & 초기화

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
            pt[i][j].Print();
            cout << "Wt";
        }
        cout << endl;
    }

    return 0;
}
```



일반 변수의 초기화 개념과 동일



## 2. 객체 포인터

### # int형 변수의 동적 할당 및 사용

```
int main(void)
{
    int *ptr;
    int i;

    ptr = new int(3);    // 동적 생성과 동시에 초기화 가능
    cout << *ptr << endl;
    delete ptr;

    ptr = new int[5];    // 배열의 동적 생성은 초기화 불가능
    for (i = 0; i < 5; i++)
        ptr[i] = i;
    for (i = 0; i < 5; i++)
        cout << ptr[i] << endl;
    delete [] ptr;

    return 0;
}
```

배열에 대한 동적 할당의 경우 **delete []** 사용

## 2. 객체 포인터

### ※ 클래스 객체를 동적으로 할당받는 방법

- 기본적으로 일반 변수의 동적 할당과 동일!

```
class CPoint {  
private :  
    int x, y;  
  
public :  
    CPoint() : x(0), y(0) { }  
    CPoint(int a) : x(a), y(a) { }  
    CPoint(int a, int b) : x(a), y(b) { }  
    void SetXY(int a, int b) { x = a, y = b; }  
    void Print() { cout << "(" << x << ", " << y <<  
};
```

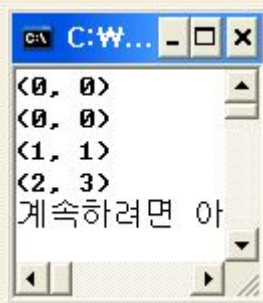
객체 포인터 선언

객체 동적 생성

메모리 해제

```
int main(void)  
{  
    CPoint *ptr;  
  
    ptr = new CPoint;  
    ptr->Print();  
    delete ptr;  
  
    ptr = new CPoint();  
    ptr->Print();  
    delete ptr;  
  
    ptr = new CPoint(1);  
    ptr->Print();  
    delete ptr;  
  
    ptr = new CPoint(2, 3);  
    ptr->Print();  
    delete ptr;  
  
    return 0;  
}
```

각각의 생성자 필요





## 2. 객체 포인터

### ▣ 객체 배열의 동적 생성

```
int main(void)
{
```

```
    CPoint *ptr;
    int i;
```

```
    ptr = new CPoint[5]; // 객체 포인터를 이용한 배열 동적 생성
```

```
    for (i = 0; i < 5; i++)
        ptr[i].SetXY(i, i);
```

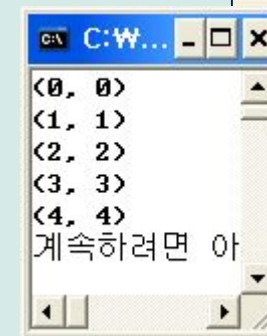
```
    for (i = 0; i < 5; i++)
        ptr[i].Print();
```

```
    delete [] ptr;
```

```
    return 0;
```

```
}
```

객체 배열을 동적으로 생성할 경우에는  
매개변수가 없는 생성자가 반드시 있어야 함



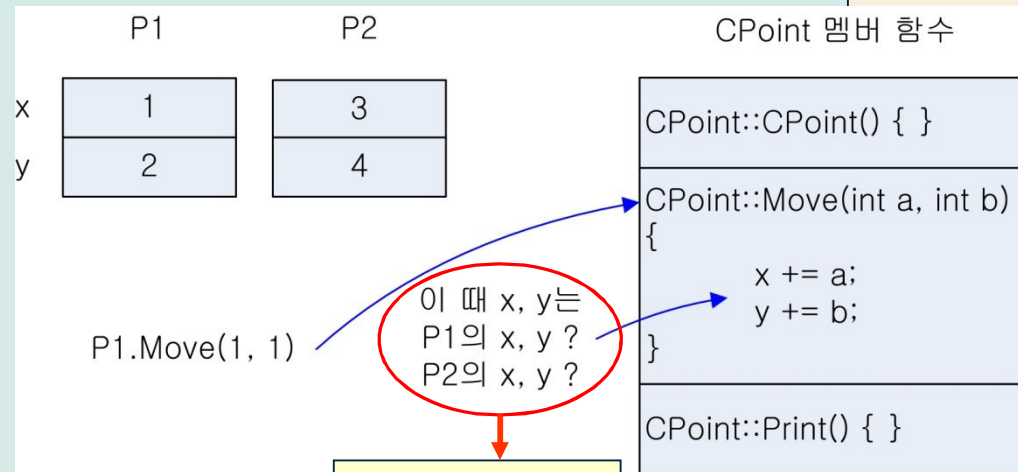
### 3. this 포인터

#### ■ 객체 생성과 메모리 구조

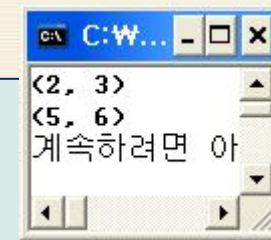
```
class CPoint {  
private :  
    int x, y;  
  
public :  
    CPoint(int a, int b) : x(a), y(b) { }  
    void Move(int a, int b) { x += a; y += b; }  
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }  
};
```

```
int main(void)  
{  
    CPoint P1(1, 2);  
    CPoint P2(3, 4);  
  
    P1.Move(1, 1);  
    P2.Move(2, 2);  
  
    P1.Print();  
    P2.Print();  
  
    return 0;  
}
```

객체에 대한 멤버 변수는 별도로 생성됨  
멤버 함수는 하나만 생성, 모든 객체가 공유



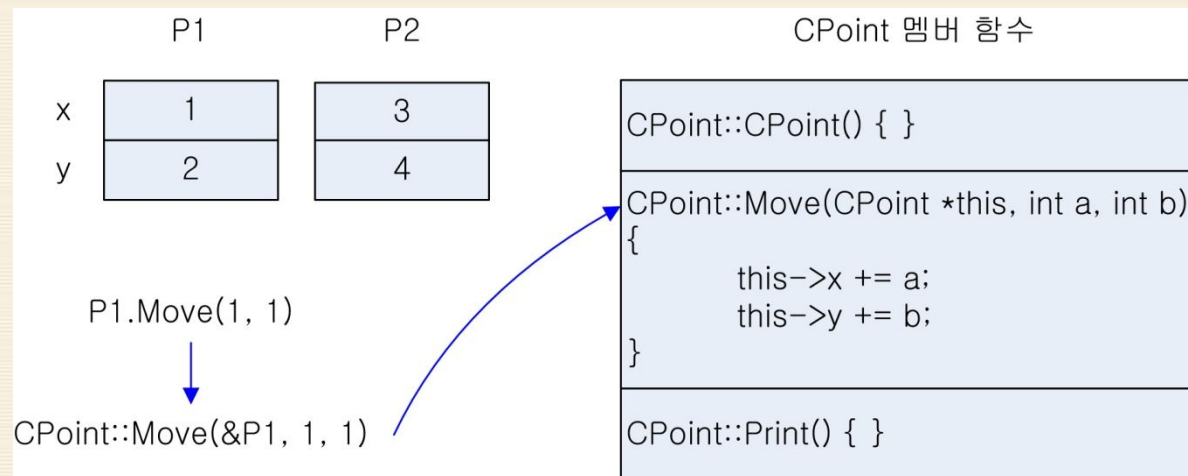
**this 포인터!!!**



### 3. this 포인터

#### ■ this 포인터

- 멤버 함수 호출 시 해당 객체에 대한 주소를 받는 형식매개변수
- 멤버 함수의 개념적 구조



```
class CPoint {  
private :  
    int x, y;  
  
public :  
    CPoint(int a, int b) : x(a), y(b) { }  
    void Move(int a, int b) { this->x += a; this->y += b; }  
    void Print() { cout << "(" << this->x << ", " << this->y << ")" << endl; }  
};
```

**this 포인터를 명시적으로 사용 가능**

### 3. this 포인터

✦ this 포인터에 대한 원리 이해 : 멤버 함수를 전역함수로 만든다면?

```
struct CPoint {  
    int x, y;  
};
```

클래스는 구조체로

```
void Move(CPoint *This, int a, int b)  
{  
    This->x += a;  
    This->y += b;  
}
```

멤버 함수는 전역 함수로  
해당 객체의 주소를 **This** 포인터로 받음

```
void Print(CPoint *This)  
{  
    cout << "(" << This->x << ", " << This->y << ")" << endl;  
}
```

```
int main(void)  
{  
    CPoint P1 = { 1, 2 };  
    CPoint P2 = { 3, 4 };  
  
    Move(&P1, 1, 1);  
    Move(&P2, 2, 2);  
  
    Print(&P1);  
    Print(&P2);  
  
    return 0;  
}
```

함수 호출 시 관련 객체의 주소를 넘김

### 3. this 포인터

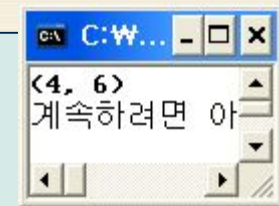
#### # this 포인터의 사용 예 (1)

```
class CPoint {
private :
    int x, y;

public :
    CPoint(int a, int b) : x(a), y(b) { }
    CPoint *MoveX(int a) { x += a; return this; }
    CPoint *MoveY(int b) { y += b; return this; }
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }
};

int main(void)
{
    CPoint P1(1, 2);
    P1.MoveX(3)->MoveY(4);      // MoveX의 결과가 P1 포인터
    P1.Print();

    return 0;
}
```



함수를 호출한 객체의  
주소를 반환

### 3. this 포인터

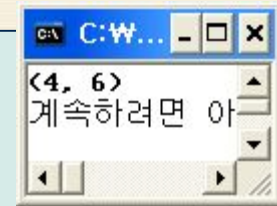
#### # this 포인터의 사용 예 (2)

```
class CPoint {
private :
    int x, y;

public :
    CPoint(int a, int b) : x(a), y(b) { }
    CPoint &MoveX(int a) { x += a; return (*this); }    // 객체 자체 반환
    CPoint &MoveY(int b) { y += b; return (*this); }    // 객체 자체 반환
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }
};

int main(void)
{
    CPoint P1(1, 2);
    P1.MoveX(3).MoveY(4); // MoveX 결과가 P1 그 자체
    P1.Print();

    return 0;
}
```



함수를 호출한 객체의 참조 반환  
반환 결과 자체가 그 객체가 됨

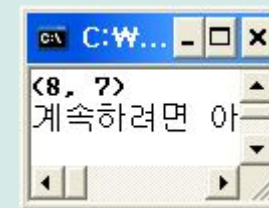
this 포인터의 실 사용 예  
→ 7장 연산자 오버로딩!



## 4. 멤버 함수 오버로딩

### ✦ 멤버 함수의 오버로딩 가능

```
class CPoint {  
private :  
    int x, y;  
  
public :  
    CPoint() : x(0), y(0) { }  
    CPoint(int a, int b) : x(a), y(b) { }  
    void Move(int a) { x += a; } // Move 함수  
    void Move(int a, int b) { x += a; y += b; } // Move 함수 오버로딩  
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }  
};  
  
int main(void)  
{  
    CPoint P1(1, 2);  
    P1.Move(3);  
    P1.Move(4, 5);  
  
    P1.Print();  
  
    return 0;  
}
```



## 5. 디폴트 매개변수의 사용

### ※ 멤버 함수 작성 시 디폴트 매개변수 사용 가능

```
class CPoint {  
private :  
    int x, y;  
  
public :  
    CPoint(int a = 0, int b = 0) : x(a), y(b) { }  
    void Move(int a, int b = 0) { x += a; y += b; }  
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }  
};
```

생성자의 디폴트 매개변수 사용

멤버 함수의 디폴트 매개변수 사용

## 6. friend 전역 함수

### private 멤버에 대한 접근

- 내부 접근만 가능 : 해당 클래스의 멤버 함수에 의한 접근 가능
- 외부 접근 불가능
  - 예외 : friend에 의한 접근 가능

### friend 선언

- 특정 클래스 입장에서 다른 전역 함수, 클래스, 다른 클래스의 멤버 함수를 자신의 친구(friend)로 선언하는 것
- friend의 종류
  - friend 전역 함수
  - friend 클래스
  - friend 멤버 함수

```
class classA {  
    private :  
        int x;  
        friend int func(classA &objA, int a);  
};  
  
int func(classA &objA, int a) { objA.x = a; }
```

func 함수를 자신의 friend로 선언

private 멤버에 접근 가능

## 6. friend 전역 함수

예 : friend 전역 함수

```
class CPoint {
private :
    int x;
    int y;

public :
    CPoint(int a = 0, int b = 0) : x(a), y(b) { }

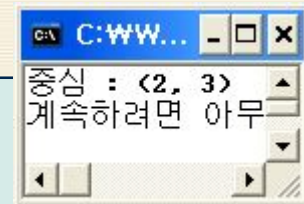
    friend void Center(CPoint P1, CPoint P2); // Center 함수를 friend로 선언
};

void Center(CPoint P1, CPoint P2)
{
    CPoint P;
    P.x = (P1.x + P2.x) / 2;
    P.y = (P1.y + P2.y) / 2;
    cout << "중심 : " << "(" << P.x << ", " << P.y << ")" << endl;
}

int main(void)
{
    CPoint P1(1, 2), P2(3, 4);
    Center(P1, P2);

    return 0;
}
```

private 멤버에 접근 가능



## 6. friend 전역 함수

### ▣ 참고 사항

- friend 함수 선언의 위치
  - 선언하는 영역(private, public)과 무관하며 의미 동일
- 클래스 내에서 friend 함수 선언 시에만 friend 키워드 추가
  - 함수 정의 시에는 friend 키워드가 없음 → 함수 자체는 일반 전역 함수
- friend 선언은 멤버 함수의 선언이 아님
  - 단지 기존 함수에 대한 friend 선언일 뿐
  - 객체를 통한 접근 불가능

## 7. friend 클래스

✦ 예 : CCar 클래스(자동차)와 CController 클래스(리모콘)

- CController 클래스의 멤버 함수에서 CCar 클래스 객체의 private에 접근
- CCar 클래스에서 CController 클래스를 friend 클래스로 선언!

```
class CCar {  
    friend class CController;  
};
```

```
class CCar {  
private :  
    bool OnOff;  
    int price;  
    int speed;  
  
public :  
    CCar(int p) : OnOff(false), price(p), speed(0) { }  
    void PrintSpeed() { cout << "현재속도 : " << speed << endl; }  
  
    friend class CController;    // friend 클래스 선언  
};
```



## 7. friend 클래스

### ✦ 코드 계속

```
class CController {
private :
    int price;

public :
    CController(int p) : price(p) { }
    void TurnOn(CCar &car) { car.OnOff = true; }           // CCar의 private 접근
    void TrunOff(CCar &car) { car.OnOff = false; }
    void SpeedChange(CCar &car, int v) { car.speed += v; }
};

int main(void)
{
    CCar MyCar(100);
    CController MyController(10);

    MyController.TurnOn(MyCar);
    MyController.SpeedChange(MyCar, 5);
    MyCar.PrintSpeed();
    MyController.TrunOff(MyCar);

    return 0;
}
```

## 7. friend 클래스 : 전방 선언

✦ 앞 예제에서 CController 클래스를 CCar 클래스 앞에 위치시킨다면

```
class CCar;

class CController {
private:
    int price;

public:
    CController(int p) : price(p) { }
    void TurnOn(CCar &car);
    void TrunOff(CCar &car);
    void SpeedChange(CCar &car, int v);
};

class CCar {
private:
    bool OnOff;
    int price;
    int speed;

public:
    CCar(int p) : OnOff(false), price(p)
    void PrintSpeed() { cout << "현재 속
    friend class CController;
};
```

전방 선언 : CCar 클래스의 존재를 알아야 됨  
사용은 불가 (예 : car.off)  
CCar 클래스의 사용은 CCar 클래스 선언 이후에 가능

CCar 클래스 객체의 사용은 CCar 클래스 선언이  
나온 이후에 가능 → CController의 멤버 함수를  
CCar 선언 이후에 작성하였음

```
void CController::TurnOn(CCar &car)
{
    car.OnOff = true;
}

void CController::TrunOff(CCar &car)
{
    car.OnOff = false ;
}

void CController::SpeedChange(CCar &car, int v)
{
    car.speed += v;
}
```

## 7. friend 클래스 : 전방 선언

### # 전방 선언이 필수적인 예

- 앞 예의 경우 CCar 클래스를 CController 클래스 앞에 위치시킨다면 전방 선언이 필요없음
- 문제 : CCar 클래스의 멤버 함수로 다음과 같은 함수를 추가하였다면?

```
void SetPrice(CController &controller, int p)
{
    controller.price = p;
}
```

- CCar, CController 클래스에서 서로 상대방 클래스 객체를 사용하는 경우
- 전방 선언 없이 해결 가능한가?
- 교재 : 예제 5.17 참고

## 8. friend 멤버 함수

- ✦ 다른 클래스의 특정 멤버 함수만을 friend로 선언 가능
- ✦ 예 : CController에서 CCar의 SetPrice 함수만을 friend로 선언

```
class CController {  
private :  
    int price;  
  
public :  
    CController(int p) : price(p) { }  
    void TurnOn(CCar &car) { car.OnOff = true; }  
    void TrunOff(CCar &car) { car.OnOff = false; }  
    void SpeedChange(CCar &car, int v) { car.speed += v; }  
  
    friend void CCar::SetPrice(CController &controller, int p); // friend 선언  
};
```

- ✦ friend의 사용
  - 정보 은닉 위배 : 가급적 사용을 피하는 것이 바람직
  - 전형적인 사용 예 : 7.9절 입출력 연산자 오버로딩

## 9. 내포 클래스 선언

### ▣ 내포 클래스(nested class)

- 다른 클래스 선언 내에 선언되어 있는 클래스

```
class CCircle {
private :
    class CPoint {          // 내포 클래스
    private :
        int x;
        int y;

    public :
        CPoint(int a = 0, int b = 0) : x(a), y(b) { }
        void Move(int a, int b) { x += a; y += b; }
        void Print() { cout << "중심 : (" << x << ", " << y << ")" << endl; }
    };

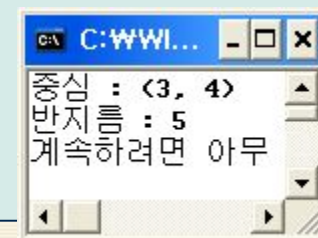
    CPoint Center;          // 내포 클래스 객체를 멤버 객체로 선언
    double Radius;

public :
    CCircle(int a, int b, double r) : Radius(r) { Center.Move(a, b); }
    void Print() {
        Center.Print();      // 멤버 객체 Center의 사용
        cout << "반지름 : " << Radius << endl;
    }
};
```

내포 클래스가 **private** 영역에 있으므로 외부에서는 사용 불가 → 다음 페이지 참고

```
int main(void)
{
    CCircle cir(3, 4, 5);
    cir.Print();

    return 0;
}
```



## 9. 내포 클래스 선언

### ▣ 내포 클래스의 사용

```
class CCircle {  
public :  
    class CPoint {    // 내포 클래스  
    private :  
        int x;  
        int y;  
  
    public :  
        CPoint(int a = 0, int b = 0) : x(a), y(b) { }  
        void Move(int a, int b);  
        void Print();  
    };  
  
    private :  
        CPoint Center;  
        double Radius;  
  
    public :  
        CCircle(int a, int b, double r) : Radius(r) { Center.Move(a, b); }  
        void Print() {  
            Center.Print();  
            cout << "반지름 : " << Radius << endl;  
        }  
};
```



## 9. 내포 클래스 선언

### ▣ 코드 계속

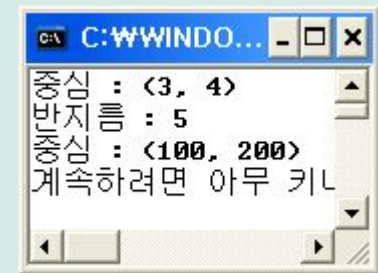
```
void CCircle::CPoint::Move(int a, int b) // 내포 클래스 멤버 함수의 외부 정의
{
    x += a;
    y += b;
}

void CCircle::CPoint::Print(void)
{
    cout << "중심 : (" << x << ", " << y << ")" << endl;
}

int main(void)
{
    CCircle cir(3, 4, 5);
    cir.Print();

    CCircle::CPoint P2(100, 200); // 내포 클래스 객체 생성
    P2.Print();

    return 0;
}
```



## 10. 지역 클래스 선언

### ⌘ 지역 클래스(local class)

- 함수 내부에 선언되어 있는 클래스
- 멤버 함수의 내부 정의만을 허용 (외부 정의 불가)
- 해당 함수 내에서만 사용 가능

```
int main(void)
{
    class CPoint {    // 지역 클래스 선언, main 함수 내에서만 사용 가능
    private :
        int x, y;

    public :
        CPoint(int a, int b) : x(a), y(b) { }
        void Print() { cout << "(" << x << ", " << y << ")" << endl; }
    };

    CPoint P1(1, 2);
    P1.Print();

    return 0;
}
```

# 11. static 멤버

## # static 멤버 변수와 static 멤버 함수

### ■ static 멤버 변수

- 클래스 당 단 하나만 생성됨
- 객체 또는 클래스명을 통해 접근 가능 (단, public인 경우에 한함)
- 변수 생성을 위해서는 초기화 과정 필수

### ■ static 멤버 함수

- static 멤버(변수 또는 함수)에만 접근 가능
- 객체 또는 클래스명을 통해 접근 가능 (단, public인 경우에 한함)

# 11. static 멤버

예 : 현재 생성되어 있는 객체의 개수

```
class CPoint {
private :
    int x, y;
    static int count;    // static 멤버 변수

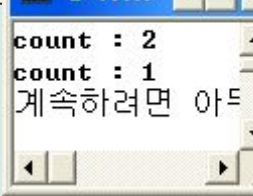
public :
    CPoint(int a = 0, int b = 0) : x(a), y(b) { count++; }
    ~CPoint() { count--; }
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }
    static int GetCount() { return count; }    // static 멤버 함수
};

int CPoint::count = 0;    // 초기값이 없을 경우 0으로 초기화

int main(void)
{
    CPoint P1(1, 2);
    CPoint *P2 = new CPoint(3, 4);

    cout << "count : " << CPoint::GetCount() << endl;    // static 함수 호출
    delete P2;
    cout << "count : " << P1.GetCount() << endl;

    return 0;
}
```



static 멤버 변수의 초기화 과정 필수

## 12. const 멤버와 const 객체

### ✦ const 멤버 변수와 const 멤버 함수

- const 멤버 변수 : 객체 생성과 동시에 초기화 필요
  - 멤버 초기화 구문 사용
- const 멤버 함수 : 멤버 변수의 값을 읽을 수 있으나 변경 불가능
  - 멤버 변수의 주소 반환 불가. 비const 멤버 함수의 호출 불가.

```
class CCircle {
private :
    double Radius;
    const double PI;

public :
    CCircle(double r = 0) : Radius(r), PI(3.14) { }           // const 변수 PI 초기화
    void SetRadius(double r) { Radius = r; }
    double GetArea() const { return (PI * Radius * Radius); } // const 함수
};

int main(void)
{
    CCircle Cir1(1);
    cout << "면적 : " << Cir1.GetArea() << endl;

    return 0;
}
```

## 12. const 멤버와 const 객체

### # const 객체

- 객체 생성 시 const 접두사 추가
- 멤버 변수의 값 변경 불가
- const 멤버 함수 이외의 멤버 함수에 대한 호출 불가

```
int main(void)
{
    const CCircle Cir1(1);      // const 객체
    // Cir1.SetRadius(2);      // X, 비const 함수 호출 불가
    cout << "면적 : " << Cir1.GetArea() << endl; // const 함수 호출 가능

    return 0;
}
```



## 13. explicit 생성자

### # explicit 생성자

- 생성자 앞에 explicit 키워드 추가
- 묵시적 형변환에 의한 객체 생성 불가

```
class CNumber {  
private :  
    int x;  
  
public :  
    CNumber() : x(0) {};  
    explicit CNumber(int a) : x(a) {}  
};
```

```
int main(void)  
{  
    CNumber N1;                // Ok!  
    CNumber N2(1);             // Ok!  
    CNumber N3 = CNumber(2);    // Ok!  
    CNumber N4 = 3;             // No! 불가능  
  
    return 0;  
}
```

3 → CNumber(3) 묵시적 형변환