

4장 클래스와 객체

- # 클래스와 객체
- # public과 private
- # 구조체와 클래스
- # 객체의 생성과 생성자
- # 객체의 소멸과 소멸자
- # 생성자와 소멸자의 호출 순서
- # 디폴트 생성자와 디폴트 소멸자
- # 멤버 초기화
- # 멤버 함수의 외부 정의
- # 멤버 함수의 인라인 함수 선언

1. 클래스와 객체

■ 추상 데이터형 : 속성(attribute) + 메서드 (method)

■ 예 : 자동차의 속성과 메서드

C++ : 주로 **class**로 표현

```
class Car {  
    속성 :  
        색상;  
        배기량;  
        현재속도;  
  
    메서드 :  
        가속하라;  
        멈춰라;  
        시동을켜라;  
};
```



- 구조체, 공용체, typedef 역시 추상 데이터형에 포함됨
→ 사용자 정의형(user-defined type)으로 불림

1. 클래스와 객체

타입 → 변수, 클래스 → 객체

- 객체 선언 방법 : 기본적으로는 변수 생성과 동일

➢ Car MyCar, YourCar;

➢ MyCar.색상 = Red;

예 : Car 클래스 구현

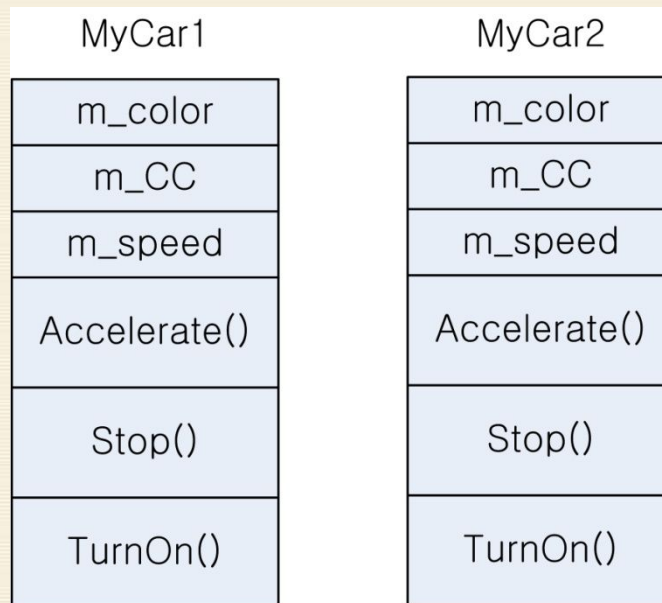
- 속성 : 멤버 변수
- 메서드 : 멤버 함수

```
class Car {  
    int m_color;           // 색상  
    int m_CC;              // 배기량  
    int m_speed;           // 속도  
  
    void Accelerate() { m_speed++; } // 가속  
    void Stop() { }        // 멈춤  
    void TurnOn() { }      // 시동켜기  
};  
  
int main(void)  
{  
    Car MyCar1, MyCar2;    // 객체 생성  
    MyCar1.m_speed = 0;  
    MyCar2.m_CC = 1000;  
    MyCar1.Accelerate();   // 멤버 함수 호출  
  
    return 0;  
}
```

1. 클래스와 객체

Car 클래스 예제에서 객체들의 메모리 구조 (개념적 표현)

- 각 객체 별로 별도의 변수와 함수 생성



```
class Car {  
    int m_color;           // 색상  
    int m_CC;              // 배기량  
    int m_speed;           // 속도  
  
    void Accelerate() { m_speed++; } // 가속  
    void Stop() { }         // 멈춤  
    void TurnOn() { }      // 시동켜기  
};  
  
int main(void)  
{  
    Car MyCar1, MyCar2;    // 객체 생성  
    MyCar1.m_speed = 0;  
    MyCar2.m_CC = 1000;  
    MyCar1.Accelerate();   // 멤버 함수 호출  
  
    return 0;  
}
```

- MyCar1.Accelerate() → MyCar1의 멤버 변수 m_speed 값 증가

실제로는 객체 별로 멤버 변수가 따로 생성되지만 멤버 함수는 단 하나만 생성됨 → 5.3절

2. public과 private

⌘ Car 클래스 예제의 컴파일 → 에러 발생

⌘ 정보 은닉 : 클래스 내부 데이터에 대한 접근 제어

- public 멤버 : 외부 접근 가능
- private 멤버 : 외부 접근 금지
→ 데이터 보호

⌘ 예 : Car 클래스

private 영역

public 영역

private, public이 여러 번 나올 수 있음
private 이후 : private 영역
public 이후 : public 영역

private 멤버에 대한
외부 접근 불가

주로 멤버 변수는 private 멤버로,
멤버 함수는 public 멤버로.
반드시 그럴 필요는 없음.

클래스 시작부에 **private, public**이
명시되어 있지 않으면
→ 디폴트로 **private**으로 인식

```
class Car {  
    private :  
        int m_color;        // 색상  
        int m_CC;           // 배기량  
        int m_speed;        // 속도  
  
    public :  
        void Accelerate() { m_speed++; } // 가속  
        void Stop() { }           // 멈춤  
        void TurnOn() { }         // 시동켜기  
};  
  
int main(void)  
{  
    Car MyCar1, MyCar2;  
    //MyCar1.m_speed = 0;  
    //MyCar2.m_CC = 1000;  
    MyCar1.Accelerate();  
  
    return 0;  
}
```

private 멤버에 대한
내부 접근 허용

3. 구조체와 클래스

✦ 예 : 평면상의 한 점을 나타내는 Point 구조체 만들기

```
struct Point {    // 평면상의 좌표(x, y)를 나타내는 구조체
    int x;
    int y;
};

int main(void)
{
    Point P1;
    P1.x = 3;
    P1.y = 4;

    cout << P1.x << " " << P1.y << endl;

    return 0;
}
```

구조체는 디폴트로 외부 접근 허용 (public)

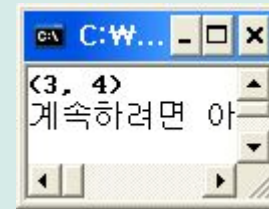
✦ C++의 구조체는 클래스와 99.8% 동일

- 멤버 변수와 멤버 함수 포함 가능
- private, public 영역 지정 가능

3. 구조체와 클래스

예 : 구조체를 클래스처럼 사용하기

```
struct Point {  
private :  
    int x;  
    int y;  
  
public :  
    void SetXY(int a, int b) { x = a; y = b; }  
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }  
};  
  
int main(void)  
{  
    Point P1;  
    P1.SetXY(3, 4);  
    P1.Print();  
  
    return 0;  
}
```



구조체와 클래스의 차이점 2가지

1. **private, public** 지정을 하지 않을 경우

- 클래스 : 디폴트로 **private**
- 구조체 : 디폴트로 **public**

2. 상속 : 8장

구조체와 클래스 중 어떤 것을 쓸 것인가?

→ 개인적 취향에 따라 선택

- **public** 멤버 변수만을 포함 → 구조체
- 함수 포함 또는 **private** 멤버 포함 → 클래스

4. 객체의 생성과 생성자

변수의 초기화 : 다음 2가지의 차이점은?

- `int a; a = 100;` ← 쓰레기값으로 초기화한 후 대입문에 의해 **100**으로 변경
- `int a = 100;` ← 메모리 생성과 동시에 **100**으로 초기화

구조체 변수의 초기화

```
struct Point {  
    int x;  
    int y;  
};  
  
int main(void)  
{  
    Point P1 = { 3, 4 };    // 구조체 변수의 초기화  
  
    return 0;  
}
```


4. 객체의 생성과 생성자

✚ 클래스 객체의 초기화 : 다음 프로그램의 문제점은?

```
class CPoint {  
private :  
    int x;  
  
public :  
    int y;  
};  
  
int main(void)  
{  
    CPoint P1 = { 3, 4 }; // X, 멤버 변수 x가 private이므로 직접 접근 불가능  
  
    return 0;  
}
```

클래스 객체 생성과 동시에 멤버 변수의 값을 초기화하는 방법 → **생성자**

4. 객체의 생성과 생성자

⌘ 생성자

- 일종의 멤버 함수. 객체가 생성되면 반드시 생성자가 호출됨
- 생성자에 대한 제약 사항
 - 생성자 이름은 클래스 이름과 같다.
 - 반환형이 없으며 어떤 것도 반환하지 않는다.
 - 매개변수는 일반 함수와 마찬가지로 자유롭게 줄 수 있다.
 - 생성자 오버로딩을 통해 다양한 생성자를 만들 수 있다.
 - 디폴트 매개변수를 사용할 수 있다.

⌘ 생성자의 예

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

4. 객체의 생성과 생성자

▣ 생성자의 호출

■ 객체 생성 시 매개변수 전달

```
int main(void)
{
    CPoint P1 = CPoint(3, 4);
    CPoint P2(5, 6);

    P1.Print();
    P2.Print();

    return 0;
}
```

둘 다 객체 생성
첫 번째는 대입이 아님!
주로 두 번째 스타일 사용.

4. 객체의 생성과 생성자

※ 생성자 오버로딩 : 여러 개의 생성자 작성 가능

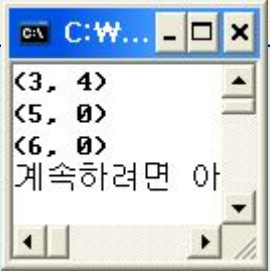
```
class CPoint {
private :
    int x;
    int y;

public :
    CPoint(int a, int b) { x = a; y = b; }    // 생성자1, 2개의 매개변수
    CPoint(int a) { x = a; y = 0; }          // 생성자2, 1개의 매개변수
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }
};

int main(void)
{
    CPoint P1(3, 4);        // 생성자1 사용
    CPoint P2(5);           // 생성자2 사용
    CPoint P3 = 6;          // 6 => CPoint(6)변환, 변환 시 생성자2 사용

    P1.Print();
    P2.Print();
    P3.Print();

    return 0;
}
```



생성자의 매개변수가 하나일 경우
이와 같은 사용 가능 → 묵시적 형변환을 통해 수행됨

4. 객체의 생성과 생성자

✦ 디폴트 매개변수 사용 가능

```
class CPoint {  
private :  
    int x;  
    int y;  
  
public :  
    CPoint(int a, int b = 0) { x = a; y = b; } // 생성자  
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }  
};  
  
int main(void)  
{  
    CPoint P1(3, 4);  
    CPoint P2(5);  
    CPoint P3 = 6;  
  
    P1.Print();  
    P2.Print();  
    P3.Print();  
  
    return 0;  
}
```

5. 객체의 소멸과 소멸자

✦ 객체 생성 → 생성자 호출, 객체 소멸 → 소멸자 호출

✦ 객체가 소멸되는 시점 (= 변수 소멸 시점)

① 지역변수 : 해당 함수의 수행이 완료되고 반환될 때

② 전역변수 : 프로그램이 종료될 때

✦ 소멸자에 대한 제약 사항

① 소멸자 이름은 클래스 이름과 동일하다. 단, 생성자와의 구별을 위해 이름 앞에 '~' 문자가 붙는다.

② 반환형 및 반환값이 없다.

③ 매개변수가 존재하지 않는다. 따라서 단 하나의 소멸자만 존재할 수 있다.

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


5. 객체의 소멸과 소멸자

✦ 소멸자가 필요한 예 : 메모리를 동적으로 생성하여 사용하는 경우

```
class CArray {  
private :  
    int count;  
    int *x;
```

```
public :  
    CArray(int a) { count = a; x = new int[count]; }    // 생성자  
    void Delete() { delete [] x; }                    // 메모리 해제  
    void Print() {  
        for (int i = 0; i < count; i++)  
            cout << x[i] << endl;  
    }  
};
```

```
int main(void)  
{  
    CArray ary(5);  
    ary.Print();  
    ary.Delete();  
  
    return 0;  
}
```

객체 생성 시 **new**를 통해 배열을 생성하였으므로
어딘가에서 **delete []**를 수행해야 함.

다음과 같은 소멸자를 추가한다면
더 이상 **Delete** 함수와 **Delete** 함수 호출이 필요없음

```
~CArray() { delete [] x; }    // 소멸자, 메모리 해제
```

6. 생성자와 소멸자의 호출 순서

※ 생성자 : 객체 생성 순, 소멸자 : 객체 생성의 역순



```
class CPoint {  
private :  
    int x;  
    int y;  
  
public :  
    CPoint(int a, int b) { x = a; y = b; cout << "생성자 : ";  
        Print();  
    }  
    ~CPoint() { cout << "소멸자 : ";  
        Print();  
    }  
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }  
};  
  
CPoint P1(1, 1);      // 전역 객체  
CPoint P2(2, 2);      // 전역 객체  
  
int main(void)  
{  
    CPoint P3(3, 3);    // 지역 객체  
    CPoint P4(4, 4);    // 지역 객체  
  
    return 0;  
}
```

멤버 함수 내에서
멤버 변수뿐만 아니라 다른 멤버 함수 호출 가능!

```
C:\C>C:\WWINDO...  
생성자 : (1, 1)  
생성자 : (2, 2)  
생성자 : (3, 3)  
생성자 : (4, 4)  
소멸자 : (4, 4)  
소멸자 : (3, 3)  
소멸자 : (2, 2)  
소멸자 : (1, 1)  
계속하려면 아무 키나
```

7. 디폴트 생성자와 디폴트 소멸자

✎ 다음 프로그램을 수행시켜 보라.

```
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 P1;  객체 생성 → 생성자 호출 → 생성자는 어디에?  
    P1.SetXY(3, 4);  
    P1.Print();  
  
    return 0;  객체 소멸 → 소멸자 호출 → 소멸자는 어디에?  
}
```

7. 디폴트 생성자와 디폴트 소멸자

디폴트 생성자와 디폴트 소멸자

- 생성자를 만들지 않을 경우 디폴트 생성자가 동작함
 - `CPoint() { }` // 매개 변수 없음. 특별히 하는 일은 없음
- 소멸자를 만들지 않을 경우 디폴트 소멸자가 동작함
 - `~CPoint() { }` // 특별히 하는 일은 없음
- 생성자를 1개 이상 명시적으로 추가하면 디폴트 생성자는 사라짐
- 소멸자를 명시적으로 추가하면 디폴트 소멸자는 사라짐

다음 프로그램의 문제점은?

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

생성자는 어디에?

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

7. 디폴트 생성자와 디폴트 소멸자

다음 코드의 의미는?

- `CPoint P1();` `// CPoint 객체 P1 생성? No!`
 - 이것은 매개변수가 없고 `CPoint` 객체를 반환하는 `P1` 함수의 프로토타입을 의미함
 - `CPoint P1(void);`
 - 비교 : `int func(void);`
- 객체 생성 시 매개 변수가 없을 경우에는 항상 다음과 같이 선언
 - `CPoint P1;`

8. 멤버 초기화

■ 객체 생성 시 멤버 변수 초기화 방법

- 생성자 내에서 대입문 사용 (기존 방법)
- 멤버 초기화 구문 사용 : 멤버 변수 메모리 생성과 동시에 초기화!

```
class CPoint {  
private :  
    int x;  
    int y;  
  
public :  
    CPoint(int a, int b) : x(a), y(b) { } // 멤버 초기화 구문  
    // CPoint(int a, int b) { x = a; y = b; }  
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }  
};  
  
int main(void)  
{  
    CPoint P1(3, 4);  
    P1.Print();  
  
    return 0;  
}
```

멤버 변수의 초기화 순서는
선언된 순서를 따름
생성자에서 기술된 순을 따르지 않음

(비교)
int a; a = 5;
int a = 5;

생성자 작성 시 멤버 초기화를 사용할 것인가?
생성자 내에서 대입문을 사용할 것인가?
→ 차이점만 알고 있다면 어느 것을 사용해도 무방
→ 단, 반드시 멤버 초기화를 사용해야 하는
상황이 있음 → 8.4절, 9.8절

9. 멤버 함수의 외부 정의

▣ 멤버 함수 정의 방법

- 내부 정의 : 클래스 선언 내부에 함수 몸체 작성
- 외부 정의 : 클래스 선언 내부에 함수 프로토타입만 선언, 함수 몸체는 클래스 외부에 작성

```
class CPoint {  
private :  
    int x, y;  
  
public :  
    CPoint(int a, int b);  
    void Move(int a, int b);  
    void Print();  
};
```

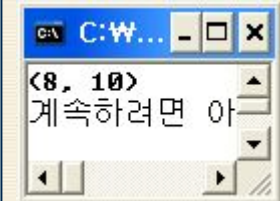
반환형 클래스명::함수명(...)

```
{  
  
}
```

```
CPoint::CPoint(int a, int b) : x(a), y(b) // 생성자의 외부 정의  
{  
}
```

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

```
void CPoint::Print() // 멤버 함수의 외부 정의  
{  
    cout << "(" << x << ", " << y << ")" << endl;  
}
```



```
int main(void)  
{  
    CPoint P1(3, 4);  
    P1.Move(5, 6);  
    P1.Print();  
  
    return 0;  
}
```

10. 멤버 함수의 인라인 함수 선언

멤버 함수를 내부 정의로 구현하는 경우

- 자동으로 인라인 함수로 인식 → 자동 인라인

외부 정의 시 인라인 함수로 선언하는 방법

- 외부 정의 시 inline 키워드만 추가하면 됨

```
class CPoint {  
private :  
    int x;  
    int y;  
  
public :  
    CPoint(int a, int b) : x(a), y(b) { }  
    inline void Print();  
};  
  
inline void CPoint::Print()    // 인라인 함수 선언  
{  
    cout << "(" << x << ", " << y << ")" << endl;  
}
```