과제

- 과제 2개 → 6_3, 6_4 → 클래스 구성, 생성자 사용
- 과제 6_3
  - gate 문제(뒤에 설명)

```
main{
        ...        // gate 문제
}
```

- 과제 6_4
  - 엘리베이터 문제 (뒤에 설명)

```
main{
        ...        // 6_3 문제
}
```

P3() gate 문제

```
int main() {
    AndGate a_and, b_and;
    OrGate c_or;

    a_and.input(1, 0);
    a_and.op();
    b_and.input(1, 1);
    b_and.op();

    c_or.input(1, 0);
    c_or.op();

    c_or.input(a_and, b_and);
    c_or.op();
    return 0;
}
```

- 멤버 변수 : **private**
  - **x, y** : **gate** 입력값을 저장
  - **output** ; **gate** 연산결과(출력값)을 저장
- **input** :
  - 입력 **2**개 값 받으면 **x, y** 멤버에 저장
  - 입력 게이트 **2**개 받으면 각 게이트 출력값을 **x, y** 멤버에 저장
- **op** : 멤버 **x, y** 값을 사용하여 각 **gate** 연산결과를 **output** 멤버에 저장하고 **gate** 종류, 입력값, 출력값을 출력
- 기타 필요 멤버함수 추가

```
AND input : 1 0 -> 0

AND input : 1 1 -> 1

OR input : 1 0 -> 1

AND input : 1 0 -> 0
AND input : 1 1 -> 1
OR input : 0 1 -> 1
```

P4() 엘리베이터 문제

```
int main( ) {
    Elevator a(1), b(8);  // 엘리베이터 2개 생성, a는 1층에, b는 8층에 존재

    a.Button(6);          // 6층 버튼 누르고 닫힘버튼 누르면 현재층에서 6층으로 이동
    a.CloseDoor();        // 한층 이동시 마다 현재 층 출력
    cout << endl;

    b.Button(3);          // 3층 버튼 누르고 닫힘버튼 누르면 현재층에서 3층으로 이동
    b.CloseDoor();        // 한층 이동시 마다 현재 층 출력

    return 0;
}
```

멤버변수 설명 없음..
멤버변수 무엇이 필요할지 고민해 작성

```
현재층: 1
현재층: 2
현재층: 3
현재층: 4
현재층: 5
현재층: 6
```

```
현재층: 8
현재층: 7
현재층: 6
현재층: 5
현재층: 4
현재층: 3
```

멤버가 다른 객체인 경우(Has a 관계) 초기화, 219쪽

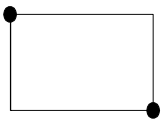
- 지금까지 대부분 멤버변수는 일반변수
  → 멤버변수는 객체도 가능

```
class Car{
    int gear;          // int type 변수
    string color;      // string 클래스 객체
    ...
};
```

```
class Point{...};

class Rect{
    Point a, b;   // Point class
    ...
};
```

멤버가 다른 객체인 경우(Has a 관계) 초기화

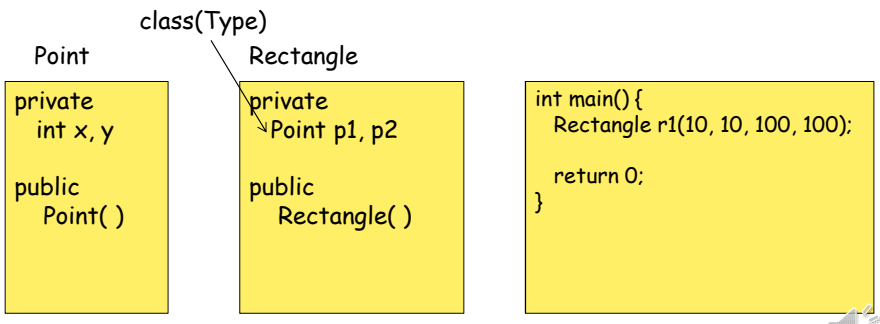
- 클래스 작성시 객체간의 관계가 있는 경우 다음 사항을 고려하여 클래스 작성
- has-a 관계(7장) → 생성자 설명으로 6장에서 설명
  - 한 객체가 다른 객체를 포함하는 관계 →" a 는 b 를 포함한다.(가지고 있다.) " 성립
    - 자동차는 바퀴를 가지고 있다.
    - 사각형은 두점(좌상단점, 하단점)을 가지고 있다.
    - 도서관은 책을 가지고 있다.



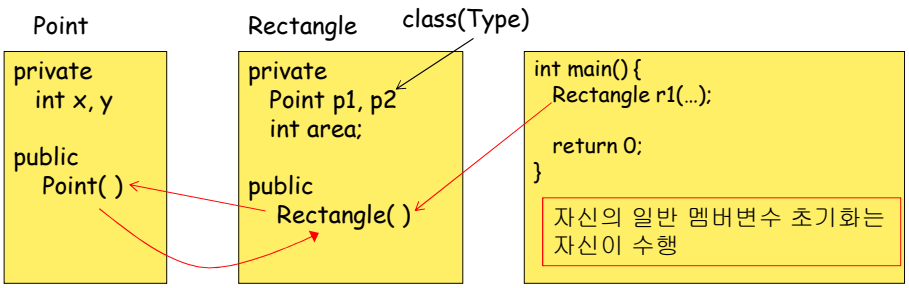
- is-a 관계(8장, 상속)
  - 한 객체가 다른 객체의 특수한 경우 → "a 는 b 이다 " 성립
    - 승용차, 트럭은 자동차 이다.
    - 사자, 개, 고양이 는 동물이다.
- → 위의 2 경우 모두 a, b 를 클래스로 만들고 두 클래스간의 관계를 만들어 줌

219쪽, 멤버가 다른 객체인 경우(Has a 관계) 초기화

- has-a 관계의 표현
  - 사각형은 두점(좌상단점, 하단점)으로 구성(포함)
  - 사각형 클래스는 멤버 변수/객체 로 점(Point) 클래스 객체를 가짐(포함)
- int 와 같이 클래스도 type 의 일종으로 생각하면 간단



219쪽, Has a 관계 생성자 호출 순서



- main 에서 Rectangle r1 객체 생성 → Rectangle 생성자 호출

1) Rectangle 생성자에서
   - p1, p2 초기화하려니 Point 클래스 객체 → Point 클래스 생성자 호출(p1, p2 생성하며 각각 호출)
   - (cf) 또한, private 객체 멤버 접근(초기화)은 해당 객체의 멤버함수만 가능
2) Point 생성자 에서 p1의 x/y 초기화, p2의 x/y 초기화 (p1 생성시 호출, p2 생성시 호출)
3) Rectangle 생성자에서 area 초기화(area 는 자신의 일반 멤버변수)

has-a 관계에서 다른 클래스의 생성자 호출

- 객체 생성시 생성자 호출된다. → 객체를 만들면 생성자 호출됨
- 생성자 수행 전에 멤버 객체/변수 생성됨 → 생성자 호출

```
class Point {
private:
  int x, y;
public:
  Point(int xx, int yy) : x(xx), y(yy) { }
    …
};

class Rectangle{
    Point p1, p2;
    int area;
public:
    Rectangle(int x1, int y1, int x2, int y2)
     : p1(x1, y1), p2(x2, y2), area(10)  {   }

    …
};
int main() {
  Rectangle r1(…);   return 0;
}
```

x(xx) → int x(xx); 로 해석 ← int x=xx;

생성자 시작 전에 멤버 객체/변수 생성됨
→ 생성자 호출

p1(x1, y1) → Point p1(x1, y1);
p2(x2, y2) → Point p2(x2, y2); 로 해석

→ Point class 생성자 두번 호출.

멤버변수 초기화는
해당 클래스 생성자에서 수행

멤버가 다른 객체인 경우(Has a 관계) 초기화

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

class Point{
    int x, y;
public:
    Point(int a, int b) : x(a), y(b) { }    // int x(a); int y(b);
};

class Rectangle{
    Point p1, p2;
public:
    Rectangle(int x1, int y1, int x2, int y2) : p1(x1, y1), p2(x2, y2)
    { }                 // Point p1(x1, y1);
                        // Point p2(x2, y2); 로 해석
};

int main() {
  Rectangle r1(10, 10, 100, 100);
  return 0;
}
```

Point 생성자 호출

has-a 관계에서 생성자 호출 순서

```
class Point {
    int x, y;
public:
    Point(int a, int b) : x(a), y(b)
    {    cout << "aaa" << endl;                }
};

class Rectangle {
    Point p1, p2;
public:
    Rectangle(int x1, int y1, int x2, int y2) : p1(x1, y1), p2(x2, y2)
    {    cout << "bbb" << endl;        }
};

int main() {
    Rectangle r1(10, 10, 100, 100);
    return 0;
}
```

출력
aaa
aaa
bbb

has-a 관계에서 멤버객체 접근

```
class Point {
    int x, y;
public:
    Point(int a, int b) : x(a), y(b) {  cout << "aaa" << endl;   }
    void prn() {      cout << x << "  " << y << endl;      }
};

class Rectangle {
    Point p1, p2;
public:
    Rectangle(int x1, int y1, int x2, int y2) : p1(x1, y1), p2(x2, y2)
    {         cout << "bbb" << endl;         }
    void prn()   {
        cout << p1.x << p1.y << p2.x << p2.y << endl;   // ??
        p1.prn();          p2.prn();
    }
};

int main() {
    Rectangle r1(10, 10, 100, 100);
    r1.prn();     // 두 점 좌표 (10, 10)  (100, 100) 출력
    return 0;
}
```

main 에서 r1.prn() 함수 호출하여 사각형 점들 좌표 출력하려면 ?

Point 멤버변수는 private 이므로 Rectangle 에서 접근 못함

참고) 문자열 리터럴 저장

- 문자열 리터럴(string literal)이란 큰 따옴표로 둘러싼 문자의 연속체 → 문자열 상수
  "abcd edfg"

- VS 2017 이상에서는 리터럴을 char* 가 가리킬 수 없다. 반드시 const char* 가 가리켜야 함

```
char* p = "school";        // error → p 가리키는 내용(문자열 상수) 수정 가능
const char* p = "school";     // ok → p 가리키는 내용(문자열 상수) 수정 불가
```

cf)

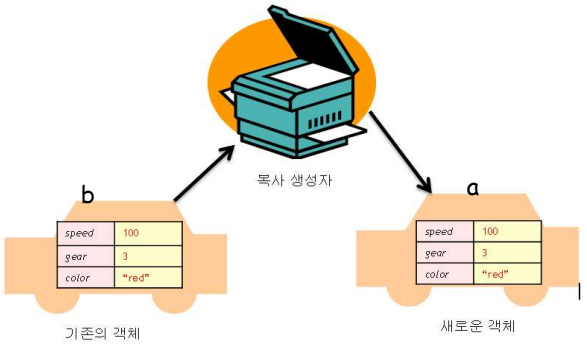
```
char* p = "school";   → p 가리키는 곳의 내용은 변수(변화 가능)

const (char* p) = "school";
```

→ ( ) 안의 것이 상수
→ p 가리키는 곳의 내용(문자열)은 상수(변화 불가능)

복사 생성자.. 생성자의 일종

- 객체 복사를 하는 생성자 → 한 객체의 멤버 값을 다른 객체의 멤버에 복사
- 객체를 선언하며 같은 타입의 객체 멤버값들로 초기화 할 때 호출되는 함수.
  - " Car a = b; " 혹은 " Car a(b); " 와 같이 객체 선언하며 초기화 할 때 사용

- 이전 코드에서 복사 생성자/대입(할당) 연산자 모두 없이 사용 → 자동으로 디폴트 복사 생성자가 생성되어 수행(이 경우 없어도 무방)
  - but, 멤버변수로 포인터 사용시 이들 모두 작성 필요.
  - 또한 포인터 멤버변수 없어도 필요한 경우도 있음. → 나중 설명



복사 생성자

```
int x = 5, z=10;
int y = x;

z = y;
```

```
Car a(1, 2, "White"), c(2, 3, "Black");  // 생성자 호출
Car b = a;                               // 복사 생성자 호출

c = b;                                   // 할당 연산자 호출(10장)
```

x
5

z
10 → 5

y
5

a
speed = 1
gear = 2
color = "White"

c
speed = 2 → 1
gear = 3 → 2
color = "Black" → "White"

b
speed = 1
gear = 2
color = "White"

- 소멸자, 복사 생성자 → 멤버로 포인터변수 있으면 작성, 아니면 작성 불필요(자동 수행)

복사 생성자

- 복사 생성자 함수 이름은 클래스 이름과 동일. 반환값 없음. → 함수이름은 생성자와 동일
- 복사 생성자 함수는 매개변수로 자신과 같은 타입의 객체를 참조자 로 받는다.
  - 복사 생성자 함수의 매개변수는 const 사용 → 값만 복사해야 하기에 매개변수 내용 불변
  - 참고) 생성자는 매개변수로 멤버변수 값들을 받음.(두 함수 이름 같지만 차이점)

class 정의 → class Car { int **speed, gear**; string **color**; ... }

```
// 생성자 → 멤버값을 인자로 받음
Car(int s, int g, string c) { ... }
```

```
// 복사 생성자 → 객체를 인자로 받음
Car(const Car &obj) { ...}
```

참고) 생성자, 복사 생성자 함수 이름은 동일

- 생성자는 매개변수로 멤버변수 값들을 받음 → Car a(2, 3, "aaa");
- 복사 생성자는 매개변수로 객체를 받음 → Car a(b); // Car a = b; 로 해석
- 복사 생성자 호출되면 생성자 호출 안됨.

복사 생성자

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

class Car {
        int speed, gear; // 속도
        string color; // 색상

public:
        Car(int s, int g, string c) : speed(s), gear(g), color(c) {
                cout << "생성자 호출" << endl;
        }
        Car(const Car &obj) : speed(obj.speed), gear(obj.gear), color(obj.color) {
                cout << "복사 생성자 호출" << endl;
        }
...
};
int main(){
        Car c1(0, 1, "yellow");
        Car c2(c1);

        c1.print();
        c2.print();
        return 0;
}
```

c1
speed = 0
gear = 1
color = "yellow"

c2
speed = 0
gear = 1
color = "yellow"

c2 의 복사 생성자 호출,
Car c2=c1; 과 동일

이 코드에서 복사 생성자는 작성하지 않아도 자동 생성(작성 불필요)
→ 일반적으로 멤버변수에 포인터변수가 없으면 작성할 필요 없음.

멤버가 다른 객체인 경우(Has a 관계) 초기화

```
class Point {
      int x, y;
public:
      Point(int a, int b) : x(a), y(b) { }
      void prn() {
            cout << x << "  " << y << endl;
      }
};

class Rectangle {
      Point p1, p2;
public:
      Rectangle(int x1, int y1, int x2, int y2)
       : p1(x1, y1), p2(x2, y2) { }       // (1)

      Rectangle(Point pp1, Point pp2)
       : p1(pp1), p2(pp2) { }             // (2)

      void prn() {
            p1.prn();
            p2.prn();
      }
};
```

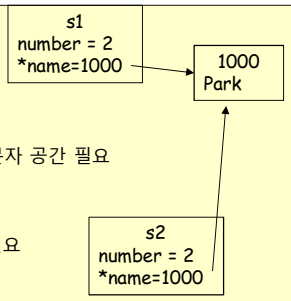
- Point 객체 사용하여 Rectangle 객체 생성시

```
int main() {
      Point p1(10, 20), p2(100, 200);
      Rectangle r1(10, 10, 100, 100);
      Rectangle r2(p1, p2);

      r1.prn();
      r2.prn();

      return 0;
}
```

- 자신 멤버 초기화는 자신이 수행
- 자신 멤버 출력은 자신이 수행 ← private
- (1) 에서 Point p1(x1, y1) → 생성자 호출
- (2)에서 Point p1(pp1) → 복사 생성자 호출 → Point class 복사 생성자 없음 → 자동 생성 수행

얕은 복사-복사생성자 작성 필요성(포인터 멤버)

```
class Student {
      char *name; // 이름
      int number;
public:
      Student(const char *p, int n) {  // 생성자
            cout << "메모리 할당" << endl;
            name = new char[strlen(p)+1];    // 널문자 공간 필요
            strcpy_s(name, (strlen(p)+1), p);
            number = n;
      }
      ~Student() { // 멤버가 포인터이면 소멸자 필요
            cout << "메모리 소멸" << endl;
            delete [] name;
      }
};

int main()
{
      Student s1("Park", 2);
      Student s2(s1);     // 복사 생성자 호출
      return 0;           // ERROR
}
```

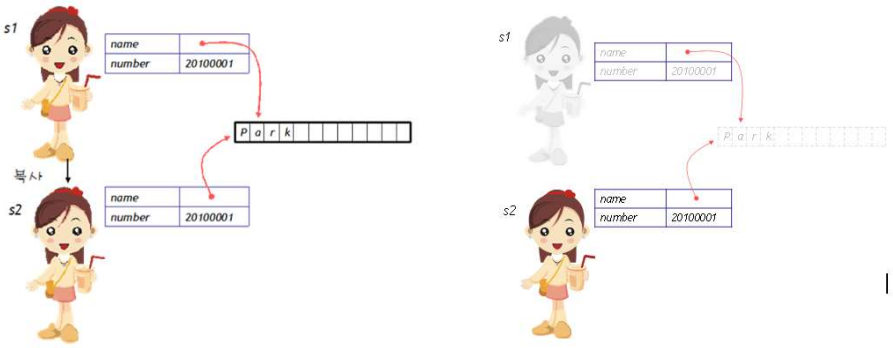


복사생성자 없음 → 자동 생성.

```
Student(Student &a){
   name = a.name;
   number = a.number;
} // 단순히 변수값만 복사
```

생성자에서 **new** 이용 저장공간 생성하면 → 소멸자에서 그 공간 꼭 해제해야함

문제점



이름을 저장하는 동적 메모리 공간이 별도로 할당되지 않았음
두 포인터가 같은 공간 지칭 → 지운 공간을 또 지우려 할때 오류 발생

포인터 멤버 있는 경우 생성자, 복사 생성자, 소멸자 할일

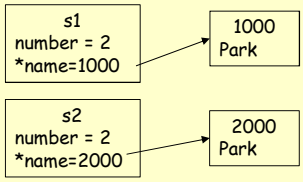
- 생성자, 복사 생성자
  - 포인터 멤버변수에 메모리 공간 할당하고 값 저장
- 소멸자
  - 포인터 변수에 할당한 메모리 공간을 해제

→ 이와 같이 하면 객체 생성시 메모리 할당, 객체 소멸시 메모리 해제가 자동으로 되어 메모리 관리 효율적, 메모리 error 가능성 적어짐

깊은 복사-다음과 같은 복사 생성자 필요(포인터 멤버)

```
class Student {
   char *name; // 이름
   int number;
public:
   // 생성자
   Student(const char *p, int n) {
        cout << "메모리 할당" << endl;
        name = new char[strlen(p)+1];
        strcpy_s(name, (strlen(p) + 1), p);
        number = n;
   }
   // 복사 생성자
   Student(const Student& s) {
       cout << "메모리할당 "<< endl;
       name = new char[strlen(s.name)+1];
       strcpy_s(name, (strlen(s.name)+1), s.name);
       number = s.number;
   }

// 복사 생성자는 매개변수 객체의 포인터 멤버변수가
// 갖는 공간 만큼 공간을 생성하여
// 자신의 멤버변수에 할당하고 그 곳에 값을 저장
```

```
~Student() {
            cout << "메모리 소멸" << endl;
            delete [] name;
   }
};

int main() {
   Student s1("Park", 20100001);
   Student s2(s1);        // 복사 생성자 호출
   return 0;
}
```




```
메모리 할당
메모리 할당
메모리 소멸
메모리 소멸
```

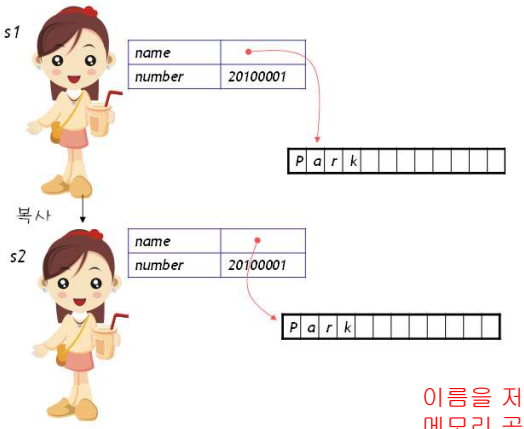


그림 6.9 깊은 복사

이름을 저장하는 동적 메모리 공간을 별도로 할당

복사 생성자

- 멤버변수에 포인터 변수가 없는 경우 → 대부분 복사 생성자 만들 필요 없다.
- 멤버변수에 포인터 변수가 있는 경우 → 복사 생성자 만든다.
  - 멤버변수값을 매개변수로 넘어온 객체의 멤버변수값으로 초기화 한다.
  - 포인터 멤버변수는 공간을 할당하고 매개변수 객체의 값을 저장

```
class Student {
        char *name;
        int number;
public:
        Student(const Student& s) {        // s는 변하지 않아야 하기에 const
                name = new char[strlen(s.name)+1];
                strcpy(name, s.name); // 2015에서는 strcpy_s(..) 사용
                number = s.number;
        }
        ...
```

복사 생성자가 호출되는 경우

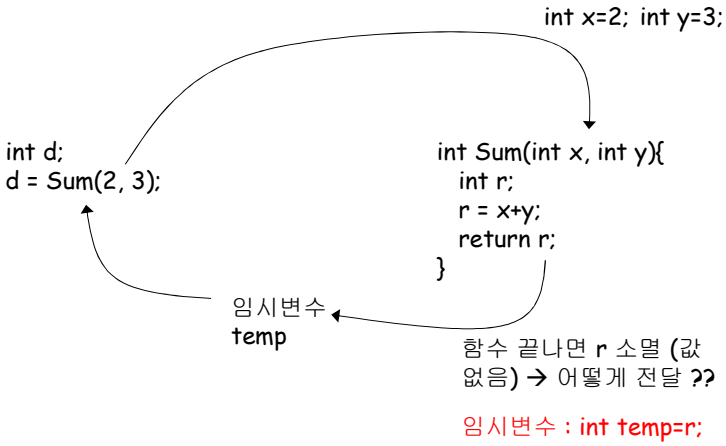
- 기존의 객체의 내용을 복사하여서 새로운 객체를 만드는 경우
  - 새로운 객체 선언시 복사

```
Car a;
Car b(a), c = b; // 복사 생성자 호출
```

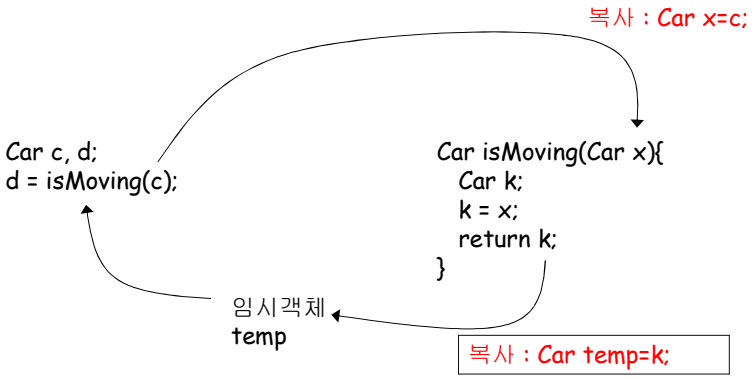
- 함수 호출시 객체를 값으로 함수의 매개 변수로 전달하는 경우
- 함수에서 객체를 반환하는 경우 → 임시 객체에 복사 전달

복사 생성자

C 함수에서 임시변수 반환



함수에서 객체 전달시 복사 생성자



예제

```
class Car {
...
        Car(int s, int g, string c) : speed(s), gear(g), color(c) {
                cout << "생성자 호출" << endl;
        }
        Car(const Car &obj) : speed(obj.speed), gear(obj.gear), color(obj.color) {
                cout << "복사 생성자 호출" << endl;
        }
...
};
void isMoving(Car obj) {            // 멤버함수 아님, 인자 전달시 Car obj = c; 로 해석
        if( obj.getSpeed() > 0 )
                cout << "움직이고 있습니다" << endl;
        else
                cout << "정지해 있습니다" << endl;
}
int main() {
        Car c(0, 1, "white");
        isMoving(c);
        return 0;
}
```

생성자 호출
복사 생성자 호출
정지해 있습니다

함수에서의 복사 생성자 예제

```
class Car {
   int speed, gear; // 속도
public:
   Car(int s=0, int g=0) : speed(s), gear(g) {
            cout << "생성자 호출" << endl;
   }
   Car(const Car &obj) : speed(obj.speed), gear(obj.gear) {
            cout << "복사 생성자 호출" << endl;
   }
   void print(){ cout << speed << " " << gear << endl;  }
};

Car isMoving(Car obj) {// 멤버함수 아님, 인자 전달시 Car obj = c; 로 해석
   Car t;                  // ... (2)
   t = obj;                // 할당 연산자
   return t;               // ... (3)
}

int main() {
   Car c(0, 1), d(10, 2), a(c), b=d;   // (0)
   d = isMoving(c);                    // ... (1)
   d.print();
   return 0;
}
```

```
생성자 호출
생성자 호출
복사 생성자 호출
복사 생성자 호출  (0) 까지
복사 생성자 호출  (1)에서 호출한 함수에 인자 전달시
생성자 호출       (2)
복사 생성자 호출  (3) 서 임시객체 전달시
0, 1, white
```

정리) 디폴트 멤버 함수

- 디폴트 생성자
- 디폴트 소멸자
- 디폴트 복사 생성자
- 디폴트 할당 연산자

포인터 멤버 있으면 작성 필요.

없으면 자동으로 추가된다.

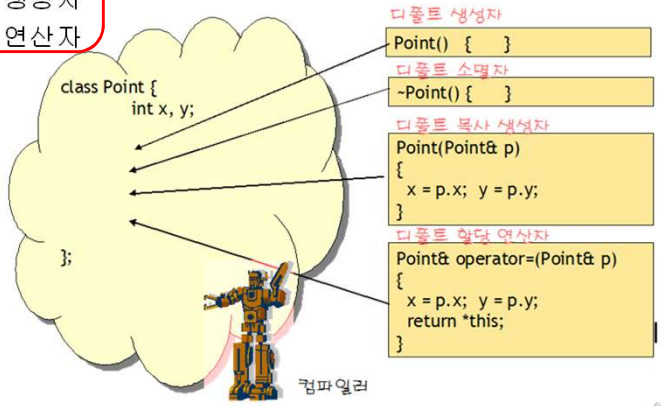


그림 10.7 디폴트 멤버 함수의 추가

키워드 explicit

C 스타일 초기화

```
int num=20;
int &ref=num;
```

C++ 스타일 초기화

```
int num(20);
int &ref(num);
```

```
SoSimple sim2=sim1;  →  SoSimple sim2(sim1);
```

- 좌변과 같이 작성하면 우변과 같이 자동으로 형변환됨.

- 이러한 형변환은 복사 생성자를 explicit으로 선언하면 막을 수 있다.

→ 복사 생성자에 explicit 사용하면 좌변과 같이 사용 못함, 우변만 가능

키워드 explicit

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

class Car {
        int speed;
public:
        Car(int s) : speed(s) {         }
        explicit Car(const Car &obj) : speed(obj.speed) {       }
        ...
};
int main(){
        Car c1(10);
        Car c2(c1);  // Car c2 = c1  사용 못함

        ...
        return 0;
}
```