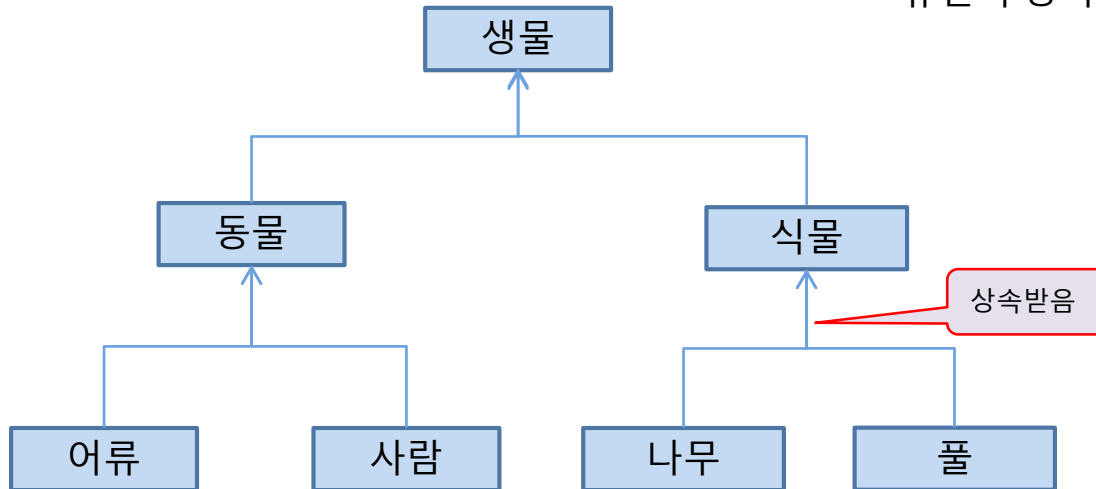
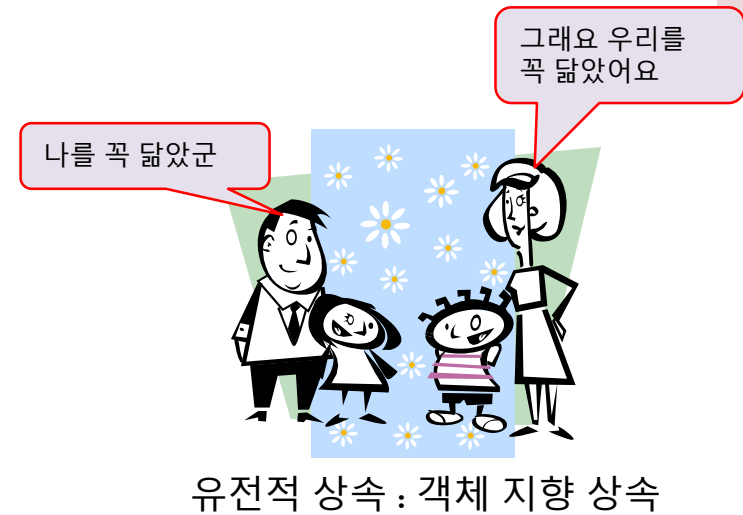


8장 상속

- 1) 상속의 개념
- 2) 클래스의 상속과 객체
- 3) 상속과 객체 포인터
- 4) `protected` 접근 지정
- 5) 상속과 생성자, 소멸자
- 6) 상속의 종류: `public`, `protected`, `private` 상속
- 7) 다중상속
- 8) 가상상속

유전적 상속과 객체 지향 상속



유전적 상속과 관계된
생물 분류

C++에서의 상속(Inheritance)

- C++에서의 상속이란?

- ▶ 클래스 사이에서 상속관계 정의
 - ▶ 자식 클래스의 객체가 생성될 때, 자신의 멤버 뿐 아니라 부모 클래스의 멤버를 포함한다.
- ▶ 기본 클래스의 속성과 기능을 파생 클래스에 물려주는 것
 - ▶ 기본 클래스(base class) - 상속해주는 클래스. 부모 클래스
 - ▶ 파생 클래스(derived class) - 상속받는 클래스. 자식 클래스
 - ▶ 기본 클래스의 속성과 기능을 물려받고 자신만의 속성과 기능을 추가하여 작성
 - ▶ Java(슈퍼 클래스, 서브클래스), C#(부모클래스, 자식클래스)
- ▶ 다중 상속을 통한 클래스의 재 활용성 높임
- ▶ C++는 다중 상속을 허용

상속의 표현



상속 관계 표현

```
class Phone {  
    void call();  
    void receive();  
};
```

Phone을 상속받는다.

```
class MobilePhone : public Phone {  
    void connectWireless();  
    void recharge();  
};
```

MobilePhone을 상속받는다.

```
class MusicPhone : public MobilePhone {  
    void downloadMusic();  
    void play();  
};
```

C++로 상속 선언



전화기



휴대 전화기



음악 기능
전화기

상속의 목적 및 장점

1. 간결한 클래스 작성

- ▶ 기본 클래스의 기능을 물려받아 파생 클래스를 간결하게 작성
 - ▶ 코드 중복 제거와 수정 용이

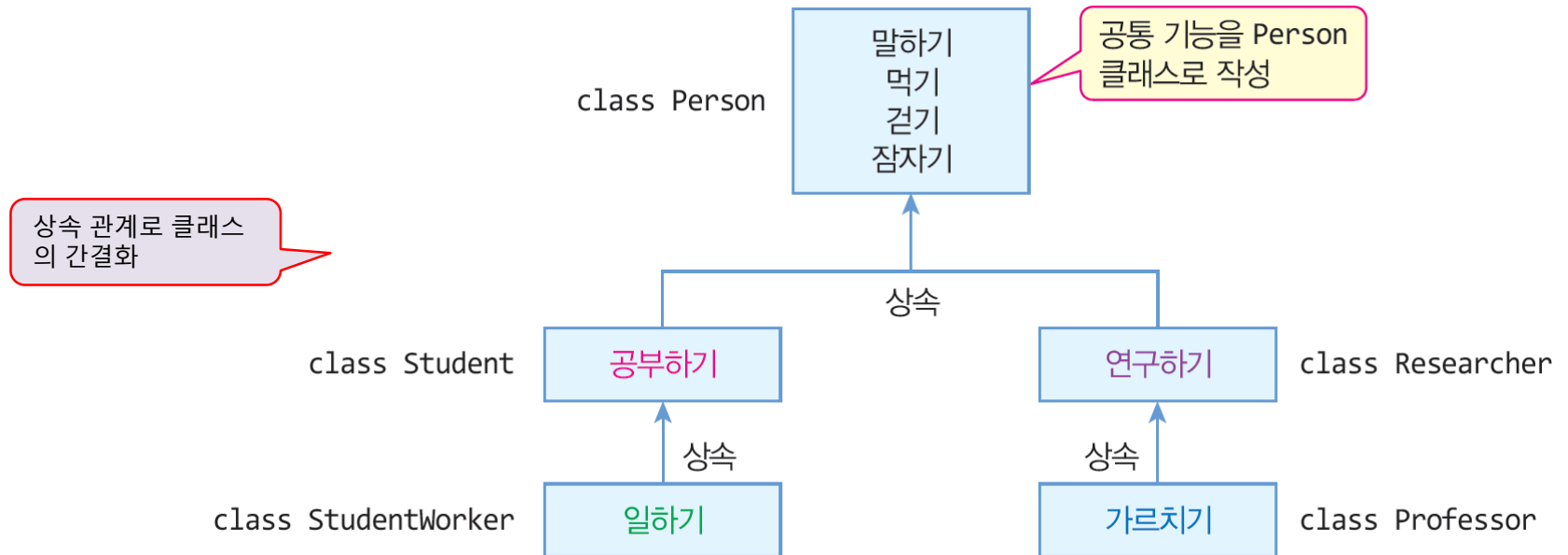
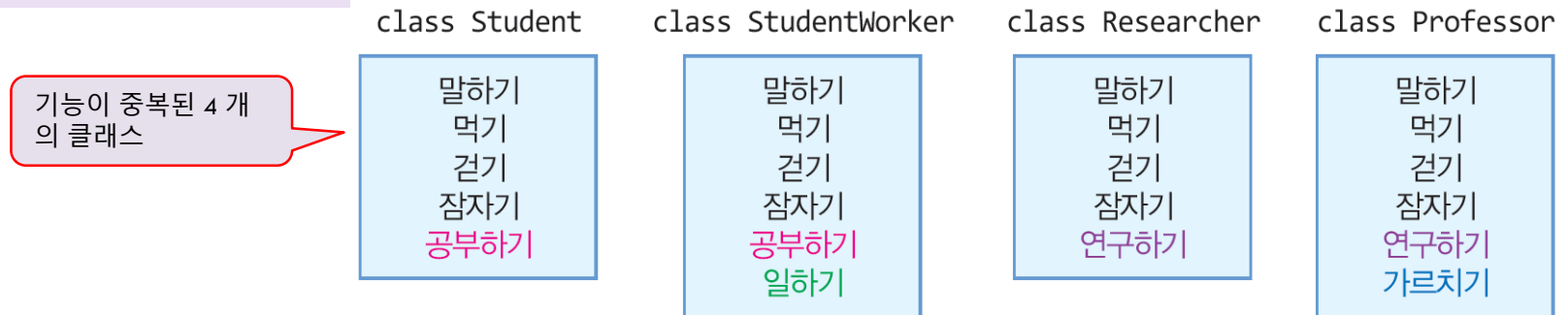
2. 클래스 간의 계층적 분류 및 관리의 용이함

- ▶ 상속은 클래스들의 구조적 관계 파악 용이

3. 클래스 재사용과 확장을 통한 소프트웨어 생산성 향상

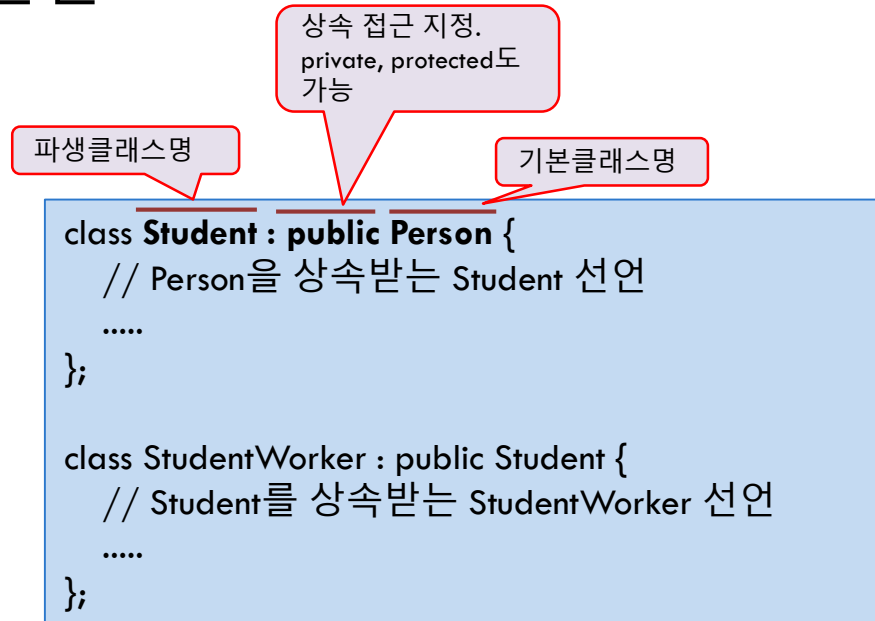
- ▶ 빠른 소프트웨어 생산 필요
- ▶ 기존에 작성한 클래스의 재사용 – 상속
 - ▶ 상속받아 새로운 기능을 확장
- ▶ 앞으로 있을 상속에 대비한 클래스의 객체 지향적 설계 필요

상속 관계로 클래스의 간결화 사례



상속 선언

● 상속 선언



- ▶ `Student` 클래스는 `Person` 클래스의 멤버를 물려받는다.
- ▶ `StudentWorker` 클래스는 `Student`의 멤버를 물려받는다.
 - ▶ `Student`가 물려받은 `Person`의 멤버도 함께 물려받는다.

예제 8-1 Point 클래스를 상속받는 ColorPoint 클래스 만들기

Point.h

```
#ifndef POINT_H
#define POINT_H
#include <iostream>
#include <string>
using namespace std;

class Point {
    int x, y; //한 점 (x,y) 좌표값
public:
    void set(int x, int y) { this->x = x; this->y = y; }
    void showPoint() {
        cout << "(" << x << ", " << y << ")" << endl;
    }
};
#endif
```

ColorPoint.h

```
#ifndef COLORPOINT_H
#define COLORPOINT_H

#include "Point.h"

class ColorPoint : public Point {
    string color; // 점의 색 표현
public:
    void setColor(string color) {this->color = color; }
    void showColorPoint();
};
#endif
```


예제 8-1 Point 클래스를 상속받는 ColorPoint 클래스 만들기

ColorPoint.cpp

```
#include "ColorPoint.h"

void ColorPoint::showColorPoint() {
    cout << color << ":";
    showPoint(); // Point의 showPoint() 호출
}
```

main.cpp

```
#include "ColorPoint.h"
#include "Point.h"

int main() {
    Point p; // 기본 클래스의 객체 생성
    ColorPoint cp; // 파생 클래스의 객체 생성

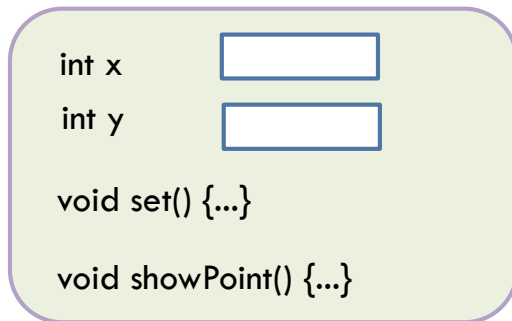
    cp.set(3,4); // 기본 클래스의 멤버 호출
    cp.setColor("Red"); // 파생 클래스의 멤버 호출
    cp.showColorPoint(); // 파생 클래스의 멤버 호출
}
```

Red:(3,4)

파생 클래스의 객체 구성

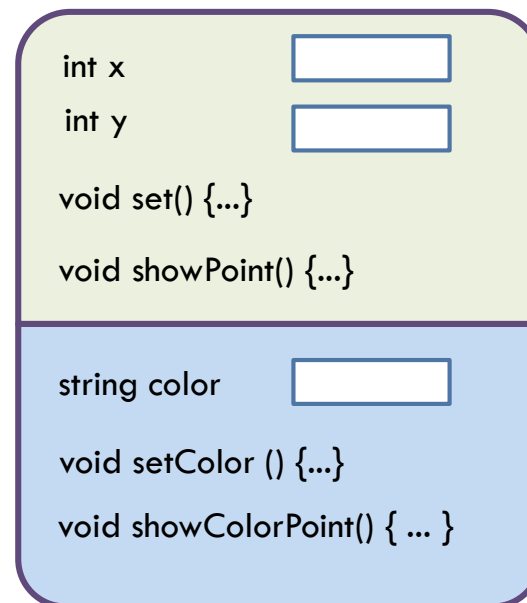
```
class Point {  
    int x, y; // 한 점 (x,y) 좌표 값  
public:  
    void set(int x, int y);  
    void showPoint();  
};
```

Point p;



```
class ColorPoint : public Point { // Point를 상속받음  
    string color; // 점의 색 표현  
public:  
    void setColor(string color);  
    void showColorPoint();  
};
```

ColorPoint cp;

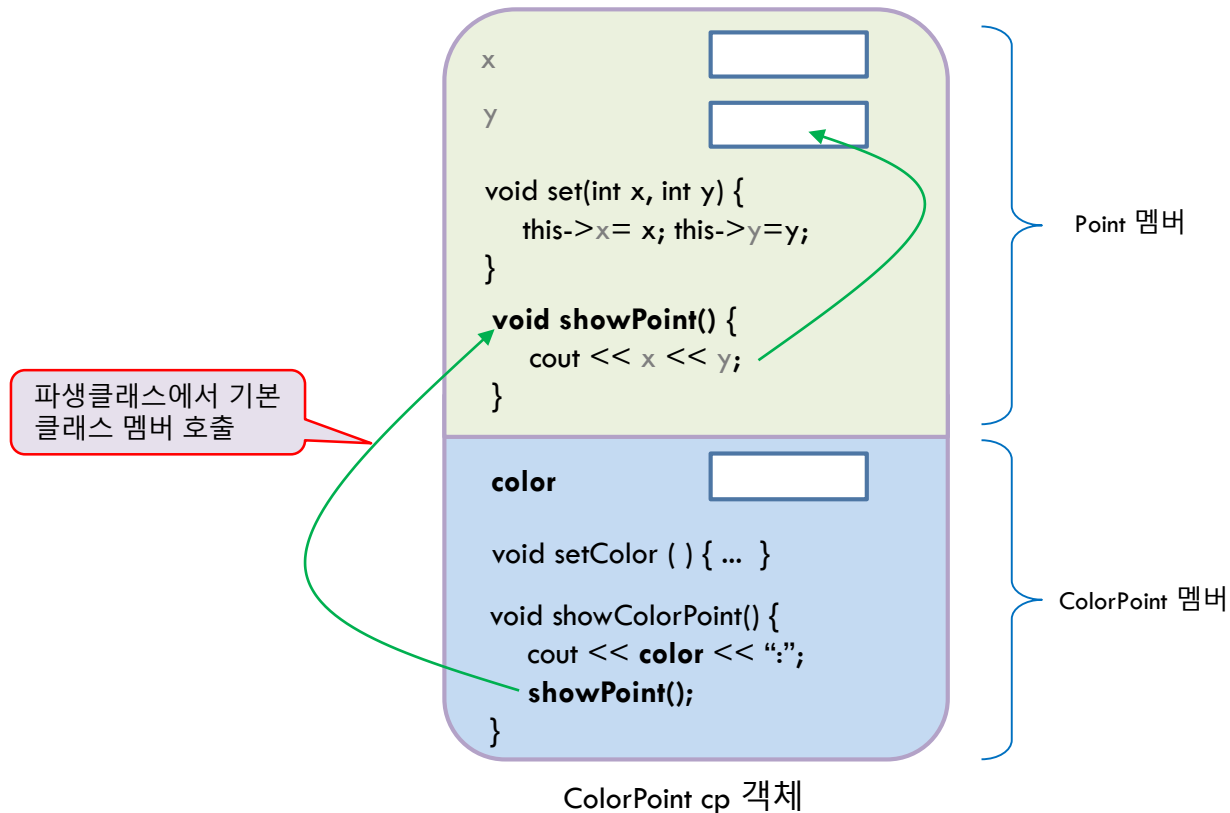


파생 클래스의 객체는 기본 클래스의 멤버 포함

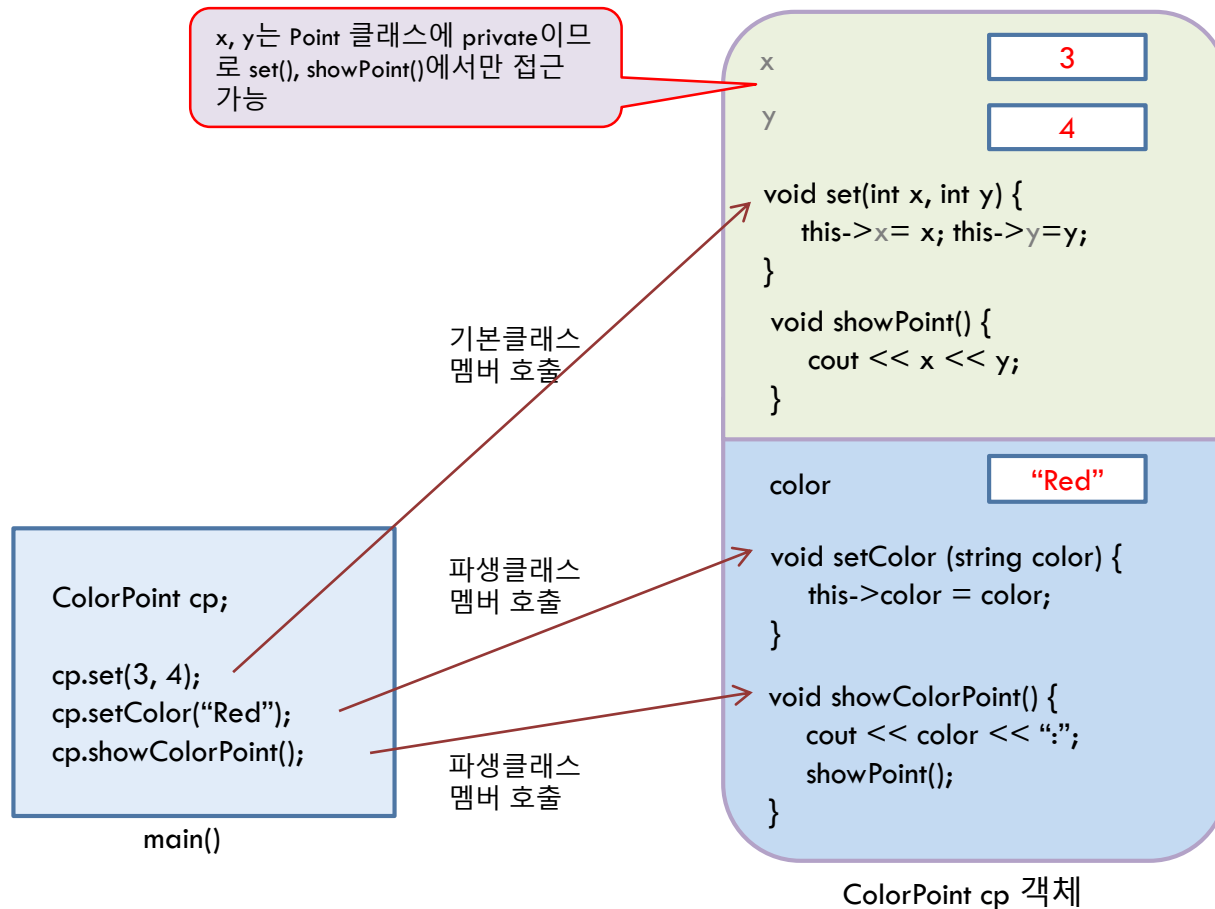
기본클래스 멤버

파생클래스 멤버

파생 클래스에서 기본 클래스 멤버 접근



외부에서 파생 클래스 객체에 대한 접근



예제 8-1 Point 클래스를 상속받는 ColorPoint 클래스 만들기(연산자 중복 <<, >>)

Point.h

```
#ifndef POINT_H
#define POINT_H
#include <iostream>
#include <string>
using namespace std;

class Point {
protected:
    int x, y; //한 점 (x,y) 좌표값
public:
    void set(int x, int y) { this->x = x; this->y = y; }
    void showPoint() {
        cout << "(" << x << "," << y << ")" << endl;
    }

    friend ostream& operator<<(ostream& os, const Point& p);
    friend istream& operator>>(istream& is, Point& p);
};
#endif
```

Point.cpp

```
#include "Point.h"
ostream& operator<<(ostream& os, const Point& p)
{
    os<< "(" << p.x << "," << p.y << ")" ;
    return os;
}

istream& operator>>(istream& is, Point& p)
{
    cout << "Point 입력(x,y 좌표 순서로):";
    is>>p.x>>p.y;
    return is;
}
```

protected로 지정한 이유 => ColorPoint에서 x, y 멤버 접근을 위해

예제 8-1 Point 클래스를 상속받는 ColorPoint 클래스 만들기

ColorPoint.h

```
#ifndef COLORPOINT_H
#define COLORPOINT_H

#include "Point.h"

class ColorPoint : public Point {
    string color; // 점의 색 표현
public:
    void setColor(string color) {this->color = color; }
    void showColorPoint();

    friend ostream& operator<<(ostream& os, const ColorPoint& p);
    friend istream& operator>>(istream& is, ColorPoint& p);
};
#endif
```

ColorPoint.cpp

```
#include "ColorPoint.h"

void ColorPoint::showColorPoint() {
    cout << color << ":";
    showPoint(); // Point의 showPoint() 호출
}

ostream& operator<<(ostream& os, const ColorPoint& p)
{
    os<< "[Color: "<< p.color << " Coord.: " << (Point)p << "]"<< endl;
    return os;
}

istream& operator>>(istream& is, ColorPoint& p)
{
    cout << "ColorPoint 입력(색상, x좌표, y좌표 순서로): ";
    is>>p.color >>p.x >> p.y;
    return is;
}
```

예제 8-1 Point 클래스를 상속받는 ColorPoint 클래스 만들기

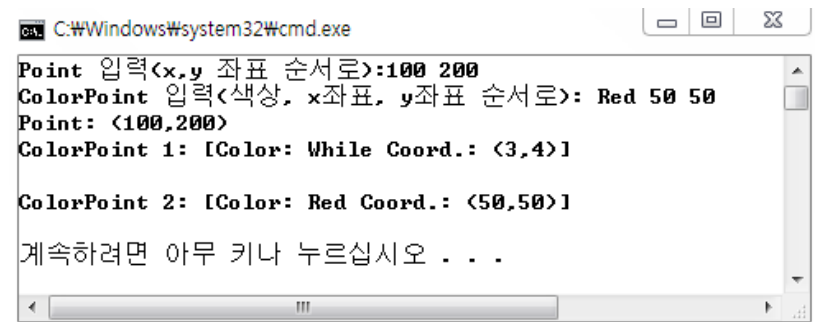
main.cpp

```
#include "ColorPoint.h"
#include "Point.h"

int main()
{
    Point p; // 기본 클래스의 객체 생성
    ColorPoint cp1, cp2; // 파생 클래스의 객체 생성

    cin >> p;
    cp1.set(3,4); // 기본 클래스의 멤버 호출
    cp1.setColor("While"); // 파생 클래스의 멤버 호출
    cin >> cp2;

    cout << "Point: " << p << endl;
    cout << "ColorPoint 1: " << cp1 << endl ;
    cout << "ColorPoint 2: " << cp2 << endl ;
}
```



```
C:\Windows\system32\cmd.exe
Point 입력<x,y 좌표 순서로>:100 200
ColorPoint 입력<색상, x좌표, y좌표 순서로>: Red 50 50
Point: <100,200>
ColorPoint 1: [Color: While Coord.: <3,4>]

ColorPoint 2: [Color: Red Coord.: <50,50>]

계속하려면 아무 키나 누르십시오 . . .
```

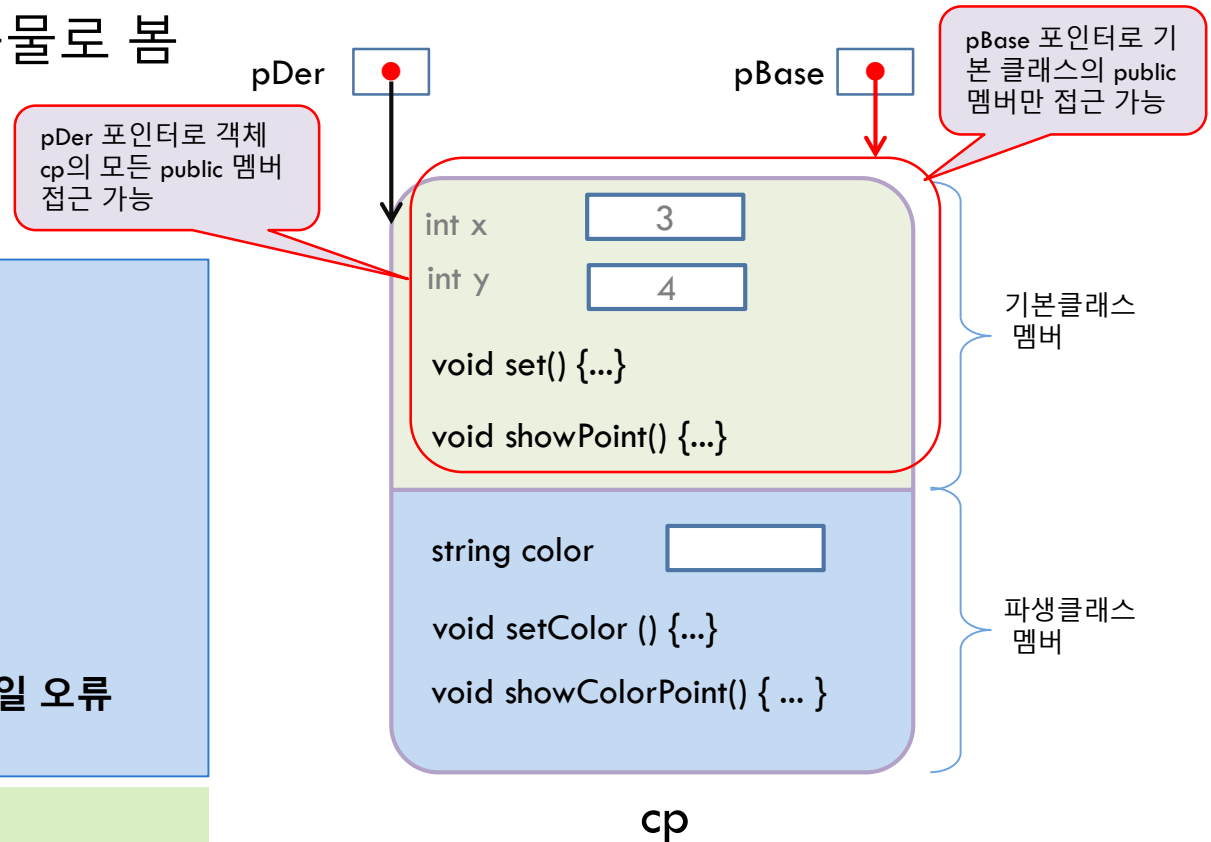
상속과 객체 포인터 – 업 캐스팅

- 업 캐스팅(up-casting)

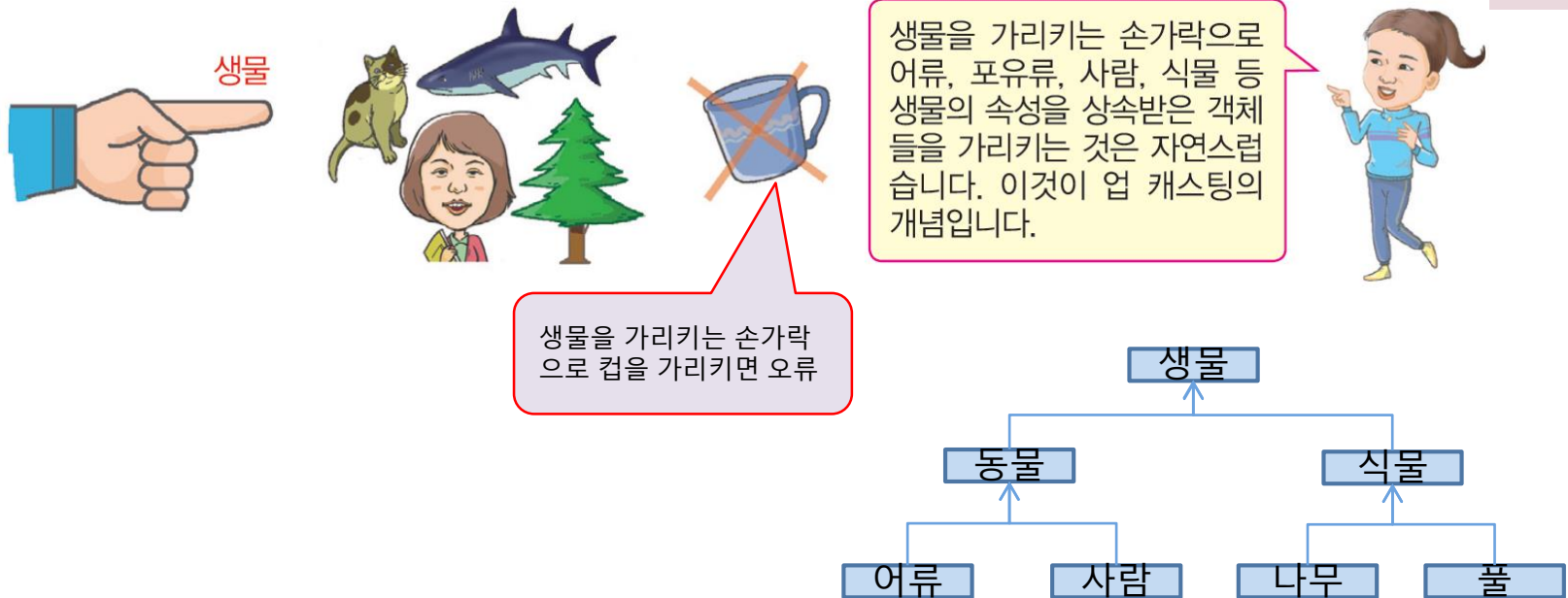
- ▶ 파생 클래스 포인터가 기본 클래스 포인터에 치환되는 것
- ▶ 예) 사람을 동물로 봄

```
int main() {  
    ColorPoint cp;  
    ColorPoint *pDer = &cp;  
    Point* pBase = pDer; // 업캐스팅  
  
    pDer->set(3,4);  
    pBase->showPoint();  
    pDer->setColor("Red");  
    pDer->showColorPoint();  
    pBase->showColorPoint(); // 컴파일 오류  
}
```

(3,4)
Red(3,4)



업 캐스팅

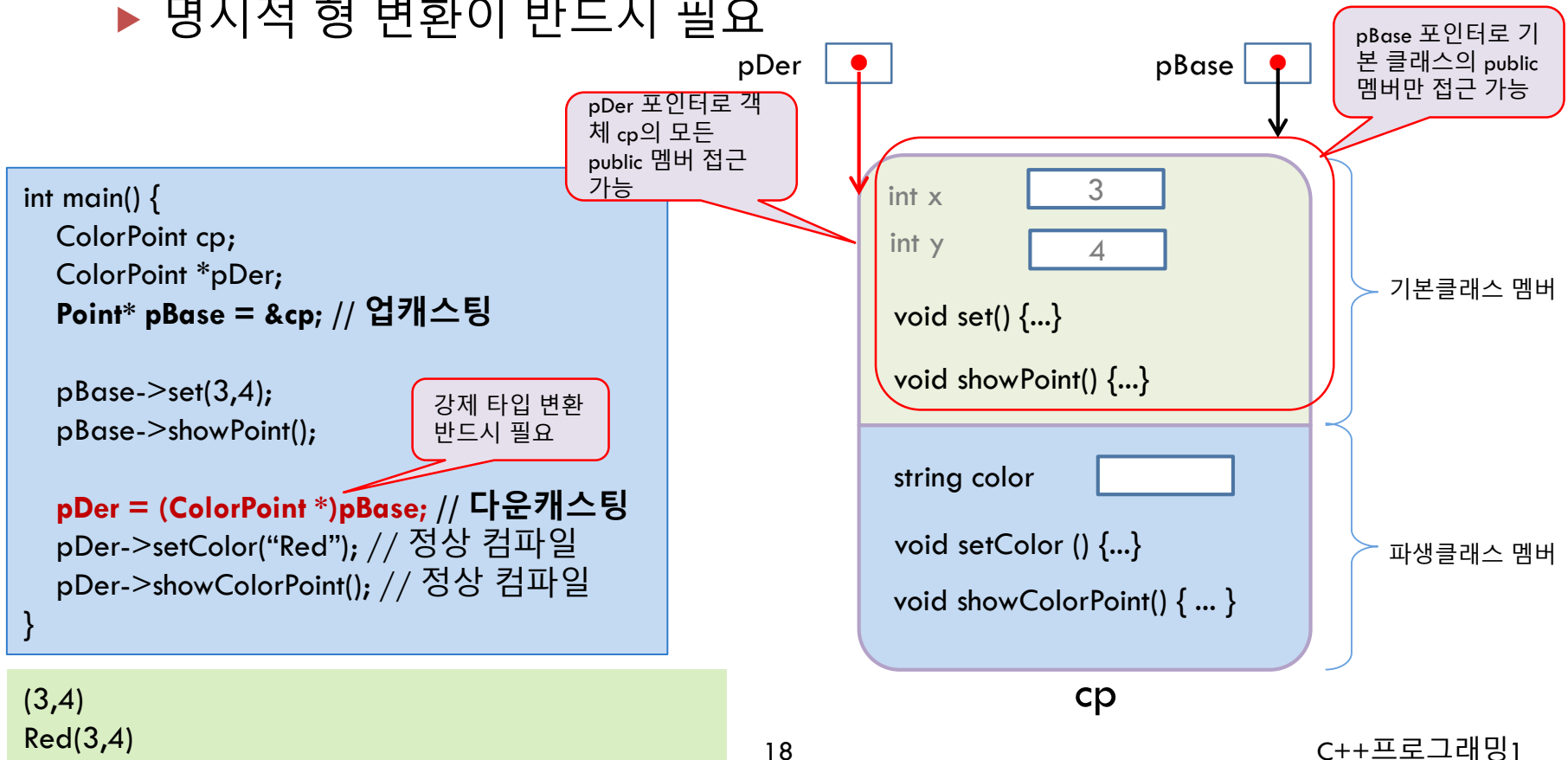


- 업 캐스팅 시 명시적 형 변환은 불필요
 - ▶ `Point* pBase = (Point*)pDer; // (Point*) 생략 가능`

상속과 객체 포인터 - 다운 캐스팅

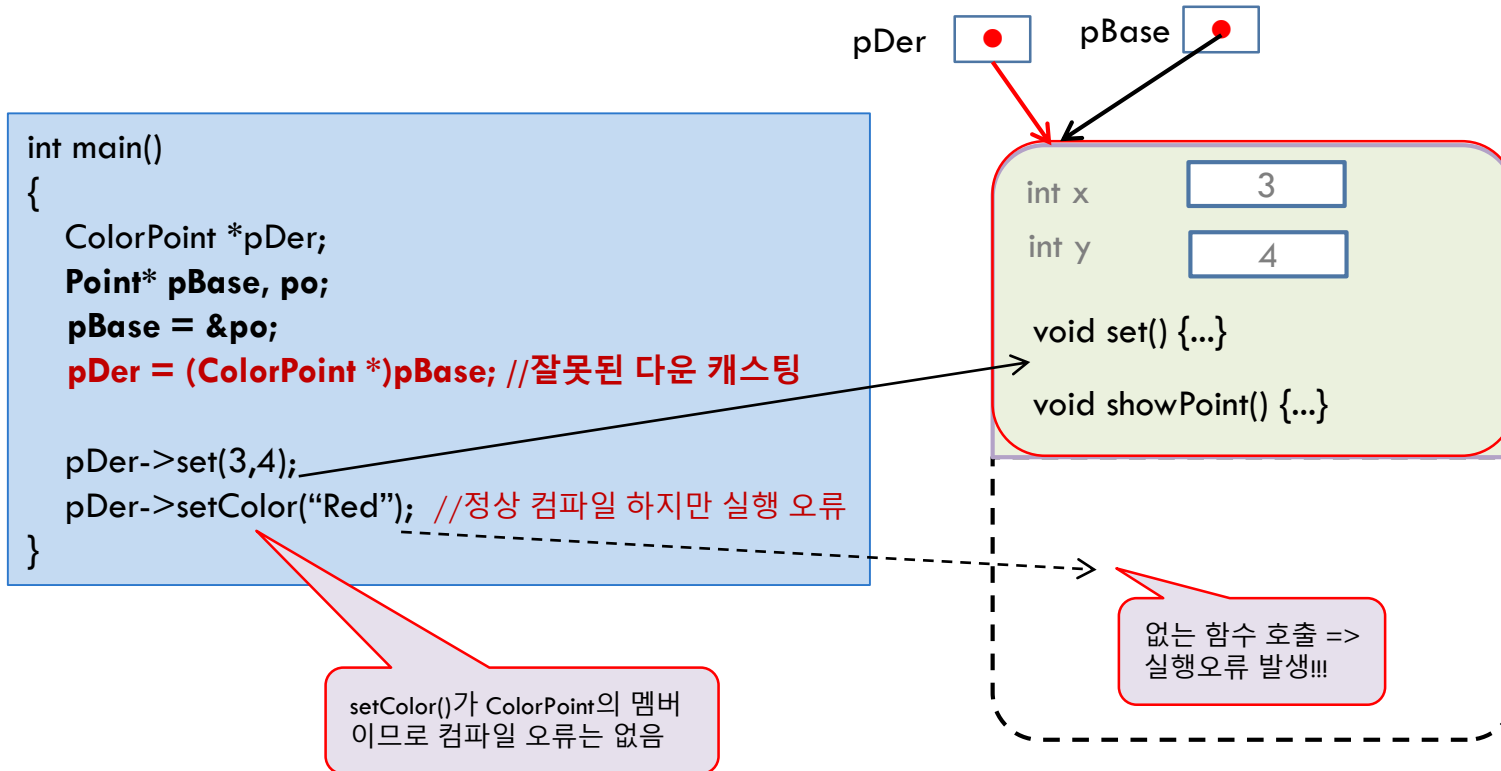
- 다운 캐스팅(down-casting)

- ▶ 기본 클래스의 포인터가 파생 클래스의 포인터에 치환되는 것
- ▶ 명시적 형 변환이 반드시 필요



상속과 객체 포인터 - 다운 캐스팅

- 다운 캐스팅(down-casting)시 주의점
 - ▶ 컴파일 오류는 없지만 실행 오류가 발생하는 경우



protected 접근 지정

- 접근 지정자

- ▶ private 멤버

- ▶ 선언된 클래스 내에서만 접근 가능
 - ▶ 파생 클래스에서도 기본 클래스의 private 멤버 직접 접근 불가

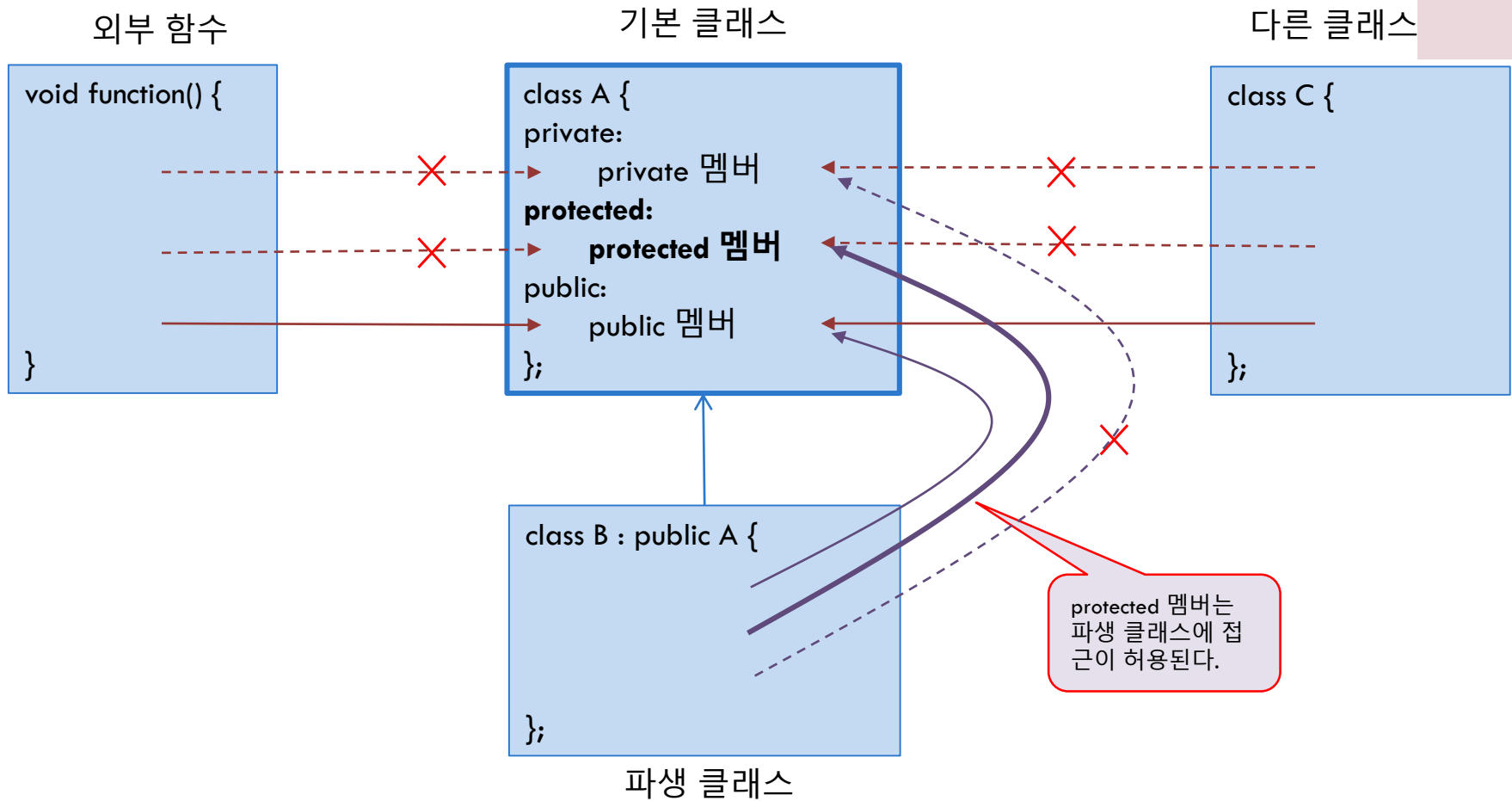
- ▶ public 멤버

- ▶ 선언된 클래스나 외부 어떤 클래스, 모든 외부 함수에 접근 허용
 - ▶ 파생 클래스에서 기본 클래스의 public 멤버 접근 가능

- ▶ protected 멤버

- ▶ 선언된 클래스에서 접근 가능
 - ▶ 파생 클래스에서만 접근 허용
 - ▶ 파생 클래스가 아닌 다른 클래스나 외부 함수에서는 protected 멤버를 접근할 수 없다.

멤버의 접근 지정에 따른 접근성



예제 8-2 protected 멤버에 대한 접근

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

class Point {
protected:
    int x, y; //한 점 (x,y) 좌표값
public:
    void set(int x, int y);
    void showPoint();
};

void Point::set(int x, int y) {
    this->x = x;
    this->y = y;
}

void Point::showPoint() {
    cout << "(" << x << "," << y << ")" << endl;
}

class ColorPoint : public Point {
    string color;
public:
    void setColor(string color);
    void showColorPoint();
    bool equals(ColorPoint p);
};

void ColorPoint::setColor(string color) {
    this->color = color;
}
```

```
void ColorPoint::showColorPoint() {
    cout << color << ":";
    showPoint(); // Point 클래스의 showPoint() 호출
}

bool ColorPoint::equals(ColorPoint p) {
    if(x == p.x && y == p.y && color == p.color) // ①
        return true;
    else
        return false;
}

int main() {
    Point p; // 기본 클래스의 객체 생성
    p.set(2,3);
    p.x = 5;
    p.y = 5;
    p.showPoint();

    ColorPoint cp; // 파생 클래스의 객체 생성
    cp.x = 10;
    cp.y = 10;
    cp.set(3,4);
    cp.setColor("Red");
    cp.showColorPoint();

    ColorPoint cp2;
    cp2.set(3,4);
    cp2.setColor("Red");
    cout << ((cp.equals(cp2))?"true":"false"); // ⑦
}
```

// ②
// ③ 오류
// ④ 오류

// ⑤ 오류
// ⑥ 오류

상속 관계의 생성자와 소멸자 실행

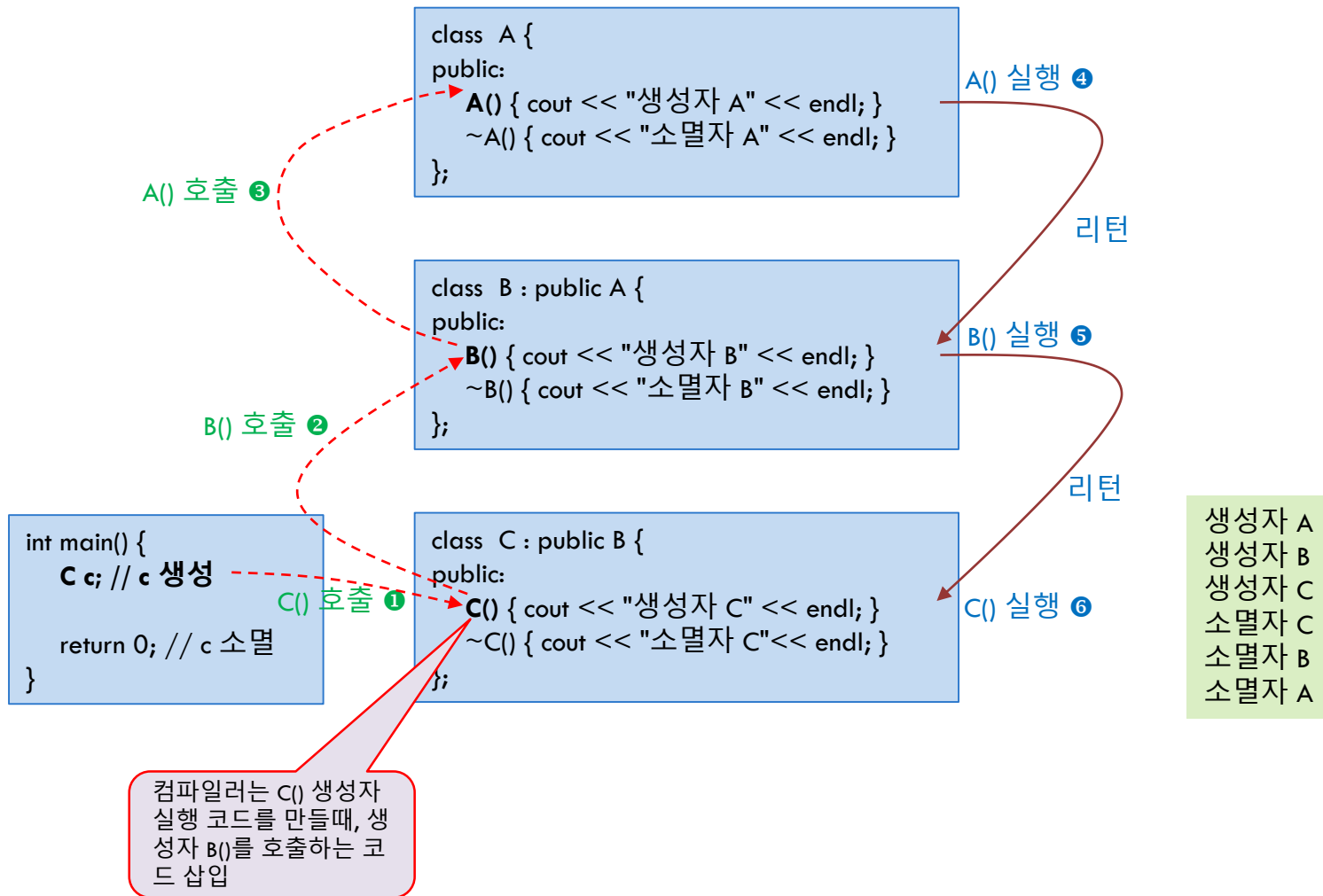
● 질문 1

- ▶ 파생 클래스의 객체가 생성될 때 파생 클래스의 생성자와 기본 클래스의 생성자가 모두 실행되는가? 아니면 파생 클래스의 생성자만 실행되는가?
 - ▶ 답 - 둘 다 실행된다. 부모 없는 자식은 없다.

● 질문 2

- ▶ 파생 클래스의 생성자와 기본 클래스의 생성자 중에서 어떤 생성자가 먼저 실행되는가?
 - ▶ 답 - 기본 클래스의 생성자가 먼저 실행된 후 파생 클래스의 생성자가 실행된다. 부모가 먼저 생성되어야 한다.

생성자 호출 관계 및 실행 순서



소멸자의 실행 순서

- 파생 클래스의 객체가 소멸될 때
 - ▶ 파생 클래스의 소멸자가 먼저 실행되고
 - ▶ 기본 클래스의 소멸자가 나중에 실행

파생 클래스에서 기본 클래스 생성자 호출

- 파생 클래스의 생성자와 기본 클래스의 생성자가 여러 개 있는 경우
 - ▶ 파생 클래스의 생성자가 실행될 때 함께 실행되는 기본 클래스의 생성자는 어떻게 결정되는가?
 - ▶ 개발자가 파생 클래스 생성자의 구현 시 함께 실행할 기본 클래스의 생성자를 지정해야 한다.
 - ▶ 명시적으로 지정되어 있지 않으면 컴파일러에 의해 묵시적으로 기본 클래스의 기본 생성자가 실행되도록 컴파일함

컴파일러에 의해 묵시적으로 기본 클래스의 생성자를 선택하는 경우

파생 클래스의 생성자에서 기본 클래스의 기본 생성자 호출

컴파일러는 묵시적으로 기본 클래스의 기본 생성자를 호출하도록 컴파일함

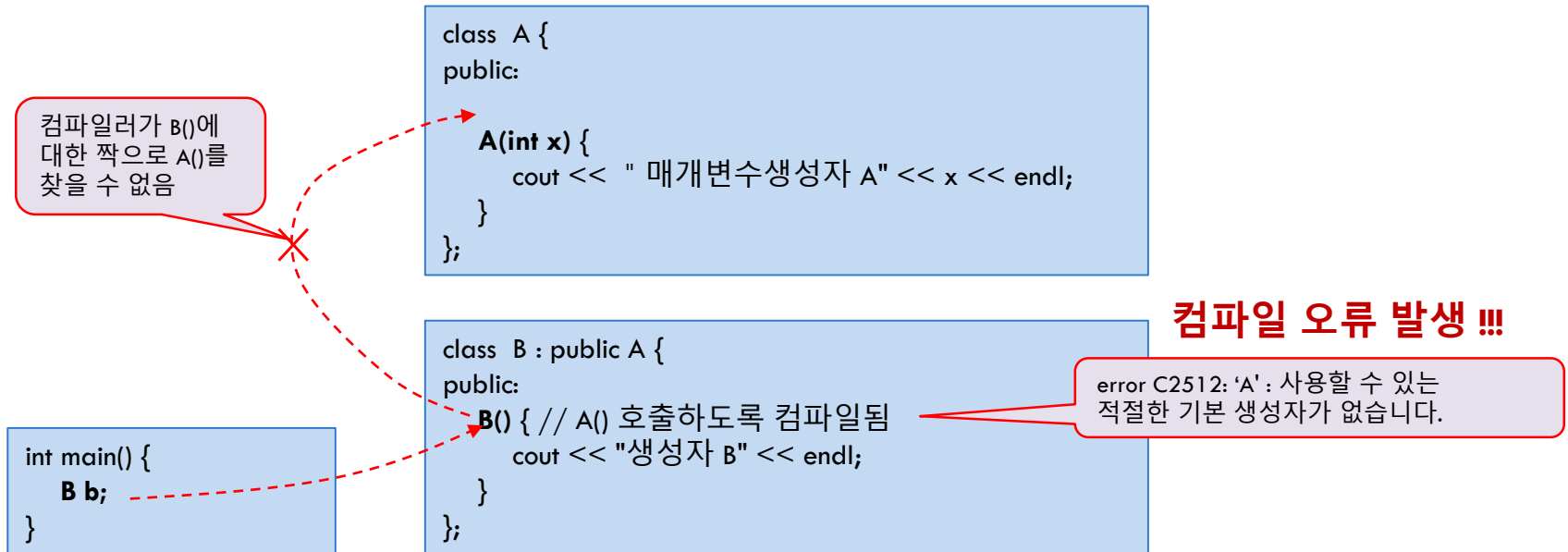
```
class A {  
public:  
    ▶A() { cout << "생성자 A" << endl; }  
    A(int x) {  
        cout << " 매개변수생성자 A" << x << endl;  
    }  
};
```

```
int main() {  
    B b;  
}
```

```
class B : public A {  
public:  
    ▶B() { // A() 호출하도록 컴파일됨  
        cout << "생성자 B" << endl;  
    }  
};
```

생성자 A
생성자 B

기본 클래스에 기본 생성자가 없는 경우



매개 변수를 가진 파생 클래스의 생성자는 묵시적으로 기본 클래스의 기본 생성자 선택

파생 클래스의 매개 변수를 가진 생성자가 기본 클래스의 기본 생성자 호출

컴파일러는 묵시적으로 기본 클래스의 기본 생성자를 호출하도록 컴파일함

```
class A {  
public:  
    A() { cout << "생성자 A" << endl; }  
    A(int x) {  
        cout << " 매개변수생성자 A" << x << endl;  
    }  
};
```

```
class B : public A {  
public:  
    B() { // A() 호출하도록 컴파일됨  
        cout << "생성자 B" << endl;  
    }  
    B(int x) { // A() 호출하도록 컴파일됨  
        cout << "매개변수생성자 B" << x << endl;  
    }  
};
```

```
int main() {  
    B b(5);  
}
```

생성자 A
매개변수생성자 B5

파생 클래스의 생성자에서 명시적으로 기본 클래스의 생성자 선택

파생 클래스의 생성자가 명시적으로 기본 클래스의 생성자를 선택 호출함

```
class A {  
public:  
    A() { cout << "생성자 A" << endl; }  
    A(int x) {  
        cout << " 매개변수생성자 A" << x << endl;  
    }  
};
```

A(8) 호출

```
class B : public A {  
public:  
    B() { // A() 호출하도록 컴파일됨  
        cout << "생성자 B" << endl;  
    }  
    B(int x) : A(x+3) {  
        cout << "매개변수생성자 B" << x << endl;  
    }  
};
```

B(5) 호출

```
int main() {  
    B b(5);  
}
```

매개변수생성자 A8
매개변수생성자 B5

컴파일러의 기본 생성자 호출 코드 삽입

```
class B {  
    B() : A() {  
        cout << "생성자 B" << endl;  
    }  
  
    B(int x) : A() {  
        cout << "매개변수생성자 B" << x << endl;  
    }  
};
```

컴파일러가 묵시적으로
삽입한 코드

컴파일러가 묵시적으로
삽입한 코드

개발자가 파생 클래스 생성자의 구현 시 명시적으로 기본 클래스의 생성자를 지정하지 않을 경우 컴파일의 수행 결과

예제 8-3 TV, WideTV, SmartTV 생성자 매개 변수 전달

파생 클래스의 생성자를 통해, 기본 클래스의 생성자에게까지 매개 변수의 값을 전달

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

```
class TV {
    int size; // 스크린 크기
public:
    TV() { size = 20; }
    TV(int size) { this->size = size; }
    int getSize() { return size; }
```

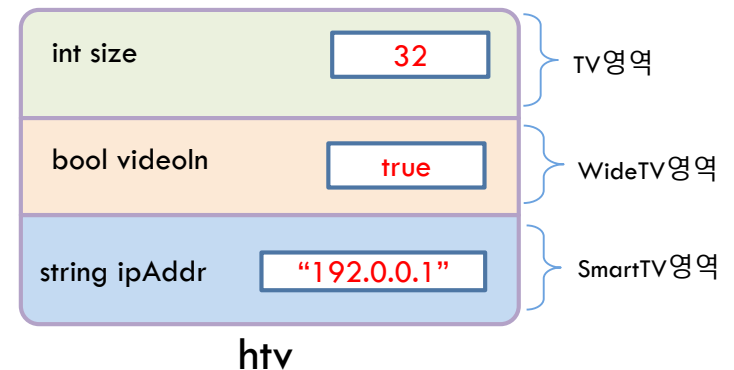
```
class WideTV : public TV { // TV를 상속받는 WideTV
    bool videoln;
public:
    WideTV(int size, bool videoln) : TV(size) {
        this->videoln = videoln;
    }
    bool getVideoln() { return videoln; }
```

```
class SmartTV : public WideTV { // WideTV를 상속받는 SmartTV
    string ipAddr; // 인터넷 주소
public:
    SmartTV(string ipAddr, int size) : WideTV(size, true) {
        this->ipAddr = ipAddr;
    }
    string getIpAddr() { return ipAddr; }
```

```
int main() {
    // 32 인치 크기에 "192.0.0.1"의 인터넷 주소를 가지는 스마트 TV 객체 생성
    SmartTV htv("192.0.0.1", 32);
    cout << "size=" << htv.getSize() << endl;
    cout << "videoln=" << boolalpha << htv.getVideoln() << endl;
    cout << "IP=" << htv.getIpAddr() << endl;
}
```

boolalpha는 불린 값을 true, false로 출력되게 하는 조작자

size=32
videoln=true
IP=192.0.0.1

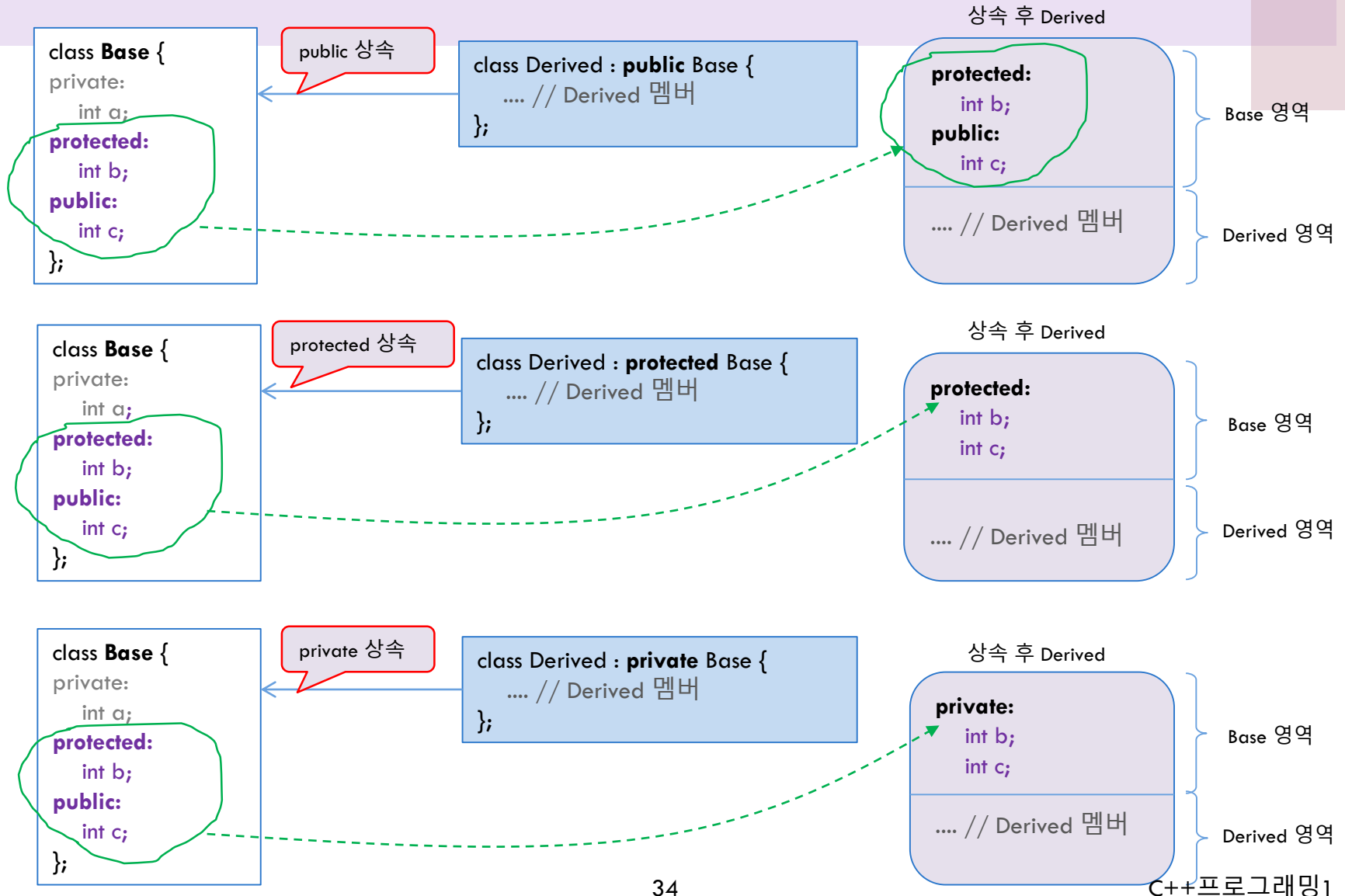


상속 지정

- 상속 지정

- ▶ 상속 선언 시 `public`, `private`, `protected`의 3가지 중 하나 지정
- ▶ 기본 클래스의 멤버의 접근 속성을 어떻게 계승할지 지정
 - ▶ `public` – 기본 클래스의 `protected`, `public` 멤버 속성을 그대로 계승
 - ▶ `private` – 기본 클래스의 `protected`, `public` 멤버를 `private`으로 계승
 - ▶ `protected` – 기본 클래스의 `protected`, `public` 멤버를 `protected`로 계승
- ▶ `public` 상속은 기본 클래스에 선언된 멤버들의 접근 속성을 그대로 계승
- ▶ `private`, `protected` 상속은 기본 클래스에 선언된 멤버들의 접근 지정을 변경

상속 시 접근 지정에 따른 멤버의 접근 지정 속성 변화



예제 8-4 private 상속 사례

다음에서 컴파일 오류가 발생하는 부분을 찾아라.

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

class Base {
    int a;
protected:
    void setA(int a) { this->a = a; }
public:
    void showA() { cout << a; }
};

class Derived : private Base {
    int b;
protected:
    void setB(int b) { this->b = b; }
public:
    void showB() { cout << b; }
};
```

```
int main() {
    Derived x;
    x.a = 5;           // ①
    x.setA(10);        // ②
    x.showA();         // ③
    x.b = 10;          // ④
    x.setB(10);        // ⑤
    x.showB();         // ⑥
}
```

컴파일 오류

①, ②, ③, ④, ⑤

예제 8-5 protected 상속 사례

다음에서 컴파일 오류가 발생하는 부분을 찾아라.

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

class Base {
    int a;
protected:
    void setA(int a) { this->a = a; }
public:
    void showA() { cout << a; }
};

class Derived : protected Base {
    int b;
protected:
    void setB(int b) { this->b = b; }
public:
    void showB() { cout << b; }
};
```

```
int main() {
    Derived x;
    x.a = 5;           // ①
    x.setA(10);        // ②
    x.showA();          // ③
    x.b = 10;          // ④
    x.setB(10);         // ⑤
    x.showB();          // ⑥
}
```

컴파일 오류

①, ②, ③, ④, ⑤

예제 8-6 상속이 중첩될 때 접근 지정 사례

다음에서 컴파일 오류가 발생하는 부분을 찾아라.

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

class Base {
    int a;
protected:
    void setA(int a) { this->a = a; }
public:
    void showA() { cout << a; }
};

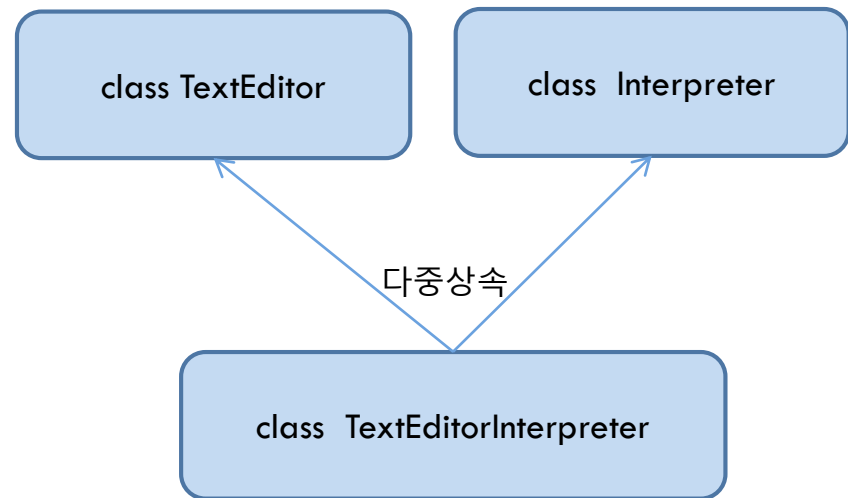
class Derived : private Base {
    int b;
protected:
    void setB(int b) { this->b = b; }
public:
    void showB() {
        setA(5);           // ①
        showA();           // ②
        cout << b;
    }
};
```

```
class GrandDerived : private Derived {
    int c;
protected:
    void setAB(int x) {
        setA(x);           // ③
        showA();           // ④
        setB(x);           // ⑤
    }
};
```

컴파일 오류

③, ④

기기의 컨버전스와 C++의 다중 상속



다중 상속 선언 및 멤버 호출

```
class MP3 {  
public:  
    void play();  
    void stop();  
};
```

```
class MobilePhone {  
public:  
    bool sendCall();  
    bool receiveCall();  
    bool sendSMS();  
    bool receiveSMS();  
};
```

상속받고자 하는 기본 클래스를 나열한다.

다중 상속 선언

```
class MusicPhone : public MP3, public MobilePhone { // 다중 상속 선언  
public:  
    void dial();  
};
```

다중 상속 활용

```
void MusicPhone::dial() {  
    play(); // mp3 음악을 연주시키고  
    sendCall(); // 전화를 건다.  
}
```

MP3::play() 호출

MobilePhone::sendCall() 호출

다중 상속 활용

```
int main() {  
    MusicPhone hanPhone;  
    hanPhone.play(); // MP3의 멤버 play() 호출  
    hanPhone.sendSMS(); // MobilePhone의 멤버 sendSMS() 호출  
}
```

예제 8-7 Adder와 Subtractor를 다중 상속 받는 Calculator 클래스 작성

Adder와 Subtractor를 다중 상속받는 Calculator를 작성하라.

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

class Adder {
protected:
    int add(int a, int b) { return a+b; }
};

class Subtractor {
protected:
    int minus(int a, int b) { return a-b; }
};
```

```
// 다중 상속
class Calculator : public Adder, public Subtractor {
public:
    int calc(char op, int a, int b);
};

int Calculator::calc(char op, int a, int b) {
    int res=0;
    switch(op) {
        case '+': res = add(a, b); break;
        case '-': res = minus(a, b); break;
    }
    return res;
}
```

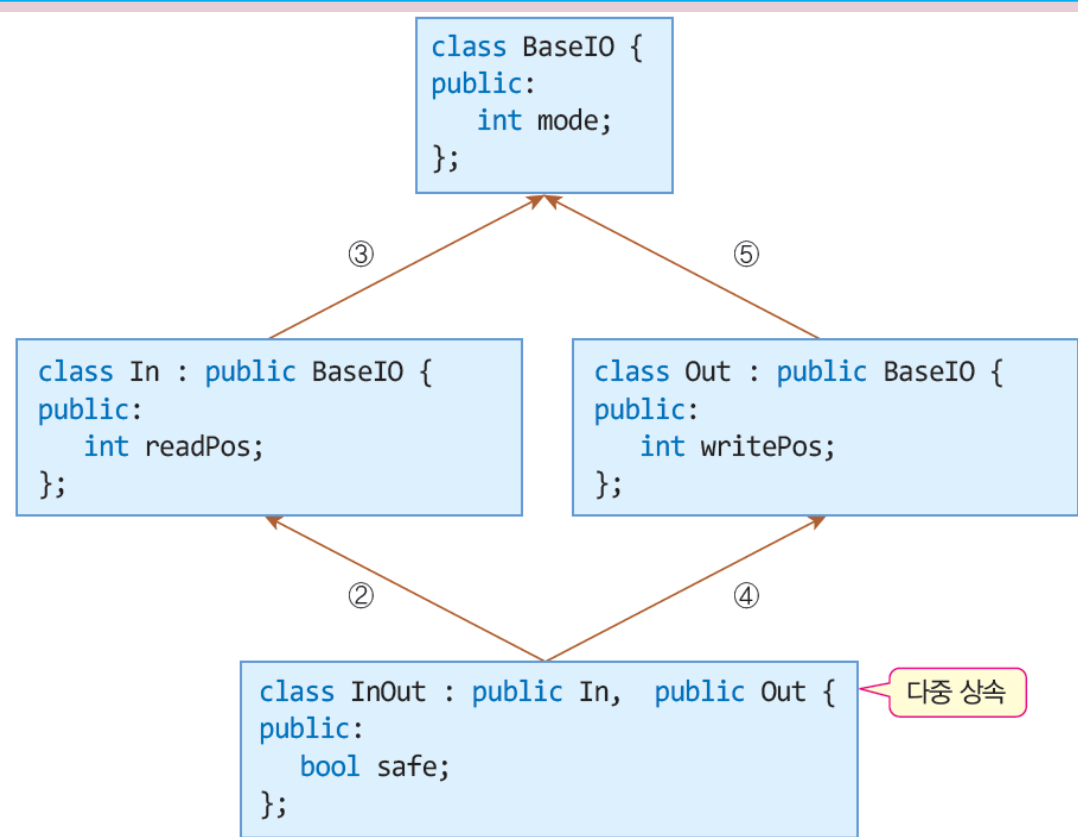
```
int main() {
    Calculator handCalculator;
    cout << "2 + 4 = "
        << handCalculator.calc('+', 2, 4) << endl;
    cout << "100 - 8 = "
        << handCalculator.calc('-', 100, 8) << endl;
}
```

2 + 4 = 6
100 - 8 = 92

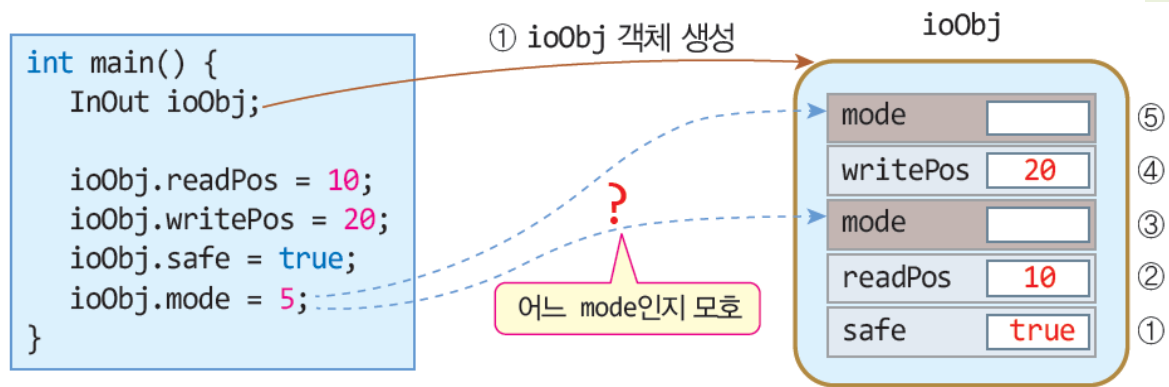
다중 상속의 문제점

- 기본 클래스 멤버의 중복 상속

- Base의 멤버가 이중으로 객체에 삽입되는 문제점.
- 동일한 x를 접근하는 프로그램이 서로 다른 x에 접근하는 결과를 낳게 되어 잘못된 실행 오류가 발생된다.



(a) 클래스 상속 관계



(b) ioObj 객체 생성 과정 및 객체 내부

가상 상속

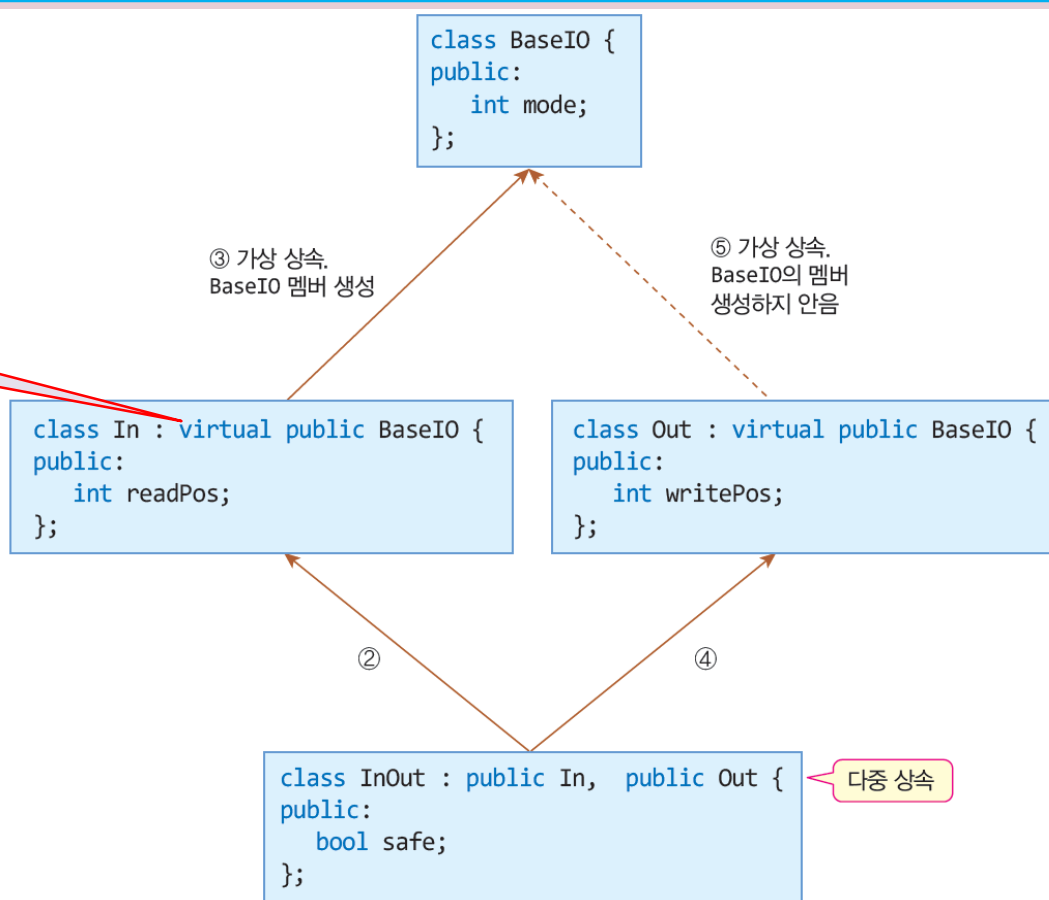
- 다중 상속으로 인한 기본 클래스 멤버의 중복 상속 해결
- 가상 상속
 - ▶ 파생 클래스의 선언문에서 기본 클래스 앞에 **virtual**로 선언
 - ▶ 파생 클래스의 객체가 생성될 때 기본 클래스의 멤버는 오직 한 번만 생성
 - ▶ 기본 클래스의 멤버가 중복하여 생성되는 것을 방지

```
class In : virtual public BaseO { // In 클래스는 BaseO 클래스를 가상 상속함
...
};

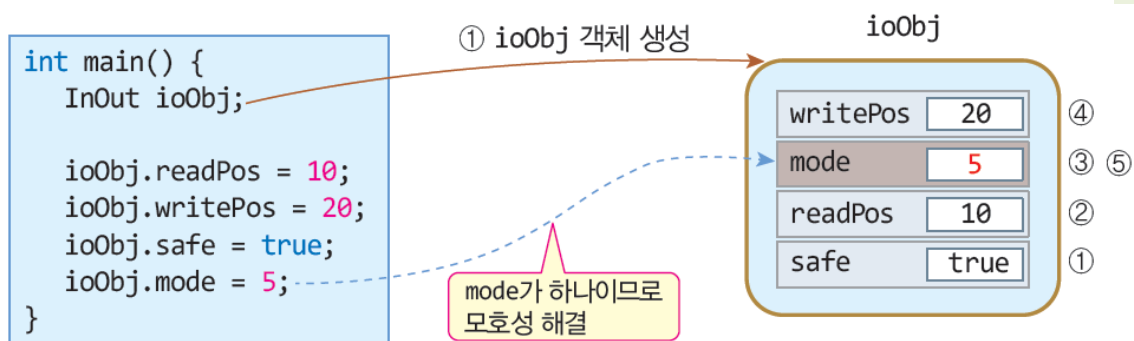
class Out : virtual public BaseO { // Out 클래스는 BaseO 클래스를 가상 상속함
...
};
```

가상 상속으로 다중 상속의 모호성 해결

가상 상속



(a) 기본 클래스를 가상 상속 받는 클래스 상속 관계



(b) 가상 기본 클래스를 가진 경우, ioObj 객체 생성 과정 및 객체 내부

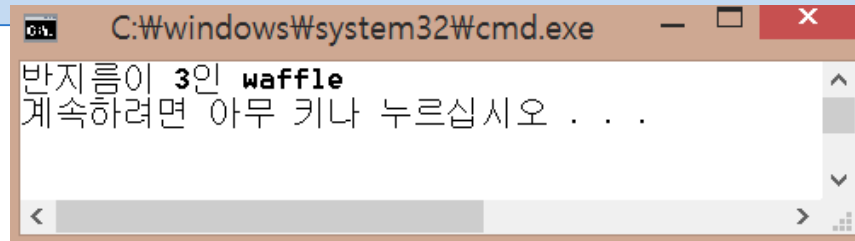
연습문제 1 – NamedCircle 클래스

- 원을 추상화한 Circle 클래스를 다음과 같이 정의하였다.

```
class Circle {  
    int radius;  
public:  
    Circle(int radius=0) { this->radius = radius; }  
    int getRadius() { return radius; }  
    void setRadius(int radius) { this->radius = radius; }  
    double getArea() { return 3.14*radius*radius; };  
};
```

- Circle을 상속받은 NamedCircle 클래스를 작성하고 전체 프로그램을 완성해 보자.

```
NamedCircle waffle(3, "waffle"); // 반지름이 3이고 이름이 waffle인 원  
waffle.show();
```

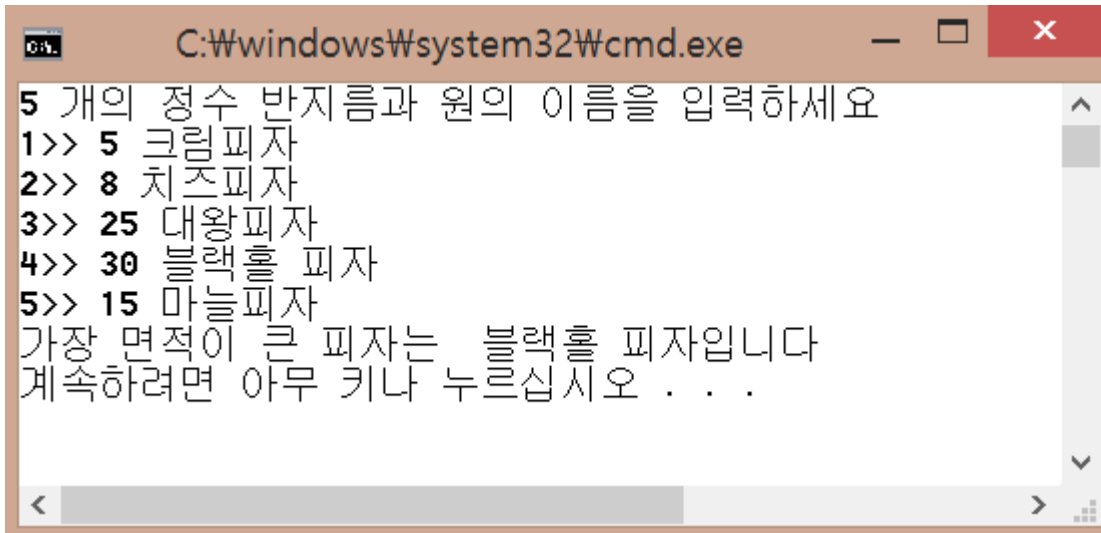


A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the output of the program: "반지름이 3인 waffle" and "계속하려면 아무 키나 누르십시오 . . .". The text is in Korean, indicating that the program is running in a Korean environment.

연습문제 2 - NamedCircle 클래스

- 다음과 같이 배열을 선언하여 그림의 실행 결과가 나오도록 NameCircle 클래스와 main()함수를 작성하시오.

```
NamedCircle c[5];
```



```
C:\Windows\system32\cmd.exe
5 개의 정수 반지름과 원의 이름을 입력하세요
1>> 5 크림피자
2>> 8 치즈피자
3>> 25 대왕피자
4>> 30 블랙홀 피자
5>> 15 마늘피자
가장 면적이 큰 피자는 블랙홀 피자입니다
계속하려면 아무 키나 누르십시오 . . .
```

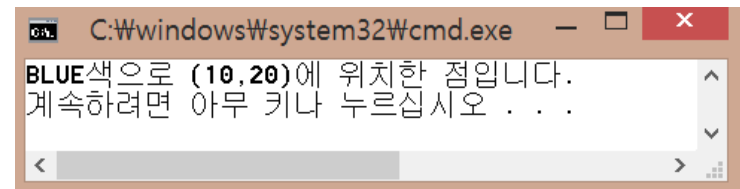
연습문제 3 – ColorPoint 클래스

- 2차원 상의 한 점을 표현하는 Point 클래스가 있다.

```
class Point {  
    int x, y;  
public:  
    Point(int x, int y) {  
        this->x = x; this->y = y;  
    }  
    int getX() { return x; }  
    int getY() { return y; }  
protected:  
    void move(int x, int y) { this->x = x; this->y = y; }  
};
```

- main()함수가 실행되도록 Point 클래스를 상속받은 ColorPoint클래스를 작성하여 프로그램을 완성하시오.

```
int main() )  
{  
    ColorPoint cp(5, 5, "RED");  
    cp.setPoint(10, 20); cp.setColor("BLUE"); cp.show();  
}
```

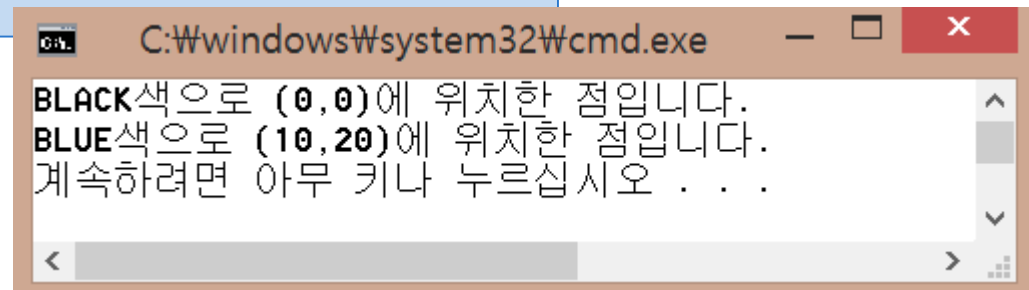


연습문제 4 - ColorPoint 클래스

- 다음의 main() 함수가 실행되도록 Point 클래스를 상속받는 ColorPoint 클래스를 작성하고 전체 프로그램을 완성하십시오.

```
int main()
{
    ColorPoint zeroPoint; // BLACK에 (0, 0) 위치의 점
    zeroPoint.show(); // zeroPoint를 출력한다.

    ColorPoint cp(5, 5);
    cp.setPoint(10, 20);
    cp.setColor("BLUE");
    cp.show(); // cp를 출력한다.
}
```



C:\Windows\system32\cmd.exe

```
BLACK색으로 (0,0)에 위치한 점입니다.
BLUE색으로 (10,20)에 위치한 점입니다.
계속하려면 아무 키나 누르십시오 . . .
```

연습문제 5 – MyQueue 클래스

- BaseArray 클래스는 다음과 같다. BaseArray를 상속받아 MyQueue 클래스와 MyStack 클래스를 구현해 보자.

```
class BaseArray {
private:
    int capacity; // 동적 할당된 메모리 용량
    int *mem;
protected:
    BaseArray(int capacity=100) {
        this->capacity = capacity; mem = new int [capacity];
    }
    ~BaseArray() { delete [] mem; }
    void put(int index, int val) { mem[index] = val; }
    int get(int index) { return mem[index]; }
    int getCapacity() { return capacity; }
};
```

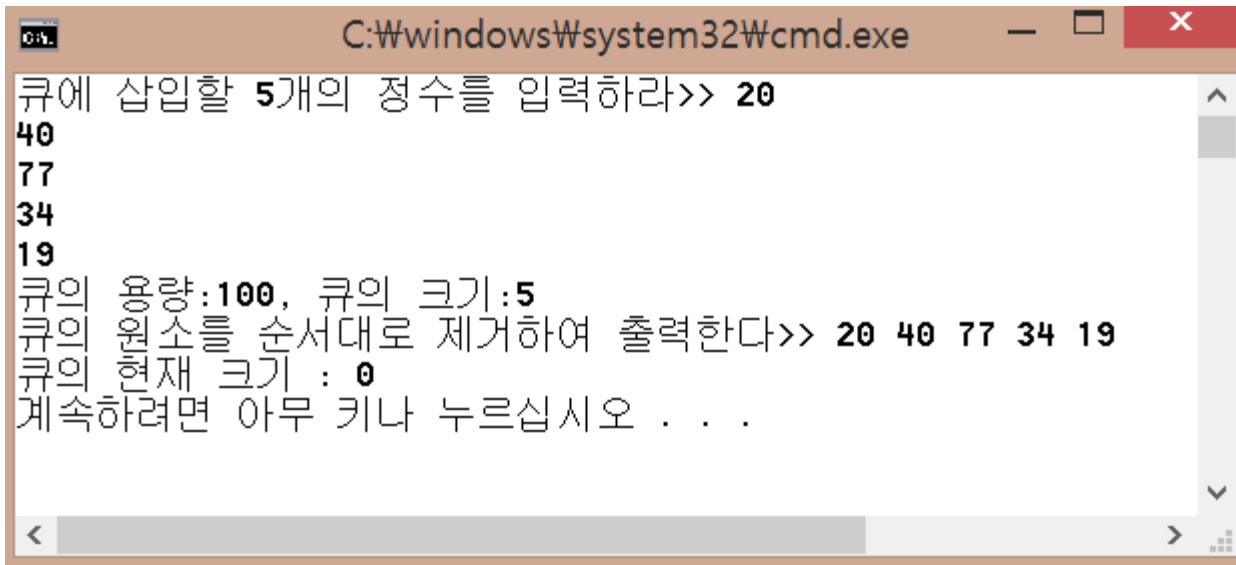

연습문제 5 – MyQueue 클래스

- BaseArray 클래스를 상속받아 큐처럼 작동하는 MyQueue 클래스를 작성하시오. MyQueue를 활용하는 main()함수는 다음과 같다.

```
int main() {
    MyQueue mQ(100);
    int n;
    cout << "큐에 삽입할 5개의 정수를 입력하라>> ";
    for(int i=0; i<5; i++) {
        cin >> n;
        mQ.enqueue(n); // 큐에 삽입
    }
    cout << "큐의 용량:" << mQ.capacity() << ", 큐의 크기:" << mQ.length() << endl;
    cout << "큐의 원소를 순서대로 제거하여 출력한다>> ";
    while(mQ.length() != 0) {
        cout << mQ.dequeue() << ' '; // 큐에서 제거하여 출력
    }
    cout << endl << "큐의 현재 크기 : " << mQ.length() << endl;
}
```

연습문제 5 – MyQueue 클래스

- 실행결과



```
C:\Windows\System32\cmd.exe
큐에 삽입할 5개의 정수를 입력하라>> 20
40
77
34
19
큐의 용량:100, 큐의 크기:5
큐의 원소를 순서대로 제거하여 출력한다>> 20 40 77 34 19
큐의 현재 크기 : 0
계속하려면 아무 키나 누르십시오 . . .
```

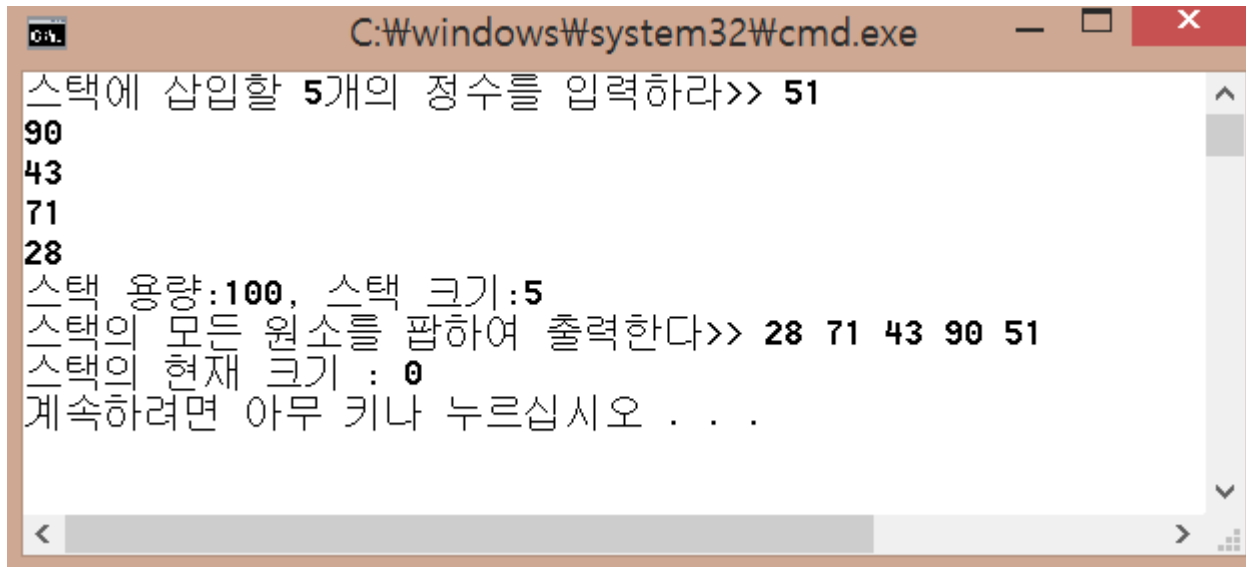
연습문제 6 - MyStack 클래스

- BaseArray 클래스를 상속받아 스택으로 작동하는 MyStack 클래스를 작성하라.

```
int main() {  
    MyStack mStack(100);  
    int n;  
    cout << "스택에 삽입할 5개의 정수를 입력하라>> ";  
    for(int i=0; i<5; i++) {  
        cin >> n;  
        mStack.push(n); // 스택에 푸시  
    }  
    cout << "스택 용량:" << mStack.capacity() << ", 스택 크기:"  
<< mStack.length() << endl;  
    cout << "스택의 모든 원소를 팝하여 출력한다>> ";  
    while(mStack.length() != 0) {  
        cout << mStack.pop() << ' '; // 스택에서 팝  
    }  
    cout << endl << "스택의 현재 크기 : " << mStack.length() << endl;  
}
```

연습문제 6 - MyStack 클래스

- 실행 결과



```
C:\Windows\system32\cmd.exe
스택에 삽입할 5개의 정수를 입력하라>> 51
90
43
71
28
스택 용량:100, 스택 크기:5
스택의 모든 원소를 팝하여 출력한다>> 28 71 43 90 51
스택의 현재 크기 : 0
계속하려면 아무 키나 누르십시오 . . .
```