C++ 프로그래밍

□ 상속

○ 상속의 개념

* 클래스 사이의 상속 : 객체 생성 시, 자신의 멤버 뿐 아니라 부모 클래스의 멤버를 포함한다.
* 기본 클래스 : 상속해주는 클래스, 부모 클래스
* 파생 클래스 : 상속받는 클래스, 자식 클래스
  - 기본 클래스의 속성과 기능을 물려받고 자신만의 속성과 기능을 추가
* Java, C#
* 간결한 클래스 작성
  - 코드 중복 제거와 수정 용이
* 클래스 간의 계층적 분류 및 관리의 용이함
* 클래스 재사용과 확장을 통한 소프트웨어 생산성 향상
* C++은 다중 상속을 허용

○ 상속 선언

* ' : ' 으로 상속을 표시

기본 형식

```
class Student : public Person {
        // Person을상속받는Student 선언
};
class StudentWorker : public Student  {
        // Student를상속받는StudentWorker선언
};
```

상속 - Point / ColorPoint

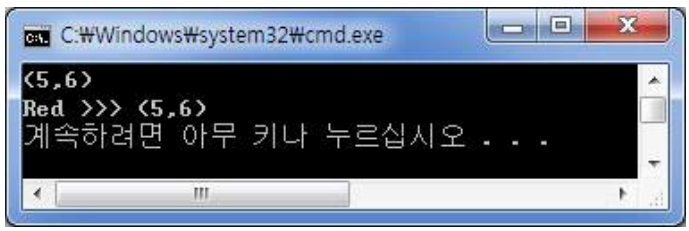
```
#include <iostream>
#include <string>

using namespace std;

#ifndef POINT_H
#define POINT_H

class Point{
private:
        int x, y;

public:
        void setPoint(int x, int y){
                this->x = x;
                this->y = y;
        }
        void showPoint(){
                cout << "(" << this->x << "," << this->y << ")" << endl;
        }
};
#endif
```




```
#include "Point.h"

#ifndef COLORPOINT_H
#define COLORPOINT_H

class ColorPoint : public Point{
protected:
        string color;

public:
        void setColor(string color){
                this->color = color;
        }
        void showColor(){
                cout << this->color << " >>> ";
                showPoint();
        }
};
#endif
```

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

int main(){
        ColorPoint cp;

        cp.setPoint(5, 6);
        cp.showPoint();

        cp.setColor("Red");
        cp.showColor();

        return 0;
}
```

○ 상속과 객체 포인터

* 업 캐스팅(up-casting) // casting : 배역을 정함
  - 위쪽 배역으로 결정한다. // 자식 -> 부모
  - 파생 클래스 포인터가 기본 클래스 포인터에 치환되는 것
  - 업 캐스팅 시 명시적 형 변환은 불필요
    // 부모 클래스는 하나이므로 형 변환이 필요하지 않다.

```
ani_a = kim;  //ani_a = (Animal*)kim;
```

* 다운 캐스팅
  - 아래쪽 배역으로 결정 // 부모 -> 자식
  - 기본 클래스의 포인터가 파생 클래스의 포인터에 치환되는 것
  - 다운 캐스팅 시 명시적 형 변환이 반드시 필요
    // 다운 캐스팅 시 자식 클래스는 부모에서 어느 클래스로 가야할지 지정해야하기 때문

```
kim = (Human*)ani_a;
```


```
int main() {
  ColorPoint cp;
  ColorPoint *pDer;
  Point* pBase = &cp; // 업캐스팅

  pBase->set(3,4);
  pBase->showPoint();

  pDer = (ColorPoint *)pBase; // 다운캐스팅
  pDer->setColor("Red"); // 정상 컴파일
  pDer->showColorPoint(); // 정상 컴파일
}
```

(강제 타입 변환 반드시 필요)

○ 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 (자식 클래스에서 사용 가능)
  - 파생 클래스에서만 접근 허용, 다른 클래스나 외부 함수에서는 protected 멤버 접근 불가

○ 상속 - 생성자 및 소멸자

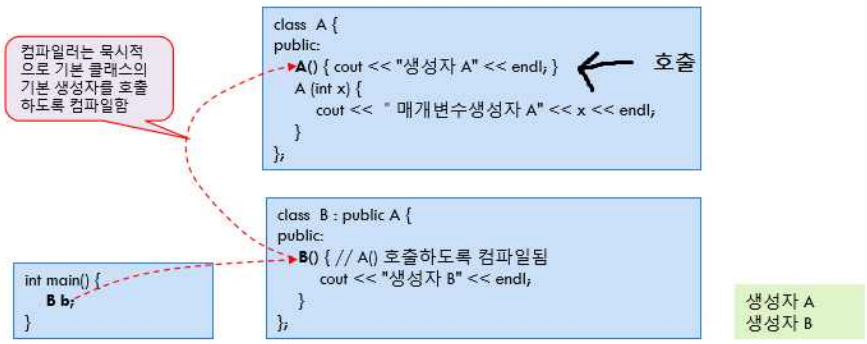
* 생성자 실행 순서 (자식 클래스의 객체 생성 시)
  - 부모 클래스의 생성자 실행 -> 자식 클래스의 생성자 실행
  - 부모가 먼저 생성 되어야 함

* 소멸자 실행 순서
  - 파생 클래스의 객체가 소멸될 때 : 파생 클래스의 소멸자 실행 -> 기본 클래스의 소멸자 실행

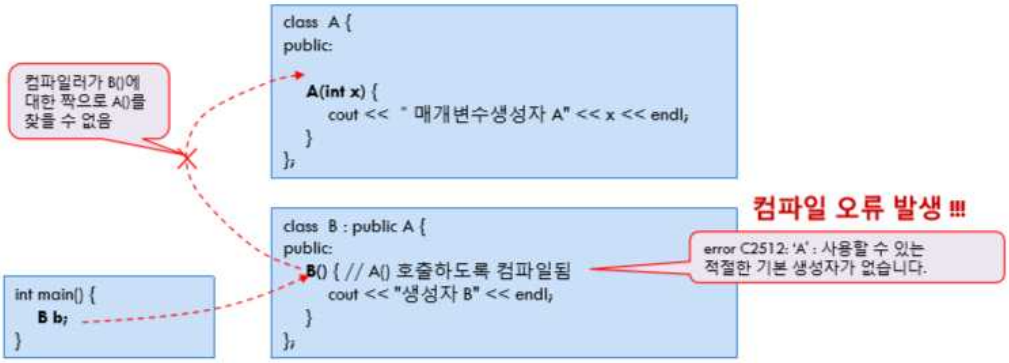
* 자식 클래스의 생성자 구현 시 함께 실행할 기본 생성자를 지정할 수 있다.

* 생성자를 지정하지 않았을 경우 컴파일러가 기본 생성자를 호출한다.

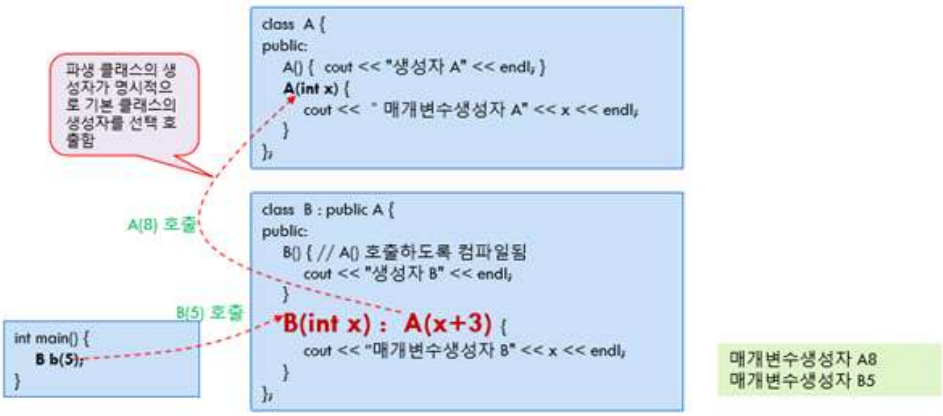
  파생 클래스의 생성자에서 기본 클래스의 기본 생성자 호출



* 기본 생성자를 지정하지 않으면 컴파일 오류가 난다.



* 자식 클래스에서 기본 클래스의 생성자를 선택



○ 상속 - 가상 상속

* 다중 상속으로 인한 기본 클래스 멤버의 중복 상속 해결
* 가상 상속
  - 자식 클래스의 선언문에 부모 클래스 앞에 virtual 선언
  - 자식 클래스의 객체가 생성될 때, 기본 클래스의 멤버는 오직 한 번만 생성

  // 기본 클래스의 멤버가 중복하여 생성되는 것을 방지

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

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

```
// 과제 연습문제 4
```