13강. 상속과 다형성

C++ 프로그래밍

jhhwang@kumoh.ac.kr

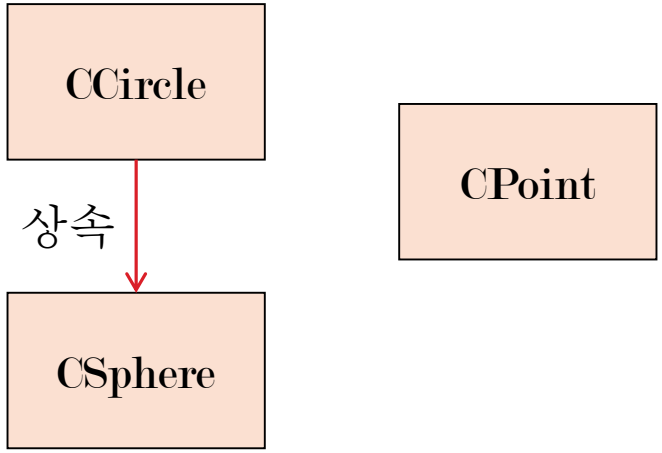
컴퓨터공학과 황준하

목차

- 문제 제기
  - 클래스 객체들 사이의 대입
  - 상속 관계인 객체와 포인터의 관계
- 가상 함수
- 추상 클래스와 순수 가상 함수
- 클래스 멤버 변수로서의 클래스 객체

클래스 객체들 사이의 대입

- 다음과 같은 클래스들과 객체들이 있을 때
  - CCircle Cir;
  - CSphere Sph;
  - CPoint Po;

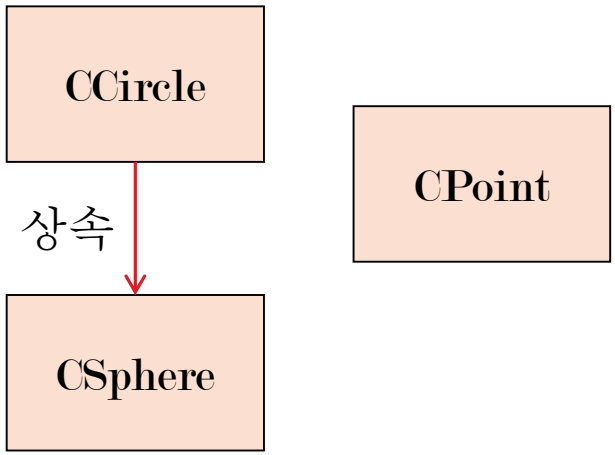


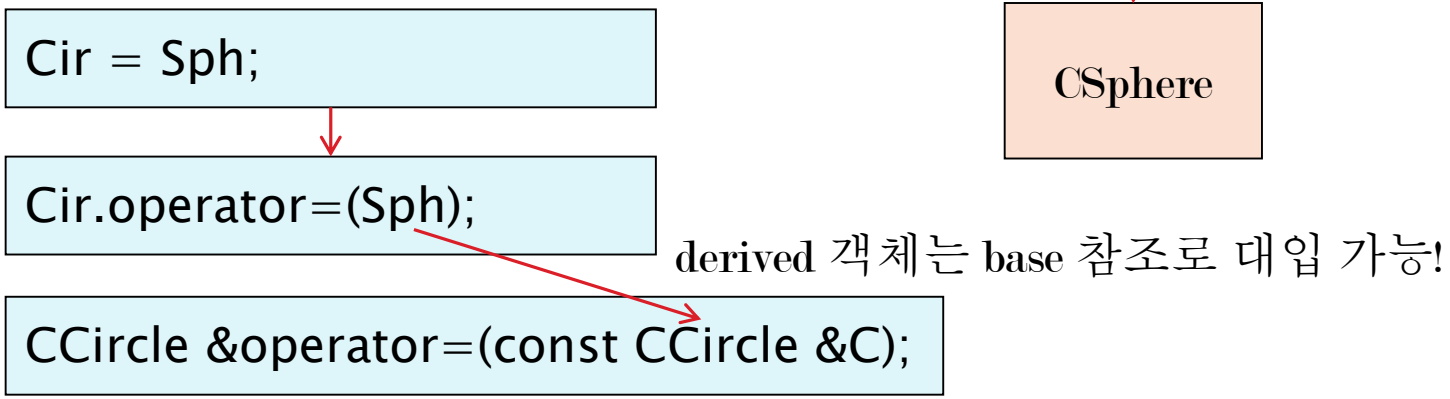
- 동일 클래스의 객체들 사이의 대입 가능(당연)
- 다음 중 가능한 대입은?

1. Cir = Sph; ○ ← derived 객체를 base 객체로 대입
2. Sph = Cir; ✕ ← base 객체를 derived 객체로 대입
3. Cir = Po; ✕ ← 무관한 클래스 객체 사이의 대입

클래스 객체들 사이의 대입

- derived 객체를 base 객체로 대입 가능한 이유(원리)
  - 명제 : derived 클래스 객체는 base 클래스의 참조 객체로 대입될 수 있다. 그 외의 관계에 있는 클래스 객체와 참조 객체 사이의 대입은 허용되지 않는다.
- Cir = Sph; 가 가능한 이유
  - 대입 연산자 수행



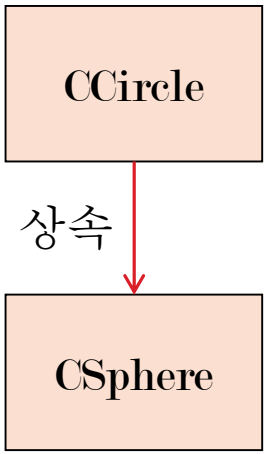


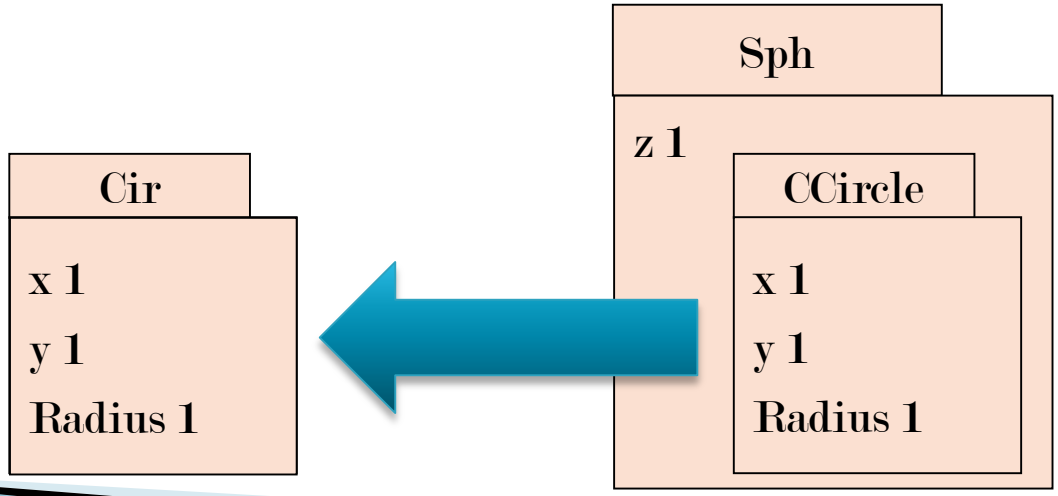
클래스 객체들 사이의 대입

- CCircle 클래스 객체와 CSphere 클래스 객체의 대입

```
void main(void)
{
        CSphere Sph(1, 1, 1, 1);
        CCircle Cir(2, 2, 100);
        Cir = Sph;
        cout << Cir.GetArea() << endl;
}
```

OK! derived를 base로 대입





클래스 객체들 사이의 대입

- 다음 프로그램의 문제점은?

```
void main(void)
{
	CSphere Sph(1, 1, 1, 1);
	CCircle &Cir = Sph;

	cout << "표면적 : " << Cir.GetArea() << endl;
	cout << Cir.GetVolume() << endl;
}
```

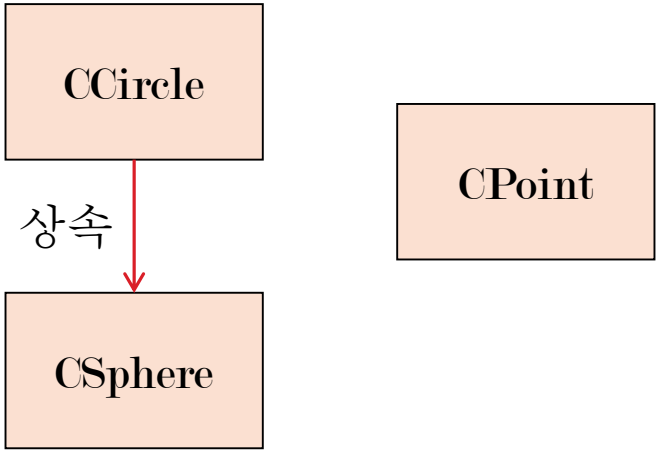
OK! derived를 base 참조로 대입

CCircle의 GetArea()?
CSphere의 GetArea()?

가능한가? No!
→ CCircle 클래스에는 GetVolume 없음

상속 관계인 객체와 포인터의 관계

- 다음과 같은 클래스들과 객체 및 포인터들이 있을 때
  - CCircle Cir; CCircle *pCir;
  - CSphere Sph; CSphere *pSph;
  - CPoint Po; CPoint *pPo;



- 동일 클래스의 객체를 포인터가 가리킬 수 있음 (당연)
- 다음 중 가능한 대입은?

1. pCir = &Sph; ✗ ← base 객체를 derived 포인터가 가리킴
2. pSph = &Cir; ✗ ← 클래스 객체를 무관한 클래스 포인터가 가리킴
3. pCir = &Po; ← derived 객체를 base 포인터가 가리킴

상속 관계인 객체와 포인터의 관계

- derived 객체를 base 객체로 대입 가능한 이유(원리)
  - 명제 : 널 포인터와 다른 타입 사이의 형변환 외에는 자동 형변환을 허용하지 않는다. 그러나 예외적으로 derived 클래스 타입의 포인터는 묵시적으로 base 클래스 타입의 포인터로 변환될 수 있다!
  - 따라서, pCir = &Sph; 가능

상속 관계인 객체와 포인터의 관계

- 다음 프로그램의 문제점은?

```
void main(void)
{
	CSphere Sph(1, 1, 1, 1);
	CCircle *pCir = &Sph;

	cout << "표면적 : " << pCir->GetArea() << endl;
	cout << pCir->GetVolume() << endl;
}
```

OK! derived를 base 포인터가 가리킴

CCircle의 GetArea()?
CSphere의 GetArea()?

가능한가? No!
→ CCircle 클래스에는 GetVolume 없음

한가지 방법 : 강제 형변환
((CSphere *) pCir)->GetVolume();

가상 함수

- 다음 프로그램에서 GetArea 함수의 결정

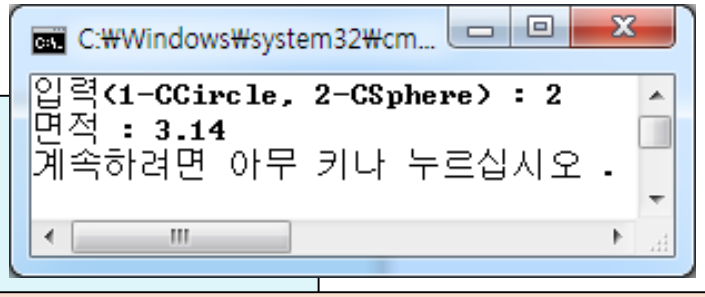
```
void main(void)
{
	CSphere Sph(1, 1, 1, 1);
	CCircle *pCir = &Sph;

	cout << "표면적 : " << pCir->GetArea() << endl;
}
```

  - 원칙 : 컴파일 시간에 결정 → pCir의 클래스인 CCircle의 GetArea 함수 실행
  - pCir 포인터가 CShpere 객체를 가리키고 있음 → CShpere의 GetArea 함수가 실행되기 원함 → 실행시간에 결정 → pCir의 클래스인 CCircle의 GetArea 함수를 가상함수로 만듦

가상 함수

- 포인터가 가리키는 실객체의 타입이 실행 시점에 결정되는 예

```
void main(void)
{
	int input;
	CCircle *pCir;

	cout << "입력(1-CCircle, 2-CSphere
	cin >> input;

	if (input == 1)
		pCir = new CCircle(1, 1, 1);
	else
		pCir = new CSphere(1, 1, 1, 1);

	cout << "면적 : " << pCir->GetArea() << endl;
}
```



디폴트 : pCir의 클래스인 CCircle의 GetArea 실행

pCir이 CCircle 객체를 가리키는지 CSphere 객체를 가리키는지 실행해 보지 않고 알 수 있을까?

가상 함수

- CCircle의 GetArea 함수를 가상 함수로 만들기

```
class CCircle {
protected :
	int x, y;
	double Radius;

public :
	CCircle(int a, int b, double r) : x(a), y(b), Radius(r) {  }
	virtual double GetArea() { return (PI * Radius * Radius); }
};
```

함수 프로토타입 앞에 virtual만 추가하면 끝!

```
class CSphere : public CCircle {
public :
	virtual double GetArea() { return (4 * PI * Radius * Radius); }
};
```

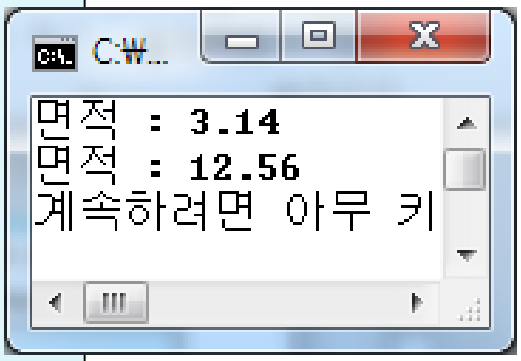
derived 클래스의 경우 virtual을 붙이지 않아도 자동으로 가상함수가 됨

가상 함수

- 가상 함수의 활용 예

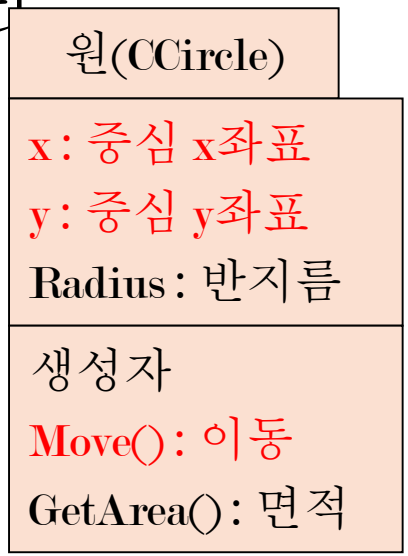
상황에 따라 실객체의 GetArea 함수가 실행됨

```
void PrintArea(CCircle &Cir)
{
	cout << "면적 : " << Cir.GetArea() << endl;
}

void main(void)
{
	CCircle Cir(1, 1, 1);
	CSphere Sph(1, 1, 1, 1);

	PrintArea(Cir);
	PrintArea(Sph);
}
```



추상 클래스와 순수 가상 함수

- 문제 : 원(CCircle) 클래스와 직사각형(CRect) 클래스를 만들려고 한다



  - 동일한 멤버 : x, y, Move 함수
  - 이름만 같은 멤버 함수 : GetArea

→ 공통 부분을 CShape이란 클래스로 만들고 상속받아 만들자!

추상 클래스와 순수 가상 함수

```
class CShape {
protected :
        int x, y;

public :
        CShape(int a, int b) : x(a), y(b) { }
        void Move(int a, int b) { x += a; y += b; }
```

```
class CCircle : public CShape {
private :
        double Radius;

public :
        CCircle(int a, int b
        double GetArea() {
};
```

```
class CRect : public CShape {
private :
        int Garo, Sero;

public :
        CRect(int a, int b, int g, int s) : CShape(a, b), Garo(g), Sero(s) { }
        double GetArea() { return (Garo * Sero); }
};
```

추상 클래스와 순수 가상 함수

- CShape에 대한 제약 사항
  - CShape 클래스의 객체는 존재할 수 없음
  - 단지 CCircle과 CRect 클래스의 base 클래스 역할 담당
  - 현재는 CShape 클래스 객체가 존재할 수 없다는 강제적 장치가 없음 → 추상 클래스로 만들면 됨
- 추상 클래스
  - 객체가 존재할 수 없는 클래스

포인터는 존재할 수 있음

```
CShape *pSpe = new CCircle(1, 1, 1)
```

  - 추상 클래스로 만드는 방법
    - 순수 가상 함수를 1개 이상 포함

```
virtual double GetArea() = 0;
```

    - 순수 가상 함수 : 몸체가 없음. derived 클래스에서 재정의해야 함
      - 재정의하지 않으면 해당 derived 클래스도 추상 클래스가 됨

추상 클래스와 순수 가상 함수

- CShape 클래스의 추상 클래스화

```
class CShape {
protected :
        int x, y;

public :
        CShape(int a, int b) : x(a), y(b) { }
        void Move(int a, int b) { x += a; y += b; }
        virtual double GetArea() = 0;
};
```

base 포인터를 통해 derived 클래스의 GetArea 함수 실행 가능

```
CShape *pSpe = new CCircle(1, 1, 1)
pSpe->GetArea();
```

클래스 멤버 변수로서의 클래스 객체

- 클래스는 int와 같은 하나의 타입
  - 어떤 클래스의 멤버 변수로 다른 클래스의 객체 선언 가능

CPoint 객체를 멤버 변수로 선언

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

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

멤버 객체 : 주로 has-a 관계
상속 : is-a 관계

```
class CCircle {
private :
        CPoint Center;
        double Radius;

public :
        CCircle(int a, int b, int r) : Center(a, b), Radius(r) { }
        void Print() {
                Center.Print();
                cout << " : " << Radius << endl;
        }
};
```

멤버 객체의 초기화
→ 반드시 멤버 초기화 구문 사용
→ 없을 경우 디폴트 생성자 사용