




고급프로그래밍

가상함수 추상클래스

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory
Kwangwoon University
(02-940-5625, mhjeong@kw.ac.kr)

Schedule

week	Topics		Homework	Quiz
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++ 	C++ 프로그래밍의 기본(3, 3/12), 클래스와 객체(4, 3/14)	1	1
3		휴강(3/17), 객체생성과 사용(5, 3/19)	2	
4		함수와 참조(6, 3/26), 복사 생성자와 함수중복(7, 3/28)	3	2, 3
5		static friend 연산자중복(8, 4/2), 연산자중복 상속(9, 4/4)	4	4
6		상속(10, 4/9), 가상함수 추상클래스(11, 4/11)	5	5
7		템플릿과 STL, 표준 입출력, 파일 입출력		
8	중간고사			
9	C++	예외처리 및 C 사용, 람다식	6	6
10		멀티스레딩	7	7
11		멀티스레딩, 고급문법	8	8
12		고급문법	9	9
13	병렬 프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15	기말고사			

오늘의 학습내용

- 가상함수
- 추상클래스

4

가상함수

파생클래스에서 함수 재정의

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

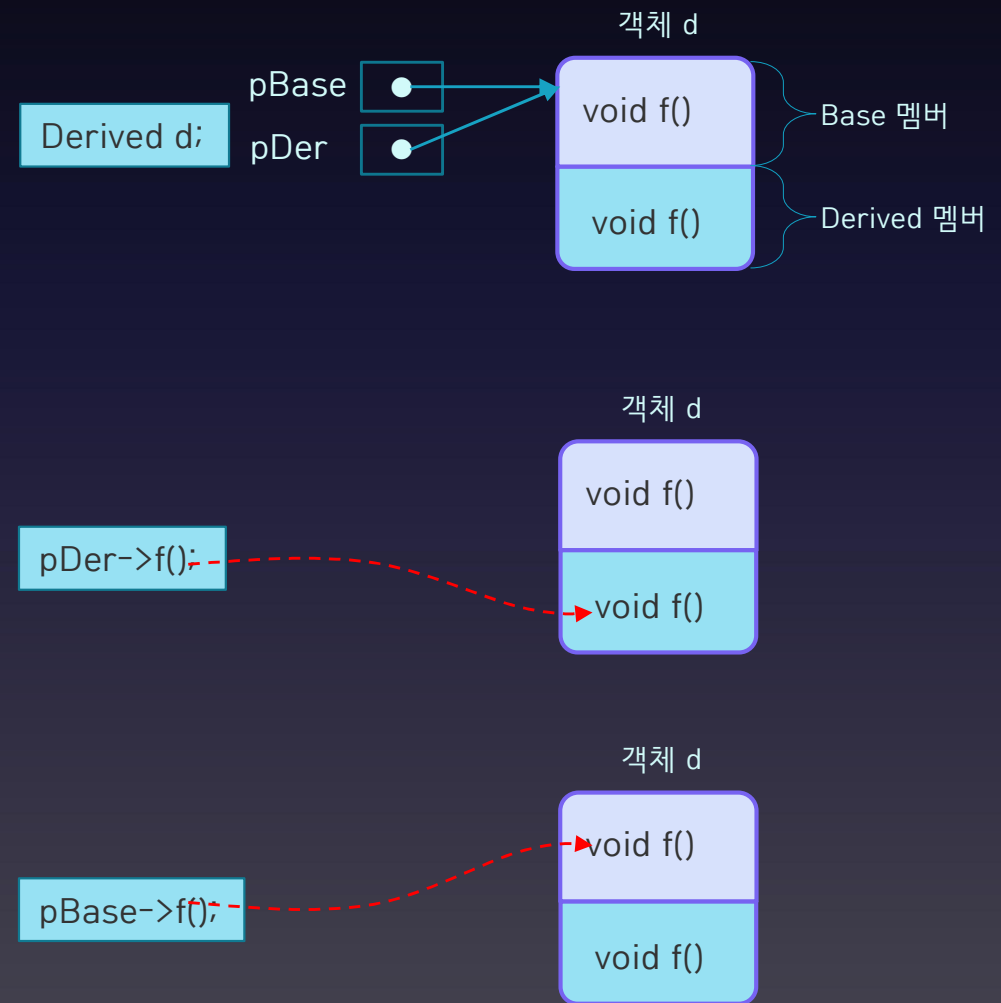
class Base {
public:
    void f() { cout << "Base::f() called" << endl; }
};

class Derived : public Base {
public:
    void f() { cout << "Derived::f() called" << endl; }
};

void main() {
    Derived d, *pDer;
    pDer = &d;
    pDer->f(); // Derived::f() 호출

    Base* pBase;
    pBase = pDer; // 업캐스팅
    pBase->f(); // Base::f() 호출
}
```

함수 중복



가상 함수와 오버라이딩

■ 가상 함수(virtual function)

- virtual 키워드로 선언된 멤버 함수
- virtual 키워드의 의미
 - 동적 바인딩 지시어
 - 컴파일러에게 함수에 대한 호출 바인딩을 실행 시간까지 미루도록 지시

```
class Base {  
public:  
    virtual void f(); // f()는 가상 함수  
};
```

■ 함수 오버라이딩(function overriding)

- 파생 클래스에서 기본 클래스의 가상 함수와 동일한 이름의 함수 선언
 - 기본 클래스의 가상 함수의 존재감 상실시킴
 - 파생 클래스에서 오버라이딩한 함수가 호출되도록 동적 바인딩
 - 함수 재정의라고도 부름
 - 다형성의 한 종류

함수 호출 비교

```
class Base {
public:
    void f() {
        cout << "Base::f() called" << endl;
    }
};

class Derived : public Base {
public:
    void f() {
        cout << "Derived::f() called" << endl;
    }
};
```

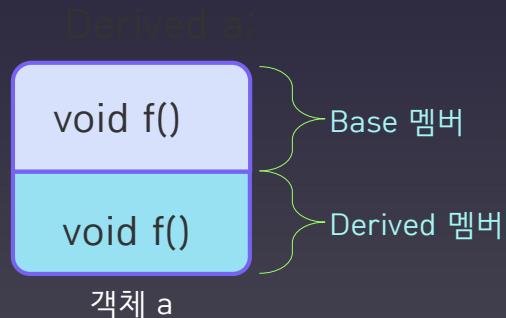
함수 재정의

```
class Base {
public:
    virtual void f() {
        cout << "Base::f() called" << endl;
    }
};

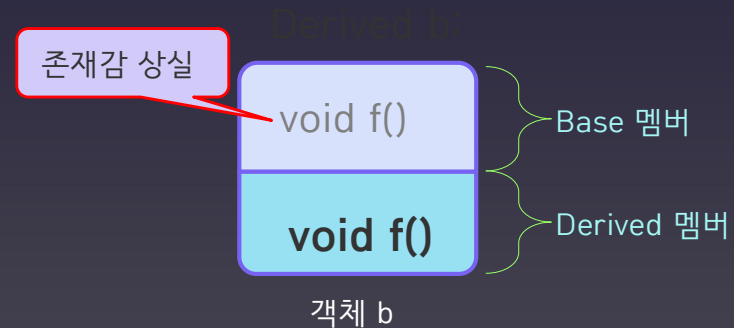
class Derived : public Base {
public:
    virtual void f() {
        cout << "Derived::f() called" << endl;
    }
};
```

가상 함수

오버라이딩



a 객체에는 동등한 호출 기회를 가진 함수 f()가 두 개 존재



존재감 상실

객체에는 두 개의 함수 f()가 존재하지만, Base의 f()는 존재감을 잃고, 항상 Derived의 f()가 호출됨

예제 - 오버라이딩과 가상 함수 호출

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

```
class Base {
public:
```

가상 함수 선언

```
    virtual void f() { cout << "Base::f() called" << endl; }
};
```

```
class Derived : public Base {
public:
```

```
    virtual void f() { cout << "Derived::f() called" << endl; }
};
```

```
int main() {
```

```
    Derived d, *pDer;
```

```
    pDer = &d;
```

```
    pDer->f(); // Derived::f() 호출
```

```
    Base * pBase;
```

```
    pBase = pDer; // 업 캐스팅
```

```
    pBase->f(); // 동적 바인딩 발생!! Derived::f() 실행
```

```
}
```

pDer->f();

객체 d

void f()

void f()

실행

pBase->f();

객체 d

void f()

void f()

동적바인딩

실행

```
Derived::f() called
Derived::f() called
```


오버라이딩의 목적

- 파생 클래스에서 구현할 함수 인터페이스 제공
- 다형성의 실현
 - draw() 가상 함수를 가진 기본 클래스 Shape
 - 오버라이딩을 통해 Circle, Rect, Line 클래스에서 자신만의 draw() 구현

```
class Shape {  
protected:  
    virtual void draw() { }  
};
```

가상 함수 선언.
파생 클래스에서 재정의할 함수에
대한 인터페이스 역할

```
class Circle : public Shape {  
protected:  
    virtual void draw() {  
        // Circle을 그린다.  
    }  
};
```

오버라이딩,
다형성 실현

```
class Rect : public Shape {  
protected:  
    virtual void draw() {  
        // Rect을 그린다.  
    }  
};
```

```
class Line : public Shape {  
protected:  
    virtual void draw() {  
        // Line을 그린다.  
    }  
};
```

```
void paint(Shape* p) {  
    p->draw();  
}
```

p가 가리키는 객체에 오
버라이딩된 draw() 호출

```
paint(new Circle()); // Circle을 그린다.  
paint(new Rect()); // Rect을 그린다.  
paint(new Line()); // Line을 그린다.
```

예제

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

class Shape {
public:
    void paint() {
        draw();
    }
    virtual void draw() {
        cout << "Shape::draw() called" << endl;
    }
};

int main() {
    Shape *pShape = new Shape();
    pShape->paint();
    delete pShape;
}
```

Shape::draw() called

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

```
class Shape {
public:
    void paint() {
        draw();
    }
    virtual void draw() {
        cout << "Shape::draw() called" << endl;
    }
};

class Circle : public Shape {
public:
    virtual void draw() {
        cout << "Circle::draw() called" << endl;
    }
};

int main() {
    Shape *pShape = new Circle(); // 업캐스팅
    pShape->paint();
    delete pShape;
}
```

기본 클래스에서 파생 클래스의
함수를 호출하게 되는 사례

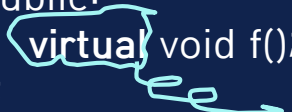
Circle::draw() called

C++ 오버라이딩의 특징

- 오버라이딩의 성공 조건
 - 가상 함수 이름, 매개 변수 타입과 개수, 리턴 타입이 모두 일치

```
class Base {  
public:  
    virtual void fail();  
    virtual void success();  
    virtual void g(int);  
};  
  
class Derived : public Base {  
public:  
    virtual int fail(); // 오버라이딩 실패  
    virtual void success(); // 오버라이딩 성공  
    virtual void g(int, double); // 오버로딩 실패  
};
```

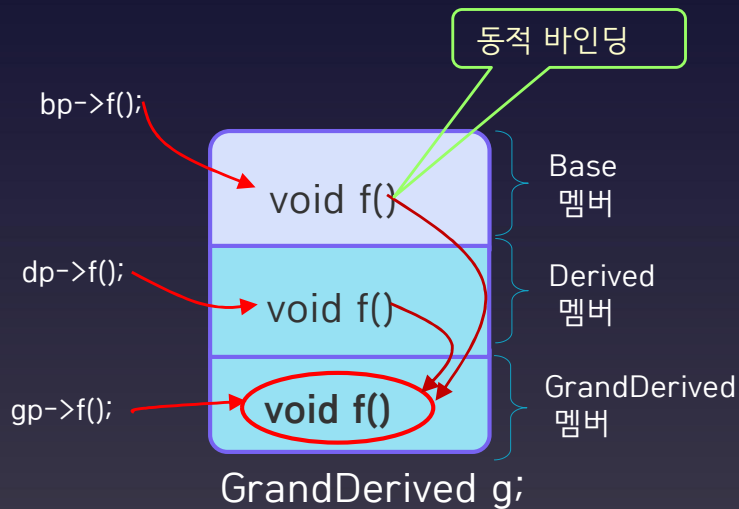
```
class Base {  
public:  
    virtual void f();  
};  
  
class Derived : public Base {  
public:  
    virtual void f();  
};
```

 생략 가능

- 오버라이딩 시 **virtual** 지시어 생략 가능
 - 가상 함수의 **virtual** 지시어는 상속됨, 파생 클래스에서 **virtual** 생략 가능
- 가상 함수의 접근 지정
 - **private**, **protected**, **public** 중 자유롭게 지정 가능

예제

- Base, Derived, GrandDerived가 상속 관계에 있을 때, 다음 코드를 실행한 결과는?



GrandDerived::f() called
GrandDerived::f() called
GrandDerived::f() called

```
class Base {
public:
    virtual void f() { cout << "Base::f() called" << endl; }
};
class Derived : public Base {
public:
    void f() { cout << "Derived::f() called" << endl; }
};
class GrandDerived : public Derived {
public:
    void f() { cout << "GrandDerived::f() called" << endl; }
};

int main() {
    GrandDerived g;
    Base *bp;
    Derived *dp;
    GrandDerived *gp;

    bp = dp = gp = &g;

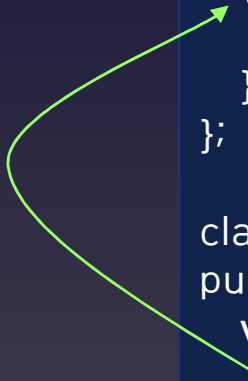
    bp->f();
    dp->f();
    gp->f();
}
```

동적 바인딩에 의해 모두 GrandDerived의 함수 f() 호출

오버라이딩과 범위 지정 연산자(::)

- 범위 지정 연산자(::)
 - 정적 바인딩 지시
 - 기본클래스::가상함수() 형태로 기본 클래스의 가상 함수를 정적 바인딩으로 호출
 - Shape::draw();

```
class Shape {  
public:  
    virtual void draw() {  
        ...  
    }  
};  
  
class Circle : public Shape {  
public:  
    virtual void draw() {  
        Shape::draw(); // 기본 클래스의 draw()를 실행한다.  
        .... // 기능을 추가한다.  
    }  
};
```



예제

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

class Shape {
public:
    virtual void draw() {
        cout << "--Shape--";
    }
};
```

정적바인딩

```
class Circle : public Shape {
public:
    virtual void draw() {
        Shape::draw(); // 기본 클래스의 draw() 호출
        cout << "Circle" << endl;
    }
};
```

동적바인딩

정적바인딩

```
int main() {
    Circle circle;
    Shape * pShape = &circle;

    pShape->draw();
    pShape->Shape::draw();
}
```

```
--Shape--Circle
--Shape--
```

가상 소멸자

■ 가상 소멸자

- 소멸자를 **virtual** 키워드로 선언
- 소멸자 호출 시 동적 바인딩 발생

```
class Base {  
public:  
    ~Base();  
};  
  
class Derived: public Base {  
public:  
    ~Derived();  
};
```

```
int main() {  
    Base *p = new Derived();  
    delete p;  
}
```

~Base() 소멸자 실행

소멸자가 가상 함수가 아닌 경우

```
class Base {  
public:  
    virtual ~Base();  
};  
  
class Derived: public Base {  
public:  
    virtual ~Derived();  
};
```

```
int main() {  
    Base *p = new Derived();  
    delete p;  
}
```

- ① ~Base() 소멸자 호출
- ② ~Derived() 실행
- ③ ~Base() 실행

가상 소멸자 경우

예제

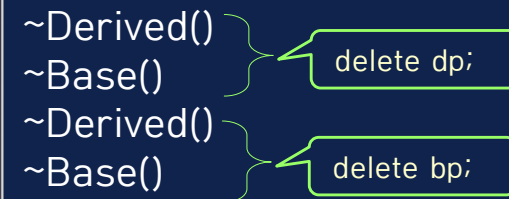
```
#include <iostream>
using namespace std;

class Base {
public:
    virtual ~Base() { cout << "~Base()" << endl; }
};

class Derived: public Base {
public:
    virtual ~Derived() { cout << "~Derived()" << endl; }
};

int main() {
    Derived *dp = new Derived();
    Base *bp = new Derived();

    delete dp; // Derived의 포인터로 소멸
    delete bp; // Base의 포인터로 소멸
}
```



순수 가상 함수

- 기본 클래스의 가상 함수 목적
 - 파생 클래스에서 재정의할 함수를 알려주는 역할
 - 실행할 코드를 작성할 목적이 아님
 - 기본 클래스의 가상 함수를 굳이 구현할 필요가 있을까?
- 순수 가상 함수
 - pure virtual function
 - 함수의 코드가 없고 선언만 있는 가상 멤버 함수
 - 선언 방법
 - 멤버 함수의 원형 = 0;

```
class Shape {  
public:  
    virtual void draw()=0; // 순수 가상 함수 선언  
};
```

추상클래스

추상 클래스

- 추상 클래스 : 최소한 하나의 순수 가상 함수를 가진 클래스

```
class Shape { // Shape은 추상 클래스
    Shape *next;
public:
    void paint() {
        draw();
    }
    virtual void draw() = 0; // 순수 가상 함수
};
void Shape::paint() {
    draw(); // 순수 가상 함수라도 호출은 할 수 있다.
}
```

- 추상 클래스의 특징

- 온전한 클래스가 아니므로 객체 생성 불가능

```
Shape shape; // 컴파일 오류
Shape *p = new Shape(); // 컴파일 오류
```

error C2259: 'Shape' : 추상 클래스를 인스턴스화 할 수 없음

- 추상 클래스의 포인터는 선언 가능

```
Shape *p;
```

추상 클래스의 목적

■ 추상 클래스의 목적

- 추상 클래스의 인스턴스를 생성할 목적 아님
- 상속에서 기본 클래스의 역할을 하기 위함
 - 순수 가상 함수를 통해 파생 클래스에서 구현할 함수의 형태(원형)을 보여주는 인터페이스 역할
 - 추상 클래스의 모든 멤버 함수를 순수 가상 함수로 선언할 필요 없음

추상 클래스의 상속과 구현

- 추상 클래스의 상속
 - 추상 클래스를 단순 상속하면 자동 추상 클래스
- 추상 클래스의 구현
 - 추상 클래스를 상속받아 순수 가상 함수를 오버라이딩
 - 파생 클래스는 추상 클래스가 아님

Shape은
추상 클래스

Circle도
추상 클래스

```
class Shape {  
public:  
    virtual void draw() = 0;  
};  
  
class Circle : public Shape {  
public:  
    string toString() { return "Circle  
객체"; }  
};
```

```
Shape shape; // 객체 생성 오류  
Circle waffle; // 객체 생성 오류
```

추상 클래스의 단순 상속



```
class Shape {  
public:  
    virtual void draw() = 0;  
};
```

Shape은
추상 클래스

Circle은
추상 클래스 아님

```
class Circle : public Shape {  
public:  
    virtual void draw() {  
        cout << "Circle";  
    }  
    string toString() { return "Circle 객체"; }  
};
```

순수 가상 함수
오버라이딩

```
Shape shape; // 객체 생성 오류  
Circle waffle; // 정상적인 객체 생성
```

추상 클래스의 구현

실습 1

- 다음 추상 클래스 Calculator를 상속받아 GoodCalc 클래스를 구현하라.

```
class Calculator {  
public:  
    virtual int add(int a, int b) = 0; // 두 정수의 합 리턴  
    virtual int subtract(int a, int b) = 0; // 두 정수의 차 리턴  
    virtual double average(int a [], int size) = 0; // 배열 a의 평균 리턴. size는 배열의 크기  
};
```

```
int main() {  
    int a[] = {1,2,3,4,5};  
    Calculator *p = new GoodCalc();  
    cout << p->add(2, 3) << endl;  
    cout << p->subtract(2, 3) << endl;  
    cout << p->average(a, 5) << endl;  
    delete p;  
}
```

```
5  
-1  
3
```

실습 2

- 사각형에 내접하는 도형을 표현하기 위한 **Shape** 클래스가 있다. 실행결과를 참조하여 **Shape**을 상속받아 타원을 표현하는 **Oval**, 사각형을 표현하는 **Rect** 클래스를 작성하라.

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

class Shape
{
protected:
    string name; // 도형의 이름
    int width, height; // 도형이 내접하는 사각형의 너비와 높이

public:
    Shape(string n="", int w=0, int h=0) { name = n; width = w; height = h; }
    virtual double getArea() = 0;
    string getName() { return name; } // 이름 리턴
};
```

실습 2

- 사각형에 내접하는 도형을 표현하기 위한 **Shape** 클래스가 있다. 실행결과를 참조하여 **Shape**을 상속받아 타원을 표현하는 **Oval**, 사각형을 표현하는 **Rect** 클래스를 작성하라.

```
int main() {  
    Shape *p[2];  
    p[0] = new Oval("빈대떡", 10, 20);  
    p[1] = new Rect("찰떡", 30, 40);  
  
    for(int i=0; i<2; i++)  
        cout << p[i]->getName() << " 넓이는 "  
              << p[i]->getArea() << endl;  
  
    for(int i=0; i<3; i++) delete p[i];  
}
```

C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
Ellipse Area : 628  
Rectangle Area : 1200  
Press <RETURN> to close this window...
```