- 이 문제를 해결하는 가장 쉬운 방법은 '소멸자를 가상화'하는 것임.
- 다음 코드처럼 기본 클래스의 소멸자를 가상함수로 선언하면 됨.

```
    // 소멸자를 가상함수로 선언
    virtual ~CTest() {
        cout << "~CTest() " << endl;
        delete[] m_pszData;
    }
```

- 실행결과

```
~CTestEx()
~CTest()
계속하려면 아무 키나 누르십시오 . . .
```

- 소멸자를 가상화하고 실행결과를 확인하면 파생클래스의 소멸자까지 제대로 호출된 것을 확인할 수 있음.

31. 가상함수 테이블

- 가상함수 테이블은 virtual function table임. 줄여서 vtable 이라고 부르기도 함.
- vtable이라는 것은 '함수 포인터 배열'로 볼 수 있음.

32. 순수 가상 클래스

- 순수 가상 클래스(Pure virtual class)는 순수 가상함수(Pure virtual function)을 멤버로 가진 클래스를 말함
- 순수 가상함수는 선언은 지금 해두지만 정의는 미래에 하도록 미뤄둔 함수로 함수 선언시 끝부분에 '=0' 이라는 표현이 붙음.

```
    virtual int GetData() const = 0;
```

- 순수 가상함수를 멤버로 가진 순수 가상 클래스의 가장 중요한 특징은 바로 인스턴스를 직접 선언할 수 없다라는 사실임.
- 순수 가상 클래스의 파생클래스는 반드시 기본 클래스의 순수 가상함수를 재정의해야 합니다.
- 순수 가상 클래스 정의

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

// 최초 설계자 코드
class CMyInterface
{
public:
    CMyInterface() {
        cout << "CMyInterface()" << endl;
    }
    // 선언만 있고 정의는 없는 순수 가상함수
    virtual int GetData() const = 0;
    virtual void SetData(int nParam) = 0;
};
```

```
// 후기 개발자 코드
class CMyData : public CMyInterface
{
public:
    CMyData() {
        cout << "CMyData()" << endl;
    }
    // 순수 가상 함수는 파생클래스에서 '반드시' 정의해야한다.
    virtual int GetData() const { return m_nData; };
    virtual void SetData(int nParam) { m_nData = nParam; };

private:
    int m_nData;
};

// 사용자 코드
int main()
{
    // 순수 가상 클래스는 인스턴스를 선언 및 정의할 수 없다.
    // CMyInterface a;
    CMyData a;
    a.SetData(5);
    cout << a.GetData() << endl;
    return 0;
}
```

- CMyInterface 클래스는 순수 가상 클래스이므로 인스턴스를 선언하거나 동적으로 생성할 수 없음.
- CMyData 클래스는 CMyInterface 클래스의 파생 클래스이므로 CMyInterface에서 순수 가상함수로 선언한 멤버(virtual int GetData() const, virtual void SetData(int nParam))를 반드시 재정의 해야함.
- a.SetData(5); 문장에서 호출하는 SetData()는 명백히 CMyData 클래스의 SetData()임. 접근형식도 실형식도 모두 CMyData 임을 확인할 수 있음.
- 만일 추상자료형으로 CMyData클래스의 인스턴스를 참조한다 하더라도 가상함수이므로 실형식을 따르기 때문에 어떤 경우라도 늘 CMyData 클래스의 함수가 호출됨.

■ 인터페이스 상속

- 인터페이스 Interface는 서로 다른 두 객체가 서로 맞닿아 상호 작용할 수 있는 통로나 방법임.
- 예를 들어 가장 대표적인 범용 인터페이스로는 USB가 있음. 많은 컴퓨터들이 외부 기기와 연결하는 방법으로 USB 인터페이스를 제공하며, 컴퓨터와 연결하려는 수많은 장치들도 USB 인터페이스를 제공함.
- 다른 장치(객체)들과 상호작용할 생각이라면 가장 많이 사용되는 보편적 인터페이스를 선택하게 됨.
- 인터페이스 예제

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

// 최초 제작자의 코드
class CMyObject
{
public:
    CMyObject() { m_nDeviceID = 0; }
    virtual ~CMyObject() { }

    // 모든 파생 클래스는 이 메서드를 가졌다고 가정할 수 있다.
    virtual int GetDeviceID() const = 0;
protected:
    int m_nDeviceID;
};

// 초기 제작자가 만든 함수
void PrintID(CMyObject* pObj) // 전역함수(멤버함수가 아님)
{
    // 실제로 어떤 것일지는 모르지만 그래도 ID는 출력할 수 있다.
    cout << "Device ID : " << pObj->GetDeviceID() << endl;
}

// 후기 제작자의 코드
class CMyTV : public CMyObject
{
public:
    CMyTV(int nID) { m_nDeviceID = nID; }
    virtual int GetDeviceID() const
    {
        cout << "CMyTV::GetDeviceID()" << endl;
        return m_nDeviceID;
    }
};
class CMyPhone : public CMyObject
{
public:
    CMyPhone(int nID) { m_nDeviceID = nID; }
    virtual int GetDeviceID() const
    {
        cout << "CMyPhone::GetDeviceID()" << endl;
        return m_nDeviceID;
    }
};
```

```
// 사용자 코드
int main()
{
    CMyTV a(5);
    CMyPhone b(6);

    // 실제 객체가 무엇이든 알아서 자신의 ID를 출력한다.
    ::PrintID(&a);
    ::PrintID(&b);
    return 0;
}
```

- 실행결과

```
CMyTV::GetDeviceID()
Device ID : 5
CMyPhone::GetDeviceID()
Device ID : 6
계속하려면 아무 키나 누르십시오 . . .
```

- TV든 전화기든 무엇이든 CMyObject 의 파생 형식이라면 모두 int GetDeviceID() 가 순수 가상함수이기 때문에 라는 int GetDeviceID()라는 멤버함수를 가지게 됨.
- 이 점을 이용해 void PrintID() 함수를 만든 것으로 볼 수 있음.
- 함수의 매개변수는 CMyObject* pObj(CMyObject 형식에 대한 포인터) 임. 따라서 CMyObject 뿐만 아니라 CMyObject 클래스의 파생형식은 모두 포인팅할 수 있음.
- ::PrintID(&a); ::PrintID(&b);를 통해 CMyTV, CMyPhone 클래스의 인스턴스 주소를 매개변수로 넘길 수 있음.
- 가상함수는 추상자료형으로 참조하더라도 언제나 실 형식의 메서드가 호출될 수 있기 때문임.