6. 가상함수와 다중상속

♲ 가상함수란?

존재하지 않는 함수라는 뜻을 가지고 있는 가상함수는 실제로 코드가 존재하는 않는 함수라는 뜻이 아니라 함수의 호출과 클래스의 참조가 컴파일 시간(compile time)에 정확히 알려지지 않고 실행시간(run time)에 정확히 결정되는 함수라는 뜻이다.

바인딩 : 함수나 변수의 주소가 결정되는 것

① 이른(초기) 바인딩 : 컴파일시간에 바인딩 되는 것
② 늦은(실행시간) 바인딩 : 실행시간에 바인딩 되는 것

가상함수의 사용법

① 기본클래스의 멤버 함수와 같은 이름을 가지는 함수를 파생클래스에서 재정의 함으로써 각 클래스마다 고유의 기능을 갖도록 변경할 때 이용한다
② 함수의 오버로딩과 유사하지만 가상함수는 기본 클래스와 함수의 반환형, 인수의 개수, 형이 같아야 한다
③ 가상함수가 포인터에 의하여 호출되는 경우에는 포인터가 가르키는 객체의 클래스에 따라 컴파일러가 결정하여 호출한다.
④ 가상함수는 하나의 인터페이스로 여러개의 수단과 방법을 제공하고자하는 객체지향 프로그래밍의 다형성을 구현한 것이다.
⑤ 가장먼저 기술되는 기본 클래스의 멤버 함수 앞에 "virtual' 라는 키워드를 기술한다.
⑥ 가상함수는 컴파일 시간에 함수의 주소가 결정되지 않는다
⑦ 클래스의 다중상속에서 virtual 을 사용하여 상속받으면 virtual 가 명시된 클래스는 가상베이스 클래스가 된다.

Ex1) 상속 관계에서 객체 포인터에 의한 함수호출 프로그램

```
#include <iostream.h>

class A {
    int i;
public:
    A() {  i = 0; }
    void Print() { cout << "A 클래스의 i = " << i << endl; }
};//class A

class B : public A {
    int i;
public:
    B() {  i = 1; }
    void Print() { cout << "B 클래스의 i = " << i << endl; }
};//class B

class C : public B {
    int i;
public:
    C() {  i = 2; }
    void Print() { cout << "C 클래스의 i = " << i << endl; }
};//class C

int main() {
    A* ap;
    B b;
    C c;
    ap = &b;
    ap->Print();    // 어떤 Print()함수가 호출될까?
    ap = &c;
    ap->Print();    // 어떤 Print()함수가 호출될까?
              return 0;
}
```

실 행 결 과	결 과 분 석
A 클래스의 i = 0 A 클래스의 i = 0 Press any key to continue...	A a; b* bp; bp = &a; 위의 문장은 에러가 발생하지만 ap = &b 는 에러가 아니다. 하지만 ap->Print() 는 A클래스의 멤버함수 Print()를 호출한다.

Ex2) 가상함수 사용예제 프로그램-1

```
#include <iostream.h>

class A {
    int i;
public:
    A() {  i = 0; }
    virtual void Print() { cout << "A 클래스의 i = " << i << endl; }
};//class A

class B : public A {
    int i;
public:
    B() {  i = 1; }
    virtual void Print() { cout << "B 클래스의 i = " << i << endl; }
};//class B

class C : public B {
    int i;
public:
    C() {  i = 2; }
    virtual void Print() { cout << "C 클래스의 i = " << i << endl; }
};//class C

int main() {
    A* ap;
    B b;
    C c;
    ap = &b;
    ap->Print();    // B 클래스의 Print()함수 호출
    ap = &c;
    ap->Print();    // C 클래스의 Print()함수 호출
              return 0;
}
```

실 행 결 과	결 과 분 석
B 클래스의 i = 1 C 클래스의 i = 2 Press any key to continue...	기본 클래스 쪽에서 가상함수를 선언하면 파생된 클래스 내의 멤버 함수도 자동으로 가상함수가 되기 때문에 재선언할 필요는 없다. 함수의 자료형과 인수의 형 및 개수는 반드시 일치해야 한다. 아니면 서로 다른 함수로 인식하게 된다.

Ex3) 가상함수 사용예제 프로그램-2

```
#include <iostream.h>
class Area {
	double side1, side2;
public:
	Area(double s1, double s2) {
		side1 = s1; side2 = s2;
	}
	void getside(double &s1, double &s2) {
		s1 = side1; s2 = side2;
	}
	virtual double getarea() {
		return 0.0;
	}
};

class Rect: public Area {
	public:
		Rect(double s1, double s2):Area(s1, s2) {
	}
	double getarea() {
		double s1, s2;
		getside(s1, s2);
		return s1 * s2;
	}
};

class Tri:public Area {
	public:
		Tri(double s1, double s2):Area(s1, s2) {
	}
	double getarea() {
		double s1, s2;
		getside(s1, s2);
		return 0.5 * s1 * s2;
	}
};

void main() {
	Area *p;
	Rect rect1(3.3, 9.8);
	Tri tri1(6.3, 5.8);
	p = &rect1;
	cout << "사각형의 면적 : " << p->getarea() << endl;
	p = &tri1;
	cout << "삼각형의 면적 : " << p->getarea() << endl;
}
```

실 행 결 과	결 과 분 석
사각형의 면적 : 32.34 삼각형의 면적 : 18.27 Press any key to continue...	기본클래스의 포인터는 이 클래스로부터 파생된 어떠한 파생클래스도 가리킬 수 있다 p가 가르키는 객체의 클래스에 따라 자동적으로 각 클래스에 소속된 멤버함수를 호출한다.

Ex4) 기본 클래스의 객체를 포인터로 정의하여 멤버함수를 호출하는 프로그램

```
#include<iostream.h>
class Master {
public:
   Master() {   }
   Master(const Master&) {   }
   virtual void  Write() const {    // 가상 함수
       cout << "₩n Master Class ....₩n";
   }
};

class First : public Master {
public:
   void Write() const {
       cout << "₩n First Class ....₩n";
   }
};

class Second : public Master {
public:
   void Write() const {
       cout << "₩n Second Class ....₩n";
   }
};

class Third : public Master {
public:
   void Write() const {
       cout << "₩n Third Class ....₩n";
   }
};

class Fourth : public Master {
public:
   void Write() const {
       cout << "₩n Fourth Class ....₩n";
   }
};

void main ( )  {
   Master ob;
   Master *binding[4];

   binding[0] = new First;
   binding[1] = new Second;
   binding[2] = new Third;
   binding[3] = new Fourth;

       for(int i = 0; i < 4; i++)
            binding[i]->Write();

   ob.Write();
}
```

실 행 결 과	결 과 분 석
First Class .... Second Class .... Third Class .... Fourth Class .... Master Class .... Press any key to continue...	기본클래스형의 객체를 포인터로 정의하는 것은 일반 포인터 변수와 동일한 기능을 가진다. 가상함수로 정의된 멤버 함수 Write()의 호출은 프로그램이 실행 될 때까지 유보되며 호출은 실행 중에 포인터 객체에 의해 이루어진다.

Ex5) 가상함수 사용예제 프로그램-3

```
#include<iostream.h>
#include<iomanip.h>
#include<string.h>
class Goods {
protected:
    char *item;
    int su, cost;
public:
    Goods();
    Goods(char *, int, int);
    virtual int sales() const {    // 가상 함수 정의
        return (su * cost);
    }
};
inline Goods::Goods()   {
    item = 0;
    su = cost = 0;
}
inline Goods::Goods(char N[], int S, int C):su(S), cost(C)  {
    item = new char[strlen(N) + 1];
    strcpy(item, N);
}

class First_class : public Goods {
    char *sts;
    int jego;
public:
    First_class(char [], int, int, char[], int);
    int sales() const   {
        return (jego - su);
    }
    void print(int J) const   {
       cout << setiosflags(ios::left) << setw(10) << "품목" << setw(10) << "상태" << setw(6)
         << "수량" << setw(10)  << "단가" << setw(10)  << "금액" << setw(8)  << "재고수량" << endl;
       cout << "----------------------------------------------------------\n";
       cout << setiosflags(ios::left) << setw(10) << item << setw(10) << sts
         << setiosflags(ios::right) << setw(6) << su << setw(10) << cost << setw(10)
         << Goods::sales() << setw(8)  << J << endl; // 기본클래스의 생성자 호출
    }
};
```

```
First_class::First_class(char N[], int S, int C, char St[], int J) :
    Goods(N, S, C), jego(J)  {
    sts = new char[strlen(St) + 1];
    strcpy(sts, St);
}

class Second_class : public Goods {
    int year, month, day, jego;
public:
    Second_class(char N[], int S, int C, int Y, int M, int D) :
Goods(N, S, C), year(Y), month(M), day(D)  { }
    void print(int B) const   {
        cout << setiosflags(ios::left) << setw(10) << "₩n품목" << setw(6) << "수량" << setw(10) <<
"단가" << setw(15) << "판매일자" << setw(10) << "금액" << endl;
       cout << "-----------------------------------------------------₩n";
       cout << setiosflags(ios::left) << setw(10) << item << setw(6) << su << setw(10) << cost <<
setw(4) << year << "." << setw(2) << month << "." << setw(6) << day << setw(10)<< B << endl;
    }
};
int main() {
    Goods *obPtr1, *obPtr2;
    int jego, BL;

    First_class ob1("TV", 4, 20000, "Best Q", 15);
    Second_class ob2("Computer", 3, 77000, 2005, 8, 19);

    obPtr1 = &ob1;     // 기본클래스형의 포인터로 파생 클래스형의 객체를 가리킴
    jego = obPtr1->sales();   // 파생 클래스의 가상 함수의 호출을 결정
    ob1.print(jego);
    cout << "₩n ********************************************************** ₩n";

    obPtr2 = &ob2;
    BL = obPtr2->sales();    // 기본 클래스의 가상 함수 호출
    ob2.print(BL);
    return 0;
}
```

실 행 결 과

```
품목    상태    수량    단가    금액    재고수량
-------------------------------------------------
TV      Best Q  4       20000   80000   11

*****************************************************
품목            수량    단가    판매일자        금액
-------------------------------------------------
Computer        3       77000   2005. 8. 19     231000

Press any key to continue...
```

순수 가상함수

기본클래스에서 특별한 연산을 수행하지 않고 단지 선언만 되어, 어떠한 동작도 정의되지 않고 함수 선언만을 하는 가상함수 이다. 함수가 반드시 필요하지만 파생클래스에서 정의해야 하는 경우 빈 함수를 만들던가 주석처리를 하게 된다.

```
// 주석으로 순수 가상함수를 대신한 예
...
class A {
public :
        virtual void print();
        // 반드시 잊어야할 주석 : print()함수는 자식 클래스에서 반드시 정의 되어야함
};      // class A
...
```

하지만 C++에서는 이 함수가 순수 가상함수 라는 것을 알려주어 구현한다.

순수 가상함수의 선언 형식

```
virtual 자료형 함수명 (인수 리스트) = 0;
```

순수 가상함수의 성질

① 순수 가상함수는 본체를 가지지 않는 함수이다.
② 순수 가상함수는 문서화에 도움을 주며, 클래스를 추상화 시킨다.
③ 가상함수에 0을 지정하면 파생 클래스에서 반드시 이 함수를 오버로딩 해야한다.
④ 함수가 반드시 필요하지만 파생클래스에서 정의해야 하는 경우에 사용한다.
⑤ 어떠한 클래스를 디자인할 때, 반드시 있어야 하지만 그 기능은 사용자에게 맡기고 싶을 때 역시 순수 가상함수를 사용한다.

Ex6) 순수가상함수의 예제

```
#include <iostream.h>

class A {
public:
    virtual void Print() = 0;   // 순수 가상 함수
};  //class A

class B : public A {
    int i;
public:
    B() {  i = 1; }
    virtual void Print() {
        cout << "A job in child only" << endl;
        cout << "i = " << i << endl;
    }
};  //class B

int main() {
    A* ap;
    B b;
    ap = &b;
    ap->Print();
                return 0;
}
```

실 행 결 과	결 과 분 석
A job in Derived only i = 1 Press any key to continue...	

※ 추상클래스

최소한 한 개 이상의 순수 가상함수를 메서드(Method)로 가지는 클래스는 완전하게 정의(Define) 되지 않았음으로, 실체화 할 수 없다. 이러한 클래스를 추상 클래스라고 한다. 그럼으로 추상클래스의 객체를 만드는 것은 당연히 불가능하다. 이처럼 최소한 1개 이상의 순수 가상 함수를 가지는 클래스는 다른 클래스의 베이스 클래스로만 사용되며, 이런 추상 클래스(abstract class)를 '객체를 만들 수 없는 클래스', '순수가상함수를 가지는 클래스' 라고 한다.

Ex7) 순수가상함수의 사용 예제 프로그램-1

```
#include <iostream.h>
class Base {
	public: virtual void show() = 0;  // 순수 가상 함수
};

class Derv1 : public Base {
	public:void show() { cout << "Wn Derv1 "; }
};

class Derv2 : public Base {
	public:void show() { cout << "Wn Derv2 "; }
};

int main() {
	Base *list[2];

	Derv1 dv1;
	Derv2 dv2;

	list[0] = &dv1;
	list[1] = &dv2;

	list[0]->show();
	list[1]->show();

	return 0;
}
```

실 행 결 과	결 과 분 석
Derv1 Derv2 Press any key to continue...	함수에서의 = 0을 값을 할당한다는 의미가 아니라 단순히 컴파일러에게 함수의 본체가 없다는 것을 알려주기 위해 사용된다. 만약, 기본클래스의 가상함수 show()를 제거하면 list[] 배열에 있는 포인터 들이 클래스 Base의 멤버들을 지정 할 수 없기 때문에 에러를 발생할 것이다.

Ex8) 순수가상함수의 사용 예제 프로그램-2

```
#include <iostream.h>
class Date {
protected:
		int year, month, day;
public:
	Date(int yy, int mm, int dd) {
		year = yy;	month = mm;		day = dd;
  }
	virtual void show() = 0;
};

class Derv1_Date : public Date {
	public:
		Derv1_Date(int yy, int mm, int dd) : Date(yy, mm, dd)   {
		}
		void show() {
			cout << year << "년 " << month << "월 " << day << "일" << endl;
	}
};

class Derv2_Date : public Date {
	public:
		Derv2_Date(int yy, int mm, int dd) : Date(yy, mm, dd)   {
		}
		void show();
};

void Derv2_Date::show() {
	static char *mon[] = {
		"Jan.", "Feb.", "Mar.", "Apr.", "May.", "June.",
		"July.", "Aug.", "Sep.", "Oct.", "Nov.", "Dec."
	};
	cout << mon[month - 1] << ' ' << day << ' ' << year << endl;
}

void main() {
	Derv1_Date date1(1995, 5, 5);
	Derv2_Date date2(2005, 5, 5);

	Date &D1 = date1, &D2 = date2;
	date1.show();
	date2.show();
}
```

실 행 결 과	결 과 분 석
1995년 5월 5일 May. 5 2005 Press any key to continue...	main()함수에서 포인터와 마찬가지로 기본클래스의 레퍼런스 D1, D2가 파생 클래스를 참조할 수 있다.

Ex9) 순수가상함수의 사용 예제 프로그램-3

```
#include <iostream.h>
#include <string.h>

class Master {
protected:
    char *str;
public:
    Master()   {
        str = new char[1];
        *str = 0;
    }
    virtual char * ret_value() = 0;    // 순수 가상 함수의 정의
    friend ostream & operator << (ostream &, Master &); // << 연산자 오버로딩
};
ostream & operator << (ostream & out, Master &temp) {
       out << temp.str;
    return out;
}

class A_class : public Master {
public:
    A_class(const char * temp)    {
        strcpy(str, temp);
    }
    char * ret_value()     {
        return (str);
    }
};

class B_class : public Master {
public:
    B_class(const char * temp)     {
        strcpy(str, temp);
    }
    char * ret_value()     {
        return (str);
    }
};

class C_class : public Master {
public:
    C_class(const char * temp)     {
        strcpy(str, temp);
    }
    char * ret_value()     {
        return (str);
    }
};

void main() {
    A_class ob1("Hello, ");
    B_class ob2("Good ");
    C_class ob3("Morning!");
    Master *obPtr1 = &ob1, *obPtr2 = &ob2, *obPtr3 = &ob3;
    cout << ob1.ret_value() << ob2.ret_value() << ob3.ret_value() << endl;
    cout << ob1 << ob2 << ob3 << endl;
    cout << obPtr1->ret_value() << obPtr2->ret_value() << obPtr3->ret_value() << endl;
    cout << *obPtr1 << *obPtr2 << *obPtr3 << endl;
}
```

실 행 결 과	결 과 분 석
Hello, Good Morning! Hello, Good Morning! Hello, Good Morning! Hello, Good Morning! Press any key to continue...	기본클래스인 Master 클래스는 추상 클래스이므로 객체 생성이 불가능하다. 그리하여 friend 함수를 선언한 Master 전달인자가 레퍼런스로 사용 되었다.

Ex10) 순수가상함수가 정의 되어 있는 추상 클래스 예제 프로그램

```
#include<iostream.h>
class BASE{       // 기반 클래스
protected:
       int a, b;
public:
       BASE();
       BASE(int, int);
       virtual void prn() = 0;
       virtual void sum() = 0;
};
BASE::BASE()  {        a=0;    b=0;  }
BASE::BASE(int aa, int bb)  {    // 생성자 함수 정의
       a = aa;   b = bb;
}
void BASE::prn() {
       cout << "Wn BASE Print : ";
       cout << " a = " << a << ", b = " << b << endl;
}

class ADD:public BASE{
protected :
       int c;
public :
       ADD(int, int);
       void sum();
       void prn();
};
ADD::ADD(int aa, int bb) : BASE() {
       a = aa; b = bb;
}
void ADD::sum() {
       c = a + b;
}
void ADD::prn() {
       cout << "Wn ADD Print : ";
       cout << a << " + " << b << " = " << c << endl;
}

void main() {
       ADD  ADD_ob(10, 20);

       BASE *pBASE;
       pBASE = &ADD_ob;

       pBASE->sum();
       pBASE->prn();
}
```

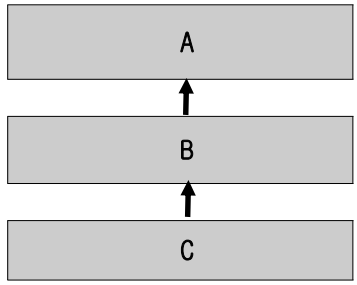
실 행 결 과	결 과 분 석
ADD print : 10 + 20 = 30 Press any key to continue...	기본클래스의 객체 포인터 pBASE는 파생 클래스 형의 객체를 가리키고, 이를 통하여 sum() 함수와 prn() 함수를 간접 호출하게 되면 늦은 바인딩으로 파생클래스의 prn()함수가 실행된다.

다중상속

두 개 이상의 이미 정의되어 있는 클래스로부터 상속받는 것을 말한다. 여기에는 세가지의 형태가 있다.

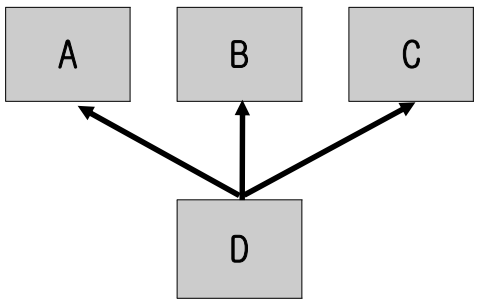
① 파생클래스가 다른 파생클래스의 기본클래스가 되는 형태

```
class A {.................};
class B : public A {..............};
class C : public B {..............};
```



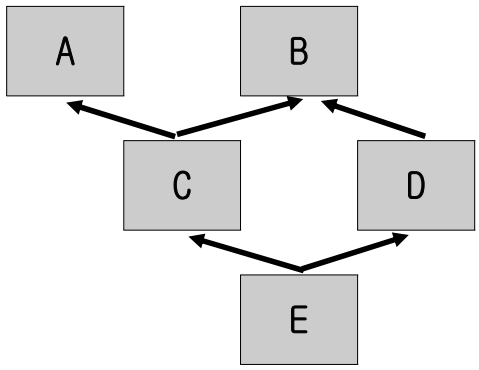
② 파생클래스가 하나 이상의 클래스를 기본 클래스로 하여 상속받는 형태

```
class A {.................};
class B {.................};
class C {.................};
class D : public A, public B, public C
{..............};
```



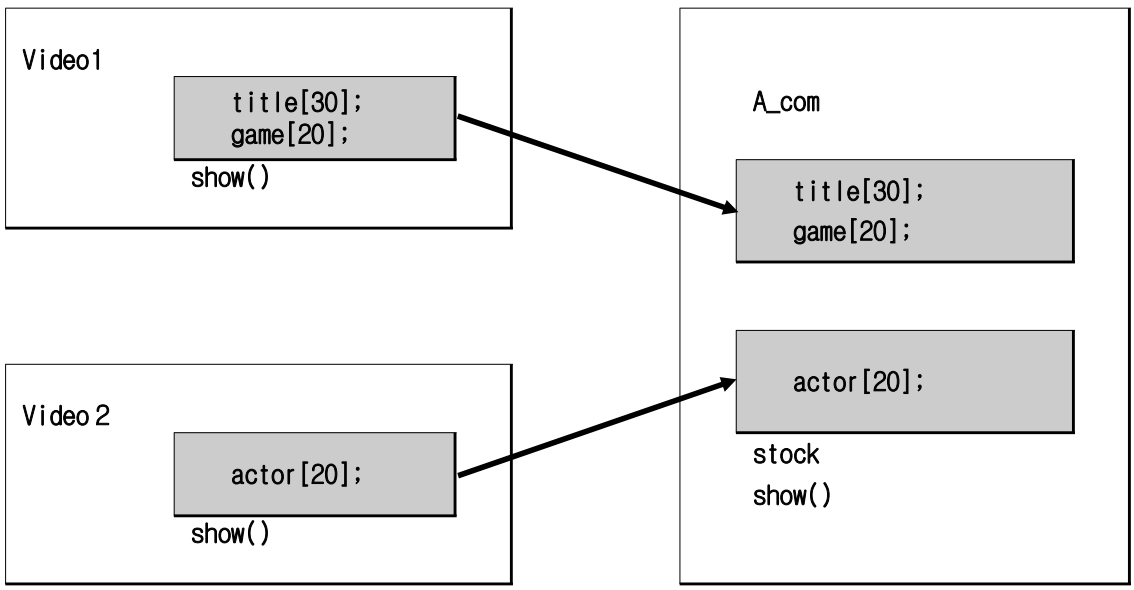
③ 위의 두 가지가 결합된 형태

```
class A {.................};
class B {.................};
class C : public A, public B
        {..............};
class D : public B {..............};
class E : public C, public D
        {..............};
```



```
class Video1 {
	protected :
		char title[30];
		char game[20];
	public :
		Video(char[], char[]);
		virture void show();
		// ...
};
class Video2 {
	protected :
		char actor[20];
	public :
		Video2(char[]);
		void show();
		// ...
};
class A_com : public Video1 , public Video2  {
	protected :
		int stock;
	public :
		A_com(char[], char[], int);
		void show();
		// ...
};
```

위 예에서 각 클래스에서의 상속관계



위 예에서 A_com의 가상함수 호출 방법

```
A.com ob;
video1 *obptr = &ob;
obptr-> show();
```

Ex12) 다중상속 예제 프로그램-1

```
#include <iostream.h>
#include <iomanip.h>
#include <string.h>

class Video1 {
       protected :
              char title[30];
              char genre[20];
       public :
              Video1(char T[], char G[])  {
                  strcpy(title, T);
                  strcpy(genre, G);
         }
              void show() {
                     cout << "Title : " << title << endl;
                     cout << "Genre : " << genre << endl;
              }
};

class Video2 {
       protected :
              char actor[20];
       public :
              Video2(char A[]) {
                     strcpy(actor, A);
         }
              void show() {
                     cout << "Actor : " << actor << endl;
              }
};

class A_com : public Video1, public Video2  {  // 다중 상속
       protected :
              int order;
       public :
              A_com(char [], char [], char [], int);
              void show();
    friend ostream & operator << (ostream &, const A_com &);
};

A_com::A_com(char T[], char G[], char A[], int O)
             : Video1(T, G), Video2(A), order(O) {
}

void A_com::show()  {
    cout << "Order : " << order << endl;
}

ostream & operator << (ostream &out, const A_com &ob)  {
    out << " 제목 : " << ob.title
        << "₩n 장르 : " << ob.genre
        << "₩n 주연 : " << ob.actor
        << "₩n 주문 : " << ob.order;
    return (out);
}
```

```
void main ( )  {
   Video1 ob1("가문의 위기", "코믹"), *obPtr1; // Video1형의 객체와 객체 포인터
   Video2 ob2("신현준"), *obPtr2;              // Video2형의 객체와 객체 포인터
   A_com ob3("더 독(The Dog)", "액션", "이연걸", 10); // 파생 클래스형의 객체
   cout << " === Object ob1 === ₩n";
   ob1.show();
   cout << "₩n === Object ob2 === ₩n";
   ob2.show();
   cout << "₩n === Object ob3 === ₩n";
   cout << ob3 << endl << endl;
   ob3.Video1::show();
   ob3.Video2::show();
   ob3.show();
   obPtr1 = &ob3;
   cout << "₩n === obPtr1->show() === ₩n";
   obPtr1->show();
   obPtr2 = &ob3;
   cout << "₩n === obPtr2->show() === ₩n";
   obPtr2->show();
}
```

실 행 결 과

```
 === Object ob1 ===
Title  : 가문의 위기
Genre  : 코믹

 === Object ob2 ===
Actor  : 신현준

 === Object ob3 ===
제목 : 더 독(The Dog)
장르 : 액션
주연 : 이연걸
주문 : 10

Title  : 더 독(The Dog)
Genre  : 액션
Actor  : 이연걸
Order  : 10

 === obPtr1->show ===
Title  : 더 독(The Dog)
Genre  : 액션

 === obPtr1->show ===
Actor  : 이연걸
Press any key to continue...
```

다중 상속에서의 생성자와 소멸자

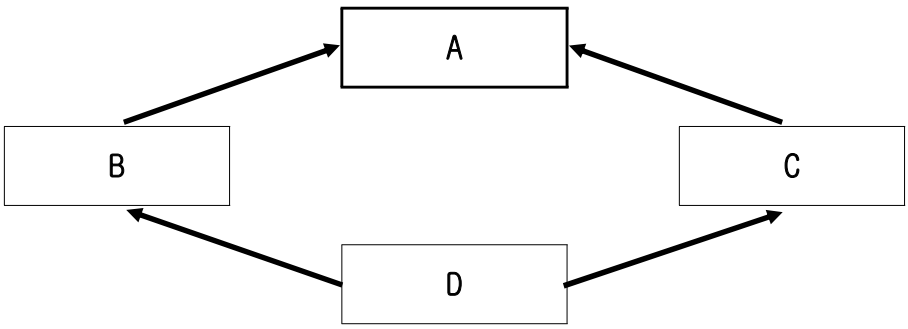
다중상속에서도 생성자는 파생된 순서대로 호출되며 소멸자는 파생된 순서의 반대로 호출된다.

생성자와 소비자 호출의 예	호출결과

class A {
public:
A() { cout << "constructing A" << endl; }
~A() { cout << "destructing A" << endl; }
}; //class A
 	
class B : public A {
public:
B() { cout << "constructing B" << endl; }
~B() { cout << "destructing B" << endl; }
}; //class B
 	
class C : public B {
public:
C() { cout << "constructing C" << endl; }
~C() { cout << "destructing C" << endl; }
}; //class C
 	
int main() {
C c;
return 0;
}
	constructing A
	constructing B
	constructing C
	destructing C
	destructing B
	destructing A
	
	파생된 순서대로 호출된다. 하지만
	이와는 다른 결과가 나올수도 있다.

가상 베이스 클래스

다중상속에 의해 발생할 수 있는 문제점 중의 하나는 공통된 조상으로부터 유도되는 다른 경로의 자손 클래스 때문에, 파생 클래스의 객체가 만들어 질 때, 여러번 참조되는 기본 클래스 때문이다.



위 그림의 파생 클래스 D에서 기본 클래스 A로의 경로는 B를 통해 혹은 C를 통해 가능하다. 이러한 정보를 컴파일 시간에 알 수 없는 것은 아니지만, 오버헤드를 초래한다. 그러므로 실행 시간에 이러한 정보를 이용하여 클래스 A는 한번만 참조되도록 해야 할 것이다.

Ex13) 다중 상속에서의 생성자와 소멸자 호출 프로그램-1

```
#include <iostream.h>

class A {
public:
    A()  { cout << "constructing A" << endl; }
    ~A() { cout << "destructing A" << endl; }
};  //class A

class B : public A {
public:
    B()  { cout << "constructing B" << endl; }
    ~B() { cout << "destructing B" << endl; }
};  //class B

class C : public A {
public:
    C()  { cout << "constructing C" << endl; }
    ~C() { cout << "destructing C" << endl; }
};  //class C

class D : public B, public C {
public:
    D()  { cout << "constructing D" << endl; }
    ~D() { cout << "destructing D" << endl; }
};  //class D

void main() {
    D d;
}
```

실 행 결 과	결 과 분 석
constructing A constructing B constructing A constructing C constructing D destructing D destructing C destructing A destructing B destructing A Press any key to continue...	예제에서 A를 참조하는 경로가 2개이므로 A의 생성자와 소멸자는 2번 호출된다.

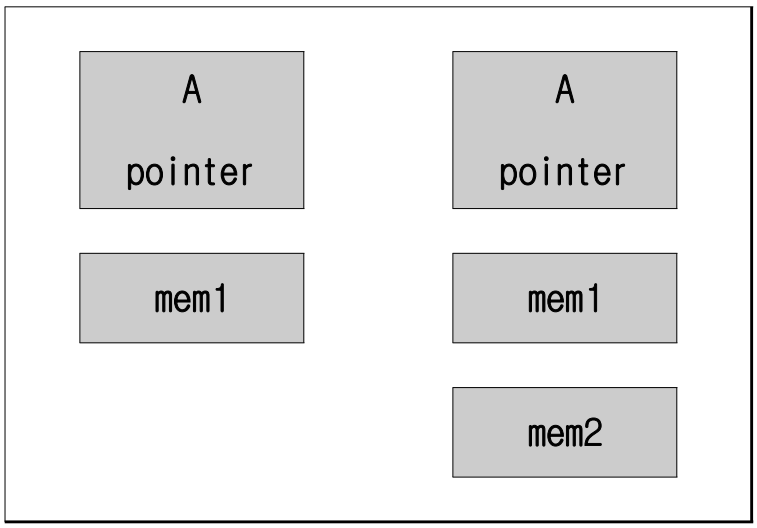
이러한 경우를 다중상속에서의 모호함(ambiguity)이라 하고, 혹은 심각한 에러를 유발시킬 수 있는 경우를 메모리 누수(memory leak)라고 한다. 위의 프로그램에서 C보다 B의 생성자가 먼저 호출되는 이유는 클래스 D를 정의할 때, 기본클래스를 명시한 순서 때문이다. 만약 클래스 E를 추가로 정의한다고 가정해보자

“ class E : public C, public B { ”

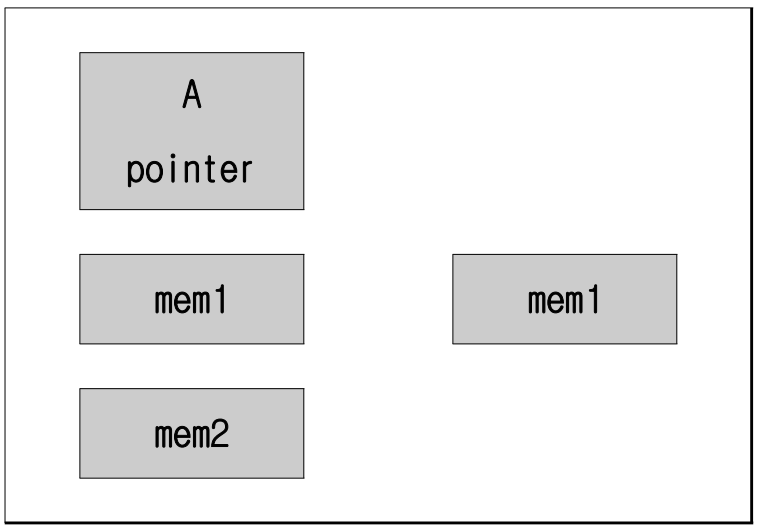
위와 같이 클래스 E를 추가 시킬 경우 D는 A, B, A, C 순으로 생성자를 호출하지만 E는 A, C, A, B 순으로 생성자를 호출한다. 대부분의 경우 문제가 없겠지만 A, B, A, C 와 A, C, A, B 는 분명히 다르며 모호하다.

또한 2개 이상의 경로가 존재하는 경우 심각한 메모리 누수가 발생할 수 있다.

클래스 A에서 동적으로 메모리를 할당할 경우 생성자의 실행 과정



클래스 A에서 동적으로 메모리를 할당할 경우 소멸자의 실행 과정



위의 문제를 실행시간의 클래스의 구조 정보를 이용함으로써 해결 할 수 있다.
즉, virtual 을 사용해 기본 클래스를 가상 베이스 클래스로 선언 한다.

Ex14) 다중 상속에서의 생성자와 소멸자 호출 프로그램-2

```
#include <iostream.h>

class A {
public:
    A()  { cout << "constructing A" << endl; }
    ~A() { cout << "destructing A" << endl; }
};   //class A

class B : virtual public A {
public:
    B()  { cout << "constructing B" << endl; }
    ~B() { cout << "destructing B" << endl; }
};   //class B

class C : virtual public A {
public:
    C()  { cout << "constructing C" << endl; }
    ~C() { cout << "destructing C" << endl; }
};   //class C

class D : public B, public C {
public:
    D()  { cout << "constructing D" << endl; }
    ~D() { cout << "destructing D" << endl; }
};   //class D

int main() {
    D d;
                return 0;
}
```

실 행 결 과

```
constructing A
constructing B
constructing C
constructing D
destructing D
destructing C
destructing B
destructing A

Press any key to continue...
```