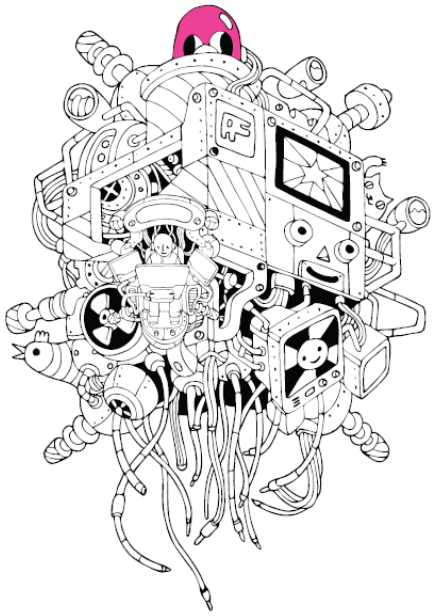
열혈 C++ 프로그래밍



윤성우 저 열혈강의 C++ 프로그래밍 개정판

Chapter 09. 가상(Virtual)의 원리와 다중상속

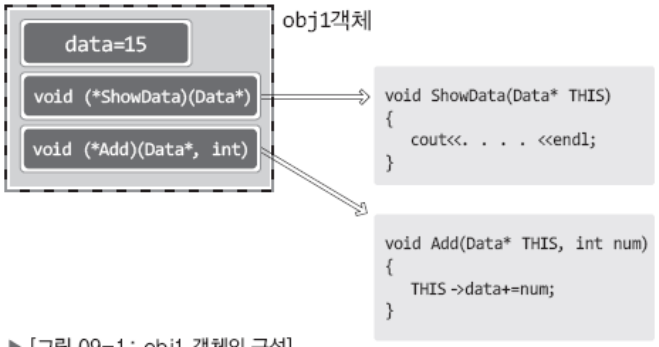
열혈 C++ 프로그래밍

Chapter 09-1. 멤버함수와 가상함수의 동작원리

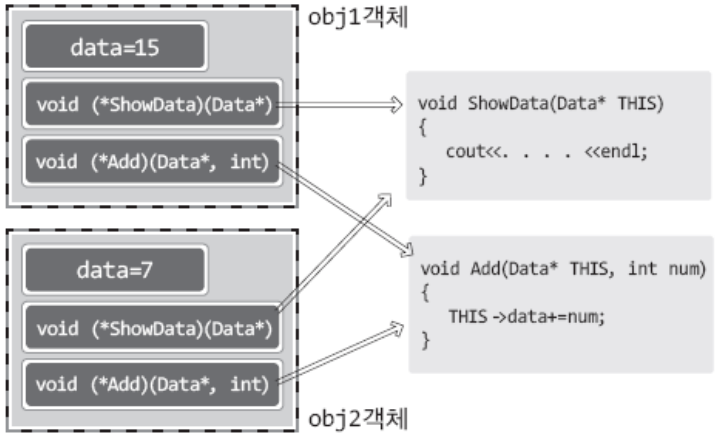
윤성우 저 열혈강의 C++ 프로그래밍 개정판

객체 안에 정말로 멤버함수가 존재하는가?

```
// 클래스 Data를 흉내 낸 영역
typedef struct Data
{
    int data;
    void (*ShowData)(Data*);
    void (*Add)(Data*, int);
} Data;

void ShowData(Data* THIS) { cout<<"Data: "<<THIS->data<<endl; }
void Add(Data* THIS, int num) { THIS->data+=num; }

// 적절히 변경된 main 함수
int main(void)
{
    Data obj1={15, ShowData, Add};
    Data obj2={7, ShowData, Add};

    obj1.Add(&obj1, 17);
    obj2.Add(&obj2, 9);
    obj1.ShowData(&obj1);
    obj2.ShowData(&obj2);
    return 0;
};
```



▶ [그림 09-1 : obj1 객체의 구성]



▶ [그림 09-2 : obj1과 obj2의 구성]

위의 예제가 보이듯이 실제로는 다수의 객체가 멤버함수를 공유하는 형태이다. 다만, 함수호출 시 객체의 정보가 전달이 되고 이를 기반으로 함수가 실행되기 때문에 논리적으로는 객체 안에 멤버함수가 존재하는 형태이다.

가상함수의 동작원리와 가상함수 테이블

```
class AAA
{
private:
    int num1;
public:
    virtual void Func1() { cout<<"Func1"<<endl; }
    virtual void Func2() { cout<<"Func2"<<endl; }
};
class BBB: public AAA
{
private:
    int num2;
public:
    virtual void Func1() { cout<<"BBB::Func1"<<endl; }
    void Func3() { cout<<"Func3"<<endl; }
};
int main(void)
{
    AAA * aptr=new AAA();
    aptr->Func1();
    BBB * bptr=new BBB();
    bptr->Func1();
    return 0;
}
```

```
Func1
BBB::Func1
```

실행결과

key	value
void AAA::Func1( )	0x1024 번지
void AAA::Func2( )	0x2048 번지

▶ [그림 09-3: AAA 클래스의 가상함수 테이블]

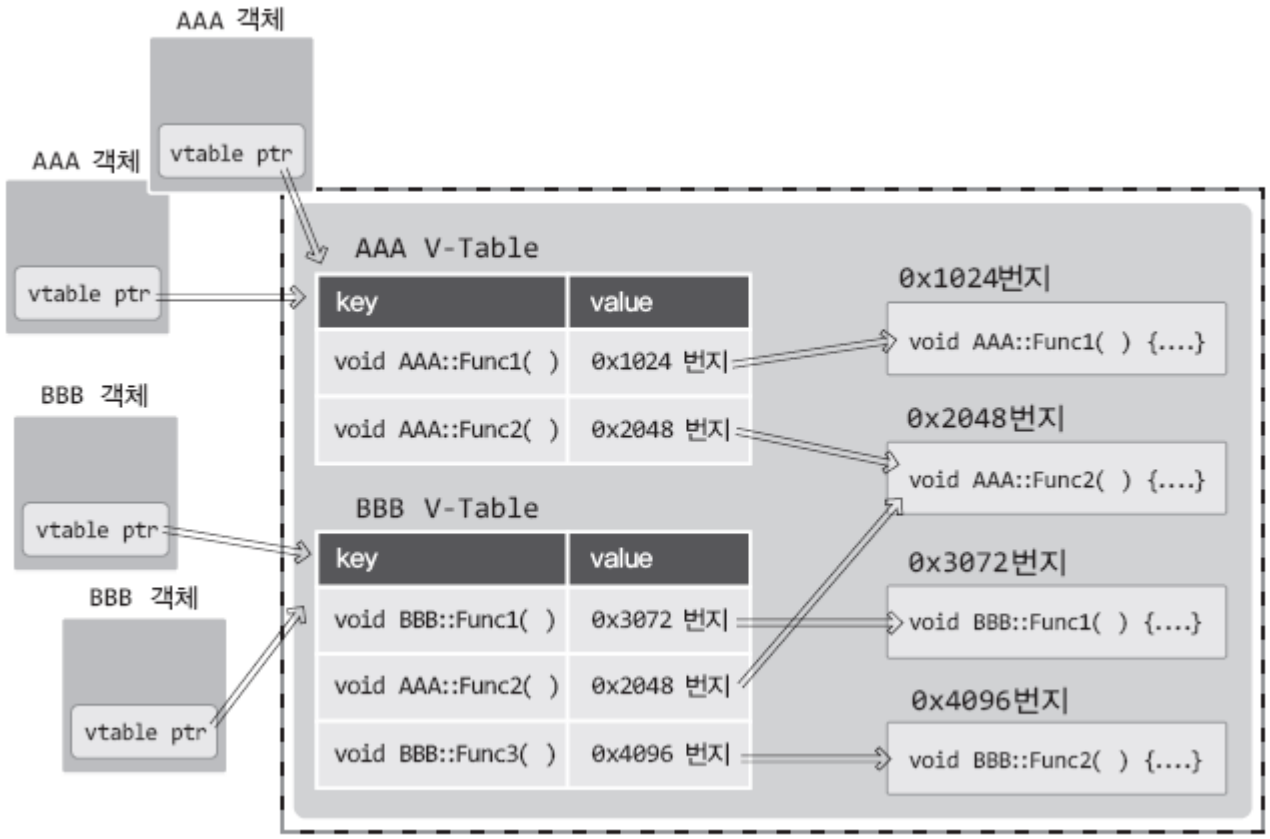
key	value
void BBB::Func1( )	0x3072 번지
void AAA::Func2( )	0x2048 번지
void BBB::Func3( )	0x4096 번지

▶ [그림 09-4: BBB 클래스의 가상함수 테이블]

하나 이상의 가상함수가 멤버로 포함되면 위와 같은 형태의 V-Table이 생성되고 매 함수호출시마다 이를 참조하게 된다.

BBB 클래스의 가상함수 테이블에는 AAA::Func1에 대한 정보가 없음에 주목하자!

가상함수 테이블이 참조되는 방식



열혈 C++ 프로그래밍

Chapter 09-2. 다중상속에 대한 이해

윤성우 저 열혈강의 C++ 프로그래밍 개정판

다중상속에 대한 견해

"다중상속은 득보다 실이 더 많은 문법이다. 그러니 절대로 사용하지 말아야 하며, 가능하다면 C++의 기본문법에서 제외시켜야 한다!"

"일반적인 경우에서 다중상속은 다양한 문제를 동반한다. 따라서 가급적 사용하지 않아야 함에는 동의를 한다. 그러나 예외적으로 매우 제한적인 사용까지 부정할 필요는 없다고 본다."

다중상속에 대한 의견은 전반적으로 매우 부정적이다!

다중상속의 기본방법

```
class BaseOne
{
public:
    void SimpleFuncOne() { cout<<"BaseOne"<<endl; }
};
class BaseTwo
{
public:
    void SimpleFuncTwo() { cout<<"BaseTwo"<<endl; }
};
class MultiDerived : public BaseOne, protected BaseTwo
{
public:
    void ComplexFunc()
    {
        SimpleFuncOne();
        SimpleFuncTwo();
    }
};
int main(void)
{
    MultiDerived mdr;
    mdr.ComplexFunc();
    return 0;
}
```

다중상속은 말 그대로 둘 이상의 클래스를 상속하는 형태이고, 이로 인해서 유도 클래스의 객체는 모든 기초 클래스의 멤버를 포함하게 된다.

본서에서 이야기한 상속의 이점과 다중상속이 어떠한 관계가 있을지 생각해보자!

실행결과

```
BaseOne
BaseTwo
```

다중상속의 모호성

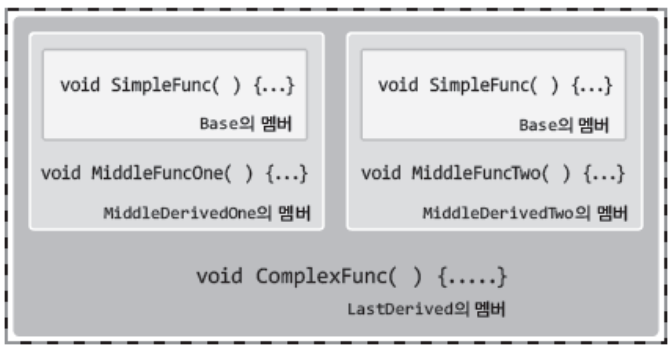
```
class BaseOne
{
public:
    void SimpleFunc() { cout<<"BaseOne"<<endl; }
};

class BaseTwo
{
public:
    void SimpleFunc() { cout<<"BaseTwo"<<endl; }
};

class MultiDerived : public BaseOne, protected BaseTwo
{
public:
    void ComplexFunc()
    {
        BaseOne::SimpleFunc();
        BaseTwo::SimpleFunc();
    }
};
```

이렇듯 호출의 대상을 구분해서 명시해야 한다.

더 모호한 상황

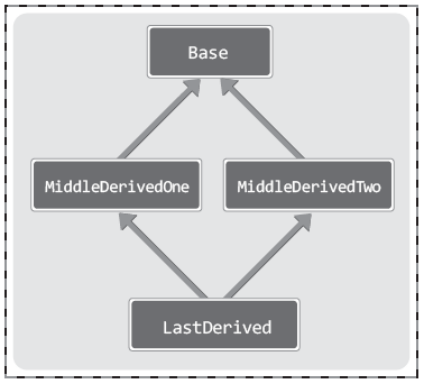
과연 LastDerived 객체에 두 개의 Base 멤버가 필요한가?

```
class Base
{
public:
    Base() { cout<<"Base Constructor"<<endl; }
    void SimpleFunc() { cout<<"BaseOne"<<endl; }
};

class MiddleDerivedOne : virtual public Base
{
public:
    MiddleDerivedOne() : Base()
    {
        cout<<"MiddleDerivedOne Constructor"<<endl;
    }
    void MiddleFuncOne()
    {
        SimpleFunc();
        cout<<"MiddleDerivedOne"<<endl;
    }
};
```



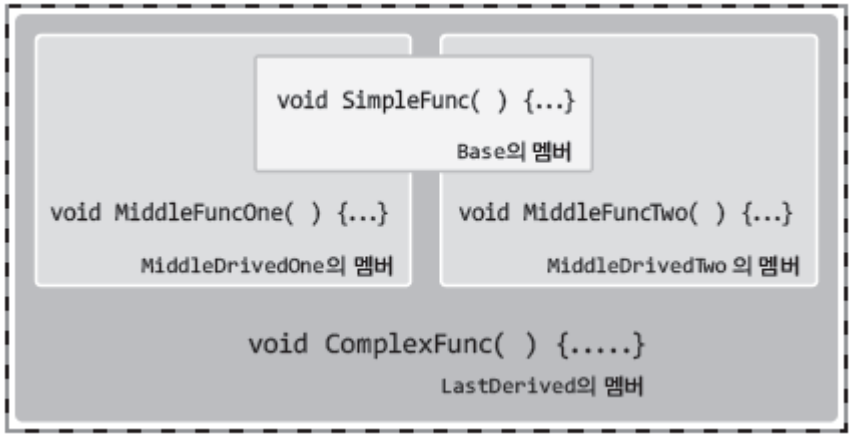

```
class MiddleDerivedTwo : virtual public Base
{
public:
    MiddleDerivedTwo() : Base()
    {
        cout<<"MiddleDerivedTwo Constructor"<<endl;
    }
    void MiddleFuncTwo()
    {
        SimpleFunc();
        cout<<"MiddleDerivedTwo"<<endl;
    }
};

class LastDerived : public MiddleDerivedOne, public MiddleDerivedTwo
{
public:
    LastDerived() : MiddleDerivedOne(), MiddleDerivedTwo()
    {
        cout<<"LastDerived Constructor"<<endl;
    }
    void ComplexFunc()
    {
        MiddleFuncOne();
        MiddleFuncTwo();
        SimpleFunc();
    }
};
```

호출의 대상파악이 불가능하다!

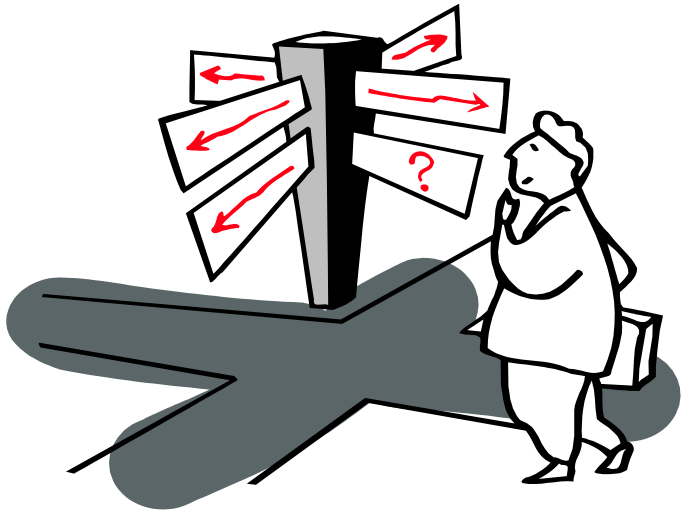


가상상속

```
class MiddleDerivedOne : virtual public Base { . . . . };

class MiddleDerivedTwo : virtual public Base { . . . . };
```



Virtual 상속으로 인해서 공통의 기초 클래스의 멤버를 하나만 포함하게 된다.



Chapter 09가 끝났습니다. 질문 있으신지요?