20191205 실습 11주차

박 건 호

KITRI BoB 8th

5422 RTDCS

devgunho.github.io

virtual

virtual 키워드는 함수에 붙일 수 있다.

이 키워드를 붙인다는 것은 가상 테이블(Virtual Table)을 작성하기 위해서이다.

이를 통해 Overriding이 가능해진다.

다시 말해 상속관계도에서 overriding을 하기 위해서는 가상 함수이여야 한다.

virtual

먼저 객체는 생성되었을 때 자기 자신의 멤버를 가리키고 있다.

여기서 부모 포인터가 자식을 가리킨다면?

가리키는 것이 객체 자체(자식)이기 때문에 원활하게 사용할 수 있을까?

virtual

게임을 구현한다고 가정했을 때,
게임에서 유닛의 종류가 수백가지 있다고 하자.
당연히 각각의 유닛의 공격 방식이나 형태는 모두 다를 것이기 때문에,
'공격' 명령을 내렸을 때 유닛이 취할 행동은 제각각 일 것이다.
그렇다면 수백가지의 각기 다른 함수를 호출해야 할까?

virtual

그보다는 부모가 자식을 가리켜 '자식에서 새롭게 정의된 기능'을 사용할 수 있다면
보다 더 효율적일 것이다.

모든 유닛이 부모로부터 '공격'을 상속받게 하고 부모 타입으로 받은 뒤,
유저가 공격 명령을 내리면 각각의 유닛들이 자신에게 맞는 공격을 할 것이다.

이를 다형성(Polymorphism)이라고 하며,
virtual 키워드는 Overriding을 성립할 수 있게 해주어 다형성을 구현하는
핵심 키워드이다. 여기서 가상 함수 포인터(virtual function pointer)가 생긴다.

virtual

그보다는 부모가 자식을 가리켜 '자식에서 새롭게 정의된 기능'을 사용할 수 있다면
보다 더 효율적일 것이다.

모든 유닛이 부모로부터 '공격'을 상속받게 하고 부모 타입으로 받은 뒤,
유저가 공격 명령을 내리면 각각의 유닛들이 자신에게 맞는 공격을 할 것이다.

이를 다형성(Polymorphism)이라고 하며,
virtual 키워드는 Overriding을 성립할 수 있게 해주어 다형성을 구현하는
핵심 키워드이다. 여기서 가상 함수 포인터(virtual function pointer)가 생긴다.

virtual

내가 원한 결과는 우측과 같은 결과인데 왜 이런 결과가 나온 것일까?

```
int main()
{
    super* superPointer = new super;
    parent* parentPointer = new parent;
    child* childPointer = new child;

    superPointer->print();

    superPointer = parentPointer;
    superPointer->print();

    superPointer = childPointer;
    superPointer->print();

    return 0;
}
```

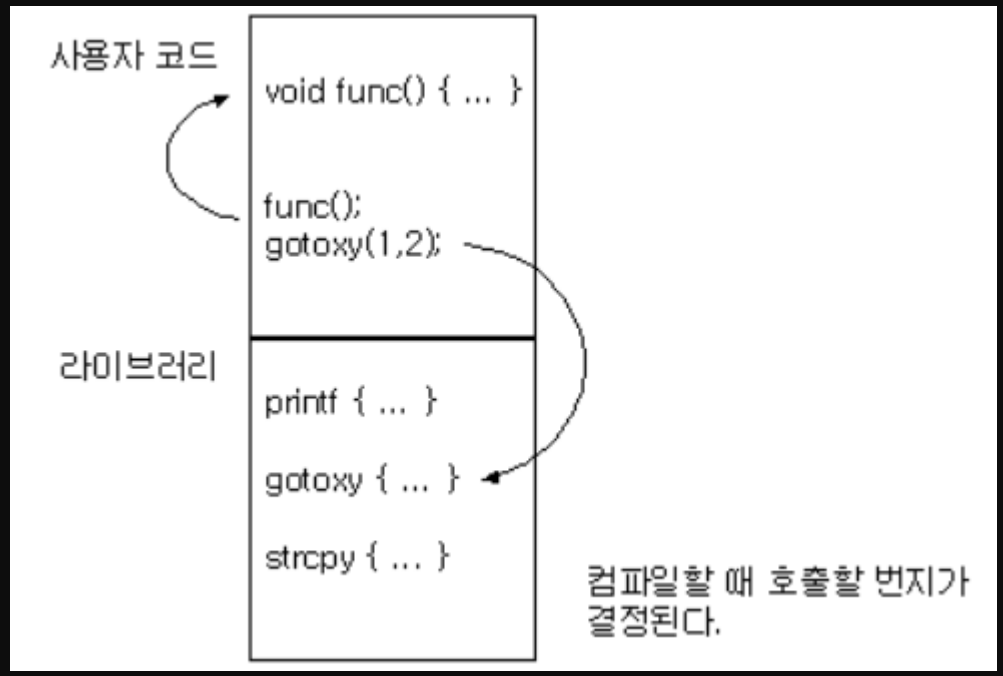
```
Microsoft Visual St
내가 조상이다.
내가 조상이다.
내가 조상이다.

G:₩SSD2_512₩_Main_v
88) exited with coc
Press any key to cl
```

```
Microsoft Visual
내가 조상이다.
내가 부모다.
내가 자식이다.
```

virtual

```cpp
#include <iostream>

using namespace std;

class super
{
public:
    void print()
    {
        cout << "내가 조상이다." << endl;
        printf("%p\n", &super::print);
    }
};

class parent : public super
{
public:
    void print()
    {
        cout << "내가 부모다." << endl;
        printf("%p\n", &parent::print);
    }
};

class child : public parent
{
public:
    void print()
    {
        cout << "내가 자식이다." << endl;
        printf("%p\n", &child::print);
    }
};
```

좌측과 같이 코드를 수정하여 함수의 주소를 확인해보자



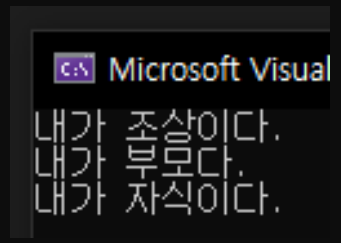
컴파일러는 함수가
어떤 주소에 있는지 알고
있기 때문에
호출문을 이 함수의
주소로 점프하는 코드로
번역하게 된다.

virtual

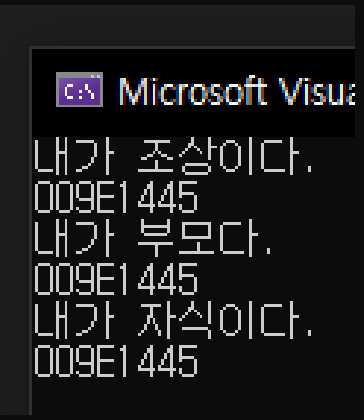
컴파일하는 시점(정확하게는 링크 시점)에 호출 주소가 결정되는 것을
정적 결합(바인딩)(Static Binding) 또는 Early Binding이라고 한다.
여기서 Binding 이란 함수 호출문에 대해 실제 호출될 함수의 번지를 결정하는 것을
의미한다.
일반적으로 함수들은 모두 정적 결합이다.
따라서 앞선 예제에서 super의 print는 처음에 Static Binding이 되어 기록된
주소로, super 입장에서는 print의 주소가 어떤 객체이든 같은 주소로 호출하는 것이다.

virtual

그렇다면 다음과 같이 virtual을 사용하여 사용자가 원하는 대로 함수를 호출 했다면,



주소값들은 어떻게 되어있을까?



가상 함수는 '동적 바인딩'되기 때문에
가상 함수를 소유한 클래스는 가상 함수 테이블을
가리키는 4바이트의 가상 함수 포인터를 가지게 된다.

virtual

여기서 동적 바인딩(Dynamic Binding)이란
동적 시간대(Runtime)에 결합되는 것을 말하는데,
처음부터 함수의 주소를 결정해두고 호출하는 것이 아니라,
호출 시에 함수의 주소를 알려주는 것을 의미한다.

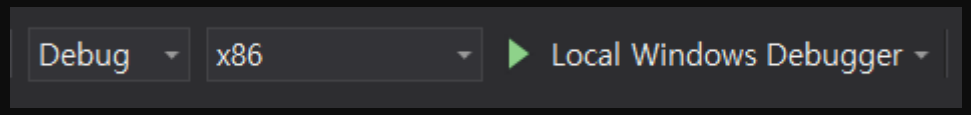
Virtual Destructor

Abstract의 소멸자에는 virtual 키워드가 없다.
이를 상속받은 Foo, Bar 2개의 class가 있는데,
Foo 클래스는 memory leak 문제가 존재하지 않는다.
왜냐하면, Foo가 가지고 있는 데이터는 순전히 HeapManager에 의해서
할당/해제가 되기 때문에 소멸자의 호출여부와는 무관하기 때문이다.
소멸자에서 하는 일이라고는 “~Foo()”를 출력하는 것 뿐이다.
하지만, Bar 클래스는 memory leak 문제가 발생 할 수 있다.
왜냐하면, char[]에 대한 해제는 소멸자가 호출되어야만 발생 할 수 있기 때문이다.

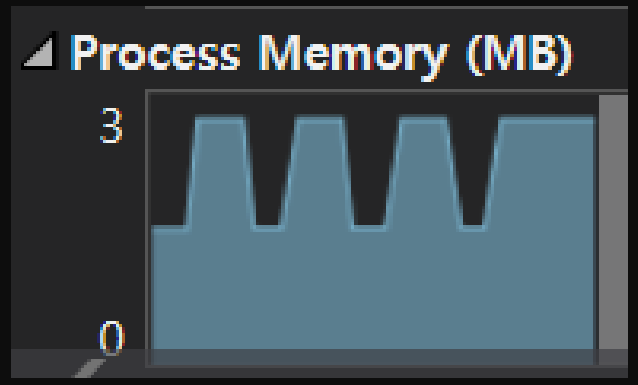
Virtual Destructor

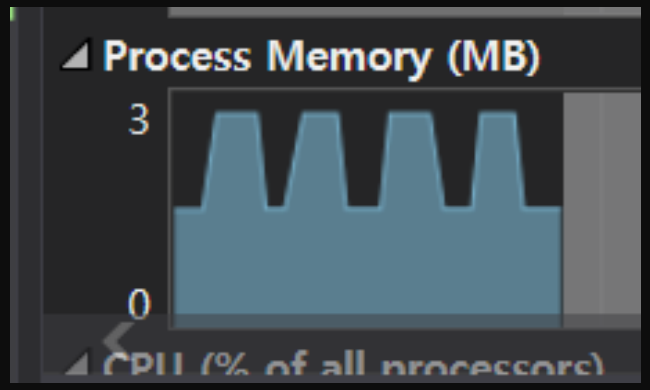
Visual Studio의 Heap Profile을 통해 메모리의 변화를 확인해보자.

```
90      return 0;
91  }
```



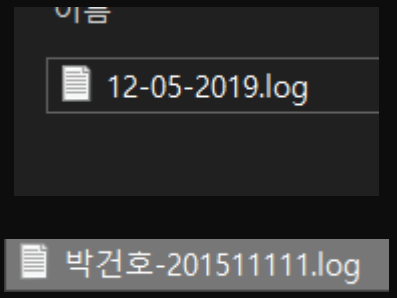
~Foo()의 소멸자 호출과는 관계없이 메모리가 해제되는 2번째 파동을 볼 수 있다.
하지만, ~Bar()의 소멸자가 호출되지 않고 메모리가 해제된다면
생기는 4번째 파동은 볼 수 없다.





기말고사 시험 (12/12 목 09:30~)

시험 시작 전부터 기록하는 프로세스 로그파일을
시험 종료 후에 바탕화면으로 복사하여 '이름-학번.log'
양식에 맞춰 이메일 제목 '이름-학번'으로 첨부파일 전송



Visual Studio 사용금지
크롬으로만 시험 응시 할 것, 구름 IDE 페이지 이외의 Web 접근 허용하지 않음.

문제 수는 10~15문제, 1시간에서 1시간 30분 시험 예정 (추후 재공지)

예상 문제

- 구구단 삼각형 출력하기
- X번째 소수 찾기
- 사용자로부터 정수를 입력 받고 3번째로 큰 수, 3번째로 작은 수 출력하기
- Point 클래스로 주어지는 두 좌표 Swap 구현하기
- 사용자로부터 문자열을 입력 받고, 깊은 복사 생성자를 구현하기
- 사용자로부터 문장을 입력 받고, 가장 길이가 긴 단어 출력하기
- 사용자로부터 문장을 입력 받고, 특정 문자 지우고 출력하기
- Box 클래스의 각 변을 곱하여 출력하는 Operation Overloading을 구현하기
- 주어진 Shape 클래스를 상속받아 Overriding을 통해 각각의 도형에 맞는 출력함수 구현하기

... (Github에 올려져 있는 예제 및 문제 변형)