default 복사 생성자 c++ is\_block\_type\_valid(header->\_block\_use) delete 할때 발생

Default 복사 생성자는 **복사 하려는 객체의 멤버변수와 복사될 대상인 객체의 멤버변수가 그대로 대입되어 포인터의 경우 그대로 주소값만 참조**하게된다. 이것을 **얇은 복사**라 하고 이러한 얇은 복사는 문제의 소지가 있다.  
// 문제를 발생시키는 함수

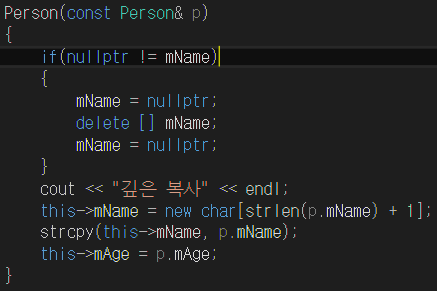
void foo(const Person p) // p가 원본 객체를 가리키고 있기 때문에 foo함수가 끝나면 p가

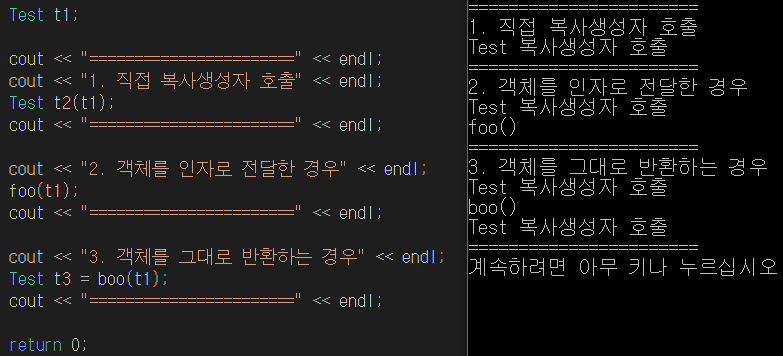
{ // 소멸 되는데 원복 객체를 가리키고 있기 때문에 원본이 소멸된다.

cout << "foo(const Person p)" << endl;

cout << "===========" << endl;

}

- 해결방법은 크게 두 가지다. 한가지는 함수의 매개변수에 객체가 사용될 때는 **const Person&와 같이 레퍼런스 타입으로 받아 사용하는 것이다.** 이렇게하면 **복사생성자를 호출하는 작업을 하지않고 기존 객체를 그대로 사용 하면서 const로 상수화 시켜 함수 내에서 값을 변경할 수 없다.**   
  
- 두번째 방법은 복사생성자를 호출 하더라도 얇은 복사가 아니라 **깊은 복사가 이뤄지도록 Person 클래스에 복사생성자를 직접 작성**해주는 방법  


복사 생성자가 호출되는 시점은 크게 3가지 경우가 있는데, 나머지 경우도 살펴보자.  
  


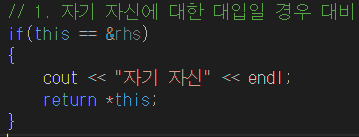
const Test& t <- 에서 &가 붙으면 복사생성자가 호출 되지 않는다. 당연하지 참조자니까

void foo(const Person p) // 인수가 const Person p 이면 복사생성자 호출하고 const Person& p 이면 호출 안함.

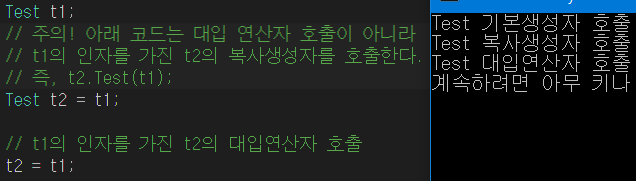
|  |
| --- |
| **대입 연산자** |

: 대입 연산자도 복사 생성자와 마찬가지로 직접 작성하지 않으면 Default로 생성된다. Default 대입 연산자도 각 멤버에 대해 대입만하는 얇은 복사가 진행되며 복사 생성자와 같은 문제가 발생할 수 있기 때문에 필요한 경우 반드시 직접 작성해서 깊은 복사를 진행해야한다.

첫 번째는 **자기 자신을 대입하는 경우에 대한 코드를 작성해두는 것**과 **대입 연산 결과를 this에 대한 참조 값으로 반환하는 관례**이다.  아래 코드를 보자.





단지 대입 '='을 하니까 대입 연산자가 호출된다고 생각했는데 복사 생성자가 호출되었다.

사실 조금만 생각해보면 간단하다. 객체가 한번도 생성된 적이 없는 경우 생성자를 호출해야하므로

동일한 객체로 대입을 하든 뭐를 하든 단지 복사 생성자가 호출된다고 생각하면 된다.

|  |
| --- |
| **상속관계에서 복사 생성자와 대입 연산자 호출** |

:  상속 관계에 있는 경우에도 위에서 언급한 **복사 생성자나 대입 연산자를 작성할 때 빠짐없이 복사해주자**는 규칙을 지키기 위해 자식 객체가 **부모 객체의 복사 생성자나 대입 연산자를 호출해주어 부모 객체의 멤버 변수 값도 모두 초기화 해줘야한다.**

출처: <http://tuyy.tistory.com/60> [Happy Coding!!]