

# 4장 복사생성자와 임시객체

## \* 복사생성자

int a(10) <sup>초깃값(상수)</sup> ⇒ int b(a) <sup>인스턴스 초깃값 원본</sup>  
 생성 (변수, 인스턴스) < 복사생성 >

생성 [ 일반생성 → int a(10)  
 - 복사생성 → int b(a), CMyString strNew(BtrData)

생성자 : ① 기본생성자

② 다중전위 (오버로드) 된 생성자 (매개변수)

③ 변환생성자 (매개변수 1개) → 형 변환자

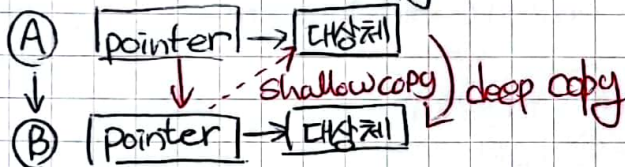
④ 복사생성자.

④+ 이동생성자 (rvalue 참조)

⇒ 이름x 임시객체  
 ↳ 코드로 드러나지x

pB = new int(pA) <sup>deep copy</sup>

복사생성자 필요? : Deep Copy - 원본까지 복사  
 Shallow Copy - 참조, 참조 복사



pA, pB → nData <sup>< shallow ></sup> ⇒ Delete 시  
 원본 1개 삭제  
 ↳ 남은 포인터 1개?  
 ⇒ dangling  
 바늘 x

디폴트 복사 생성자 ⇒ shallow copy  
 ↳ deep copy 실사코드 따로 작성.

복사생성자 형태 : CTest(const CTest &ctest)

↳ call 시점 : CTest b(a);

↳ shallow copy code : this → 변수명 = ctest. 변수명

↳ 멤버변수를 단순대입으로 복사.

↳ if 포인터 ⇒ 디폴트일 때 프로그램 중단 위험 ⇒ 존재하지 않는 메모리 반환.

↳ deep copy code : this → 변수명 = new int(\*ctest. 변수명)

## \* 대입연산자 (연산자 overloading)

: + - \* / ⇒ 함수를 치환 가능 : 연산자 함수 (C++)

단순대입 연산자 (=) l-value = r-value

locator 변수 <sup>overwrite</sup> 변수/상수

⇒ class에 대해서만 연산자 함수 가능.

\* 복사생성자 정의 시 모든 객체는 단순대입 연산자 반드시 재정의

클래스명 & operator = (const 클래스명 &객체)

{ \*포인트변수 = \*객체.포인터 변수  
 return \*this; }

↳ 사용 : b.operator = (a) or b = a



# \* 묵시적 변환 **explicit**

변환생성자: 클래스명 (int nParam) ⇒ 매개변수 1개인 생성자.

Default: CTest a;  
Copy: CTest b(a);  
변환생성자: CTest c(5);

함수명 (상수) ⇒ 변환생성자로 캐치 생성됨.

↳ const CTest& param = 5;    ↳ int    ⇒ 오류 X / 컴파일러가 변환생성자로 캐치 생성.

참조자: int &nData = nData.

explicit ⇒ 묵시적으로 다른 자료형이 클래스의 객체화 되는 것을 방지

## \* 허용되는 변환.

내부적으로 타 자료형과 비슷한 클래스를 자동변환하는 것은 허용.

⇒ operator (int)(void)

```

{
    return *포인터자료;
}
    
```

→ 각종 형변환 자원.  
int로 변환.

↳ 다른 자료형과 높은 호환성을 가지게 됨.

## \* 임시객체와 이동시맨트.

이름없는 임시객체: 변수의 return 등에서 쓰이는 임시객체.

↳ 함수 결과값: 어셈블리 EAX register. 임시결과 저장.    ↳ 메모리 사용에 유의. (Peak - 임시객체) 용량까지.)

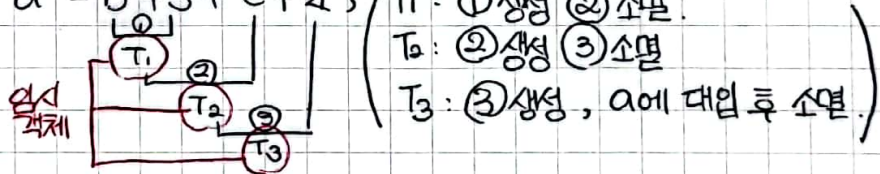
int a;  
a = 3 + 4;    ↳ 동일형식 강제. → 연산.  
7 ⇒ 임시결과.

수(자료) 표현 [ 변수 vs Instance (추상성+)

a = 3 + 4    ↳ r-value.  
변수    상수    ↳ 상수\*

< 숨겨진 임시객체 >  
a = 3 + 4 ..... 7 임시객체.  
↳ int 인스턴스의 개수: 3 (코드 수준)  
↳ " : 4 (연산 후)  
↳ let CmyData a = b + c  
↳ CmyData 인스턴스의 개수: 4개  
↳ 생성과 소멸: 4회 (선언 및 정의 시 3회)

\* Instance life cycle: a = b + 3 + c + 4;



↳ T2 차후 필요시: z = b + 3 + c;    ↳ 메모리 절약 (생성)    ↳ int &  
a = T2 + 4;    ↳ r-value 참조 필요.    ↳ int && → r-value

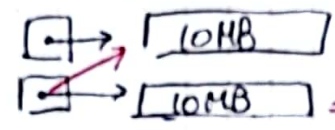
상속 ⇒ 시간의 흐름 이해.

**explicit**

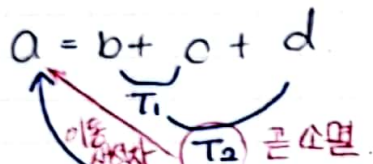
[생성자] ① 디폴트 : `CMyData()` ② 변환생성자 : `CMyData(int)` ③ 복사생성자  
 ↳ 호출시점 : 객체 생성 ↳ 매개변수가 1개인 생성자 ↳ 호출 : `CMyData b(a);`  
 ↳ 함수호출시.

`TestFunc(CMyData(5)); int(10)` ④ 다중의뢰 생성자. ⑤ 이동생성자.  
 ↳ 이름 없는 객체. 임시객체 : 구문이 끝나면 소멸. ↳ 반환형이 클래스일 때  
 ↳ nvalue 참조.

\* 이동생성자 (이동 시멘트) : `CMyData(const CMyData &&hs)`

Deep Copy 필드 a.  순리메모리 20MB  
 ↳ 큰 소멸할 객체 a에 대해. **이동수**

↳ r-value가 임시객체일 때 사용.

$a = b + c + d$   


인스턴스 4개 + 임시객체 2개.

↳ 이동생성자 사용 : 성능향상 기대.

+ 대입연산자 &&.