

- 실체가 없는 (null) 것을 가리키기 위하여 `nullptr`을 이용해 포인터를 null로써 초기화할 수 있다. (기초 C의 NULL은 그냥 0을 의미)

Class 2  
- 객체의 배열도 동적으로 할당 가능하지만, 클래스에 기본 생성자가 존재해야 한다.

- **(++의 Type Casting (다형성을 위한))**: C-style 형 변환은 상수 포인터의 형 변환 등 일반적이지 않은 형 변환도 허용하기에 사용 권유지 않음.

✓ `dynamic_cast<T>(expr)`: 상속 관계의 두 클래스에서 '유도 클래스의 포인터 및 참조형 데이터 → 기초 클래스의 포인터 및 참조형 데이터' 변환에 사용. 동시에 변환되기는 하나, 안전한 변환을 위하여 사용 주제.

✓ `static_cast<T>(expr)`: 상속 관계의 두 클래스에서 '기초 클래스 → 유도 클래스' 변환에 사용. 실제 형 변환이 가능하나 변환된 객체가 가지고 있지 않은 멤버에 대해서 프로그래머가 확실히 체크해야 함.

✓ `const_cast<T>(expr)`: 포인터와 참조자의 `const / volatile` 성향 제거. 함수의 인자 전달 시 `const` 선언으로 인한 형 불일치 발생으로 인한 인자 전달 불가에 유용.

→ `volatile`: 하드웨어 제어 관련 상황에서 컴파일러의 최적화를 막아주는 키워드. 사용 빈도 매우 낮음.

✓ `reinterpret_cast<T>(expr)`: 서로 전혀 상관없는 클래스의 포인터 관련 모든 유형의 형 변환 허용. 원하는 데이터의 비트 수로 접근 등 특정 상황에서는 유용하게 사용.  
하지만 크게 의미를 부여해주는 어려움!

- 참조자 (\*) 는 포인터 (\*) 와 내부적인 구현은 동일하다. 단, 포인터는 '주소(의 변수)'이고, 참조자는 '객체(의 별명)'임을 헷갈리지 말 것.

- 컴파일 타입에 생성되는 코드를 이용하여 프로그래밍하는 방법을 '메타 프로그래밍'이라 하는데, 템플릿이 컴파일 타입에 코드가 만들어지는 점을 이용하여 프로그래밍 하는 방법을 '템플릿 메타 프로그래밍 (TMP)'라 부른다.

✓ 런타임에 계산될 작업을 컴파일 타입에 해결할 수 있어 컴파일 타입은 길지만 실행 속도가 매우 좋아진다. 하지만 복잡하고 구성하기 어려워며, 디버그가 어려워진다는 단점으로 인해 혼입에서 자주 사용되지는 않는다. (참고로 TMP는 동일 완전성을 가지며 모든 코드를 TMP로 구성할 수 있다)

✓ 보통 재귀적인 방법으로 이용하며, 컴파일마다 재귀 가능 횟수가 어느정도 정해져 있어 재귀 횟수가 큰 경우에는 컴파일이 실패할 수도 있다.

✓ TMP를 편하게 구성하기 위하여 `boost::mpl` 라이브러리를 활용할 수 있다.

- 현실에 존재하는 값들과 연관된 연산에서는 항상 '단위'에 주의하여야 한다. TMP 등을 이용하여 단위(Unit) 클래스를 제작하여 사용할 수도 있다.

- `'Divide < N, two > :: result'` (`N, two`에 의존하는 '타입')과 같은 '의존 타입'을 사용하기 위해서는 컴파일러에게 해당 키워드가 확실히 타입임을 알려주기 위하여 앞에 `typename` 키워드를 붙여줘야 한다.