Static\_cast<type>(캐스팅대상)

Const\_cast<type>(캐스팅대상)

Dynamic\_cast<type>(캐스팅대상)

Reinterpret\_cast<type>(캐스팅대상)

모두 대부분 포인터(=객체)에 대한 캐스팅에 쓰인다.

**Static\_cast<type>(캐스팅대상)**

c언어의 묵시적 캐스팅과 동일 기능이지만 가독성이 좋음 ( 컴파일 중 캐스팅 )

서로 다른 타입은 형 변환 불가.

(\*단! Void\* 를 거쳐서 변환 가능. (하지만 권장 X))

상속관계가 존재하는 두 클래스간의 다운캐스팅에 주로 사용.(가상함수 존재 유무와 관계X)

**Const\_cast<type>(캐스팅대상)**

포인터에만 사용 가능

상속 속성, volatile 속성 제거.

Char\* str= “abcd” 이런식으로 가면 “abcd”는 const char\* 가 되는데 이걸

Char\* str= const\_cast<char\*>(“abcd”)

**Reinterpret\_cast<type>(캐스팅대상)**

c타입의 강제 캐스팅과 거의 동일

포인터 타입에만 사용

관련없는 포인터간의 캐스팅도 가능.

상수성 제거에는 사용 X

**Dynamic\_cast<type>(캐스팅대상)**

상속관계의 클래스 사이에 다운캐스팅을 안전하게 하기 위해 실행중에 캐스팅.

가상함수가 반드시 하나 이상 있어야 한다.

왜냐? RTTI 정보를 사용해야해서 가상함수테이블이 필요하다.

가상함수테이블의 최초 4byte에 들어있음

**RTTI**

**구성요소**

Type\_info class

Typeid 연산자

Dynamic\_cast 연산자.

가상함수 테이블에는 virtual 이 선언된 함수에 대한 주소를 가지고 있다.

Class base

{

Virtual func1()

Virtual func2()

}

가상함수 테이블

100 func1주소

200 func2주소

Class Derived :public Base

{

Virtual func1()

Virtual func3()

}

가상함수테이블

300 override 된 func1

200 base의 func2

400 derived func3

Base의 객체가 생기면 메모리구조가

가상함수테이블의 주소

Base 멤버변수

...

가상함수 메모리 구조

Rtti 메모리주소

가상함수주소들

가상함수가 있는 객체의 경우 메모리 접근이 여러 번 되기 때문에 시간이 걸림.

상속받은게 있는 객체의 경우

Rtti 메모리주소 구조

상위 rtti포인터

현재 객체의 클래스 정보

누구를 상속받았는지도 알아올 수 있다.

Const type\_info& tyinfo = typeid(객체포인터)

객체의 클래스 타입 참조가 리턴됨.

그리고 클래스의 크기는 멤버변수의 합인데 가상함수가선언되어 있으면 가상테이블 크기까지 더한다. +4byte