|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **작성자** | **2014132002 구태균** | **팀명** | - |
| **주차** | **1 주차** | **기간** | **2020.01.13~2020.01.19** | **지도교수** | **정 내 훈** (서명) |
| **이번주 한일** | shared\_ptr 상속 클래스 구조 분석 | | | | |

**---------------------------<shared\_ptr\_base.h>---------------------------**

// Helper traits for shared\_ptr of array:

// A pointer type Y\* is said to be compatible with a pointer type T\* when

// either Y\* is convertible to T\* or Y is U[N] and T is U cv [].

// Y\*가 T\*로 변환 가능하거나 Y가 U[N]이고 T가 U cv[] 인 경우

// 포인터 유형 Y\*는 포인터 유형 T\*와 호환 가능

// \_\_sp\_compatible\_with

template<typename \_Yp\_ptr, typename \_Tp\_ptr>

struct \_\_sp\_compatible\_with

: false\_type { };

template<typename \_Yp, typename \_Tp>

struct \_\_sp\_compatible\_with<\_Yp\*, \_Tp\*>

: is\_convertible<\_Yp\*, \_Tp\*>::type { };

template<typename \_Up, size\_t \_Nm>

struct \_\_sp\_compatible\_with<\_Up(\*)[\_Nm], \_Up(\*)[]>

: true\_type { };

template<typename \_Up, size\_t \_Nm>

struct \_\_sp\_compatible\_with<\_Up(\*)[\_Nm], const \_Up(\*)[]>

: true\_type { };

template<typename \_Up, size\_t \_Nm>

struct \_\_sp\_compatible\_with<\_Up(\*)[\_Nm], volatile \_Up(\*)[]>

: true\_type { };

template<typename \_Up, size\_t \_Nm>

struct \_\_sp\_compatible\_with<\_Up(\*)[\_Nm], const volatile \_Up(\*)[]>

: true\_type { };

// \_\_sp\_is\_constructible

// 유효하지 않은 유형 Y[N]을 형성하지 않고 Y(\*)[N]에서 U(\*)[N]으로의 변환 테스트.

template<typename \_Up, size\_t \_Nm, typename \_Yp, typename = void>

struct \_\_sp\_is\_constructible\_arrN

: false\_type { };

template<typename \_Up, size\_t \_Nm, typename \_Yp>

struct \_\_sp\_is\_constructible\_arrN<\_Up, \_Nm, \_Yp, \_\_void\_t<\_Yp[\_Nm]>>

: is\_convertible<\_Yp(\*)[\_Nm], \_Up(\*)[\_Nm]>::type { };

// 유효하지 않은 유형 Y[]를 형성하지 않고 Y(\*)[]에서 U(\*)[]로의 변환 테스트

template<typename \_Up, typename \_Yp, typename = void>

struct \_\_sp\_is\_constructible\_arr

: false\_type { };

template<typename \_Up, typename \_Yp>

struct \_\_sp\_is\_constructible\_arr<\_Up, \_Yp, \_\_void\_t<\_Yp[]>>

: is\_convertible<\_Yp(\*)[], \_Up(\*)[]>::type { };

// shared\_ptr <T>가 Y\*에서 구성 될 수 있는지 확인하는 특성

template<typename \_Tp, typename \_Yp>

struct \_\_sp\_is\_constructible;

// T가 U[N] 일 때, Y(\*)[N]은 T\*로 변환 가능해야한다.

template<typename \_Up, size\_t \_Nm, typename \_Yp>

struct \_\_sp\_is\_constructible<\_Up[\_Nm], \_Yp>

: \_\_sp\_is\_constructible\_arrN<\_Up, \_Nm, \_Yp>::type { };

// T가 U[] 일 때, Y(\*)[]은 T\*로 변환 가능해야한다.

template<typename \_Up, typename \_Yp>

struct \_\_sp\_is\_constructible<\_Up[], \_Yp>

: \_\_sp\_is\_constructible\_arr<\_Up, \_Yp>::type { };

// 그렇지 않으면 Y \*는 T \*로 변환 할 수 있습니다.

template<typename \_Tp, typename \_Yp>

struct \_\_sp\_is\_constructible

: is\_convertible<\_Yp\*, \_Tp\*>::type { };

**[\_\_shared\_ptr]**

template<typename \_Tp, \_Lock\_policy \_Lp>

class \_\_shared\_ptr : public \_\_shared\_ptr\_access<\_Tp, \_Lp>

{

public:

using element\_type = typename remove\_extent<\_Tp>::type;

// (using a = b) == (typedef b a) template에서 사용하기 편리.

// remove\_extent<T>::type : 원소의 타입을 리턴 (배열이여도 원래 원소타입 리턴, 템플릿::멤버이름)

// \_Tp로 배열이 들어와도 원래의 타입으로 element\_type

private:

template<typename \_Yp>

using \_SafeConv = typename enable\_if<[\_\_sp\_is\_constructible](#sp_is_constructible)<\_Tp, \_Yp>::value>::type;

// \_Yp\* 유형의 포인터 구성가능 여부

// enable\_if : 템플릿 인자를 받아 참일 경우에 type aliasing (컴파일 에러)

// 설명 <http://egloos.zum.com/sweeper/v/3059985>

// \_\_sp\_is\_constructible : 구성 가능한지 식별하는 템플릿 return true or false.

template<typename \_Yp, typename \_Res = void>

using \_Compatible = typename enable\_if<[\_\_sp\_compatible\_with](#sp_compatible_with)<\_Yp\*, \_Tp\*>::value, \_Res>::type;

// shared\_ptr 구성가능 여부

template<typename \_Yp>

using \_Assignable = \_Compatible<\_Yp, \_\_shared\_ptr&>;

// shared\_ptr 할당가능 여부

template<typename \_Yp, typename \_Del, typename \_Res = void,

typename \_Ptr = typename unique\_ptr<\_Yp, \_Del>::pointer>

using \_UniqCompatible = typename enable\_if<\_\_and\_<\_\_sp\_compatible\_with<\_Yp\*, \_Tp\*>,

is\_convertible<\_Ptr, element\_type\*>>::value, \_Res>::type;

// unique\_ptr 생성 관련

template<typename \_Yp, typename \_Del>

using \_UniqAssignable = \_UniqCompatible<\_Yp, \_Del, \_\_shared\_ptr&>;

// unique\_ptr 할당 관련

…

protected:

friend class \_\_weak\_ptr<\_Tp, \_Lp>;

private:

template<typename \_Tp1, \_Lock\_policy \_Lp1>

friend class \_\_shared\_ptr;

template<typename \_Tp1, \_Lock\_policy \_Lp1>

friend class \_\_weak\_ptr;

element\_type\* \_M\_ptr; // object pointer

\_\_shared\_count<\_Lp> \_M\_refcount; // reference counter

};

**[\_\_shared\_count]**

template<\_Lock\_policy \_Lp>

class \_\_shared\_count

{

template<typename \_Tp>

struct \_\_not\_alloc\_shared\_tag { using type = void; };

template<typename \_Tp>

struct \_\_not\_alloc\_shared\_tag<\_Sp\_alloc\_shared\_tag<\_Tp>> { };

public:

…

private:

friend class \_\_weak\_count<\_Lp>;

\_Sp\_counted\_base<\_Lp>\* \_M\_pi;

};

**[\_\_Sp\_counted\_base]**

template<\_Lock\_policy \_Lp = \_\_default\_lock\_policy>

class \_Sp\_counted\_base : public \_Mutex\_base<\_Lp>

// \_Mutex\_base 설명 <https://gcc.gnu.org/onlinedocs/libstdc++/libstdc++-html-USERS-4.3/a00768.html>

{

public:

…

private:

\_Atomic\_word \_M\_use\_count; // shaed\_count

\_Atomic\_word \_M\_weak\_count; // weak\_count

// \_Atomic\_word : 원자 연산을 지원하는 부호있는 정수 유형

// 설명 <https://gcc.gnu.org/onlinedocs/libstdc++/manual/ext_concurrency.html>

// 코드 <https://github.com/gcc-mirror/gcc/blob/master/libstdc%2B%2B-v3/config/cpu/m68k/atomicity.h>

};

**---------------------------<shared\_ptr.h>---------------------------**

**[shared\_ptr]**

template<typename \_Tp>

class shared\_ptr : public \_\_shared\_ptr<\_Tp>

{

template<typename... \_Args>

using \_Constructible = typename enable\_if<is\_constructible<\_\_shared\_ptr<\_Tp>, Args...>::value>::type;

// is\_constructible : 구성 가능한지 식별하는 템플릿 return true or false.

// 설명 <http://www.cplusplus.com/reference/type_traits/is_constructible/>

template<typename \_Arg>

using \_Assignable = typename enable\_if<is\_assignable<\_\_shared\_ptr<\_Tp>&, \_Arg>::value, shared\_ptr&>::type;

// is\_assignable : 뒤의 인자 형식값을 앞의 인자 형식에 할당할 수 있는지 여부

// 설명 <https://docs.microsoft.com/ko-kr/cpp/standard-library/is-assignable-class?view=vs-2019>

public:

using element\_type = typename \_\_shared\_ptr<\_Tp>::element\_type;

private:

friend class weak\_ptr<\_Tp>;

};

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | **해결 방안** |  |
| **다음 주차** | **2 주차** | **다음 기간** | **2020.01.20~2020.01.26** |
| **다음주 할 일** | class shared\_ptr 멤버함수 분석 | | |
| **지도교수**  **Comment** |  | | |