|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **작성자** | **2014132002 구태균** | **팀명** | - |
| **주차** | **14 주차** | **기간** | **2020.04.13 ~ 2020.04.19** | **지도교수** | **정 내 훈** (서명) |
| **이번주 한일** | * Enable\_shared\_from\_this 분석 | | | | |

**Shared\_ptr<T> 생성 1**

std::shared\_ptr<lfspTest> stdsptr = std::make\_shared<lfspTest>(10);

template<typename Tp, typename... Args>

shared\_ptr<Tp> make\_shared(Args&&... \_Args)

{

// Create ptr

Tp\* new\_Tp = new Tp(std::forward<Args>(\_Args)...);

// Create ctr\_block

// init : ptr, use\_count, weak\_count

ctr\_block<Tp>\* new\_ctr = new ctr\_block<Tp>(new\_Tp);

// Create shared\_ptr

shared\_ptr<Tp> \_Ret;

\_Ret.\_Set\_ptr\_rep\_and\_enable\_shared(new\_Tp, new\_ctr);

return (\_Ret);

}

template<typename Tp, typename = void>

struct \_Can\_enable\_shared : std::false\_type

{ // detect unambiguous and accessible inheritance from enable\_shared\_from\_this

};

template<typename Tp>

struct \_Can\_enable\_shared<Tp, std::void\_t<typename Tp::\_Esft\_type>>

: std::is\_convertible<std::remove\_cv\_t<Tp> \*, typename Tp::\_Esft\_type \*>::type

{ // is\_convertible is necessary to verify unambiguous inheritance

};

template<typename Tp>

void \_Enable\_shared\_from\_this1(const shared\_ptr<Tp>& \_This, Tp\* \_Ptr, std::true\_type)

{ // enable shared\_from\_this

// 1. Dynamic\_cast 이용

/\*enable\_shared\_from\_this<Tp>\* enable\_shared\_from\_this\_base

= dynamic\_cast<enable\_shared\_from\_this<Tp>\*>(\_Ptr);

if(enable\_shared\_from\_this\_base)

enable\_shared\_from\_this\_base->Wptr = \_This;\*/

// 2. Reinterpret\_cast 이용

enable\_shared\_from\_this<Tp>\* enable\_shared\_from\_this\_base

= reinterpret\_cast<enable\_shared\_from\_this<Tp>\*>(\_Ptr);

enable\_shared\_from\_this\_base->Wptr = \_This;

}

template<typename Tp>

void \_Enable\_shared\_from\_this1(const shared\_ptr<Tp>&, Tp\*, std::false\_type)

{ // don't enable shared\_from\_this

}

template<typename Tp>

void \_Enable\_shared\_from\_this(const shared\_ptr<Tp>& \_This, Tp \* \_Ptr)

{ // possibly enable shared\_from\_this

// Tp가 enable\_shared\_로 상속 받았는지 구분해야함.

\_Enable\_shared\_from\_this1(\_This, \_Ptr,

std::bool\_constant<std::conjunction\_v<\_Can\_enable\_shared<Tp>>>{});

}

bool\_constant< conjunction\_v< negation< is\_array< \_Other > >, negation< is\_volatile< \_Yty > >, \_Can\_enable\_shared< \_Yty > > >{}); **구문으로 enable\_shared\_from\_this를 구분해야함**

**상속을 받는 경우,**

template<class \_Other,class \_Yty>

void \_Enable\_shared\_from\_this1(const shared\_ptr<\_Other>& \_This, \_Yty \* \_Ptr, true\_type)

{ // enable shared\_from\_this

if (\_Ptr && \_Ptr->\_Wptr.expired())

\_Ptr->\_Wptr = shared\_ptr<\_Yty>(\_This, const\_cast<\_Yty\*>(\_Ptr));

// enable\_shared\_fron\_this Wptr 초기화

}

**상속을 안받는 경우,**

template<class \_Other, class \_Yty>

void \_Enable\_shared\_from\_this1(const shared\_ptr<\_Other>&, \_Yty \*, false\_type)

{ // don't enable shared\_from\_this

}

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | **해결 방안** |  |
| **다음 주차** | **16 주차** | **다음 기간** | **2020.04.27 ~ 2020.05.03** |
| **다음주 할 일** |  | | |
| **지도교수**  **Comment** |  | | |