**2023 12 04**

**std::unique\_ptr**

**class** Myclass**;**

**int** main**()**

**{**

**using** **namespace** std**;**

unique\_ptr**<**Myclass**>;** ***// incomplete type olarak kullanabiliriz***

**}**

***/\****

***uptr.release(): sarmaladığı adresi döndürür mülkiyeti bırakır (dtor çağırmaz)***

***uptr.get(): sarmaladığı adresi döndürür***

***\*/***

**int** main**()**

**{**

**using** **namespace** std**;**

**auto** uptr **=** make\_unique**<**string**>(**"alican korkmaz"**);**

***// ikiside aynı***

cout **<<** uptr **<<** "\n"**;**

cout **<<** uptr**.**get**()** **<<** "\n"**;**

***// tanımsız davranış***

**auto** p **=** uptr**.**get**();**

unique\_ptr**<**string**>** upx**(**p**);**

***// geçerli***

**auto** p **=** uptr**.**release**();**

unique\_ptr**<**string**>** upx**(**p**);**

**}**

**template<typename** T**>**

**struct** DefaultDelete

**{**

**void** **operator()(**T**\*** p**)**

**{**

**delete** p**;**

**}**

**};**

**template<typename** T**,** **typename** D **=** std**::**default\_delete**<**T**>>**

**class** UniquePtr

**{**

**public:**

**~**UniquePtr**()**

**{**

**if** **(**mp**)**

**{**

D**{}(**mp**);**

**}**

**}**

**private:**

T**\*** mp**;**

**}**

#include <iomanip>

**struct** SDeleter

**{**

**void** **operator()(**std**::**string**\*** p**)const** **noexcept**

**{**

std**::**cout **<<** std**::**quoted**(\***p**)** **<<** " delete ediliyor\n"**;**

**delete** p**;**

**}**

**}**

**void** fdeleter**(**std**::**string **\***p**)**

**{**

std**::**cout **<<** std**::**quoted**(\***p**)** **<<** " delete ediliyor\n"**;**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

**{**

***//1***

unique\_ptr**<**string**,** SDeleter**>** uptr**{new** string**{**"tamer dundar"**}};**

***//2***

unique\_ptr**<**string**,** **decltype(&**fdeleter**)>** uptr**{new** string**{**"tamer dundar"**}};**

***//3***

**auto** fdel **=** **[](**string **\***p**)**

**{**

std**::**cout **<<** std**::**quoted**(\***p**)** **<<** " delete ediliyor\n"**;**

**delete** p**;**

**};**

unique\_ptr**<**string**,** **decltype(**fdel**)>** uptr**{new** string**{**"tamer dundar"**}};**

***//4***

**decltype([])**string **\***p**)**

**{**

std**::**cout **<<** std**::**quoted**(\***p**)** **<<** " delete ediliyor\n"**;**

**delete** p**;**

**};**

unique\_ptr**<**string**,** **decltype([])**string **\***p**)**

**{**

std**::**cout **<<** std**::**quoted**(\***p**)** **<<** " delete ediliyor\n"**;**

**delete** p**;**

**};>** uptr**{new** string**{**"tamer dundar"**}};**

**}**

std**::**cout **<<** "main devam ediyor\n"

***/\****

***tamer dundar delete ediliyor\n***

***main devam ediyor***

***\*/***

**}**

Neden kendi deleter'imiz kullanmalıyız:

* Delete etmek dışında kaynağı başka bir şekilde sonlandırmak istersek.
* Delete ederken yanında başka işlemler yapmak istersek.

#include<cstdio>

**int** main**()**

**{**

**auto** fdel **=** **[](**std**::**FILE**\*** f**)**

**{**

std**::**cout **<<** "file is being closed!\n"**;**

std**::**fclose**(**f**);**

**};**

std**::**unique\_ptr**<**std**::**FILE**,** **decltype(**fdel**)>** uptr**{**fopen**(**"melike.txt"**),** "w" **};**

fprintf**(**uptr**.**get**(),** "Emre Bahtiyar"**);**

**}**

***// C-style Ctor Dtor***

***// unique\_ptr ile herhangi bir kaynağı sarmalayabiliriz.***

**struct** Data

**{**

**};**

Data **\*** createData**(void);**

**void** do\_something**(**Data **\*);**

**void** do\_this**(**Data **\*);**

**void** do\_that**(**Data **\*);**

**void** destroyData**(**Data **\*);**

**int** main**()**

**{**

**auto** fdel **=** **[](**Data **\***p**)**

**{**

destroyData**(**p**);**

**};**

unique\_ptr**<**Data**,** **decltype(**fdel**)>** uptr**(**createData**());**

do\_something**(**uptr**.**get**());**

do\_this**(**uptr**.**get**());**

do\_that**(**uptr**.**get**());**

destroyData**(**uptr**.**get**());**

**}**

**struct** Nec

**{**

Nec**()**

**{**

std**::**cout **<<** "Nec default ctor this : " **<<** **this** **<<** "\n"**;**

**}**

**~**Nec**()**

**{**

std**::**cout **<<** "Nec default ctor this : " **<<** **this** **<<** "\n"**;**

**}**

**char** buf**[**256**]{};**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

***// tanımsız davranış 4 defa ctor çağrılır ama bi kere dtor çağrılır***

unique\_ptr**<**Nec**>** uptr**(new** Nec**[**4**]);**

***// 4 kez ctor çağrılır 4 kez dtor çağrılır***

**auto** fd **=** **[](**Nec **\***p**)** **{delete** **[]**p**;};**

unique\_ptr**<**Nec**,** **decltype(**fd**)>** uptr**(new** Nec**[**4**]);**

***// unique\_ptr için partial spec for array pointer***

unique\_ptr**<**Nec**[]>** uptr**(new** Nec**[**4**]);**

**auto** up **=** make\_unique**<**Nec**[]>(**5**);**

**}**

**int** main**()**

**{**

Date **\***p **=** **new** Date**{**3**,** 12**,** 1872**};**

**{**

unique\_ptr**<**Date**>** upx**(**p**);**

**}**

***// tanımsız davranış***

unique\_ptr**<**Date**>** upy**(**p**);** ***// p dangling pointer oldu***

**}**

***// unique\_ptr -- container'da tutma***

**int** main**()**

**{**

**using** **namespace** std**;**

vector**<**std**::**unique\_ptr**<**Date**>>** dvec**;**

dvec**.**reserve**(**10**);**

**for** **(auto** i **=** 1**;** i **<** **=** 10**;** **++**i**)**

**{**

dvec**.**push\_back**(new** Date **{**i **,**i 2000 **+**i**});**

**}**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

vector**<**unique\_ptr**<**Date**>>** dvec**;**

**auto** uptr **=** make\_unique**<**Date**>(**1**,** 5**,** 1986**);**

dvec**.**push\_back**(**move**(**uptr**));**

dvec**.**push\_back**(**make\_unique**<**Date**>(**1**,** 5**,** 1986**));**

dvec**.**emplace\_back**(new** Date**{**5**,** 6**,** 1965**});**

**}**

**void** fsink**(**std**::**unique\_ptr**<**Date**>** uptr**)**

**{**

std**::**cout **<<** **\***uptr **<<** '\n'**;**

***// Date sınıfı bu scopeta yok olur***

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

fsink**(**make\_unique**<**Date**>(**3**,** 6**,** 1987**));**

cout **<<** "main devam ediliyor\n"**;**

**}**

***// pass-through***

std**::**unqiue\_tr**<**Date**>** pass\_through**(**std**::**unique\_ptr**<**Date**>** uptr**)**

**{**

std**::**cout **<<** **\***uptr **<<** '\n'**;**

**return** uptr**;**

**}**

**int** main**()**

**{**

**using** **namespace** std**;**

**auto** up **=** pass\_through(make\_unique**<**Date**>(**3**,** 6**,** 1987**));**

***// uptr main scope sonunda sonlanır***

cout **<<** "main devam ediyor\n"**;**

**}**

**std::shared\_ptr**

***// std::shared\_ptr -- bir nesnenin birden fazla sahibi olabilir***

**int** main**()**

**{**

**using** **namespace** std**;**

std**::**cout **<<** "sizeof(unique\_ptr<string>)" **=** **<<** **sizeof(**unique\_ptr**<**string**>)** **<<** "\n"**;** ***// 4***

std**::**cout **<<** "sizeof(shared\_ptr<string>)" **=** **<<** **sizeof(**shared\_ptr**<**string**>)** **<<** "\n"**;** ***// 8***

**}**

**template<typename** T**>**

**class** SharedPtr

**{**

**};**

**int** main**()**

**{**

**using** **namespace** std**;**

shared\_ptr**<**Date**>** sp1**(new** Date**{** 1**,** 2**,** 1998**});**

**{**

**auto** sp2 **=** sp1**;**

***// sp2 burda biter ama dtor çağrılmaz***

**}**

std**::**cout **<<** "main devam ediyor\n"**;**

***// sp1 burda sonlanır ve dtor çağrılır***

**}**

***// use\_count()***

**int** main**()**

**{**

**using** **namespace** std**;**

shared\_ptr**<**Date**>** sp1**(new** Date**{** 1**,** 2**,** 1998**});**

**auto** sp2 **=** sp1**;**

***//use count = 2***

cout **<<** "use count = " sp1**.**use\_count**()** **<<** "\n"**;**

cout **<<** "use count = " sp2**.**use\_count**()** **<<** "\n"**;**

**{**

**auto** sp3 **=** sp2**;**

cout **<<** "use count = " sp3**.**use\_count**()** **<<** "\n"**;** ***// use count = 3***

**}**

cout **<<** "use count = " sp2**.**use\_count**()** **<<** "\n"**;** ***// use count = 2***

**}**

**void\*** **operator** **new(**std**::size\_t** n**)**

**{**

std**::**cout **<<** "operator new called n = " **<<** n **<<** "\n"**;**

**void\*** vp **=** std**::**malloc**(**n**);**

**if** **(!**vp**)**

**{**

**throw** std**::**bad\_alloc**{};**

**}**

std**::**cout **<<** "the adress of the allocated block is: " **<<** vp **<<**"\n"**;**

**return** vp**;**

**}**

**struct** Nec

**{**

**char** buf**[**512**]{};**

**};**

**void** foo**()**

**{**

std**::**cout **<<** "foo cagrildi\n"**;**

**auto** pnec **=** **new** Nec**;**

std**::**shared\_ptr**<**Nec**>** sptr**(**pnec**);**

***// burda önce Nec için blok açılıyor***

***//sonra shared\_ptr'in kontrol bloğu için yer açılıyor***

**}**

**void** bar**()**

**{**

std**::**cout **<<** "foo cagrildi\n"**;**

**auto** pnec **=** **new** Nec**;**

std**::**shared\_ptr**<**Nec**>** sptr**(**pnec**);**

***// burda Nec ve kontrol bloğu için ayrılan yer aynı anda açılıyor***

***// derleyici optimazyon yapıyor***

**}**

**int** main**()**

**{**

foo**();**

bar**();**

**}**

***// type erasure -- shared\_ptr***

**class** Myclass

**{**

**public:**

**~**Myclass**()**

**{**

std**::**cout **<<** "Myclass dtor\n"

**}**

**};**

**struct** MyclassDelete

**{**

**void** **operator()(**Myclass**\*** p**)const**

**{**

std**::**cout **<<** "the object at the adress of " **<<** p **<<** "is being deleted\n"**;**

**delete** p**;**

**}**

**};**

**int** main**()**

**{**

**using** **namespace** std**;**

**{**

shared\_ptr**<**Myclass**>** sptr**(new** Myclass**,** MyclassDeleter**{});**

**}**

std**::**cout **<<** "main devam ediyor\n"**;**

**}**

***// unique\_ptr to shared\_ptr***

**int** main**()**

**{**

**using** **namespace** std**;**

**auto** uptr **=** make\_unique**<**Date**>(**1**,** 1**,** 2024**);**

shared\_ptr**<**Date**>** sptr**(**move**(**uptr**));**

***/\****

***std::shared\_ptr kontrol bloğu ne zaman oluşur:***

***- eğer default dtor ile oluşturulduysa kontrol bloğu oluşmaz***

***- eğer 2. ya da daha fazla shared\_ptr oluşturulduysa kontrol bloğu oluşmaz.***

***- unqiue\_ptr'den shared\_ptr dönüştürüyorsak kontrol bloğu oluşur***

***- 1. kez shared\_ptr oluşturuyorsak kontrol bloğu oluşur.***

***\*/***

**}**

***// shared\_ptr fonksiyonları***

**int** main**()**

**{**

**using** **namespace** std**;**

shared\_ptr**<**Date**>** sp1**(new** Date**{**1**,** 1**,** 2024**});**

**auto** sp2 **=** sp1**;**

**auto** sp3 **=** sp2**;**

***// üçünde aynı pointer sarmalar yani get fonksiyonları aynı adresi döndürür***

cout **<<** sp1**.**get**()** **<<** "\n"**;**

cout **<<** sp2**.**get**()** **<<** "\n"**;**

cout **<<** sp3**.**get**()** **<<** "\n"**;**

**if** **(**sp**)** ***// boş mu dolu mu operator bool ile yapılabilir***

sp1**.**reset**();** ***// mülkiyeti bırakıyor unique\_ptr'deki release gibi***

**}**

**void** foo**(**std**::**unqiue\_ptr**<**std**::**string**>)**

**{**

***// reference sayıcı artıcak sonra fonksiyon sonlanınca azalcak***

**}**

**void** bar**(**std**::**unique\_ptr**<**std**::**string**>&** r**)**

**{**

***// reference sayıcı artmıcak***

***// eğer shared\_ptr değerini değiştirmiceksek, reset yapmıcaksak***

***// fonksiyon parametresini reference yapmaya gerek yok***

**}**