**2023 12 01**

**Dinamik Ömürlü Nesneler**

* new ifadesiyle oluşturduğumuz bellek alanını free edemeyiz.
* malloc ile oluşturduğumuz bellek alanını delete edemeyiz.

**class** Nec

**{**

**public:**

Nec**();**

Nec**(int);**

Nec**(int,** **int);**

Nec**(const** **char\*);**

**};**

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

**{**

Nec **\***p1 **=** **new** Nec**;**

Nec **\***p2 **=** **new** Nec**(**234**);**

Nec **\***p3 **=** **new** Nec**(**234**,** 21**);**

**auto** p **=** **new** **const** Nec**(**243**);** ***// const Nec\****

***// array new***

Nec **\***p **=** **new** Nec**[**10**];** ***// 5 tane Nec\* oluşacak***

**delete[]** p**;** ***// array delete***

**}**

**placement new**

***//new (x, y, z)Myclass***

#include <new>

**void\*** **operator** **new(size\_t,** **void\*** ptr**)**

**{**

**return** ptr**;**

**}**

**class** Nec

**{**

**public:**

Nec**(){**

std**::**cout **<<** "CTOR: " **<<** **this** **<<** "\n" **;**

**}**

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

std**::**cout **<<** "DTOR: " **<<** **this** **<<** "\n" **;**

**}**

**private:**

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

**};**

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

**{**

**unsigned** **char** buffer**[sizeof(**Nec**)];**

std**::**cout **<<** "buffer dizisinin adresi : " **<<** **static\_cast<void\*>**

***// buffer ile aynı adreste Nec oluşturuz.***

Nec **\***p **=** **new(**buffer**)**Nec**;** ***// void\* operator new(size\_t, void\* ptr)***

***// tanımsız davranış çünkü bellek alanı operator new ile elde edilmedi***

**delete** p**;**

p**->~**Ñec**()** ***// delete yerine dtor çağırıcaz***

**}**

**placement new -- emplace\_back**

***// placement new -- emplace\_back***

**template<typename** **...**Args**>**

**void** vector**<**T**>::**emplace\_back**(**Args**&&** **...**args**)**

**{**

**new(**adres**)**T**(**std**::**forward**<**Args**>(**args**)...);**

**}**

/\*

void\* operator new( std::size\_t count, const std::nothrow\_t& tag);

new(nothrow)

başarısız olursa exceptions throw etmez nullptr döndürür

\*/

**class** Myclass**;**

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

**{**

**auto** p **=** **new(**nothrow**)**Myclass**;** ***// exception throw etmez***

**if** **(!**p**)** ***// başarısız olduysa nullptr döndürür***

**{**

**}**

**}**

**new ifadesini neden doğrudan kullanmamalıyız?**

1) dinamik ömürlü, otomatik ömürlü yada statik ömürlü mü belirsiz

2) nullptr değerinden mi?

3) delete etmeli miyim?

4) hangi delete ifadesini kullanmalıyım?

5) sadece delete yeterli mi?

**smart pointer**

* unique\_ptr : bir kaynağın tek sahibi var
* shared\_ptr : bir nesneyi gösteren birden fazla pointer olabilir
  + - weak\_ptr

**std::unique\_ptr**

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

**struct** DefaultDelete

**{**

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

**{**

**delete** p**;**

**}**

**};**

***// D ==> deleter parametre***

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

**class** UniquePtr

**{**

**public:**

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

**{**

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

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

**}**

**private:**

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

**}**

#include <memory>

#include "date.h"

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

**{**

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

unique\_ptr**<**Date**>** up**;**

**if** **(**up**)**

std**::**cout **<<** "dolu\n"**;**

**else**

std**::**cout **<<** "bos\n"**;**

**if** **(**up **==** **nullptr)**

**if** **(**up **!=** **nullptr)**

**if** **(**up**.**get**()** **==** **nullptr)**

***// eğer unique\_ptr boşsa tanımsız davranış***

cout **<<** **\***up **<<** "\n"**;**

cout **<<** up**->**month\_day**()** **<<** "\n;

**}**

**class** Myclass

**{**

**public:**

Myclass**()**

**{**

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

**}**

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

**{**

std**::**cout **<<** "Myclass dtor this = " **<<** **this** **<<** "\n"**;**

**}**

**};**

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

**{**

std**::**cout **<<** "main basladi\n"**;**

**{**

***// unique\_ptr explicit oldugu için hatalı***

***// std::unique\_ptr<Myclass> uptr = new Myclass;***

std**::**unique\_ptr**<**Myclass**>** uptr**{new** Myclass**};**

***// scope sonunda uptr'in ömürü biter***

**}**

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

**}**

**template** **<typename** T**,** **typename** **...**Args**>**

std**::**unique\_ptr**<**T**>** MakeUnique**(**Args**&&** **...**args**)**

**{**

**return** std**::**unique\_ptr**<**T**>(new** T**(**std**::**forward**<**Args**>(**args**)...));**

**}**

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

**{**

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

**auto** up **=** make\_unique**<**Date**>(**3**,** 5**,** 1898**);**

**}**

**void** foo**(**std**::**unique\_ptr**<**std**::**string**>** up**);**

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

**{**

***// ctor explicit olduğu için syntax hatası***

foo**(new** std**::**string**{**"necati"**});** ***// syntax hatası***

unique\_ptr**<**Date**>** x**{new** Date**};**

**auto** y **=** x**;** ***// syntax hatası çünkü copy ctor deleted***

**}**

**class** Tamer **{**

**public:**

***/\****

***unique\_ptr'in copy ctor ve assignment delete olduğu için Tamer sınıfının***

***copy ctor ve assignment'i delete eder derleyici.***

***\*/***

**private:**

std**::**unique\_ptr**<**Date**>** uptr**;**

**}**

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

**{**

Tamer tx**;**

Tamer ty**;**

tx **=** ty**;** ***// sytnax hatası***

tx **=** std**::**move**(**ty**);** ***// geçerli***

**}**

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

**{**

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

**auto** upx **=** make\_unique**<**Date**>(**2**,** 5**,** 1982**);**

**auto** upy **=** make\_unique**<**Date**>(**2**,** 5**,** 1982**);**

***// upy'in dtor çağrılır move yapınca***

upx **=** move**(**upy**);**

**}**

std**::**unique\_ptr**<int>** foo**()**

**{**

**auto** up **=** std**::**make\_unique**<int>(**24**);**

**return** up**;** ***// move only class'ları döndürebiliriz***

**}**

*///*

**void** foo**(**std**::**unique\_ptr**<int>)**

**{**

**}**

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

**{**

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

**auto** up **=** make\_unique**<int>(**345**);**

***// foo(up); // legal değil --copy ctor deleted***

foo**(**move**(**up**));** ***// legal***

foo**(**make\_unique**<int>(**7123**));** ***// legal***

foo**(**unique\_ptr**<int>(new** **int));** ***// legal --temp obje***

***// R value olanların hepsini foo'ya arguman olarak verebiliriz.***

**}**

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

**{**

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

**auto** **\***p **=** **new** Date**(**3**,** 6**,** 1923**);**

***// tanımsız davranış***

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

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

***/\****

***upx'ın hayatı bittiğinde upy'in hayatı bitmemiş olacak***

***yani dangling pointer olacak***

***\*/***

**}**

***// unique\_ptr'in hayatını sonlandırma***

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

**{**

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

**auto** up **=** make\_unique**<**Date**>(**31**,** 12**,** 2023**);**

***// sonlandırma yöntemleri***

up**.**reset**();** ***//up.reset(nullptr);***

up **=** **nullptr;**

up **=** unique\_ptr**<**Date**>{};**

up **=** **{};**

**}**

***// unique\_ptr reset() fonksiyonu***

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

**{**

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

**auto** up **=** make\_unique**<**Date**>(**31**,** 12**,** 2023**);**

up**.**reset**(new** Date**(**1**,** 2**,** 2024**));**

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

**}**

***// unique\_ptr release() fonksiyonu***

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

**{**

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

**auto** up **=** make\_unique**<**Date**>(**31**,** 12**,** 2023**);**

Date **\***p **=** up**.**release**();** ***// up'a boşa çıkar ama dtor çağrılmaz***

***// ikisi aynı şey***

**auto** x **=** move**(**up**);**

unique\_ptr**<**Date**>** x**(**up**.**release**());**

**}**