**2023.08.23metin, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**class** Date **{**

**private:**

**int** x**{}** ***//default init edilecek***

std**::**string str ***// string class'ının default ctor cagırılcak böyle default***

**};**

**move-only type class**

**class** Neco**{**

**public:**

Neco**();**

Neco**(const** Neco**&)** **=** **delete;**

Neco **&operator=(const** Neco**&)** **=** **delete;**

Neco**(**Neco**&&);**

Neco**&** **operator=(**Neco**&&);**

**}**

**copy ve move ctor yok**

**class** Neco**{**

**public:**

Neco**();**

Neco**(const** Neco**&)** **=** **delete;**

Neco **&operator=(const** Neco**&)** **=** **delete;**

**}**

***// Asla asla move member'ları delete etme!!! Çünkü move member delete edersem copy memberleri bloke ederiz***

**class** Member **{**

**public:**

Member**(int);**

Member**(int,int);**

Member**()** **=** **delete;** ***// durum2***

**private:**

Member**()** ***// durum 1***

***//durum 3 hiç default ctor olmaması***

**}**

**class** Nec

**{**

**private:**

Member mx**;** ***// burda syntax hatası yok***

***/\****

***Nec sınıfın default ctor delete edilir çünkü derleyicinin yazdığı default ctor mx'i default init edecek***

***ancak mx'in default ctor olmadığı için kendi default ctor'ın delete edicek***

***\*/***

**}**

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

**{**

Nec nec**;** ***// burda nec nesnesi oluşamıcak çünkü default ctor delete edilmiş durumda***

**}**

* **temporary objects C++17**
* **explicit ctor**
* **conversion ctor**

**Temporary Objects Cpp17**

**class** Myclass**{**

**public:**

Myclass**();**

Myclass**(int** x**);**

**}**

**void** func**(const** Myclass **&);**

**void** foo**(**Myclass**);**

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

**{**

Myclass**{};** ***// PR Value expr***

Myclass**(**12**);**

***// iki durumda da ctor çağrılır böyle objelere temporary objects denir***

Myclass m**;**

func**(**m**);**

***/\****

***eğer bu m nesnesine func ya da foo gibi sadece fonksiyon çağrısı için ihtiyacımız varsa***

***böyle kullanmalıyız çünkü yukarıdaki gibi kullanırsak ( func(m) )***

***hemen nesnenin daha sonra kullanacağımızı düşünülebilir***

***hem de mandatory copy elision (c++17)***

***\*/***

func**(**Myclass**{**12**});**

Myclass**&** r **=** Myclass**{};** ***// l ref r value'ya bağlanmak istiyor hatalı***

**const** Myclass**&** r **=** Myclass**{};** ***// const l value ref olur hatasız***

Myclass**&&** r **=** Myclass**{};** ***// r value ref olur hatasız***

**}**

**void** foo**(**Myclass**);**

**void** bar**(**Myclass **&);**

**void** baz**(const** Myclass **&);**

**void** func**(**Myclass **&&)**

main**()**

**{**

foo**(**Myclass**{})** ***// geçerli***

bar**(**Myclass**{})** ***// geçersiz***

baz**(**Myclass**{})** ***// geçerli***

func**(**Myclass**{})** ***// geçerli***

**}**

***/\****

***normalde bir geçici nesne oluştuğunda, hayata gelen geçici nesnenin hayatı geçici nesneyi içine***

***alan ifadenin yürütülmesiyle sona erer***

***\*/***

main**()**

**{**

std**::**cout **<<** "[1]\n"**;**

Myclass **{};** ***// ctor ve dtor 1 ve 2 arasında çağrılır***

std**::**cout **<<** "[2]\n"**;**

std**::**cout **<<** "[3]\n"**;**

**const** Myclass **&**r **=** Myclass**{};** ***// ctor 3 'ten sonra ama dtor scope bitince çağrılır buna life extension denir***

std**::**cout **<<** "[4]\n"**;**

**}**

**moved-from state (taşınmış nesne durumu)**

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

**{**

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

string s**;**

s **=** foo**();** ***// move assignment***

***/\****

***foo() r value expr o yüzden move assignment çağrılır ancak***

***l value olsaydı copy assignment çağrılırdı***

***\*/***

s **=** string**(**100100**,**'A'**)** ***// move assignment***

string str**(**200000**,**'Z'**);**

string sx **=** str**;** ***// copy assignment***

string sxx **=** std**::**move**(**str**);** ***// move assignment***

***//mesela***

string sx**;**

**{**

string str**(**2000**,** 'Z'**);**

sx **=** std**::**move**(**str**);**

***// str 'in dtor bu scope sonunda çağrılır***

***// str is in moved\_from state***

***// str yani kaynağı çalınmış nesne geçerlidir ancak değeri bilinmez s.empty() --> true döner***

**}**

***// sx burda kaynağı gittiği için kullanılamaz.***

**}**

***/\****

***Tipik olarak (böyle bir zorunluluk yok) move member'lar kaynağı çalınmış diğer nesneyi***

***default ctor edilmiş state'te bırakırlar.***

***Kaynağı çalınmış bir nesne***

***a) geçerli bir durumda (in valid state)***

***b) bilinmeyen değerde (its value unknown)***

***standart kütüphanede böyle tasarlanmştır.***

***\*/***

std**::**string foo**();**

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

**{**

string sx**(**20000**,** 'a'**);**

string str **=** std**::**move**(**sx**);** ***// sx'i gözden kaçırdığımız düşünülebilir***

**}**

Örnek Kod (Moved-from state)

**class** Vector **{**

**public:**

push\_back**(const** string **&);** ***// l value buraya copy ederek***

push\_back**(**string **&&);** ***// r value buraya move ederek***

**};**

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

**{**

ifstream ifs **{**"notlar.txt"**};**

string sline**;**

vector **<**string**>** svec**;**

***// satır satır dosyadan okuyoruz***

**while(**getline**(**ifs**,** sline**))**

**{**

cout **<<** sline **<<** '\n'**;**

svec**.**push\_back**(**sline**);**

svec**.**push\_back**(**std**::**move**(**sline**));** ***// böyle yaparak daha verimli olur***

***/\****

***burada sline string'i std::move kullanarak svec taşınıyor.***

***sline taşınıyor ama tekrar kullanabilir hale de tekrar atama yapılabiliyor***

***ama bu garantiye veren stardart kütüphane başka bir kütüphane sline'in tekrar kullanabileceği***

***garantisi vermez.***

***\*/***

**}**

**}**

**Conversion Constructor (Dönüştüren kurucu işlev)**

**class** Myclass **{**

**public:**

Myclass**()** **=** **default;**

Myclass**(int);**

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

**}**

main**()**

**{**

Myclass mx**;**

std**::**cout **<<** "main [1]\n"**;**

mx **=** 5**;** ***// eğer Myclass(int) diye ctor yoksa hata verir***

***/\****

***Myclass(int) ctor'u çağrılır ve mx'in dtor çağrılır***

***çünkü mx temp object olarak oluşturuldu***

***mx = 5 move assignment ile çağrılır ama biz copy assignment yazarsak class'ın içinde***

***copy assignment çağrılır.***

***\*/***

std**::**cout **<<** "main [2]\n"**;**

***// ilk tanımladığım mx'in dtor çağrılır***

**}**

Function overload resolution

- variadic conversion

- user defined conversion

**class** Myclass **{**

**public:**

Myclass**();**

Myclass**(int);**

Myclass**(bool);**

**}**

**void** func**(**Myclass**);**

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

**{**

func**(**12**);** ***// bu geçerli çünkü burada user defined conversion gerçekleşti***

Myclass m**;**

m **=** 4.9073**;** ***// double olmasına rağmen legal***

**int** x **=** 10**;**

**int\*** ptr **=** **nullptr;**

m **=** **&**x**;** ***// legal***

m **=** ptr**;** ***// legal***

***// ptr türünden bool türüne dönüşüm var***

**}**

Eğer bir dönüşüm aşağıdaki dönüşüm sekanslarından biriyle gerçekleştirebiliyor

ise derleyici bu dönüşümü örtülü olarak yapmak zorunda

user-defined conversion + stardart conversion

standart conversion + user-defined conversion

Myclass m **;**

**double** dval **=** 32.5**;**

m **=** dval**;** ***// Önce standart dönüşüm (double -int) sonra user-defined dönüşüm (int - Myclass)***

**Explicit Ctor**

**class** Myclass **{**

**public:**

Myclass**();**

**explicit** Myclass**(int);** ***// anahtar sözcükelerle dönüşüm ile olur (static\_cast gibi)***

**}**

main**()**

**{**

Myclass m**;**

m **=** 23**;** ***// hata döner***

m **=** **static\_cast<**Myclass**>(**23**);** ***// hata dönmez böylece yanlışlıkla dönüşüm yapmayı engeller***

Myclass m1**(**19**);** ***// geçerli direct init***

Myclass m2**{**35**};** ***// geçerli direct list init***

Myclass m3 **=** 21**;** ***// geçersi copy init***

***/\****

***ctor explicit ise copy init syntax hatası oluşturur ama***

***direct init ve direct list init hata oluşturmaz***

***\*/***

**}**

Cpp Core Guidelines

Bir sınıfı (özellikle) tek parametreli ctor'larını (aksi yönde karar almanıza gerektirecek bir neden olmadığı sürece) explicit yapınız.

Explicit Ctor Örnek:

main**()**

**{**

unique\_ptr**<int>** p1**{new** **int};**

unique\_ptr**<int>** p2**{new** **int};**

unique\_ptr**<int>** p3 **=** **new** **int** ***// geçersiz çünkü unique\_ptr explicit ctor sahip***

**}**

**class** Myclass **{**

**public:**

Myclass**(int);**

**explicit** Myclass**(double);**

**}**

main**()**

**{**

Myclass m **=** 45.98**;** ***// hata olmaz çünkü explicit ctor overload sette hiç girmiyor***

**}**

**void** func**(**Myclass**)**

Myclass foo**()**

**{**

***//return Myclass();***

***//return Myclass {};***

**return** **{};** ***// ctor explicit olursa geçersiz***

***// üçüde geçerli***

**}**

main**()**

**{**

func**(**Myclass**());**

func**(**Myclass**{});**

func**({});** ***// ctor explicit olursa geçersiz***

***// üçüde geçerli***

**}**