**28.08.2023**

**Sınıfların Static Veri Elemanları**

***// sınıfların static veri elemanları***

***//nec.h***

**class** Nec **{**

**public:**

**static** **int** x**;** ***// derleyici x için bir yer ayırmaz. Sadece bildirimdir***

**};**

***//nec.cpp***

**int** Nec**::**x**{};** ***// static veri elemanları cpp dosyasında yapılır***

***// forward declaration***

**class** Myclass**;**

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

**{**

Myclass m**;** ***// incomplete type***

**}**

**incomplete type'lar ile neler yapabiliriz?**

- fonksiyon bildirimlerinde kullanabiliriz

* Neco foo(Neco);
* Neco& bar(Neco&);

- type alias declaration

* typedef Neco\* NecoPtr;
* typedef Neco& NecoRef;

- pointer ya da referans değişkenler tanımlayabiliriz.

* Myclass \*p = nullptr;

- extern bildirim yapabiliriz.

* extern Myclass ge;
* extern Myclass ga[];

**class** Nec**;**

**class** Myclass **{**

**static** Nec senc**;** ***// incomplete type***

**}**

**class** Data **{**

**int** mx**;**

***// Data data; hatalıdır çünkü derleyici Data'nın size bilmek zorunda***

**static** Data data ***// hatalı değil size bilmesine gerek yok derleyicinin***

**}**

**void is a type and void is a incomplete type**

Header-Only Library

**class** Myclass **{**

**inline** **static** std**::**vector**<int>** x **{**1**,**23**,**4**,**50**};**

**static** **int** sx**;** ***// ODR'i ihlal eder eğer inline kullanmazsam***

***/\****

***sınıfların static, const ve integral type veri elemanlarına sınıf içinde***

***ilk değer verilebilir***

***\*/***

**static** **const** **int** ck **=** 10**;** ***// ilk değer verilebilir***

**static** **constexpr** **double** x **=** 5.4**;** ***// implicitly inline***

**static** **int** k**;** ***// ilk değer veremeyiz***

**}**

**class** Nec **{**

**public:**

Nec**(int** i**):** x**(**i**)** **{};**

***// legal değil çünkü ctor x'e ilk değer veriyor***

***// x'e ilk değer veremeyiz***

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

**{**

x **=** 6**;**

***// legal ama static olmasa legal olmazdı***

**}**

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

**{**

x **=** 5**;** ***// legal***

Nec**::**x **=** 5**;** ***// legal***

**this->**x **=** 5**;** ***// legal***

**}**

**int** y**;**

**private:**

**static** **int** x**;**

**}**

**int** y **=** 5**;**

**int** Nec**::**x **=** y**;**

***//hatalı olur çünkü burda y class scope aranıyor global scope'ta değil***

***// ortada bir x değişkeni olmadığı içinde syntax hatası verir.***

**Sınıfların Static Üye fonksiyonları(Static Member Functions)**

Static üye fonksiyonları

a) class scope'ta bulunurlar (global'den fark olarak)

b) sınıfın private elemanlarına erişebilirler.

-Sınıflarn static üye fonksiyonları sınıfların non-static veri

elemanlarını kullanamaz

**class** Nec **{**

**public:**

**void** foo**();** ***// non-static gizli bir parametre değişkene sahiptir***

**static** **void** foo**();**

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

***//static void foo() const syntax hatasıdır.***

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

**{**

***// burada this anahtarını kullanamayız. Çünkü***

***// this'e sahip değildir.***

***// mx = 5; hatalıdır çünkü this pointeri yok***

***// func(); // legal değil this pointeri yok***

Nec myNec**;**

myNec**.**mx**;** ***// legal***

myNec**.**func**();** ***// legal***

**}**

**private:**

**int** mx**;**

**}**

**class** Nec **{**

**public:**

**static** **double** foo**()**

**{**

**return** 3.9**;**

**}**

**static** **int** ival**;**

**};**

**int** foo**()**

**{**

**return** 2**;**

**}**

**int** Nec**::**ival **=** foo**();**

**int** Nec**::**ival **=** **::**foo**();** ***// global'daki foo() çağrılır ival = 2***

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

**{**

std**::**cout **<<** Nec**::**ival **<<** "\n"**;** ***// ival = 3***

***/\****

***static veriyi init eden isimler önce class scope'ta aranır***

***\*/***

**}**

**Named Constructor**

**class** Myclass

**{**

**public:**

**static** Myclass createObject**();**

**};**

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

**{**

**auto** m1 **=** Myclass**::**createObject**();**

**}**

Overloading Constructor’a farklı bir bakış

**class** Complex

**{**

**public:**

**static** Complex create\_polar**(double** a**,** **double** d**)**

**{**

**return** Complex**(**a**,** d**,** 0**);**

**}**

**static** Complex create\_cartesian**(double** r**,** **double** i**)**

**{**

**return** Complex**(**r**,** i**);**

**};**

**private:**

Complex**(double** r**,** **double** i**);**

Complex**(double** a**,** **double** d**);**

**}**

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

**{**

***// mandatory copy elision***

**auto** c1 **=** Complex**::**create\_cartesian**(**3.5**,**1.2**);**

**auto** c2 **=** Complex**::**create\_cartesian**(**.2352 **,** 4.5767**);**

**}**

Bir class’tan sadece dinamik nesne oluşturulmasını istiyorsak

**class** DynamicOnly

**{**

**public:**

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

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

DynamicOnly**\*** create\_object**()**

**{**

**return** **new** DynamicOnly**{};**

**}**

**private:**

DynamicOnly**();**

**}**

Singleton Pattern ( Tek Nesne Örüntüsü): Bir sınıf türünden tek bir nesne oluşturabiliyor

**class** Singleton

**{**

**public:**

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

**static** Singleton**\*** get\_instance**()**

**{**

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

**{**

mp **=** **new** Singleton**();**

**}**

**return** mp**;**

**}**

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

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

**private:**

**inline** **static** Singleton **\***mp**{};**

**}**

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

**{**

***// Hep aynı nesneyi kullanıyoruz ve bu nesne programın sonuna kadar***

***// dtor olmayacak***

**auto** p **=** Singleton**::**get\_instance**();**

Singleton**::**get\_instance**()->**bar**();**

Singleton**::**get\_instance**()->**foo**();**

**}**

Meyers' signleton

* -lazy initialization
* thread-safe

***// Meyers' signleton***

**class** Singleton

**{**

**public:**

**static** Singleton**&** instance**()**

**{**

**static** Singleton object**;**

**return** object**;**

**}**

**private:**

Singleton**();**

**}**

Hayatta olan nesnelerin sayılması

***// Hayatta olan nesnelerin sayılması***

**class** Myclass

**{**

Myclass**()**

**{**

**++**live\_object\_count**;**

**++**lived\_object\_count

**}**

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

**{**

**--**live\_object\_count**;**

**}**

**static** **int** get\_live\_count**()**

**{**

**return** live\_object\_count**;**

**}**

**private:**

**inline** **static** **int** live\_object\_count**{};**

**inline** **static** **int** live\_object\_count**{};**

**}**

Hayatta kalan diğer dövüşcülerden yardım isticek

**class** Fighter

**{**

**public:**

**void** call\_fighters\_for\_help**()**

**{**

**}**

**};**

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

**{**

Fighter f1**{**"Emre"**};**

Fighter f2**{**"Mehmet"**);**

Fighter f3**{**"Necati"**};**

f3**.**call\_fighters\_for\_help**();**