**08.09.2023**

**Arrow operator overloading**

**class** Myclass

**{**

**public:**

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

**{**

std**::**cout **<<** "Myclass::foo()\n"**;**

**}**

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

**{**

std**::**cout **<<** "Myclass::bar(int)\n"**;**

**}**

**}**

**class** MyPtr **{**

**public:**

Myclass**\*** **operator->();**

**}**

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

**{**

Myclass m**;**

Myclass**\*** p **{&**m**};**

MyPtr myPtr**;**

p**->**foo**();**

p**->**bar**(**12**);**

myPtr**->**foo**();**

***//myPtr.operator->()->foo();***

myPtr**->**bar**(**12**);**

***//myPtr.operator->()->bar(12);***

**}**

**Örnek:**

Bir DatePtr nesnesi new ifadesi ile oluşturulmuş dinamik ömürlü bir Date nesnesini gösterir ancak DatePtr nesnesinin hayatı bittiğinde gösterdiği Date nesnesini delete eder.

**class** Date **{**

**public:**

Date**(int** d**,** **int** m**,** **int** y**)** **:** md**{**d**},** mm**{**m**},** my**{**y**}**

**{**

std**::**cout **<<** "\*this" **<<** "değerindeki" **<<** **this** **<<** " adresinde"

**}**

**~**Date**();**

**friend** std**::**ostream**&** **operator<<(**std**::**ostream**&** os**,** **const** Date**&** dt**);**

**int** get\_month\_day**()** **const;**

**int** get\_month**()** **const;**

**int** set\_year**(int);**

**int** set\_month**();**

**private:**

**int** md**{**1**},** mm**{**1**},** my**{**1900**};**

**};**

/\*

bir DatePtr nesnesi new ifadesi ile oluşturulmuş

dinamik ömürlü bir Date nesnesini gösterir

ancak DatePtr nesnesinin hayatı bittiğinde gösterdiği Date nesnesini

delete eder

\*/

**class** DatePtr **{**

**public:**

DatePtr **=** **default;**

**explicit** DatePtr**(**Date **\***p**);**

**~**DatePtr**()** **{**

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

**delete** mp**;**

**}**

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

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

Date**&** **operator\*()**

**{**

**return** **\***mp**;**

**};**

Date**\*** **operator->(){**

**return** mp**;**

**};**

**private:**

Date**\*** mp **{nullptr};**

**};**

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

**{**

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

**if** **(true)**

**{**

DatePtr p **{new** Date**(** 3**,** 5**,** 1987**}};**

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

std**::**cout **<<** "ay = " **<<** p**->**get\_month**()** **<<** "\n"**;**

p**->**set\_year**(**2022**);**

**}**

***// Date nesnesi hayata veda eder***

std**::**cout **<<** "main sona eriyor\n"**;**

**}**

**Overloading function call operator ()**

C'de foo(3, 5) üç ayrı varlık kategorisinden birine ait olması gerekiyor

Bunlar:

* fonksiyon ismi (function designator)
* fonksiyon pointer
* function-like macro

Kurallar:

* fonksiyon ismi operator() olmalı
* üye fonksiyon olmak zorunda
* varsayılan arguman alabilir diğer operator overloadingler alamaz
* ismiyle çağrılabilir
* overload edilebilir

***// Function object***

**class** Myclass

**{**

**public:**

**void** **operator()()**

**{**

std**::**cout **<<** "Myclass::operator()() this:" **<<** **this** **<<** "\n"**;**

**}**

**void** **operator()(int** x **=** 5 **);**

**}**

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

**{**

Myclass m**;**

std**::**cout **<<** "&m = " **<<** **&**m **<<** "\n"**;**

m**();**

***//m.operator()()***

**}**

**Örnek:**

**class** Random **{**

**public:**

Random**(int** low**,** **int** high**)** **:** mlow**{**low**},** mhigh**{**high**}** **{}**

**int** **operator()()**

**{**

**return** std**::**rand**()** **%** **(**mhigh **-**mlow **+** 1**)** **+** mlow**;**

**}**

**private:**

**int** mlow**,** mhigh**;**

**};**

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

**{**

Random rand1 **{** 45**,** 72 **};**

**for** **(int** i **=** 0**;** i **<** 10**;** i**++)**

**{**

std**::**cout **<<** rand1**()** **<<** "\n"**;**

**}**

**}**

**Tür dönüşüm operatörleri**

Kurallar:

* const fonkisyon olmalı
* tür dönüşüm değeri yazılmaz ama o türü dönüşür
* her türlü dönüşüm türüne uygun (class, pointer, double vb.)

**class** Nec

**{**

**public:**

**operator** **int()** **const;**

**};**

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

**{**

Nec myNec**;**

**int** ival **{**0**};**

ival **=** myNec**;**

ival **=** myNec**.operator** **int();**

**}**

User Define Conversion:

* conversion ctor
* operator T

Standart Conversion -> User Define Conversion

User Define Conversion -> Standart Conversion

User Define Conversion -> User Define Conversion // bu olmaz

**Conversion Ctor**

**class** A **{**

**};**

**class** B

**{**

**public:**

B**();**

B**(**A**);**

**}**

**class** C

**{**

**public:**

C**();**

C**(**B**);**

**}**

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

**{**

B bx**;**

A ax**;**

bx **=** ax**;** ***// user define conversion***

cx **=** ax**;** ***// hatalı çünkü User Define Conversion -> User Define Conversion***

**}**

**Operator Conversion**

**class** Nec

**{**

**public:**

**explicit** **operator** **int()** **const;**

***/\****

***explicit kullanabiliriz böyle tür dönüşüm operator***

***kullanamk zorunda kalırız.***

***\*/***

**};**

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

**{**

**double** dval**{};**

Nec mynec**;**

dval **=** mynec ***// explicit olduğu için hatalı***

***/\****

***önce mynec -> int olacak user define conversion***

***sonra int -> double olacak standart conversion***

***\*/***

***// geçerli***

ival **=** **static\_cast<int>** mynec**;**

ival **=** **(int)**mynec**;**

ival **=** **int(**mynec**)**

**}**

Example:

**class** Counter

**{**

**public:**

Counter**()** **=** **default;**

Counter**(int** val**)** **:** mc**{**val**}** **{}**

Counter**&** **operator++()**

**{**

**++**mc**;**

**return** **\*this;**

**}**

Counter **operator++(int)**

**{**

Counter temp**(\*this)**

**++\*this;**

**return** temp**;**

**}**

**operator** **int()const**

**{**

**return** mc**;**

**}**

**private:**

**int** mc**{};**

**}**

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

**{**

**}**

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

**{**

Counter c **{**632**}**

**return** c**;**

**}**

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

**{**

Counter c **{** 4 **};**

**for** **(int** i **=** 0**;** i **<** 10**;** **++**i**)**

**{**

**++**c**;**

**}**

**int** ival **=** c**;**

***//int ival = c.operator int()***

foo**();** ***// hatasız çalışır***

bar**(**c**)** ***// hatasız çalışır***

**}**

**Operator bool**

**class** Myclass

**{**

**public:**

**operator** **bool()** **const**

**{**

**return** **true;**

**}**

**};**

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

**{**

Myclass m1**,** m2**;**

**auto** **int** x **=** m1 **+** m2**;**

***//auto int x = m1.operator bool() + m2.operator bool();***

cout **<<** "x = " **<<** x **<<** "\n"**;** ***// x = 2***

**}**

Sınıfların operator bool fonkisyonları hem her zaman explicit

olmak zorundadır.

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

**{**

unique\_ptr**<int>** uptr **{** **new** **int{**35**}** **}**

***// operator bool fonkisyon'a sahiptir unique\_ptr***

**if(**uptr**)**

**{**

***//***

**}**

**else** **{**

***//***

**}**

**}**