**2023 08 21**

**class** Address

**{**

**public:**

Address**(const** **char\*** p**)** **:** mlen**(**std**::**strlen**(**p**)),**

mp**(static\_cast<char\*>(**std**::**malloc**(**mlen **+** 1**))){**

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

**throw** std**::**runtime\_error**{** "not enough memory" **};**

**}**

std**::**cout **<<** **static\_cast<void\*>(**mp**)** **<<** "adresindeki bellek blogu edinildi\n"**;**

std**::**stpcpy**(**mp**,** p**);**

**}**

Address**(const** Address**&** other**)** **:** mlen**{**other**.**mlen**},** mp**(static\_cast<char\*>(**std**::**malloc**(**mlen **+** 1**)){**

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

**throw** std**::**runtime\_error**{** "not enough memory" **};**

**}**

std**::**strcpy**(**mp**,** other**.**mp**);**

**}**

Address**&** **operator=(const** Address**&** other**)** **{**

**if** **(this** **=** **&**other**)** **{**

***// avoid self assignment***

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

**}**

std**::**free**(**mp**);**

mlen **=** other**.**mlen**;**

mp **=** **static\_cast<char** **\*>(**std**::**malloc**(**mlen **+** 12**));**

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

**throw** std**::**runtime\_error**{** " not enough memory "**};**

**}**

std**::**strcpy**(**mp**,** other**.**mp**);**

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

**}**

**~**Address**(){**

std**::**cout **<<** **static\_cast<void\*>(**mp**)** **<<** " adresindeki bellek blogu geri verildi\n"**;**

std**::**free**(**mp**);**

**}**

**void** print**()const** **{**

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

**}**

std**::size\_t** length**()const** **{**

**return** mlen**;**

**}**

**private:**

std**::size\_t** mlen**;**

**char\*** mp**;**

**};**

**void** process\_address**(**Address x**)** **{**

***// copy ctor cagrilir***

std**::**cout **<<** "process\_address fonksiyonu cagrildi\n"**;**

x**.**print**();**

**}**

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

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

Address adx**{**" sultangazi "**};**

adx**.**print**();**

cout **<<** "adres uzunlugu" **<<** adx**.**length**()** **<<** "\n"**;**

process\_address**(**adx**);**

***// kopyaladık sonra dtor cagrildi adx danglig pointer oldu***

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

adx**.**print**();**

***//***

Address adx1**{** " gop " **};**

**if** **(**adx1**.**length**()** **>** 10**)**

**{**

Address ady1 **{**" bayrampasa" **};**

ady1**.**print**();**

ady **=** adx**;** ***// copy assigment***

ady**.**print**();**

**}**

adx**.**print**();** ***//***

**}**

copy ctor oluşturma durumunu çok nadir de kullanılsa da eğer bir pointer gibi parametre tutuyorsak adresi işaret eden bir fonksiyona bu nesneyi arguman olarak verdiğimiz de o nesne kopyalanır ve o fonksiyon sonlanınca o nesne dtor olur bu da adresteki nesnenin dtor olmasına sebep olur. Yani main'e tekrar döndüğümüzde o nesneyi kullanamayız.

**class** Student

**{**

***// copy ctor yazmaya gerek yok sorun sadece sınıf veri elemanı pointer olduğunda var***

**private:**

**int** m\_id**;**

std**::**string m\_name**;**

std**::**string m\_address**;**

std**::**vector**<int>** m\_grades**;**

**}**

**Copy Assignment**

**class** Myclass **{**

**public:**

Myclass**&** **operator=(const** Myclass**&** other**)**

**{**

ax **=** other**.**ax**;**

bx **=** other**.**bx**;**

cx **=** other**.**cx**;**

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

**}**

**private:**

A ax**;**

B bx**;**

C cx**;**

**}**

***/\****

***Eski C++'da***

***Big Three***

***Destructor release resourcers***

***Copy Constructor deep copy***

***Copy Assignment release resourcers and deep copy***

***\*/***

**Move Constructor**

**class** Myclass **{**

**public:**

Myclass**();** ***// default ctor***

**~**Myclass**();** ***// destructor***

Myclass**(const** Myclass**&);** ***// copy ctor***

Myclass**&** **operator=(const** Myclass**&);** ***// copy assigment***

Myclass**(**Myclass**&&);** ***// move ctor***

Myclass**&** **operator=(**Myclass**&&);** ***// move assigment***

**};**

**class** Myclass**{**

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

Myclass**(const** Myclass**&)**

**{**

std**::**cout **<<** "copy ctor\n"**;**

**}**

Myclass**(**Myclass**&&)**

**{**

std**::**cout **<<** "move ctor\n"**;**

**}**

**};**

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

**{**

std**::**cout **<<** "const Myclass&\n"**;**

**}**

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

**{**

std**::**cout **<<** "Myclass&&\n"**;**

**}**

**void** foo**(**Myclass**&&** r**)**

**{**

func**(**r**);**

**}**

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

**{**

Myclass m**;**

func**(**m**);** ***// const Myclass&***

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

func**(static\_cast<**Myclass**&&>(**m**));** ***// Myclass&&***

***// taşıma yapmıyor l value'yu r value yapıyor***

***// ne copy ne move ctor çağırlırs***

func**(**std**::**move**(**m**));** ***// Myclass&&***

foo**(**std**::**move**(**m**));** ***// func(const Myclass&) çağırılır***

**}**

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

**{**

Myclass m**;**

***// hayata gelen bir nesne yok o yüzden ne move ne copy ctor cağrılır***

Myclass**&&** r **=** std**::**move**(**m**);**

**}**

**class** Myclass**{**

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

Myclass**(const** Myclass**&)**

**{**

std**::**cout **<<** "copy ctor\n"**;**

**}**

Myclass**(**Myclass**&&)**

**{**

std**::**cout **<<** "move ctor\n"**;**

**}**

**};**

***// kaynağı çalan m objesi***

**void** foo**(const** Myclass**&**other**)**

**{**

***// copy ctor***

Myclass m**(**other**);**

**}**

**void** foo**(**Myclass**&&** other**)**

**{**

***// move ctor***

Myclass m**(**std**::**move**(**other**));**

**}**

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

**{**

Myclass m**;**

foo**(**std**::**move**(**m**));**

**}**

***// derleyicinin yazdığı move ctor***

**class** Myclass

**{**

***// primative türler ve pointerlar için taşıma olmaz***

**public:**

Myclass**(**Myclass**&&** other**)** **:** ax**(**std**::**move**(**other**.**ax**)),** x**(**other**.**x**),** ptr**(**other**.**ptr**)**

**{**

**}**

Myclass**&** **operator=(**Myclass**&&** other**)**

**{**

ax **=** std**::**move**(**other**.**ax**);**

x **=** other**.**x**;**

ptr **=** other**.**ptr**;**

**}**

**private:**

A ax**;**

**int** x**;**

**char** **\***ptr**;**

**};**