11 Mayıs 2022

İçindekiler

Special members
big five
Rule of zero
How Did Move Semantics Get Started?
copyable and moveable classes
Dinamik ömürlü nesneler
new'e bir overload yapmak
array new

11 Mayıs 2022

special members (sınıfın özel üye fonksiyonları): belirli şartlar sağlandığında kodları derleyici tarafından yazılan fonksiyonlardır. Bunları özel yapan derleyicinin bu fonksiyonları default etmesi ya da sizin bu fonksiyonları derleyiciden default etmesini talep etmenizdir.

Special members hangi durumlarda olabilir ?

a) hiç olmayabilir

b) user declared (programcı tarafından bildirilmiş)
Myclass();
Myclass() = default; // derleyiciden bu fonksiyonun kodunun tanımlanmasını talep etmek
Myclass() = delete; // bu fonksiyona yapılan çağrı sentax hatası olsun demek

c) implicitly declared : derleyici durumdan vazife çıkartarak bu fonksiyonu bildirmesi
defaulted :
deleted : default etme sürecinde sentax hatası oluşursa fonksiyonu deleted olarak bildirir.

big three : eskiden copy constructor, copy assignment ve destructor'ın oluşturduğu üç fonksiyona denirdi**.**

big five : günümüzde copy constructor, copy assignment, destructor, move constructor ve move assigment'ın oluşturduğu beş fonksiyona denir.

Rule of zero: ideali bu fonksiyonların hepsinin derleyici tarafından tanımlanmasıdır. (derleyicinin default etmesi)
tek sıradışı durum default constructor, bunun derleyici tarafından yazılması uygun olmayabilir.
Öyle durumlar varki biz bu fonksiyonlardan copy constructor, copy assignment ve destructor'ı hatta move member'ları yazmak zorunda kalabiliyoruz.

How Did Move Semantics Get Started?

herşey vector sınıfı ile başladı.

vector => dinamik dizidir.
Heap'te tutar. 3 pointer tutar. begin, end, capacity

Eğer program esnasında boyutları bildirilmiş değişmez bir değer kullanıyorsak stack, değişebilir bir değer kullanıyorsak heap bizim için uygun olacaktır.

vector'ü diğer bir vector'e kopyalarken kopyalanan nesnenin hayatı bitmek üzere ise o zaman kaynağı kopyalamak yerine kaynağı çalabilir.
pointer'ları kopyalarız. Hayati bitecek nesnenin de pointer'larını nullptr yaparız. Böylece hayati bitecek nesnenin destructor'ı çağırıldığında bu bellek bloğunu free etmez.
Sonunda büyük bir kopyalama maaliyetinden kaçınmış oluruz.

special members:

default constructor : Myclass()
destructor: ~Myclass()
copy constructor: Myclass(Myclass const&) or Myclass(const Myclass&)
copy assignment: Myclass& operator = (Myclass const&)
move constructor: Myclass(Myclass&&) // kaynak çalma
move assignment: Myclass& operator = (Myclass&&) // kaynak çalma

copy assignment ile move assignment birbirlerinin overload'larıdır.
bu fonksiyona bir çağrı yapıldığında gönderilen argümanın değer kategorisinin Lvalue veya Rvalue olmasına göre copy assignment veya move assignment çağrılacak.

copy constructor ile move constructor birbirlerinin overload'udur.
bu fonksiyona bir çağrı yapıldığında gönderilen argümanın değer kategorisinin Lvalue veya Rvalue olmasına göre değişir.

special members can be:
- not declared
- implicitly declared (deleted, defaulted)
- user declared (deleted, defaulted, user-defined)

```
struct X
{
    X() {};  // user declared (user-defined)
    X(); // user declared  (user-defined)
    X() = default;  // user declared (defaulted)
    X() = delete;  // user declared (deleted)
}
```

Hangi koşullarda sınıfın özel üye fonksiyonları implicitly sağlanıyor ?

1) if user declares nothing = > bütün üye fonksiyonlar default edilir.

```
class Nec{

}
```

Nec sınıfının bütün özel üye fonksiyonları implicitly declared.

```
class Nec{
public:
    Nec() = default;  // default const.
    ~Nec() = default;  //  destructor
    Nec(const Nec&) = default;  //copy constructor
    Nec(Nec&&) = default;  //
    Nec& operator = (const Nec&) = default; //
    Nec& operator = (Nec&) = default;  //
}
```

- default edilmesi delete edilmesi anlamına gelebilir eğer derleyici tarafından default edilen özel üye fonksiyon illegal bir durum oluştururursa. Bu durum örneğin const. bir nesneyi init. etmek, private bir üye fonksiyona çağrı yapmak gibi.

- derleyici eğer move member'lardan birini default ediyorsa ve illegal durum oluşuyorsa bunlar hiç bildirilmemiş sayılıyor. bu duruma da move yerine copy yapılıyor.

Deep copy ile move arasında fark ney?

```
class Myclass{
public:
    Myclass();
    Myclass(const Myclass&);  // copy constructor
    Myclass(Myclass&&);  // move constructor
};

void func(Myclass);

int main()
{
    Myclass mx;

    func(mx);  // copy constructor çağırılır, lvalue expression
    func(Myclass{});  // move constructor, pr value expression, geçici sınıf nesnesi sonra öğrenilecek
    //  move constructor olmasa idi copy constructor çağırılırdı çünkü const Myclass& değeri sol veya sağ taraf değerine bağlanabilir.
}
```

not : Eğer bir sınıfın hem copy constructor'ı hem move constructor'ı varsa ve bir sınıf nesnesine ilk değer vermek için sağ taraf değeri kullanırsam move constructor, sol taraf değeri kullanırsam copy constructor çağırılır.

2) if default const. is user declared then other special members are defaulted.

```
class Nec{
public:
    Nec();  // default const.  is user-declared.
    // other special members are defaulted.
};
```

*----------
-

3) if user declares any non-special constructor then default constructor will not be declared (doesn't exist)
other member functions defaulted

```
class Nec{
public:
    Nec(int);  // default const. not declared.
    // other special members are defaulted.
};
```

4) destructor is user-declared, then default const. copy assingment impliclity declared but move members are not declared (son derece tehlikeli bir durum)

then you should declare copy members **(derleyicinin yazdığı copy constructor, copy assignment shallow copy yapar.)**

```
class Nec{
public:
    // Nec();   // implicitly declared
    ~Nec();  // destructor is user declared
    // Nec(const Nec&);  //  implicitly declared

};
```

*----------

5) copy constructor is user-declared then destructor is defaulted
but move member are not declared then you should declare copy members

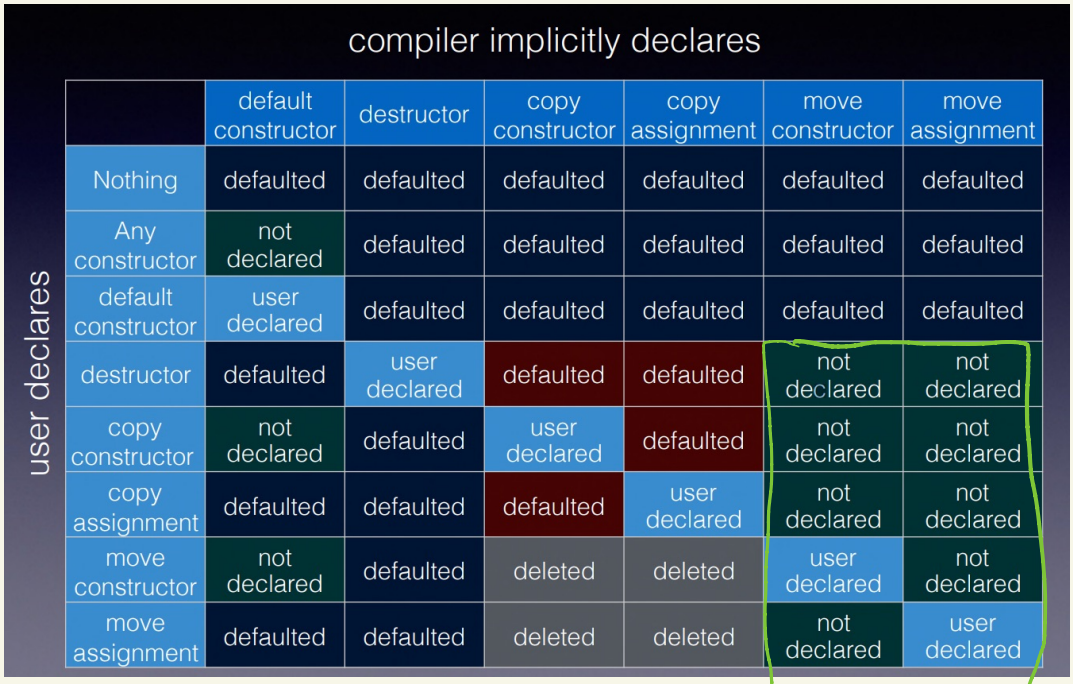
```
class Nec{
public:

    // ~Nec();  // implicitly declared
     Nec(const Nec&);  // copy const. user declared
    // Nec& operator  = (const Nec&);  // implicitly declared
};
```

6) copy assignment(bir constructor değil) is user-declared then default const. , destructor and copy constructor
are defaulted but move members are not declared.

7) if move contructor is user-declared then copy members are deleted and default contructor is not declared (because move constructor is a constructor)

8) move assignment is user-declared



1) move'lar user declared olucak
2) move'lar tanımlanınca copy'ler deleted olur.

// copyable and moveable classes
Eğer bir sınıf copyable and moveable ise (bu şu demek sınıfın copy const. ve move const. var) taşıma gereken yerde move const.
copy gereken yerde copy const. ve copy'layarak atama gereken yerde copy assignment, taşıyarak atama gereken yerde move assignment çalışır. string sınıfı, vector , std'nin sınıflarından çoğu böyle

//non-copyable but movable classes

// non-copyable and non-moveable classes

```
int main()
{

    string s1;  // string class is copyable and moveable

    string s2 = s1;
    string s3 = std::move(s1);

}

*-----------------------------------------------------------------------------------------------

int main()
{

    unique_ptr<int> uptr{new int};   //unique_ptr sınıfı non-copyable but movable'dır.
    unique_ptr<int> uptr2{uptr1};   // sentax hatası
    unique_ptr<int> uptr2{std::move(uptr1)};
}

*-----------------------------------------------------------------------------------------------

int main()
{
    std::mutex mx;  // std::mutex sınıfı non-copyable and non-moveable'dır

    std::mutex my = mx; // sentax hatası
    std::mutex my = std::move(mx); // sentax hatası

}


*-----------------------------------------------------------------------------------------------


   // bir sınıfı copyable but not moveable yapmak için copy member'ları bildirmek
   move member'ları bildirmemek gerekir
   class Myclass{

   public:
       Myclass();
       Myclass(const Myclass&); // copy constructor
       // move const. not declared
       // Myclass sınıfı copyable but not moveable
   };
```

C++'daki çoğu sınıf copyable, movable
özellikle move açısından verim artacaksa kopyalamak yerine taşımak önem kazanıyor.
Örnek: string sınıfı, vector sınıfı, diğer container sınıfları

öyle türler varki taşımanın sağlayacağı bir avantaj yok, move yapılabilir ama avantaj sağlamaz copy'e denk olur.

```
class Nec(){

private:
    int ma[100];  // her sınıf için move const. , move assignment fayda sağlamaz.
    // move member'ların fayda sağlaması için handle gerekir.
};
```

*----------

derleyici move member'lari default ettiğinde copy elemalardan farklı oluyor mu ?

```
class Myclass{

public:
    Myclass(const Myclass& other) : ax(other.ax), bx(other.bx), cx(other.cx) {}  // derleyicinin yazdığı copy constructor
    Myclass(const Myclass&& ) : ax(std:move(other.ax)), bx(std:move(other.bx)), cx(std:move(other.cx)) {}  // derleyicinin yazdığı move constructor
private:
    A ax;
    B bx;
    C cx;
};
```

*----------

```
Myclass func()
{
    Myclass m;

    return m;
};

int main()
{
    m1 = func();
}
```

otomatik ömürlü nesne ile geri dönerseniz
derleyici bir optimizasyon yapacak (named return value optimization) zorunlu değil veya taşıma semantiği devreye girecek . Nesne sadece taşınabilir bir nesne copy'alabilir değilse taşıma semantiği devreye girer.

Dinamik Ömürlü Nesneler (heap alanında)

programcının dilediği noktada hayatlarını başlatabileceği, dilediği noktada hayatlarını bitirebileceği nesnelerdir. Storage'ları heap alanında elde edilir.

otomatik, statik, dinamik ömürlü sınıf olabilir.

default kullanmamız gereken otomatik ömürlü nesneler:
a) maliyetleri (tipik olarak) çok daha fazla
b) hata yapma riski çok daha fazla
c) smart pointer olarak std::unique_ptr, std::shared_ptr. vs kullanılabilir.
d) kodun okunması, yazılması, test edilmesi

new operatörleri (new expressions)
delete operatörleri (delete expressions)

new Myclass
new int
new std:string

new Myclass
new Myclass()
new Myclass{}
new Myclass(x,y,z)
new Myclass{x,y,z}

Dikkat !!!
new ifadesi bir adres üretir. Üretilen adres hayata gelen nesnenin adresidir !!!

```
int* ptr = new int;
int* ptr = new int(5);
```

hayata gelen nesne adresi

```
int main()
{
    Myclass* p( new Myclass );  // ileride smart pointer'ları kullanacağız.
    Myclass* p{ new Myclass };
}
```

*----------

pointer-like classes: kendisi pointer değil ama pointermış gibi görev yapıyor. smart pointer demiyoruz. örnek : iteratör sınıfı

smart pointer: pointer-like class ama dinamik ömürlü nesnelerin hayatlarını kontrol etmek için nesnelere pointermış gibi kullanılan sınıflar

raw pointer (naked pointer): int*, Myclass*

*----------

new operator ile
operator new farklı ⇒ buradaki: operator new

```
void* operator new(std::size_t)  // malloc gibi
// malloc başarısız olursa nullptr döndürür, new ise exception throw eder.
```

derleyici new ifadesi karşılığı önce standart kütüphanenin operator new fonksiyonunu çağırır.
derleyici çağırdığı op. new sınıfı türünün sizeof değerini gönderir.
çağırılan bu fonksiyon parametresine aktarılan büyüklükte bir bellek alanın allocate edemez ise bu durumda exception throw eder.

```
int main()
{
    auto p1 = new Myclass;
    // operator new fonk. çağrılır
    // sizeof(Myclass ) kadar bellek alanı ayrılır.
    // elde edilen bellek bloğu adresinin sınıf nesnesinin
    //  adresi olarak kullanılarak sınıfın constr. çağrılır.


    /*  pseudo code
        auto p = (Myclass *) operator new(sizeof(Myclass));
        p->Myclass();  // constructor'ı ismiyle çağırmayız, pseudo code olduğu
için
    */
}
```

dikkat !!
eğer hayata getirdiğiniz dinamik ömürlü nesneyi delete etmezseniz şu iki sonuç oluşur:
a) d.ö. sınıf nesnesinin destructor'ı çağırılmaz
b) operator new fonksiyonu ile elde edilmiş sizeof(Myclass) büyüklüğünde bellek bloğu deallocate edilmez.

mülakat sorusu: new operatörü ile nesne oluşturdum ama delete etmedim, ne olur ?

cevap : 1) mutlaka bellek bloğu geri verilmemesi yüzünden memory leak olur. (yarım doğru)
2) destructor çağırılmadığı için resource leak olabilir. (kaynak sızıntısı)

derleyici delete ifadesi karşılığı nasıl bir kod üretir ?

void operator delete(void *vp);
delete p;

p->Myclass; // destructor nesnesinin çağırılması
operator delete(p); // bellek bloğunun geri verilmesi

delete bir operatör, operator delete bir fonksiyon

new operatörü ile operator new (bir fonksiyon) farklı
new ifadesini kullandığımızda operator new fonksiyonunu çağırıyor.

```
int main()
{
    std::cout << "sizeof(Myclass) = " << sizeof(Myclass) << "\n ";

    auto p = new Myclass;
    std::cout << "p = " << p << "\n";

    delete p;  // önce destructor çağırılır
    // sonra operator delete fonk. çağırılır
    // operator delete fonksiyonuna delete operatorünün operandı olan adres geçilir.
}
```

operator new fonksiyonu global olarak overload edilmiş. Hangi türden elemanla çağırılırsa çağırılsın bu new fonksiyonu çağrılır.

// array new :
birden fazla dinamik nesneyi dizi şekline oluşturma

```
int main()
{
    int n;
    std::cin >> n;
    Myclass* p = new Myclass[ n ];  // n kez const. çağrılır

    // p[2] = = *(p+2)

    delete p;  // tanımsız davranış
    delete[ ]  p;  // array delete  // n kez destr. çağırılır
    // const. ile destr. çağırılmas sırası tam tersi !!
}
```