01.09.2023

Bir Fighter hayatta kalan diğer dövüşcülerden yardım isticek

```
class Fighter
 {
 public:
     Fighter()
     {
         fvec_.push_back(this);
     }
     Fighter(std::string name, int age); name_(std::move(name)) , age_(age)
     {
         fvec_.push_back(this);
     }

     ~Fighter()
     {
         //std::erase(fvec_, this);
         if (auto iter = std::find(fvec_.begin(), fvec_.end(), this); iter !=
fvec_.end())
             fvec_.erase(iter);
        else
           std::cer << "this pointer cannot be found in the container\n";

     }
     Fighter(const Fighter&) = delete;
     Fighter& operator=(const Fighter&) = delete;
     void ask_for_help()
     {
         std::cout << "yetisin basım belada... \n";
         for(auto p: fvec_)
         {
             if ( p!= this && p->get_age > 15)
             {
                 std::cout << p->get_name() << " ";
             }
         }
     };
     std::string get_name() const {return name_ ;};
     int get_age() const {return age_ ;}};
 private:
        std::string name_;
        int age_{0};
        inline static std::vector<Fighter* > fvec_;
 };
 int main()
 {
     Fighter f1{"Emre" , 50};
     Fighter f2{"Mehmet", 25);
     Fighter f3{"Necati", 19};
     auto p1 {new Fighter("ganoş", 23)};
     f3.ask_for_help();
     delete p1;

     f1.ask_for_help();
     f3.call_fighters_for_help();
 }
```

Delegating Ctor (modern cpp)

Çoğu zaman sınıfların ctor'ları overload ediliyor ancak overload edilen ctor'ların bazen ortak bir kodu oluyor. Unutmayınız ki kod tekrarı her türlü belayı beraberinde getirir eski Cpp'de ctor'ların ortak bir kodu bir fonksiyon şekline tanımlanıyor ve ctor'lar bu fonksiyon çağrıyordu

```
class Myclass
{
    public:
        Myclass(int) : Myclass(x, x, x)
        {
            //common_code();
        };

        Myclass(int, int) :  Myclass(x, x, 0)
        {

        }
        Myclass(int, int, int) : mx(x), my(y), mz(z)
        {
            //common_code();

            //belki ortak kod

        };
        Myclass(const char *p) :Myclass(std::atio(p))
        {
            //common_code();

        };

    private:
        common_code();
        int mx, my, mz;

}
```

Friend Bildirimleri

friend bildirimleri (hemen her zaman sınıfın kendi kodlarına veriliyor)

Sınıfın:

a) global fonksiyonları (free functions)

b) yardımcı türler

Örnek1)

```
class Myclass
{
    public:
        // friend bildirim private veya public yapılabilir. Bir fark yok
        friend void ff(Myclass);
    private:
        int mx{};
        void foo(Myclass);
};

void ff(Myclass m)
{
    m.foo();
    Myclass myc;
    myc.mx = 5;
    // private olmasına rağmen eriştik
}
```

Örnek2)

```
class Myclass
{
    public:
        // hidden friend
        friend void bar(Myclass, int x )
        {
            return x * x; // bu fonksiyon class'ın memberi değil
        }
    private:
        int mx{};
        void foo(Myclass);
};
```

Örnek3)

```
class Erg
{
    public:
        Erg(int);
        void foo(int);
};
class Nec
{
    private:
        friend void Erg::foo(int);
        friend Erg::Erg(int); // ctor friend'Lik verebiliriz.
        int mx;
}

void Erg::foo(int x)
{
    Nec necx;

    necx.mx = x; // private eriştil
}
```

Bir sınıf bir başka sınıfa "frined"lik verebilir. Bu durumda friend bildirime konu sınıf incomplite type olabilir.

```
class Nec
{
    private:
        friend class Erg; // Erg bildirimi olmaması rağmen friend'Lik verdik
}
```

1) A sınıf'ı B sınıfına friend'Lik verirse A sınıfı B'nin private bölümüne erişemez.

2) A B'ye friend'lik vermiş olsun
   B C'ye friend'Lik vermiş olsun
   C A'nın private bölümüne erişemez. Yani frienddlik geçişken değildir.

3) Base sınıfından Der sınıfı kalıtım yoluuyla elde edilmiş olsun. Base sınıfı global foo işlevine friend'lik vermiş olsun. foo işlevi Der sınıfın private bölümüne erişemez.

4) Bazen ihtiyaç olsa da bir sınıf kendi seçilmiş öğeleri içinbir başka kodda friend'lik veremez .Böyle durumarda bazı idioms/ techniques kullanabilir. (attorney)

Operator Overloading

1) Run time maliyeti yok tamamen derleme zamanında çalışır.
2) Keyfi isimlendirme yok. isim operator anahtar sözcüğünü içerecek
    a. operator+
    b. operator!
    c. operator++
    d. operator==
3) Olmayan bir operator overload edilemez
    a. a @ b // böyle bir operator olmadığı için @ ile yapamayız

4) Operandlar en az birinin user defined olmalı
5) Global operator function ya da member operator function olmalı
6) Her operator overload edilemiyor. Dilin kuralları bazı opearatörlerin overload edilmesini yasaklıyor.
    a. . operatorü overload edilemiyor
    b. ? : (ternarny operator overload edilemiyor)
    c. sizeof operator overlaod edilemiyor
    d. .* operatörü (C'de olmayan Cpp dilinde olan bir operator)
    e. typeid operatoru
7) Bazı operatorler yalnızca member operator function overload edilir.
    a. function call operator (fonksiyon çağrı operator)
    b. subscript operator a[b]
    c. assignment operators
    d. tür dönüştürme operatorleri
    e. -> arrow operatör
8) Bir istisna hariç operator fonksiyonları "default arguman" alamıyor (function call operatör)
9) Bu mekanizmada operatör öncelik seviyesi ve operator öncelik yönü değiştirilemiyor
    a. a * b + c > 10 denklemi böyle yazılıyor ((a * b) + c) > 10 ve bu öncelik değiştirilemiyor.
10) Bütün operatör fonksiyonları isimleriyle çağırılabiliyor.

```
a) a + b  ya da operator+(a + b)

b)  class Myclass
    {

    };

    main()
    {
        Myclass a, b, c;
        a = b; // a.operator(b)
    }
```

Bu mekanizmada ile operatörlerin "arity" sini değişitiremeyiz

arity ==> operator unary ya da binary olmasına

```
 a > b // binary çünkü 2 variable aldı
 !a // unary tek variable aldı

 bool operator >(Myclass); // hatalı tek parametre var
 bool operator >(Myclass, Myclass); // hata yok
 bool operator >(Myclass, Myclass, Myclass); // hatalı 3 parametre var
```

Eğer member operator function ise

```
   a > b  myclass::operator>;
          a.operator>(b);   // yani sol operator this'i alır

class Myclass {

    public:
       bool operator>(Myclass)const;
       //burda hata olmaz çünkü sol operand this'tir
       bool operator>(Myclass,Myclass)const; // hatalıdır

       bool operator!()const; // this operand olarak kullanılır sadece
       bool operator!(Myclass)const; // hatalıdır
}
```