04.09.2023

Global Operator Function vs Member Operator Function

```
/*
global operator function
    a>b – operator>(a, b)
    !x operator!(x)

*/
/*
member operator functions
non-static member function

a>b         myclass::operator>
            a.operator>(b);

x /y        x.opertator/(y);

!x          x.operator!();

a*b         operator*(a, b);

~x          operator~(x);

*/
```

```
class Nec
{
    //Member operator function

    Nec operator*(const Nec&) const;
    Nec operator/(const Nec&) const;
    Nec operator+(const Nec&) const;
    Nec operator>(const Nec&) const;

};
//Global operator function
Nec operator*(const Nec&, const Nec&);
Nec operator/(const Nec&, const Nec&);
Nec operator+(const Nec&, const Nec&);
Nec operator>(const Nec&, const Nec&);

main()
{
    Nec n1, n2, n3, n4;
    auto b = n1 * n2 + n3 / n4 > n5;

    //Global operator function (derleyicinin çevirdiği)
    // operator>(operator+(operator*(n1,n2), operator/(n3,n4)) , n5);

    //Member operator function (derleyicinin çevirdiği)
     // n1.operator*(n2).operator+(n3.operator/(n4)).operator>(n5);
}
```

Operator Overloading'te Function Overloading

```
class Nec {
    public:
        Nec operator+()const; // burada "+" işaret operator'u
        Nec operator+(const Nec&)const; // burada "+" toplama operator'u
        // function overloading var
}
```

Neden member operator function ve global operator function ayrı ayrı var?

```
class Nec {

    public:
        Nec operator+(int) const;

};
int main()
{
    Nec myNec;

    auto x = myNec + x; // bunda hata yok çünkü myNec sol tarafta
    x = 5 + myNec  // hatalı
}
```

```
class Matrix
{

};

std::ostream& operator<<(std::ostream& os, const Matrix&);

int main()
{

    int ival = 10;

    cout << ival;
    cout.operator<<(ival);

    Matrix m;
    cout << m;  // normalde hatalı ama operator<< ile mümkün kıldık

    operator<<(cout, m).operator<<(ival);
}
```

Const Correctness & Operator Functions

```
class Matrix
{
    public:
        Matrix operator*(const Matrix&)const;
        // m1 ve m2 böyle const oldu

};

Matrix operator*(Matrix&, Matrix&); // uygun değil
Matrix operator*(const Matrix&, const Matrix&);
// m1 ve m2 değişmiyor bu yüzden const olmalı
main()
{
    Matrix m1, m2;

    m1 * m2;
}
```

Operator Overloading ile Value Category İlişkisi

```
Bigint& operator+(const Bigint&, const Bigint&) // hatalı
{
    Bigint result; // otomatik ömürlü nesneyi ref döndürüyoruz
    static Bigint result; // static ömürlü olunca hep aynı nesneyi döner
    // (x + y) * (a + b)  x + y ile  a + b aynı nesne

    Bigint* result = new Bigint;
    // delete edilemez

    // Code makes result

    return result;
}
```

```
Bigint operator+(const Bigint&, const Bigint*);
/*
    performansı etkilemez çünkü copy elision ya da
    taşıma semantiği olacak
*/


/*
    a = b;
    x+=y;
    ++x;

    L value expr olduğu için  L value ref dönmesi gerekiyor

*/
```

```cpp
class Bigint {
    public:
        Bigint operator+(const Bigint&) const; // R value
        Bigint& operator=(const Bigint&);  // L value

        bool operator==(const Bigint&)const;
};

main()
{
    Bigint b1, b2;

    auto flag = b1 != b2; // Cpp 20 ile geçerli oldu
}
```

Özel bazı durumlar söz konusu değilse

- binary simetrik operatorler

- global operator fonksiyonu olarak
- -(a < b)
- -(a + b)

bazen global operator fonksiyonu class'a friend olarak yaparak sınıfın private elemanlarıan erişme imkanımız olabilir.

```
int main{


    vector vec = {1, 2, 3};

    auto val = vec.front()++; // hata olmaz

    const vector v1 = {1, 2, ,3 };

    val = v1.front()++; // hatalı olur
    /*
        Çünkü front fonksiyonun geri dönüş değeri const oldu
        const overloading deniyor
    */

}
```

```
/*

    yıldız dereferencing / indirection
    . dot operator
    -> arrow operator


    func fonksiyonun parametre değişkenin türü int ref ref
    fonksiyonun içindeki  x ifadesinin türü int
    (bir ifadenin türü refarasn türü olamaz)
    x ifadesinin value category'si ise l value

*/
void func(int&& x)
{
    x
}
```

```
void bar(int &)
{
    std::cout << "1"; // ikinci olarak buraya gelir
    /*
        x ifadesinin value category'si l value olduğu için
        bu fonksiyon çalışır
    */
}


void bar(int&&)
{
    std::cout << "2";
}

void foo(int&& x)
{
    bar(x);  // birinci olarak buraya gelir
    // x ifadesinin value category'si l value
}

main() {

    foo(5);
}
```

Arrow Operator Overloading Fonksiyon

```
class Myclass
{
    public:
        void foo();
        void bar(int);
};

class PointerLike
{
    public:
        PointerLike(Myclass*);
        Myclass* operator->();
};

main()
{
    PointerLike p = new Myclass;

    p->foo();
    p.operator->()->foo();

    p->bar(12);
    p.operator->()->bar(12);
}
```