\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

saat 21:00-22.00

->iki string arasında +, += çalışıyo ama -, -= çalışmıyo

->string "[]" (stringde overloadlanmış subscript operatörü) boundary check yapmıyor(stringin uzunluğunu aştın mı uyarı vermiyor):

string x = "alaz"; x[15] = "a" gibi bir işlem yapmamaya dikkat etmem lazım

x.at(15); yazınca out of range exception atıyor(programi crashliyor ve hatayi yazıyor)

-> yani string.at() operatörü boundary check yapıyor.

->c++ yeni operatör oluşturmaya izin vermiyor, fakat var olan operatörleri overloadlamaya izin veriyor

->operatör overloading otomatik değil, kendim yapmam lazım

->classlarda overloaded operatör function kullanmak için bu overloaded operatörleri definelamam lazım, şu 3üne dikkat:

**operator=** ----> "=" i overloadlerken "="in sağındaki objenin datalarını "="in solundaki objenin içine kopyalarız. fakat eğer classta pointer varsa, bu kopyalama tehlikeli olabilir(shallow copy). yani kopyalarken sadece sağ objedeki pointerin işaret ettiği adres sol objenin pointerine kopyalanırsa, bir objenin pointeri üzerinde yaptığımız işlem diğer objede de uygulanmış olur. kısaca dinamik bellek yönetiminde problem oluşturur.

**operator(&)** ----> objeyi point eden bir pointer returnler // address operator

**operator(,)** ----> virgül operatörü önce virgülün solundaki ifadeyi değerlendirir, sonrasında sağı değerlendirir ve sağdaki ifadenin sonucunu geri döndürür:

*int x = (5,10); // output: x = 10* a

-> overloadlanamayan operatörler:

“.” “.\*”(pointer to member) “::” “ ?:”

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Saat 22.00-23.00

-> “+” operatörünü overloadlayarak iki int sayıyı toplamak yerine çıkartma işlemi yapmasını sağlayamam. Operatör aşırı yükleme, yalnızca kullanıcı tanımlı türler (class, struct) veya temel türler(int, string) ile kullanıcı tanımlı türlerin karışımları için geçerlidir.

**-> Bazı operatörler sadece class member fonksiyonu olarak overloadlanabilir:** Eğer (), [], -> veya herhangi bir atama operatörü (= gibi) overloadlanacaksa bu operatörler classın member fonksiyonu olarak tanımlanmalıdır. Diğer overloadlanabilir operatörler hem class member hem de global fonksiyonlar olarak tanımlanabilir.

-> **Operatörün aldığı operand sayısı değiştirilemez:** overloadlanan bir operatör, aldığı operand sayısını değiştiremez. Yani, tekil (unary) operatörler tekil kalır, ikili (binary) operatörler ikili kalır. &, \*, +, ve - operatörleri hem unary hem de binary versiyonları olduğu için her iki versiyon da ayrı ayrı overloadlanabilir:

***&, \*, +, ve – operatörlerinin binary ve unary overloadlanmış helleri “mainPCcalisma” projesinde var.***

Unary (&): adres döndür, **myClass\* operator&()** şeklinde yazılır.

Binary (&): ne yaptığını bilmiyorum

Unary (\*): dereference operatörü, **int operator\*()** şeklinde yazılabilir fakat fundamental typesi int olmak zorunda değil, classtaki dataya göre değişebilir. Bu operator display fonksiyonunun farklı bir versiyonu gibi de düşünülebilir.

Binary (\*): çarpma işlemi, classın dataları içinde çarpma işlemi yapar. **myClass operator\*(const myClass& obj)** şeklinde yazılabilir.

Unary (+): değer neyse aynen döner, bir işlem yapmaz. **myClass operator+()** şeklinde yazılabilir. (implementasyonunda sadece **return \*this;** yazıyor.)

Binary (+): normal toplama işlemi.

Unary (-): girilen değer neyse negatife döndürür**. myClass operator-()** şeklinde yazılır. Implementationunda **return myClass(-value);** yazar.

Binary (+): normal çıkarma işlemi.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Saat 22.00-23.00

**-> Member Function Olarak Overloadlanan Binary Operatörler:**

Bir String **(kullanıcı tarafından oluşturulmuş bir class)** classı tanımladığınızı ve iki String nesnesini < operatörü ile karşılaştırmak istediğinizi düşünün. < operatörünü, String sınıfı içinde bir member fonksiyon olarak overloadladığınızda, y < z ifadesi, aslında y.operator<(z) olarak değerlendirilir. Yani sol operand (y) String classına ait bir obje olduğu için bu fonksiyon, z objesini argüman olarak alır ve < operatörünü uygular.

Bu durumda, **binary operatörler member fonksiyonlar olarak sadece sol operand sınıfın bir nesnesi olduğunda** tanımlanabilir. Yani y < z ifadesinde sol taraftaki y objesi mutlaka String classına ait olmalıdır.

**bool operator< (const String& str)**

**-> Üye Olmayan Fonksiyonlar Olarak Aşırı Yüklenen Binary Operatörler:**

Binary operatör < bir non-member/global fonksiyon olarak tanımlandığında, iki argüman almalıdır. Bu argümanlardan biri mutlaka String classının bir objesi (veya referansı) olmalıdır. Yani, y < z ifadesi, programda aslında operator<(y, z) şeklinde değerlendirilir ve iki String objesini veya referanslarını karşılaştıran bir fonksiyon çağrısı yapılır.

**bool operator<(const String &y, const String &z);**

-> **Stream insertion(<<) ve stream extraction(>>) operatörlerinin yazımı:**

**ostream& operator<<( ostream& os, const myClass& obj) {  
 os << obj.value;  
 return os;  
}**  
  
**istream& operator>>( istream& is, myClass& obj) {  
 is >> obj.value;  
 return is;  
}**

***Main’e yazılanlar:***

**Compiler’in gördüğü: cout << obj;**

**Compiler’in çağırdığı: operator<<( cout, obj );**

**Compiler’in gördüğü: cin >> obj;**

**Compiler’in çağırdığı: operator>>( cin, obj );**

>> operatörü, istream referansı (örneğin, cin) döndürür. Bu, myClass nesneleri üzerinde giriş işlemlerinin diğer myClass nesneleri veya başka veri tipleriyle birlikte cascadelenmesine(zincirlenmesine) olanak tanır. Örneğin, bir program iki myClass nesnesine tek bir ifadeyle şu şekilde veri girişi yapabilir:

cin >> obj1 >> obj2;

İlk olarak, cin >> obj1 ifadesi çalışır ve bu, aşağıdaki gibi bir fonksiyon çağrısı yapar:

operator>>( cin, obj1 );

Bu çağrı, cin'i referans olarak döndürür. Böylece cin >> obj1 ifadesi cin'i döndüğü için, ifadenin kalan kısmı sadece cin >> obj2 olarak değerlendirilir. Bu, şu şekilde başka bir üye olmayan fonksiyon çağrısı yapar:

operator>>( cin, obj2 );

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*