Dept. of Comp. Sc. & Engg., NIT Durgapur

CSC 404 Object Oriented Programming

Operator Overloading in C++

- *subrata.nandi@cse.nitdgp.ac.in*

"Operator overloading is just syntactic sugar, which means it is simply another way for a user to make a function call" – *Bruce Eckel (Thinking in C++, Vol 1)*

Overloading of operators…

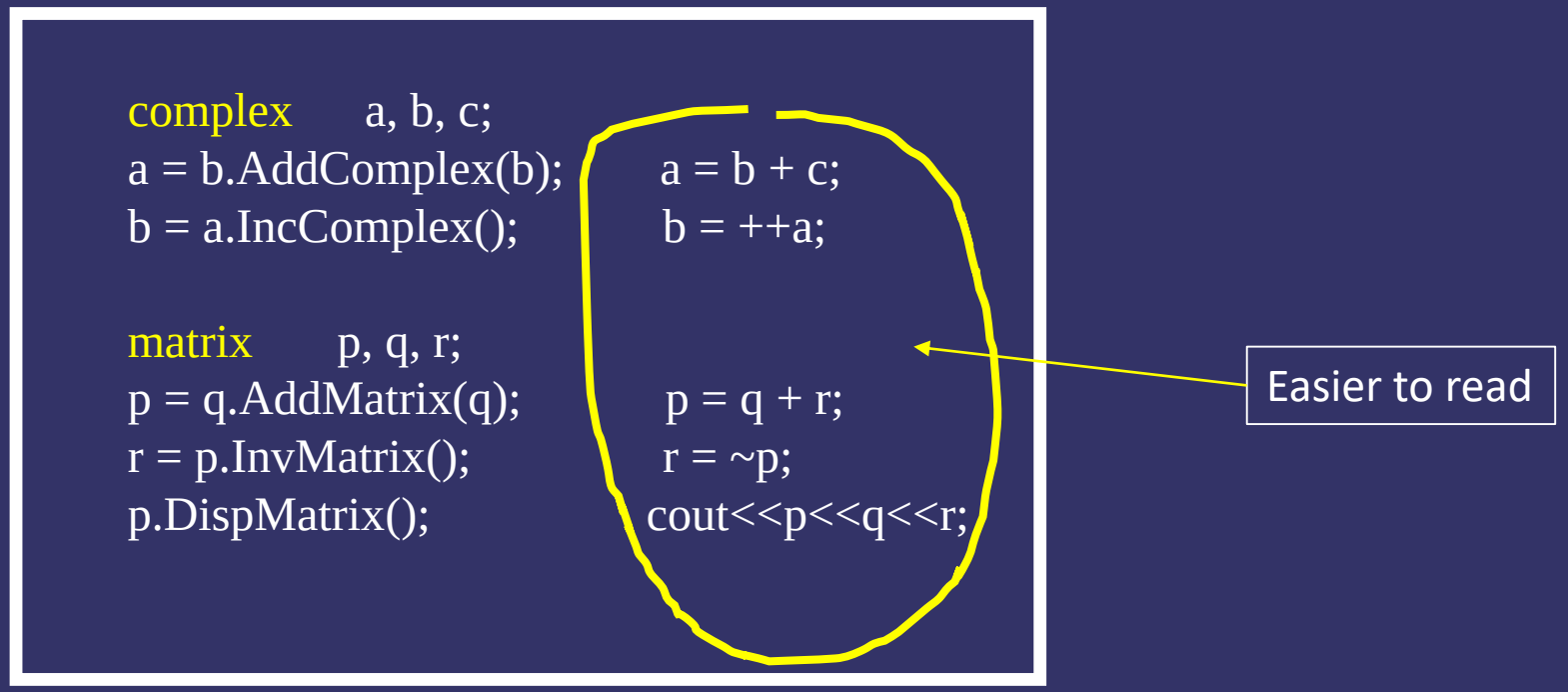
```
float  x, y, c;
int    a, b, c;
-----------------------------------------------
a = b + c;     // + adds two integers
x = y + z;     // + adds two float variables
-----------------------------------------------
b = a<<1;      // <<  acts as bit-wise left shift operator
cout<<a<<b;  // << acts as insertion operator
-----------------------------------------------
a = b & c;     // & acts as bit-wise AND operator
int *p = &a;   //  & acts as address-of operator
```

+ and << are binary operators

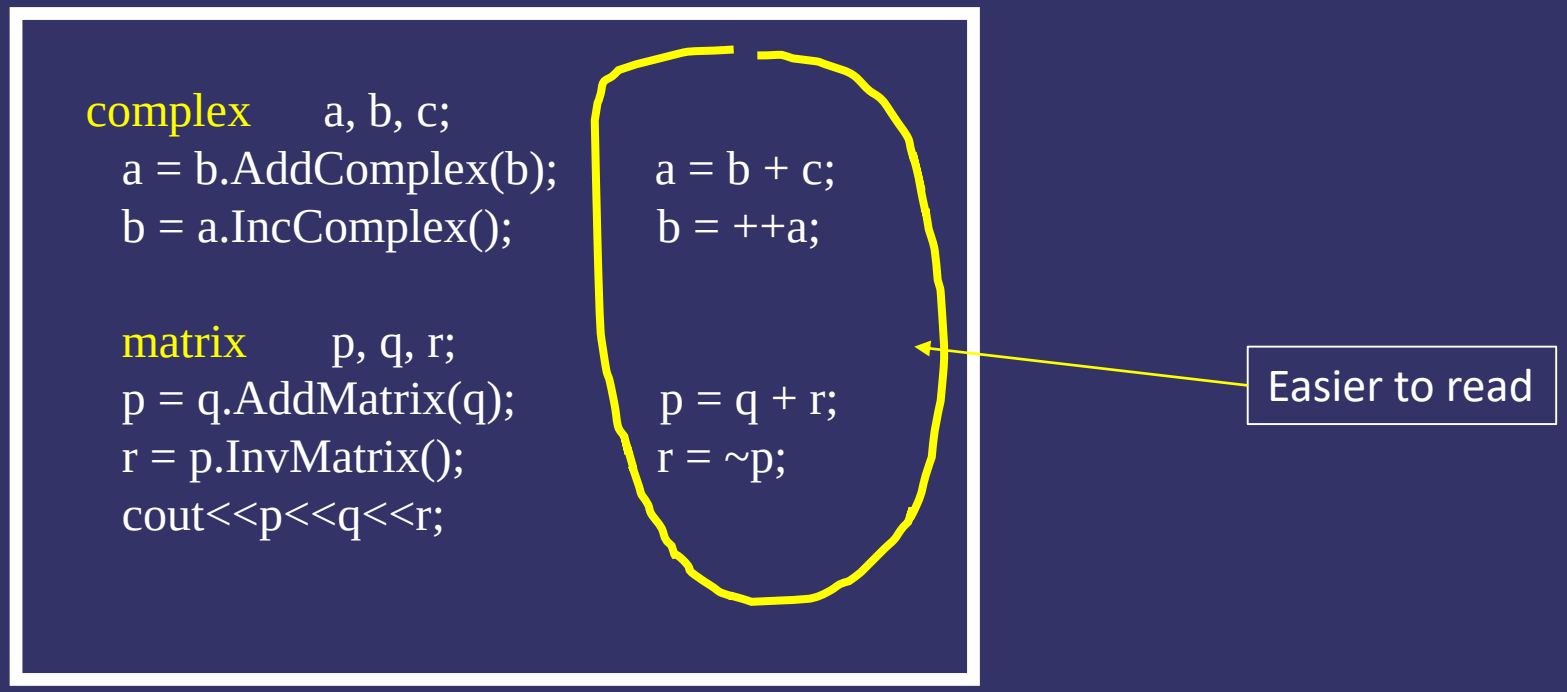
& is an unary operator

Many of the existing C++ operators are already overloaded for built-in datatypes;
Overload means assign multiple responsibilities based on the context

Can this overloading feature be extended to user-defined data types (class) also?


complex a, b, c;
a = b.AddComplex(b);
b = a.IncComplex();
matrix p, q, r;
p = q.AddMatrix(q);
r = p.InvMatrix();
p.DispMatrix();
a = b + c;
b = ++a;
p = q + r;
r = ~p;
cout<<p<<q<<r;
Easier to read

Operator overloading…



- Operator overloading is an object-oriented feature
  - to assign more responsibility to existing C++ operators (e.g.: '+', '++', '~', etc.)
  - so that they can *work meaningfully* with user-defined class (e.g.: complex, matrix, stack, etc.) objects;
  - the way they work in association with built-in class (e.g: int/float, etc.) objects

- It just makes the code involving your class easier to *read*.

An user-defined class - complex

Addition of Complex objects: using AddComplex() method

```
class complex
{
    float rl, img;   // private members
    public:
       float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}
        //  other required methods……

        //body of  the special operator function
        complex    & AddComplex (const complex &c)
        {
              static complex   t; //  local object to store diff
                 t.rl =  rl + c.rl; //  t.rl=this.rl + c.rl
                 t.img =  img + c.img;
              return t;
        }
};// End of  class definition
```

```
int main(void)
{
      complex    a, b(2,3.5);

     // subtracting two complex objects

      complex    d = a.AddComplex (b );


       return 0;
}
```

Addition of Complex objects: using SubComplex() method

```
class complex
{
    float rl, img;   // private members
    public:
       float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}
        //  other required methods……

        //body of  the special operator function
        complex    & SubComplex (const complex &c)
        {
            static complex   t; //  local object to store diff
                t.rl =  rl - c.rl;  // OR t.rl = this.rl – c.rl
                t.img =  img - c.img;
            return t;
        }
};  // End of  class definition
```

```
int main(void)
{
     complex    a, b(2,3.5),d;

    // subtracting two complex objects

        d = a.SubComplex ( b );

        return 0;
}
```

Note:

- Instead of call by value use call by const object reference; to avoid overhead of temporary object constr/destruction
- Instead of returning value of the object from function return object reference; make sure the object to be returned exists (for local object declare it static) even outside the function

Overloading Binary operators

Overloading Binary '-' operator: Case 1

Overloading options

```
int main(void)
{
     complex    a, b(2,3.5);

    // subtracting two complex objects
     //CASE 1
     complex    d = a - b ;

      return 0;
}
```

Overloading Binary '-' operator: Case 1

Overloading options

```
int main(void)
{
     complex    a, b(2,3.5);

    // subtracting two complex objects
    //CASE 1
     complex    d = a - b ;

     return 0;
}
```

1st operand 2nd operand

d = a . operator-(b);

Compiler translates it to a special function call

function name operator@;
@ stands for theoperator symbol

operator is a keyword

Overloading Binary '-' operator: Case 1

```
class complex
{
    float rl, img;
    public:
       float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}
        //  other required methods……

        //body of  the special operator function
        complex    & SubComplex(const complex &c)
        {
              static complex   t; //  local object to store diff
                 t.rl =  rl - c.rl;
                 t.img =  img - c.img;
              return t;
        }
};// End of  class definition
```

Overloading options

```
int main(void)
{
      complex    a, b(2,3.5);

     // subtracting two complex objects
      //CASE 1
      complex    d = a - b ;

       return 0;
}
```

d = a . operator-(b);

Compiler translates it to a special function call

Overloading Binary '-' operator: Case 1

Body of special function to support operator overloading syntax

```
class complex
{
    float rl, img;
    public:
       float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}
        //  other required methods……

        //body of  the special operator function
        complex    & operator-(const complex &c)
        {
              static complex   t; //  local object to store diff
                 t.rl =  rl - c.rl;
                 t.img =  img - c.img;
              return t;
        }
};// End of  class definition
```

Overloading options

```
int main(void)
{
      complex    a, b(2,3.5);

     // subtracting two complex objects
      //CASE 1
      complex    d = a - b ;

       return 0;
}
```

d = a . operator-(b);

Compiler translates it to a special function call

Overloading Binary ‘-’ operator: Case 2

Overloading options

```
int main(void)
{
      complex     a, b(2,3.5);

     // consider subtracting 3.0 from real part of b
      //CASE 2
      complex     d = a - 3.0 ;




       return 0;
}
```

Overloading Binary ‘-’ operator: Case 2

Overloading options

```
int main(void)
{
     complex    a, b(2,3.5);

    // consider subtracting 3.0 from real part of b
    //CASE 2
     complex    d = a - 3.0 ;

      return 0;
}
```

→ d = a . operator-(3.0);

Compiler translates it to a special function call

Overloading Binary '-' operator: Case 2

Body of special function to support operator overloading syntax

```
class complex
{
    float rl, img;
    public:
       float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}

        //  other required methods……

       //body of  the special operator function
        complex    & operator-(float x)
        {
              static complex   t; //  local object to store diff
                 t.rl = rl - x;
                 t.img =  img;
                 return t;
        }
}; // End of  class definition
```

Overloading options

```
int main(void)
{
     complex    a, b(2,3.5);

    // consider subtracting  3.0 from real part of b
     //CASE 2
     complex    d = a - 3.0 ;

      return 0;
}
```

→ d = a . operator-(3.0);

Compiler translates it to a special function call

Overloading Binary ‘-’ operator: Case 3

Overloading options

```
int main(void)
{
     complex    a, b(2,3.5);

    // consider subtracting  real part of b from 3
    //CASE 3
     complex    d = 3.0 - b ;




      return 0;
}
```

Overloading Binary '-' operator: Case 3

Overloading options

```
int main(void)
{
     complex    a, b(2,3.5);

    // consider subtracting  real part of b from 3
     //CASE 3
     complex    d = 3 - b ;

      return 0;
}
```

d = 3 . operator (b);

Can't be translated as a special *member function; here* the 1st operand is a literal (non-object)…

Overloading Binary '-' operator: Case 3

Overloading options

```
int main(void)
{
     complex    a, b(2,3.5);

    // consider subtracting  real part of b from 3
     //CASE 3
     complex    d = 3.0 - b ;

      return 0;
}
```

→ d = ~~3 . operator (b);~~

Can't be translated as a special *member function; here* the 1st operand is a literal (non-object)…

The only option is to write a *friend* operator function…..

A *friend function* is a non-member function

- which can still access the private class members
- don't need an object to get invoked
- not referred through class scope
- Requires the class object as input argument
- To be declared as friend by the class

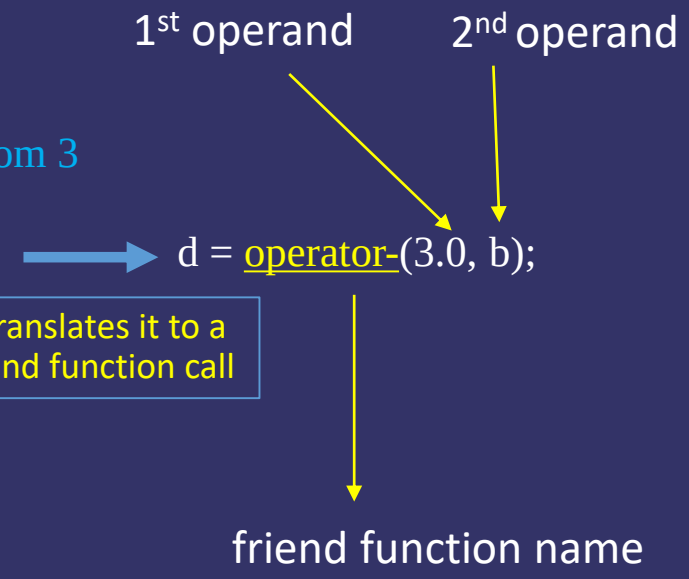
Overloading Binary ‘-’ operator: Case 3

Overloading options

```
int main(void)
{
     complex    a, b(2,3.5);

    // consider subtracting  real part of b from 3
     //CASE 3
     complex    d = 3.0 - b ;

      return 0;
}
```



Overloading Binary '-' operator: Case 3

Body of special function to support operator overloading syntax

```
class complex
{
    float rl, img;
    public:
       float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}

        //  other required methods……

       //body of  the friend operator function
friend  complex    & operator-(float x, const complex  &c)
{
            static complex   t; //  local object to store diff
               t.rl = c.rl - x;
               t.img =  c.img;
               return t;
        }
}; // End of  class definition
```

Overloading options

```
int main(void)
{
     complex    a, b(2,3.5);

    // consider subtracting  real part of b from 3
     //CASE 2
     complex    d = 3.0  - a;

      return 0;
}
```

→ d = operator-(3.0, a);

Compiler translates it to a special function call

Overloading Binary '-' operator: Case 3

Body of special function to support operator overloading syntax

```
class complex
{
    float rl, img;
    public:
       float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}

        //  other required methods……

       // friend operator function declaration
 friend   complex    & operator-(float x, const complex  &c);

}; // End of  class definition

  //  body of the friend function
complex    & operator-(float x, const complex  &c)
 {
              static complex   t; //  local object to store diff
                 t.rl = c.rl + x;
                 t.img =  c.img;
                 return t;
  }
```

Overloading options

```
int main(void)
{
      complex    a, b(2,3.5);

     // consider subtracting  real part of b from 3
      //CASE 2
      complex    d = a - 3.0 ;

       return 0;
}
```

→ d = operator-(3.0, a);

Compiler translates it to a special function call

Overloading Binary operators: Points to note

- Use of an overloaded binary operator (‘-’) with user-defined objects (a and b)
  gets translated to a special operator function call with the name *operator@*; @ being the name of the overloaded operator (e.g.: operator-)

Overloading Syntax	Operator Function name	Options	Compiler translation
a - b;	operator-	i) Member function	a.operator-(b);
		ii) Friend function	operator-(a,b);
a - 3.0;		i) Member function	a.operator-(3.0)
		ii) Friend function	operator-(a,3.0);
3.0 - b;		Friend function	operator-(3.0,b);

One can add both the options simultaneously;
Availability of member function is checked first if not found only then it searches for friend option

- To make it possible
  The behavior of the operator function need to be defined in the class definition
  a binary operator function takes one argument (the 2nd operand) when implemented as member function
  a binary operator function takes two arguments (1st operand and 2nd operand) when implemented as friend function

- Difference with function call
  function don’t appear inside parentheses (e.g.: d=a.SubComplex(b)), *but instead*
  surrounded or are next to characters (e.g.: d = a - b or a++) – a syntactic sugar; just another way of calling a function

```
class complex
{     float rl, img;
    public:
        float arg, amp;
         complex(float f1=1.0,float f2=1.0) {..}
         ~complex() {}
       //  other required methods……

 complex    & SubComplex(const complex &c);

   //prototype of  the operator functions
  complex    & operator+(const complex &c);
  friend complex    &  operator+(const complex  &c, float x);
friend  complex    & operator+(float x, const complex  &c);


}; // End of  class definition


// body of   SubComplex method
complex    & complex:: SubComplex(const complex &c)
        {
              static complex    t;
                 t.rl =  rl - c.rl;
                 t.img =  img - c.img;
              return t;
        }
```

```
//body of  the special operator function; CASE 1
  complex    & complex:: operator-(const complex &c)
        {
              static complex    t;
                 t.rl =  rl - c.rl;
                 t.img =  img - c.img;
              return t;
        }
//body of  the special operator function; CASE 2
complex    &  operator+(const complex  &c, float x)
        {
              static complex    t;
                 t.rl = rl - x;
                 t.img =  img;
                 return t;
        }
// body of friend operator function; CASE 3
complex    & operator-(float x, const complex  &c)
     {
              static complex    t;
                 t.rl = c.rl - x;
                 t.img =  c.img;
                 return t;
        }
```

```
int main(void)
{
     complex    a, b(2,3.5);

      complex    d;

    d = a.SubComplex(b);

    d = a - b ;
    d = a - 3.0 ;
    d = 4.5 - b;

      return 0;
}
```

Putting all possible options together for '-' operator in the class definition….

Opting for friend function instead of member function for the case a - 3.0

Overloading Unary operators

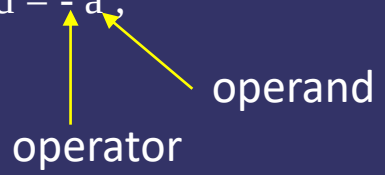
Overloading Unary ‘-’ operator

Overloading Unary Operator

```
int main(void)
{
      complex     a, b(2,3.5);

       // overloading unary ‘-’
       //  flips the sign of real & imaginary components
      complex     d = - a ;

       return 0;
}
```



Overloading Unary '-' operator: using member function

Overloading Unary Operator

```
int main(void)
{
    complex    a, b(2,3.5);

    // overloading unary '-'
    //  flips the sign of real & imaginary components
    complex    d =  - a ;

    return 0;
}
```

operand

d = a.operator-();

Compiler translates it to a special function call

function name

Note: the name of the operator function is same in both unary and binary form of the symbol '-' but operator- (unary) accepts no arguments operator- (binary) accepts one argument

Overloading Unary '-' operator: using member function

```
class complex
{
    float rl, img;
    public:
       float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}
        //  other required methods……

        //body of  the special operator function
        complex    & operator-(void)
        {
              static complex   t;
                 t.rl =  -rl ;
                 t.img =  -img ;
              return t;
        }
};// End of  class definition
```

Overloading Unary Operator

```
int main(void)
{
      complex    a, b(2,3.5);

       // overloading unary '-'
       //  flips the sign of real & imaginary components
      complex    d =  - a ;

       return 0;
}
```

operand

d = a.operator-();

Compiler translates it to a special function call

function name

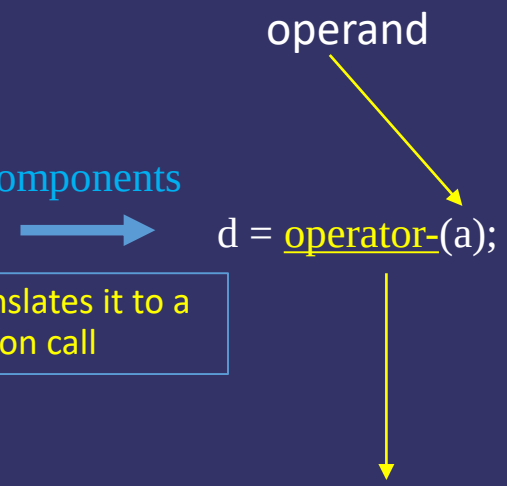
Overloading Unary '-' operator: using friend function

Overloading Unary Operator

```
int main(void)
{
    complex    a, b(2,3.5);

    // overloading unary '-'
    //  flips the sign of real & imaginary components
    complex    d =  - a ;

    return 0;
}
```



Compiler translates it to a special function call

Overloading Unary '-' operator: using friend function

```
class complex
{
    float rl, img;
    public:
       float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}
        //  other required methods……

        //body of  the special friend function
        friend  complex    & operator-(const complex &c)
        {
              static complex   t;
                 t.rl =  - c.rl ;
                 t.img =  - c.img ;
              return t;
        }
};// End of  class definition
```

Overloading Unary Operator

```
int main(void)
{
      complex    a, b(2,3.5);

       // overloading unary '-'
       //  flips the sign of real & imaginary components
      complex    d =  - a ;

       return 0;
}
```

d = operator-(a);

Compiler translates it to a special function call

Overloading Unary operators: Points to note

- Use of an overloaded unary operator (‘-’) with user-defined object
  gets translated to a special operator function call with the name *operator@*; @ being the name of the overloaded operator (e.g.: operator-)

Overloading Syntax	Operator Function name	Options	Compiler translation
- b;	operator-	i) Member function	b.operator-()
		ii) Friend function	operator-(b);

- An unary operator function takes either no arguments (member function) or one argument (friend function)
- Overloading increment (++) and decrement (--) operators

Overloading Syntax	Operator Function name	Options	Operator function call
++a	operator++	i) Member function	a.operator++()
		ii) Friend function	operator++(a);
a++	operator++	i) Member function	a.operator++(int)
		ii) Friend function	operator++(a, int);
- - a	operator- -	i) Member function	a.operator- -()
		ii) Friend function	operator- -(a);
a - -	operator- -	i) Member function	a.operator- -(int)
		ii) Friend function	operator- -(a, int);

dummy int argument is passed to post-inc/dec to distinguish it from pre-inc/dec case

Overloading Unary '--' operator

```
class complex
{
    float rl, img;
    public:
        float arg, amp;
        complex(float f1=1.0,float f2=1.0) {..}
        ~complex() {}
        //  other required methods……

        //operator function prototypes
     complex    & operator-- (void);
 friend    complex  & operator--(const complex &c, int dummy);

}; // End of  class definition

 complex    & complex:: operator-- (void)  // pre-decrement
        {
                rl -- ;    //  (this->rl) -- ;
                img --; //  (this->img) -- ;

            return *this;
        }
```

```
   complex    & operator -- (const complex &c, int dummy)   // post-decrement
        {
            static complex   t = c;

                c.rl --;
                c.img --;

            return t;
        }
```

```
int main(void)
{
    complex    a, b(2,3.5);

    // overloading operator '--'

    b =  -- a ;

    b =  a -- ;


    return 0;
}
```

Operator function call

b = a.operator-- ();

b = operator-- (a, int);

Overloadable operators…

- Except existing operators no new operators like +-, ^^, ^% can be overloaded

- Operators that can be overloaded:
  + - * / % & | ~ = += -= *= /= %= && || ++ -- >> << <<= >>= == != > < <= >= ( ) [ ] ->* -> new delete

- Operators that cannot be overloaded: sizeof, .* (member access through poniter to member)
  . (member access) - difficult to infer if it is for object reference or overloading
  :? (ternary conditional) - difficult to implement that either exp2 or exp3 be executed when overloaded like exp1 ? exp2 : exp3
  :: (scope resolution) – performs compile time scope resolution rather than an expression evaluation

- The arity, precedence and associativity of the operators cannot be changed

- Operators , || && when overloaded losses their special properties (short-circuit evaluation and sequencing); better to avoid overloading them

- Operators = [] () cannot be overloaded using friend function;

"*harmless restriction that eliminated the possibility of some obscure errors because these operators invariably depend on and modify the state of their left-hand operand. However it is probably the case of unnecessary nannyism…..*" **Bjarne Stroustrup**