```
class myclass
{
    int a,b;
    public:
     myclass(int i,int j)
      {   a=i;
       }  b=j;
     void display()
     {  cout<<a<<" "<<b<<"\n";
      }
};
```

```
int main()
{
    int x=2, y=5, z;
    float m=7.6, n=8.2, o;
    double p=88.2, q=98.3, r;
    myclass ob1(5,6), ob2(7,8);
     myclass ob3;
      z = x+y;
      o = m+n;
      r = p+q;

      return 0;
}
```

```
class myclass
{
    int a,b;
    public:
     myclass(int i,int j)
      {   a=i;
       }  b=j;
     void display()
     {
       Cout<<a<<" "<<b<<"\n";
      }
};
```

```
int main()
{
    int x=2, y=5, z;
    float m=7.6, n=8.2, o;
    double p=88.2, q=98.3, r;
    myclass ob1(5,6), ob2(7,8);
     myclass ob3;
      z = x+y;
      o = m+n;
      r = p+q;
      ob3 = ob1+ob2;
      return 0;
}
```

```
class myclass
{
    int a,b;
    public:
     myclass(int i,int j)
      {   a=i;
       }  b=j;
     void display()
     {  cout<<a<<" "<<b<<"\n";
      }
};
```

```
int main()
{
    int x=2, y=5, z;
    float m=7.6, n=8.2, o;
    double p=88.2, q=98.3, r;
    myclass ob1(5,6), ob2(7,8);
     myclass ob3;
      z = x+y;
      o = m+n;
      r = p+q;
      ob3= ob1+ob2;
      return 0;
}
```

```
class myclass
{
    int a,b;
    public:
     myclass(int i,int j)
      {   a=i;
       }  b=j;
     void display()
     {
       cout<<a<<" "<<b<<"\n";
      }
```

```
int main()
{
    int x=2, y=5, z;
    float m=7.6, n=8.2, o;
    double p=88.2, q=98.3, r;
    myclass ob1(5,6), ob2(7,8);
     myclass ob3;
      z = x+y;
      o = m+n;
      r = p+q;

      cout<<z<<o<<r;
       return 0;
}
```

```
class myclass
{
    int a,b;
    public:
     myclass(int i,int j)
      {   a=i;
       }  b=j;
     void display()
    {  cout<<a<<" "<<b<<"\n";
     }
```

```
int main()
{
    int x=2, y=5, z;
    float m=7.6, n=8.2, o;
    double p=88.2, q=98.3, r;
    myclass ob1(5,6), ob2(7,8);
     myclass ob3;
      z = x+y;
      o = m+n;
      r = p+q;

      cout<<z<<o<<r;
       return 0;
}
```

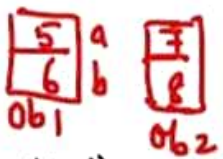
```
{
    int a,b;
    public:
    myclass(int i,int j)
    {
        a=i;
    }   b=j;
    void display()
    {
        cout<<a<<" "<<b<<"\n";
    }
    myclass(){ }
};
```

```
→ int x=2, y=5, z;
→ float m=7.6, n=8.2, o;
→ double p=88.2, q=98.3, r;
→ myclass ob1(5,6), ob2(7,8);
→ myclass ob3;
    z = x+y; ✓
    o = m+n; ✓
    r = p+q; ✓
    r = x+p; ✓
    ob3= ob1+ob2;
    cout<<z<<o<<r;
    cout<<ob3;
}   return 0;
```

```
{
    int a,b;
    public:
    myclass(int i,int j)
    {
        a=i;
    }   b=j;
    void display()
    {
        cout<<a<<" "<<b<<"\n";
    }
    myclass(){ }
};
```

```
→ int x=2, y=5, I;
→ float m=7.6, n=8.2, o;
→ double p=88.2, q=98.3, r;
→ myclass ob1(5,6), ob2(7,8);
→ myclass ob3;
    I = x+y; ✓
    o = m+n; ✓
    r = p+q; ✓
    r = x+p; ✓
    ob3= ob1+ob2;
    cout<<I<<o<<r;
    cout<<ob3;
    return 0;
}
```

```
myclass(int i, int j)
{
    a=i;
    b=j;
}
void display()
{
  cout<<a<<" "<<b<<"\n";
}
myclass(){ }
myclass sum(myclass ob)
{
    myclass o;
    o.a= a+ob.a;
    o.b= b+ob.b
    return o;
};
```

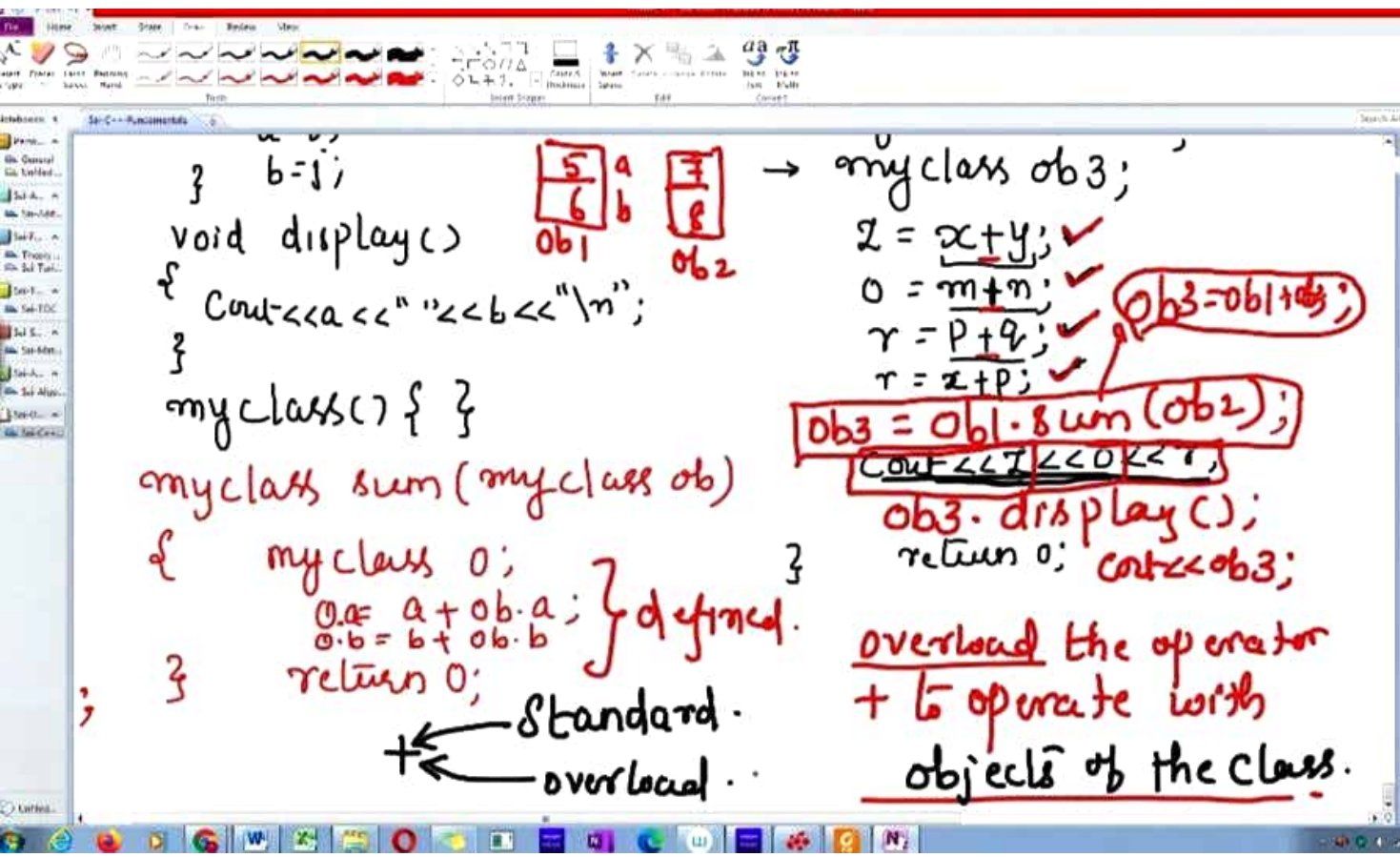



```
→ double p=88.2, r=76.5,
→ myclass ob1(5,6), ob2(7,8);
→ myclass ob3;
  z = x+y; ✓
  o = m+n; ✓
  r = p+q; ✓
  r = z+p; ✓
ob3 = ob1.sum(ob2)
  cout<<z<<o<<r;
  cout<<ob3;
}   return 0;
```

```
    a=1;
  } b=3;
void display()
{ cout<<a<<" "<<b<<"\n";
}
myclass(){ }
myclass sum(myclass ob)
{   myclass o;
    o.a= a+ob.a;
    o.b= b+ob.b
  } defined.
  } return o;
};
```

ob1: a = 5, b = 6 ; ob2: a = 7, b = 8

```
→ myclass ob3;
z = x+y;
o = m+n;
r = p+q;
r = x+p;
ob3=ob1+ob2;
ob3 = ob1.sum(ob2);
cout<<z<<o<<r;
ob3.display();
return 0;
cout<<ob3;
```


} b=j;
void display()
{ cout<<a<<" "<<b<<"\n";
}
myclass(){ }
myclass sum(myclass ob)
{ myclass 0;
0.a= a+ob.a;
0.b= b+ob.b
} defined.
return 0;
};
Standard.
overload..
5 a
6 b
ob1
3 a
8 b
ob2
myclass ob3;
z = x+y;
0 = m+n;
r = p+q;
r = x+p;
ob3=ob1+ob2;
ob3 = ob1.sum(ob2);
cout<<z<<0<<r;
ob3.display();
return 0;
cout<<ob3;
overload the operator + to operate with objects of the class.

# Operator overloading

In C++

↳ we can overload most operators

↳ so that

↳ these operators can perform special operations relative to classes that programmer

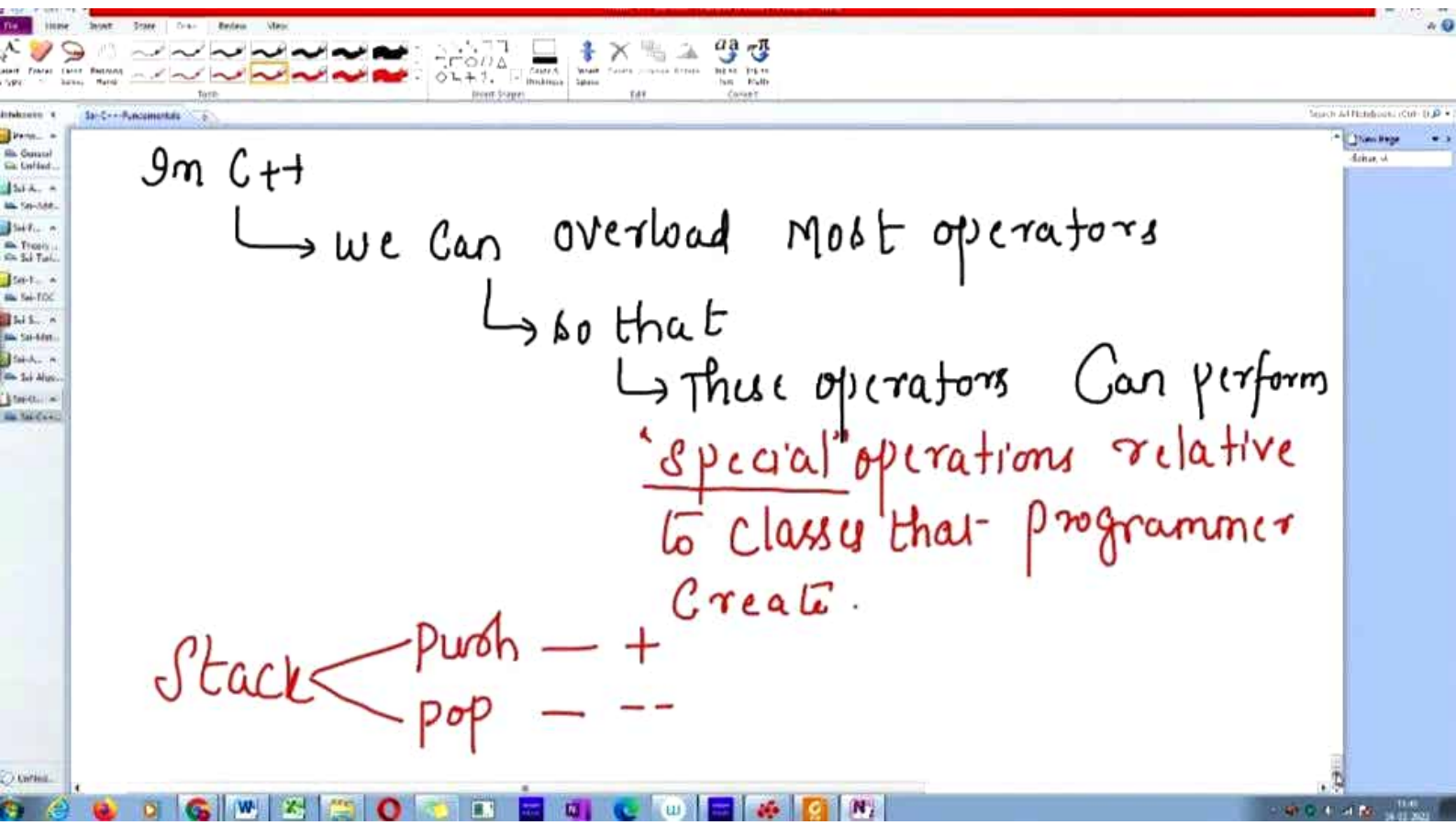
In C++

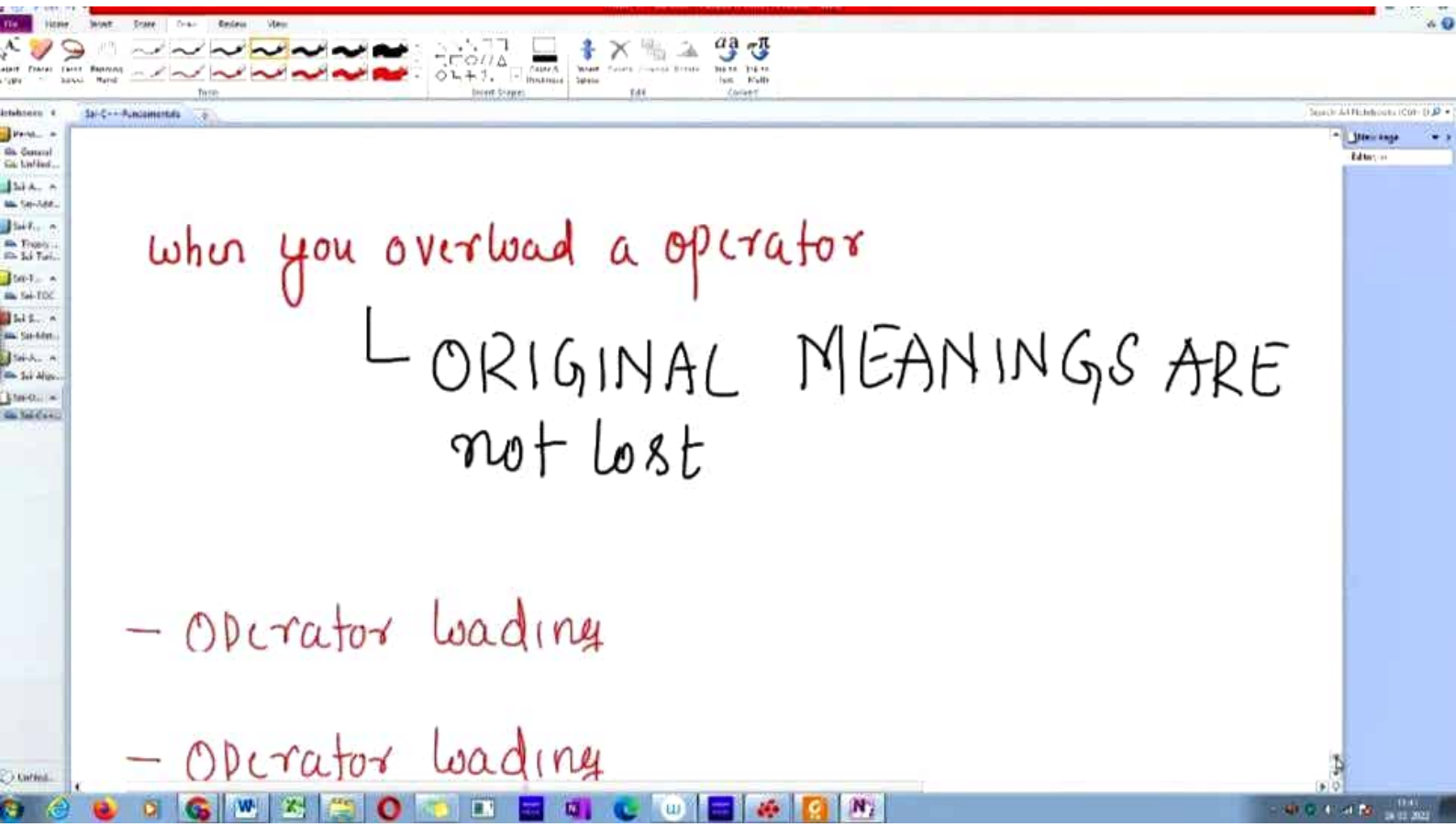
↳ we can overload most operators

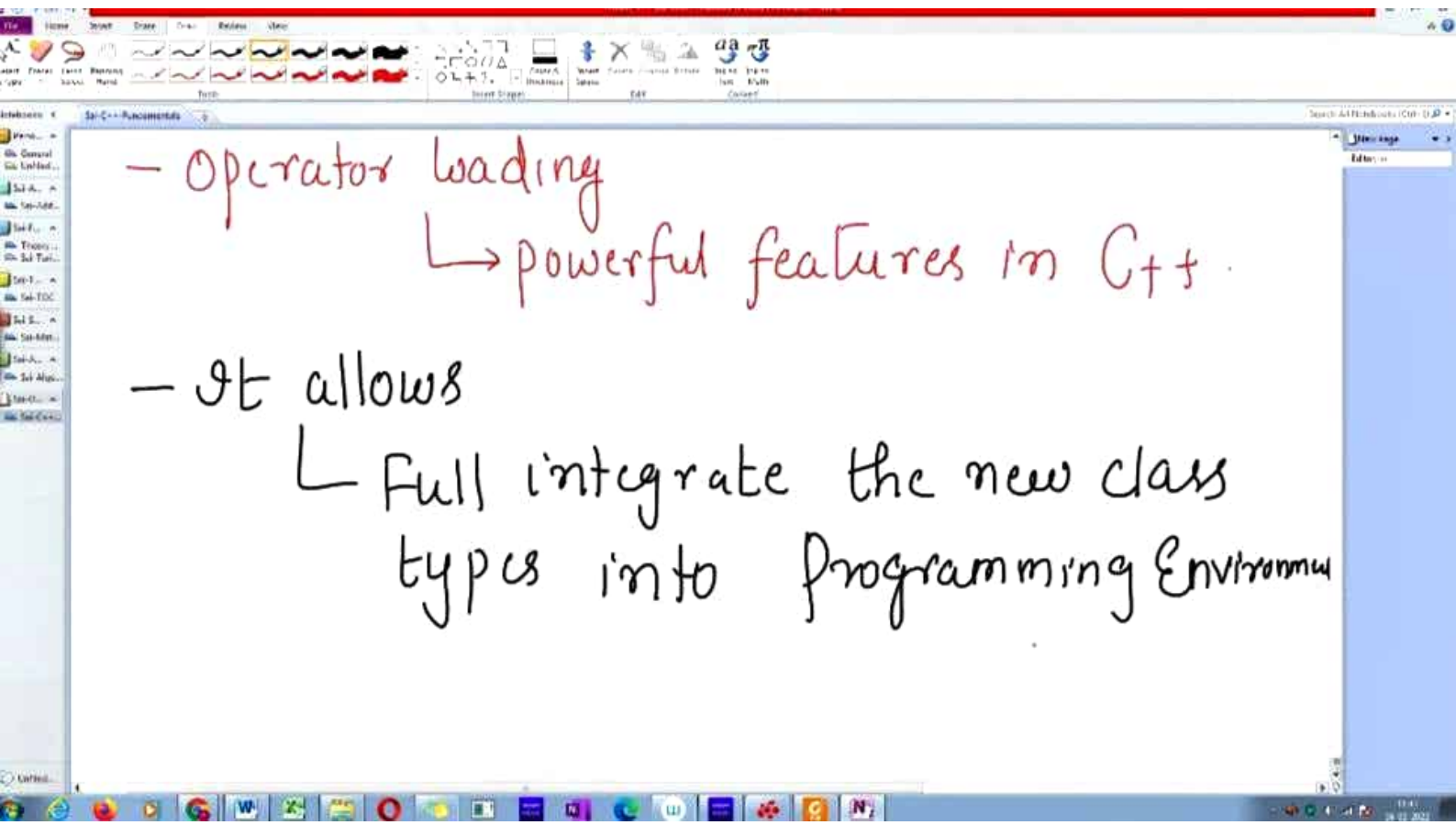
↳ so that

↳ these operators can perform 'special' operations relative to classes that programmer create.

Stack < push — +
pop — --




When you overload a operator
└ ORIGINAL MEANINGS ARE not lost
- Operator loading
- Operator loading


- Operator loading
↳ powerful features in C++.
- It allows
↳ Full integrate the new class
types into Programming Environmu

└ Full integrate the new class types into Programming Environmu

Once you overload operators

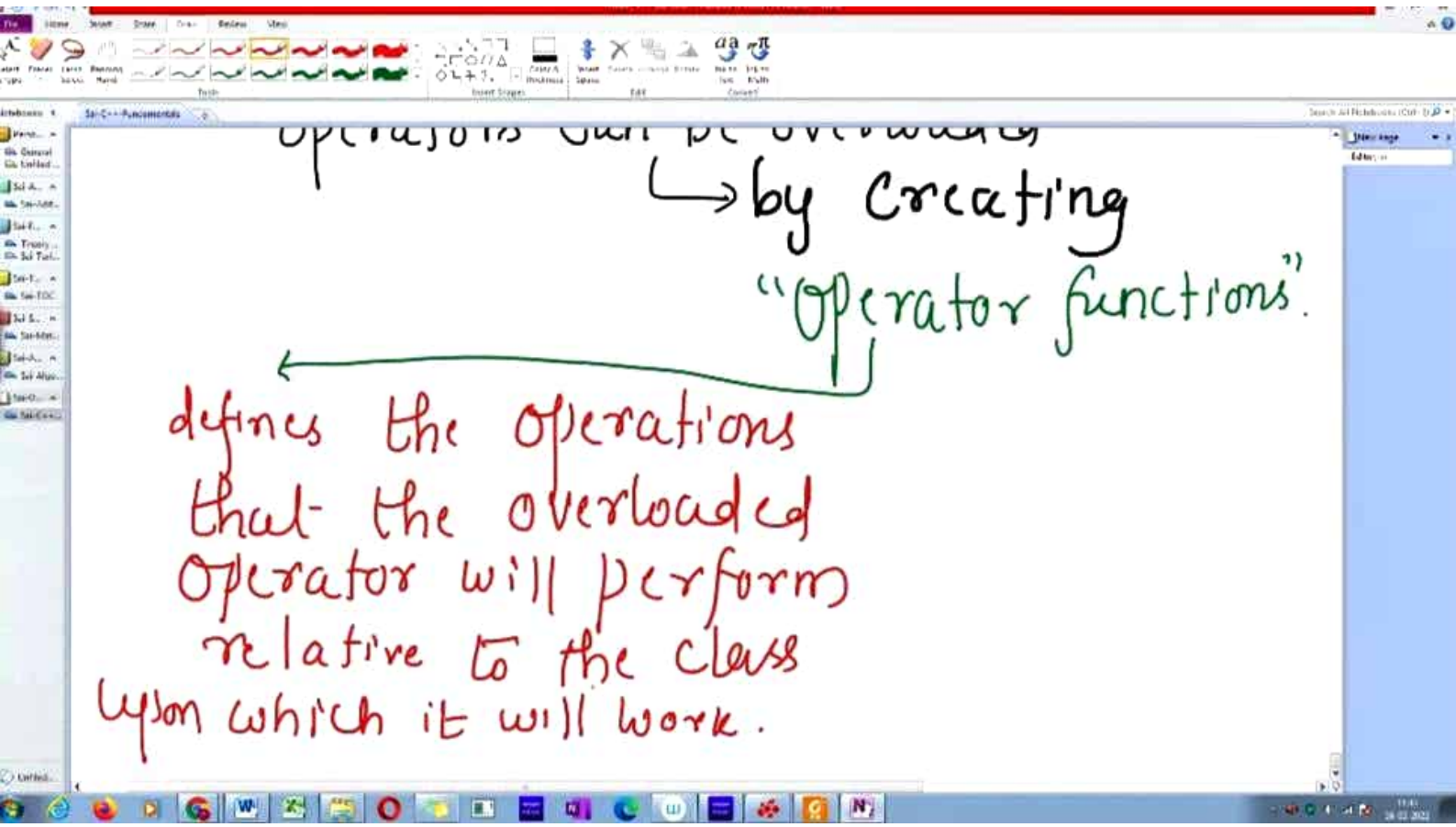
x+y
p+q
ob1+ob2

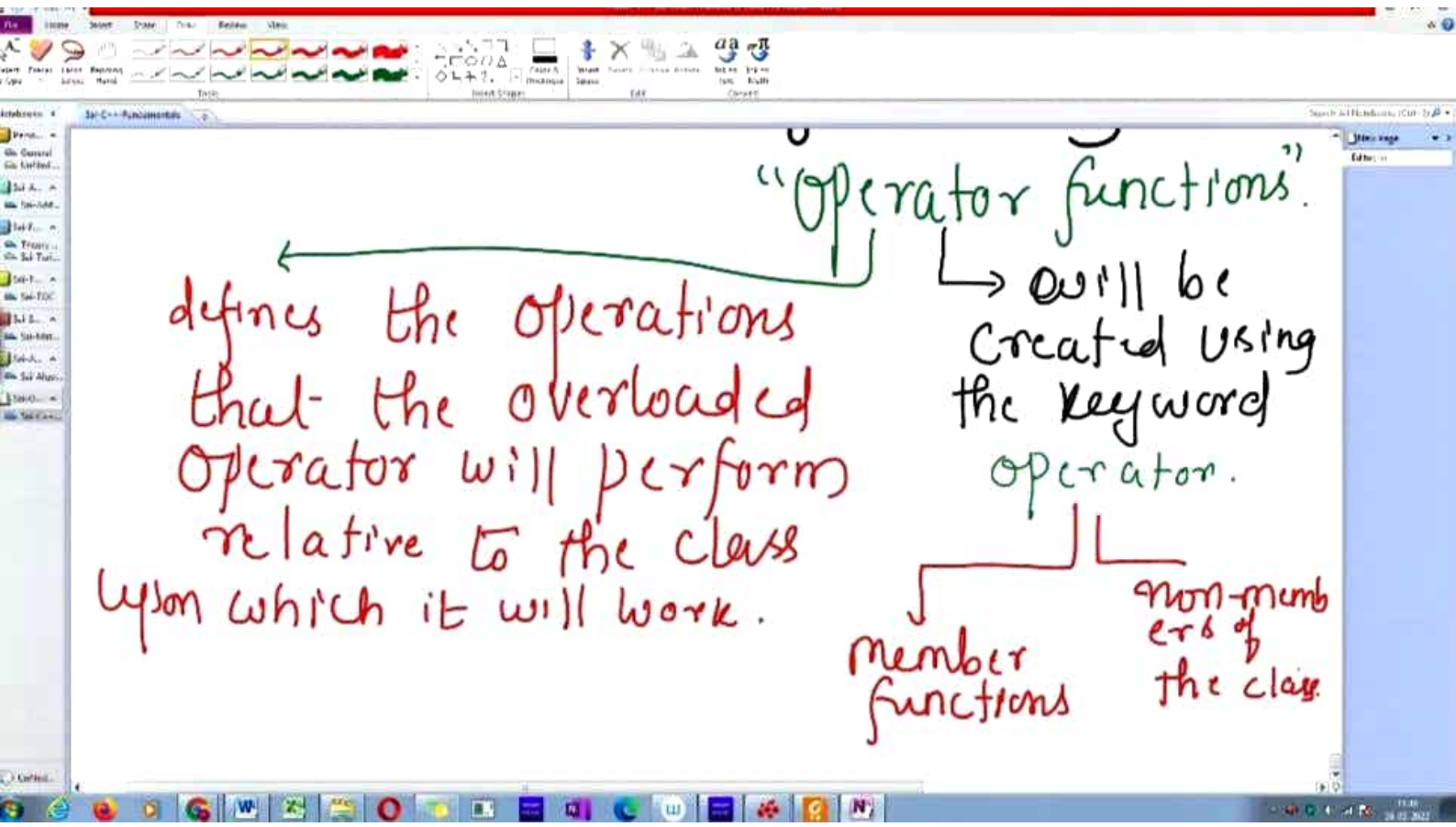
↳ I can use the objects as operands in Expressions just the same way we use Built-in data types.

# How to overload operators?

- Operators can be overloaded
  - ↳ by creating "operator functions".

defines the operations that the overloaded operator

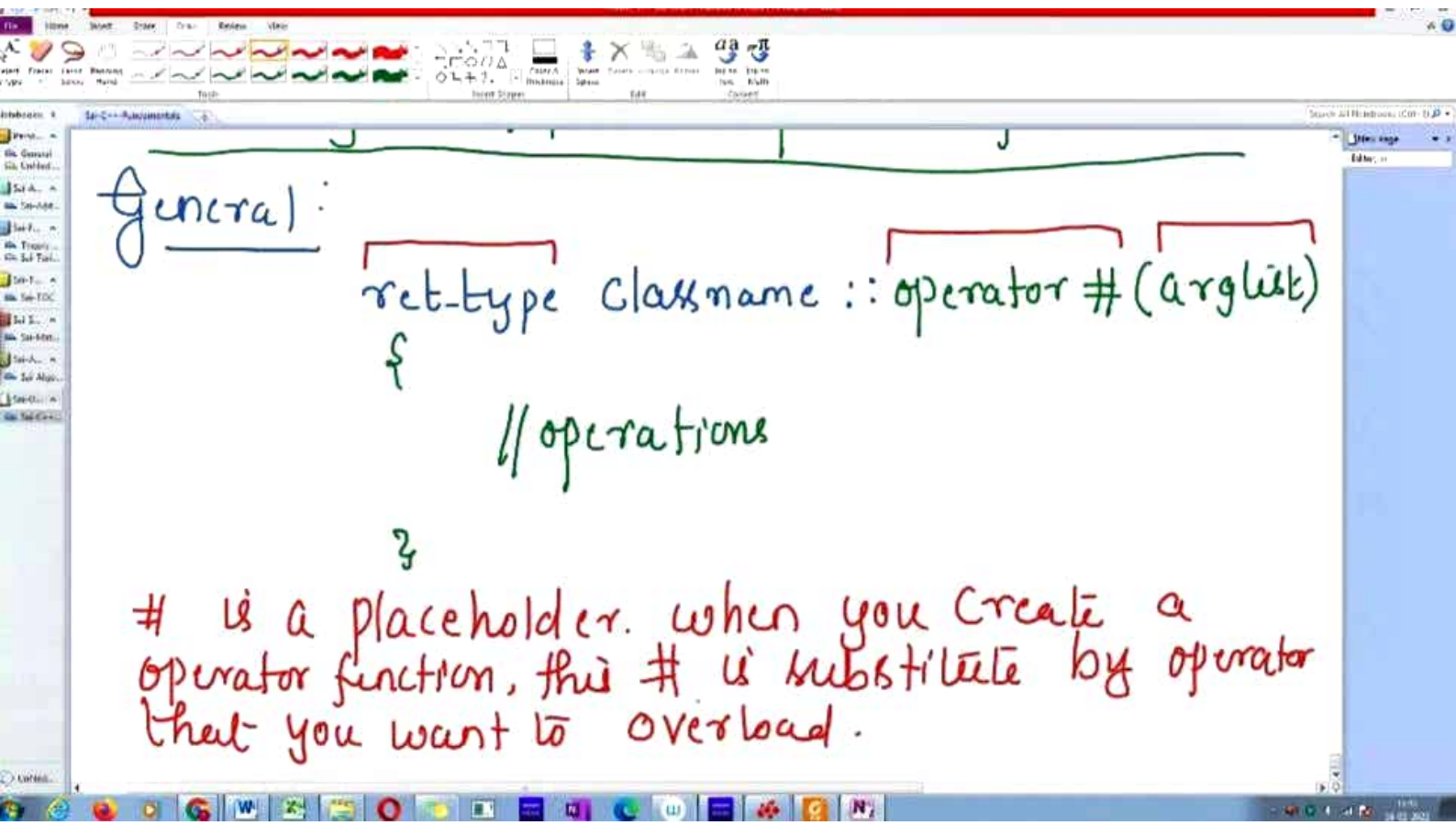

operations can be overloaded
by creating
"Operator functions".
defines the operations
that the overloaded
operator will perform
relative to the class
upon which it will work.


"Operator functions".
defines the operations that the overloaded operator will perform relative to the class upon which it will work.
will be created using the keyword operator.
member functions
non-members of the class.

# Creating a Member operator function.

General:

```
ret_type Classname :: operator # (arglist)
{
    // operations
}
```


General:
ret-type Classname :: operator # (arglist)
{
// operations
}
# is a placeholder. when you create a operator function, this # is substitute by operator that you want to overload.

General:

ret-type Classname :: operator # (arglist)

Classname.

{

// operations

}

# is a placeholder. when you create a operator function, this # is substitute by operator that you want to overload.

```
Class myclass
{  int a, b;
   public:
   myclass(){ }
   myclass(int i, int j)
   {   a=i;
       b=j;
   }
void display(){cout<<a<<" "<<b;}


};
```

```
int main()
{ myclass ob1(1,2);
  myclass ob2(3,4);
  myclass ob3;
  ob3 = ob1 + ob2;
   ob3.display();
   return 0;
}
```

```
Class myclass
{  int a, b;
   public:
   myclass(){ }
   myclass(int i, int j)
   {   a=i;
       b=j;
   }
void display(){cout<<a<<" "<<b;}
   myclass operator+(myclass o);

};
```

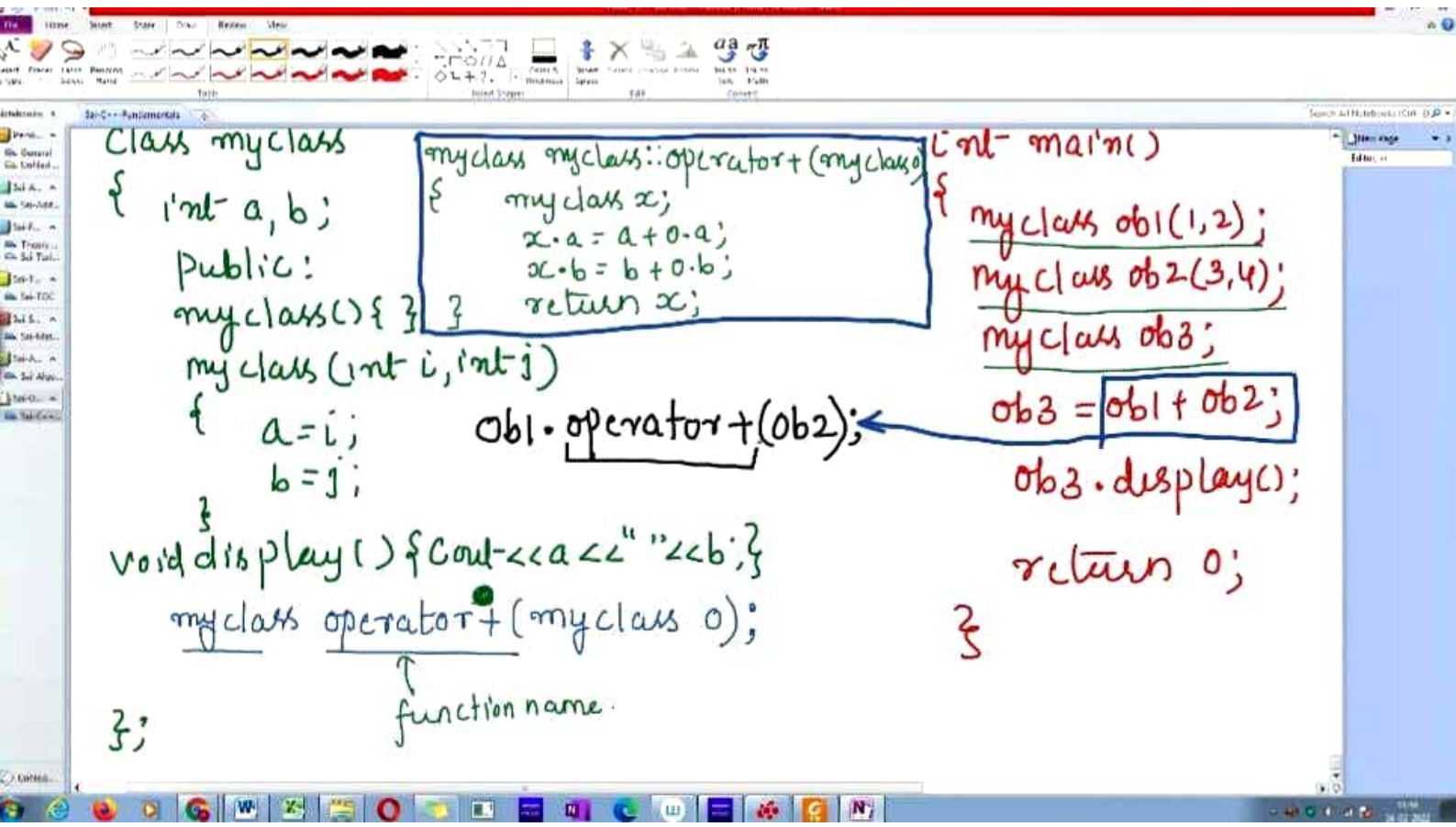
```
int main()
{  myclass ob1(1,2);
   myclass ob2(3,4);
   myclass ob3;
   ob3 = ob1 + ob2;
   ob3.display();
   return 0;
}
```

```
Class myclass
{   int a, b;
    public:
    myclass(){ }
    myclass(int i, int j)
    {   a=i;
        b=j;
    }
void display(){cout<<a<<" "<<b;}
    myclass operator+(myclass o);
};
```

```
myclass myclass::operator+(myclass o)
{   myclass x;
    x.a = a+o.a;
    x.b = b+o.b;
}   return x;
```

function name.

```
int main()
{
    myclass ob1(1,2);
    myclass ob2(3,4);
    myclass ob3;
    ob3 = ob1 + ob2;
    ob3.display();
    return 0;
}
```


Class myclass
{ int a, b;
public:
myclass(){ }
myclass (int i, int j)
{ a=i;
b=j;
}
void display(){cout<<a<<" "<<b;}
myclass operator+(myclass o);
};
function name.
myclass myclass::operator+(myclass o)
{ myclass x;
x.a = a+o.a;
x.b = b+o.b;
return x;
}
int main()
{
myclass ob1(1,2);
myclass ob2(3,4);
myclass ob3;
ob3 = ob1 + ob2;
ob3.display();
return 0;
}
ob1.operator+(ob2);

Suppose you are overloading Binary Operator Using the operator function as a member furiction, how many parameteers you have to pass?

Suppose you are overloading a Unary Operator Using the Operator function as a member functions

Using the operator function as a member function, how many parameters you have to pass?

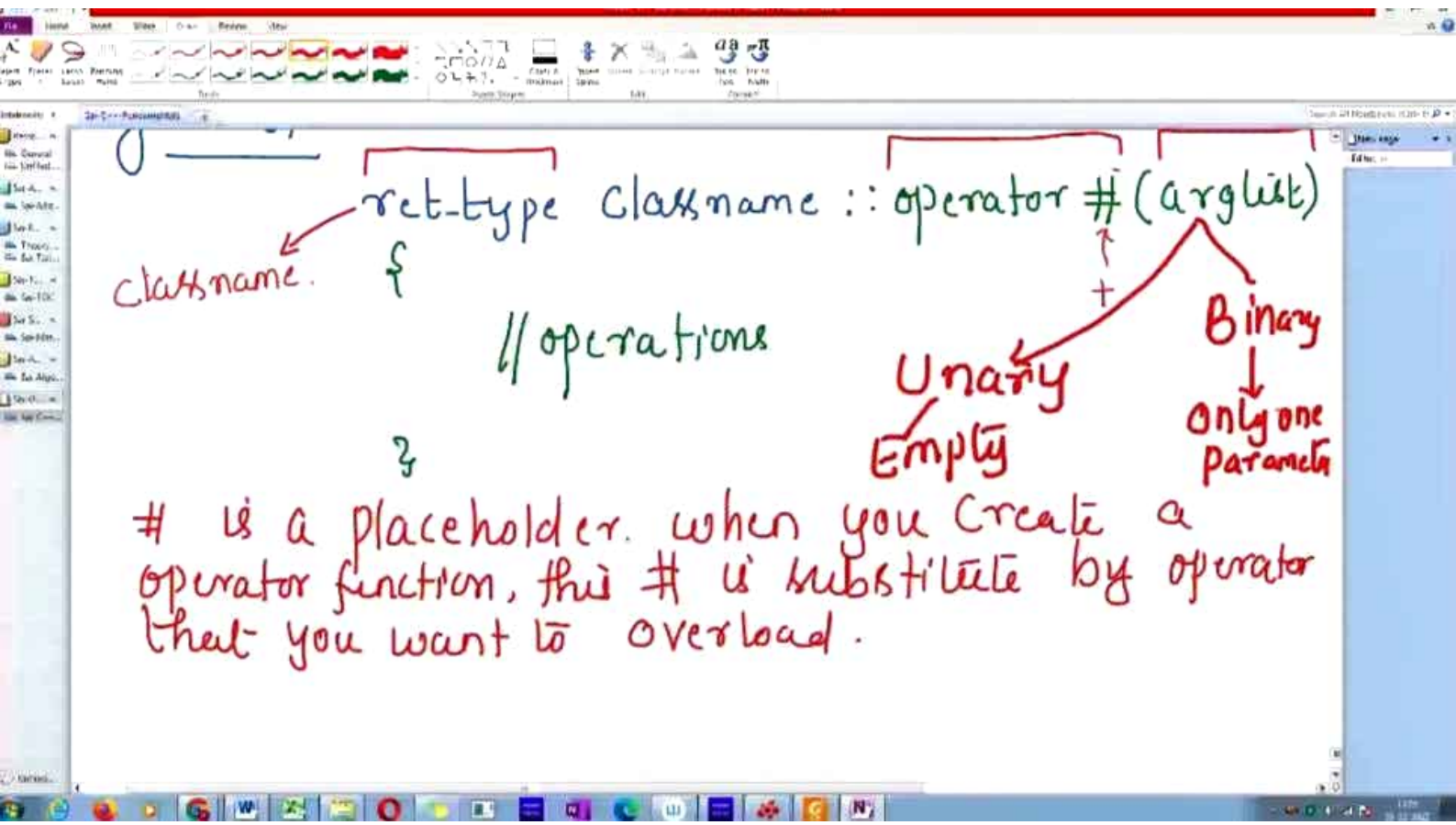
Suppose you are overloading a unary operator using the operator function as a member functions, how many parameters you have to pass?

Using the operator function as a member function, how many parameters you have to pass?

ONE

Suppose you are overloading a unary operator using the operator function as a member functions, how many parameters you have to pass?

NONE



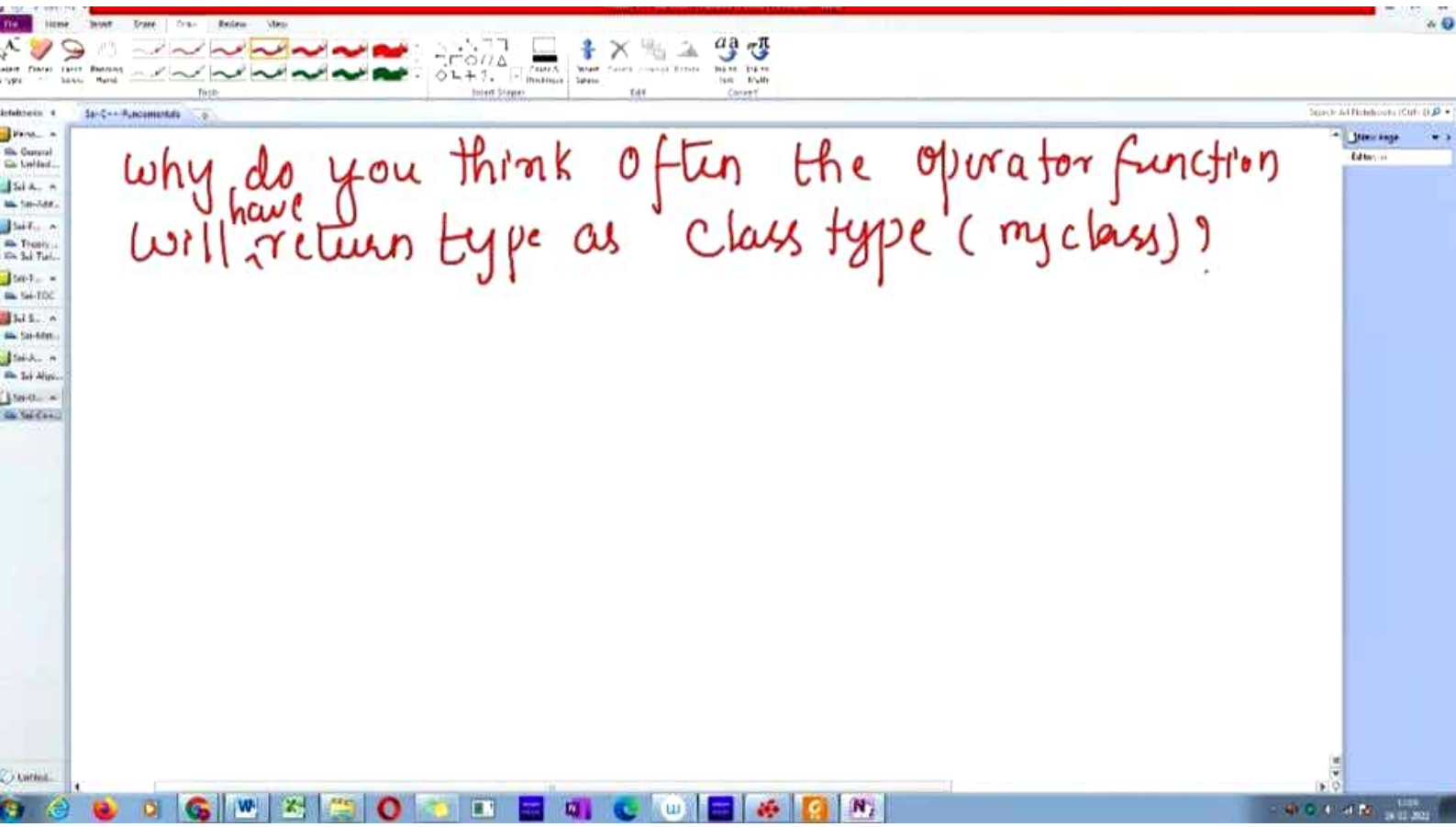

# is a placeholder. when you create a operator function, this # is substitute by operator that you want to overload.

```
myclass myclass::operator+(myclass *this, myclass o)
{
        myclass x;
        x.a = this->a + o.a;
        x.b = this->b + o.b;
        return x;
}
```

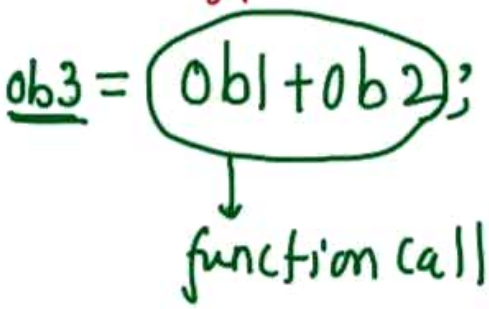
```
myclass myclass::operator+(myclass *this, myclass o)
{
        myclass x;
        x.a = this->a + o.a;
        x.b = this->b + o.b;
        return x;
}
```

ob1.operator+(ob2)

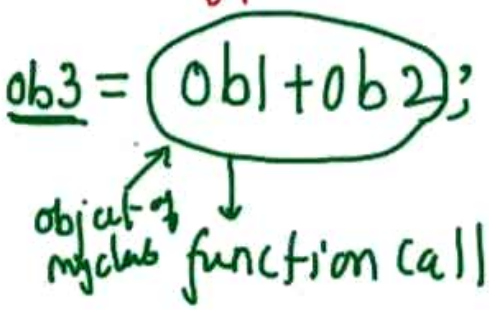
ob1 + ob2;


Why do you think often the operator function will have return type as class type (myclass)?

Why do you think often the operator function will have return type as class type (myclass)?



Why do you think often the operator function will have return type as class type (myclass)?

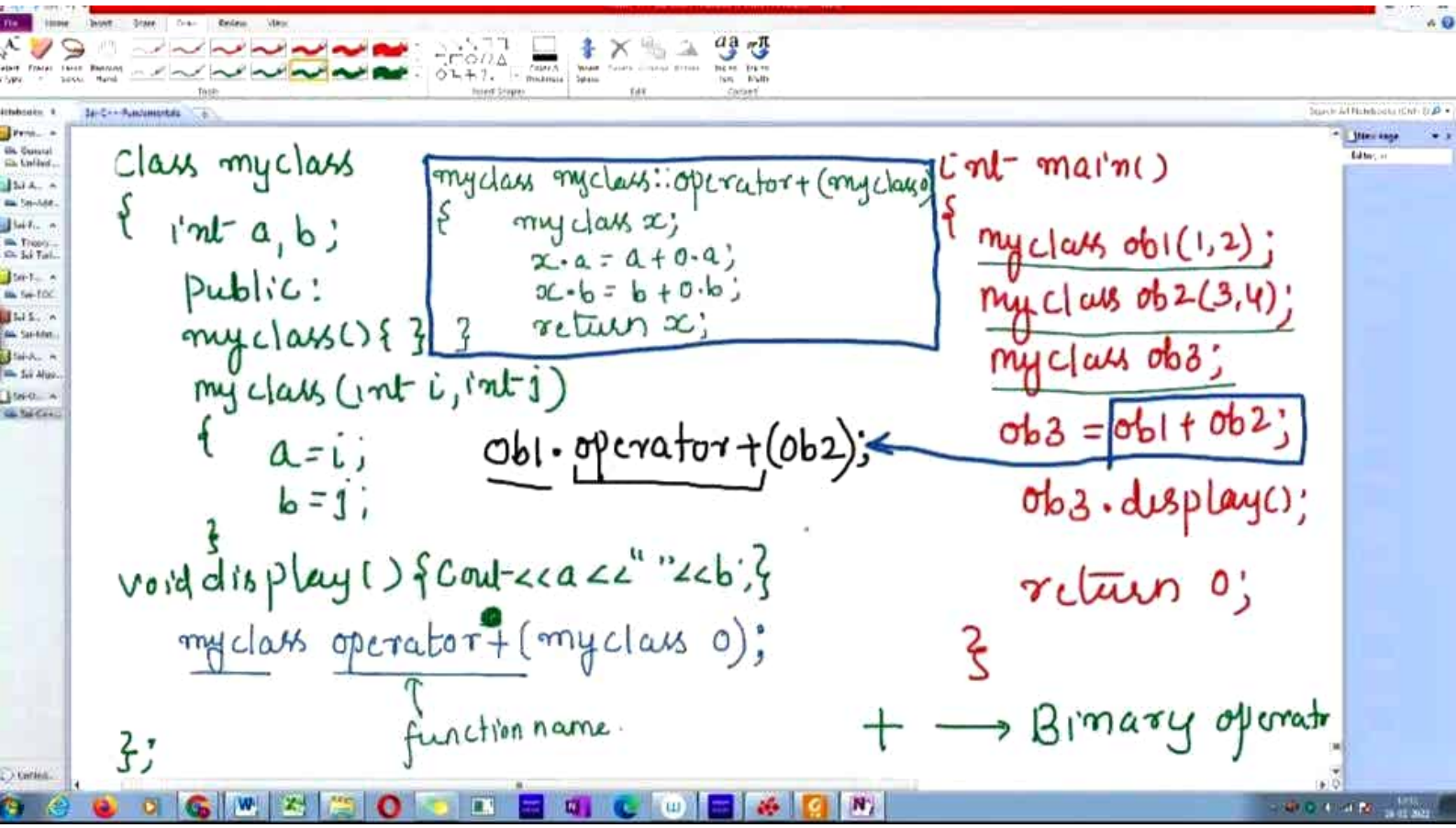


Operator Using the Operator function as a member functions, how many parameters you have to pass? NONE

on ob1 + ob2 $\Longleftrightarrow$ ob1.operator+(ob2);

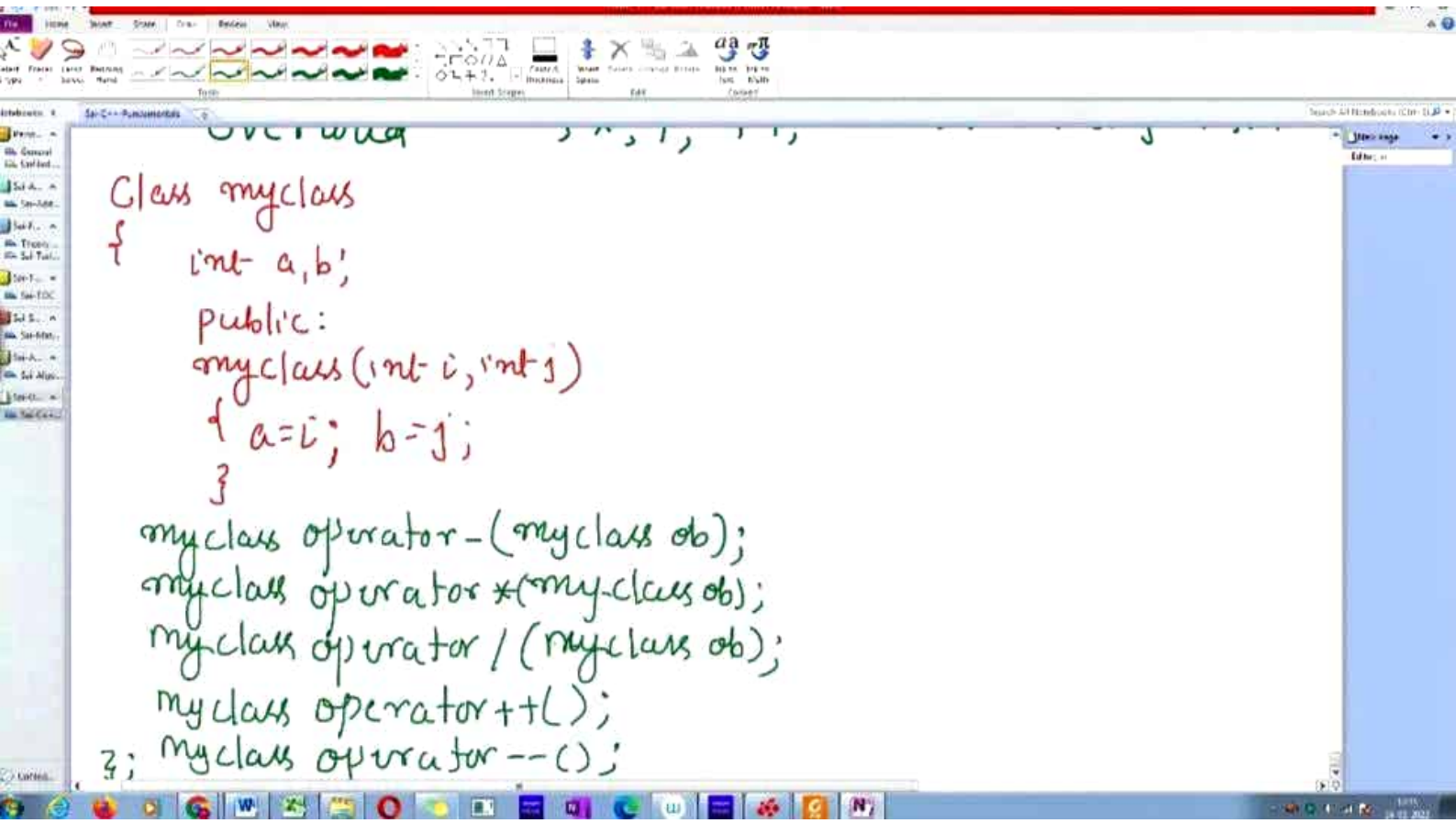
Invoking object

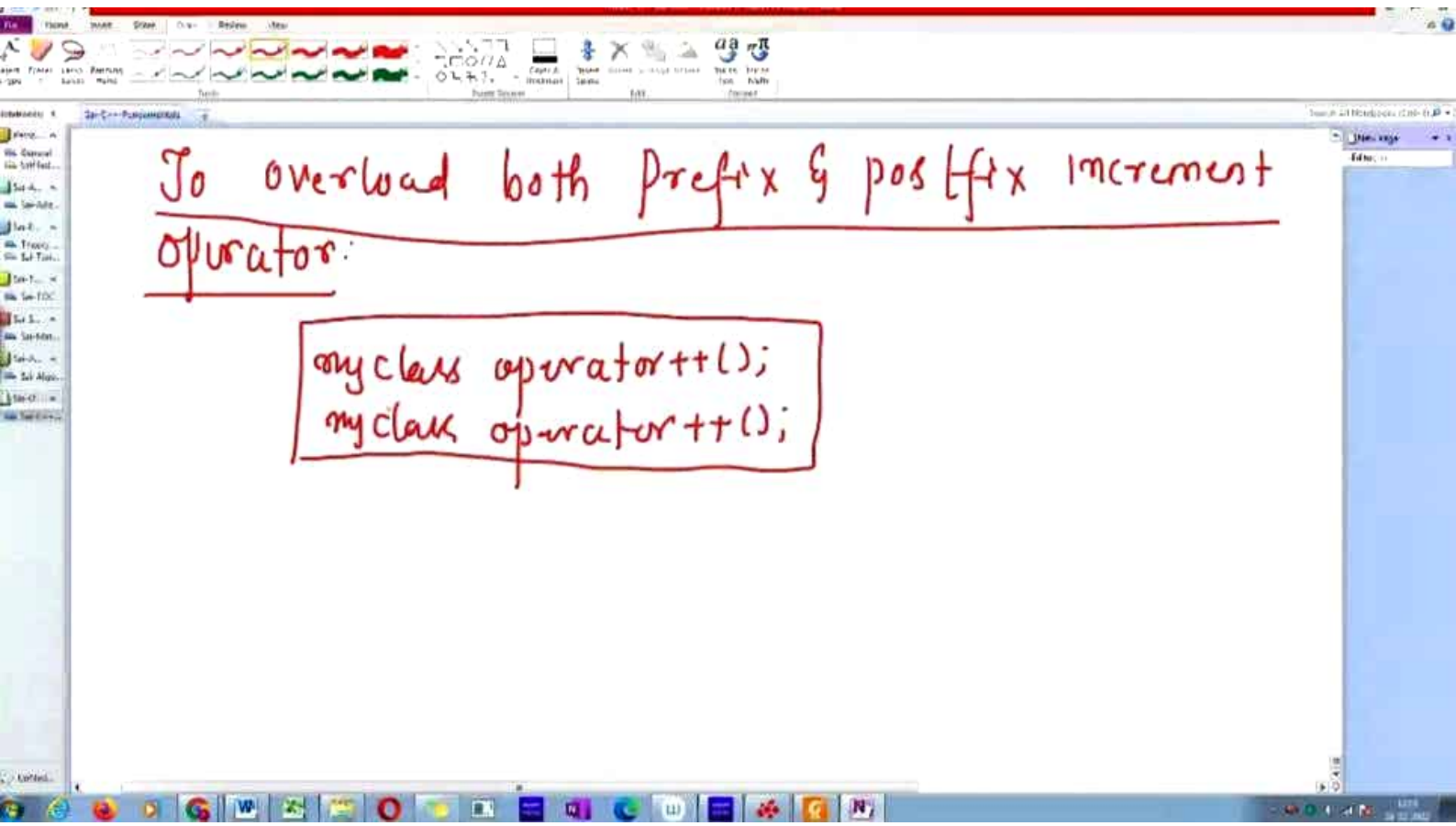
It will be passed


class myclass
{ int a, b;
public:
myclass() { }
myclass(int i, int j)
{ a=i;
b=j;
}
void display() {cout<<a<<" "<<b;}
myclass operator+(myclass o);
};
myclass myclass::operator+(myclass o)
{ myclass x;
x.a = a + o.a;
x.b = b + o.b;
return x;
}
int main()
{
myclass ob1(1,2);
myclass ob2(3,4);
myclass ob3;
ob3 = ob1 + ob2;
ob1.operator+(ob2);
ob3.display();
return 0;
}
function name.
+ → Binary operator

Overload +, −, *, /, ++, −− as member functions.

```
Class myclass
{
    int a,b;
    public:
    myclass(int i,int j)
    { a=i; b=j;
    }
```


Class myclass
{
int a,b;
public:
myclass(int i,int j)
{ a=i; b=j;
}
myclass operator-(myclass ob);
myclass operator*(myclass ob);
myclass operator/(myclass ob);
myclass operator++();
}; myclass operator--();

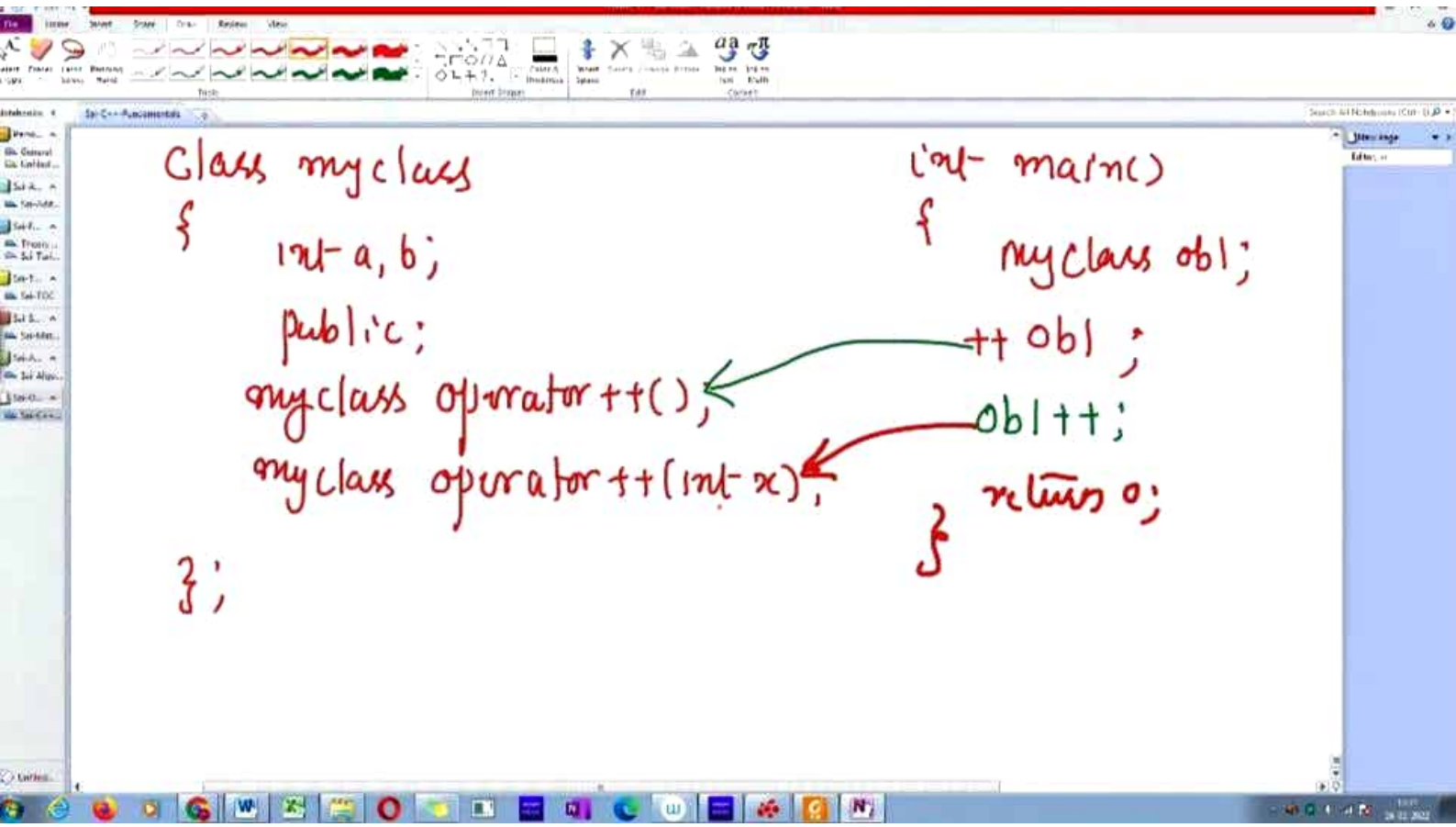

To overload both prefix & postfix increment operator:
myclass operator++();
myclass operator++();

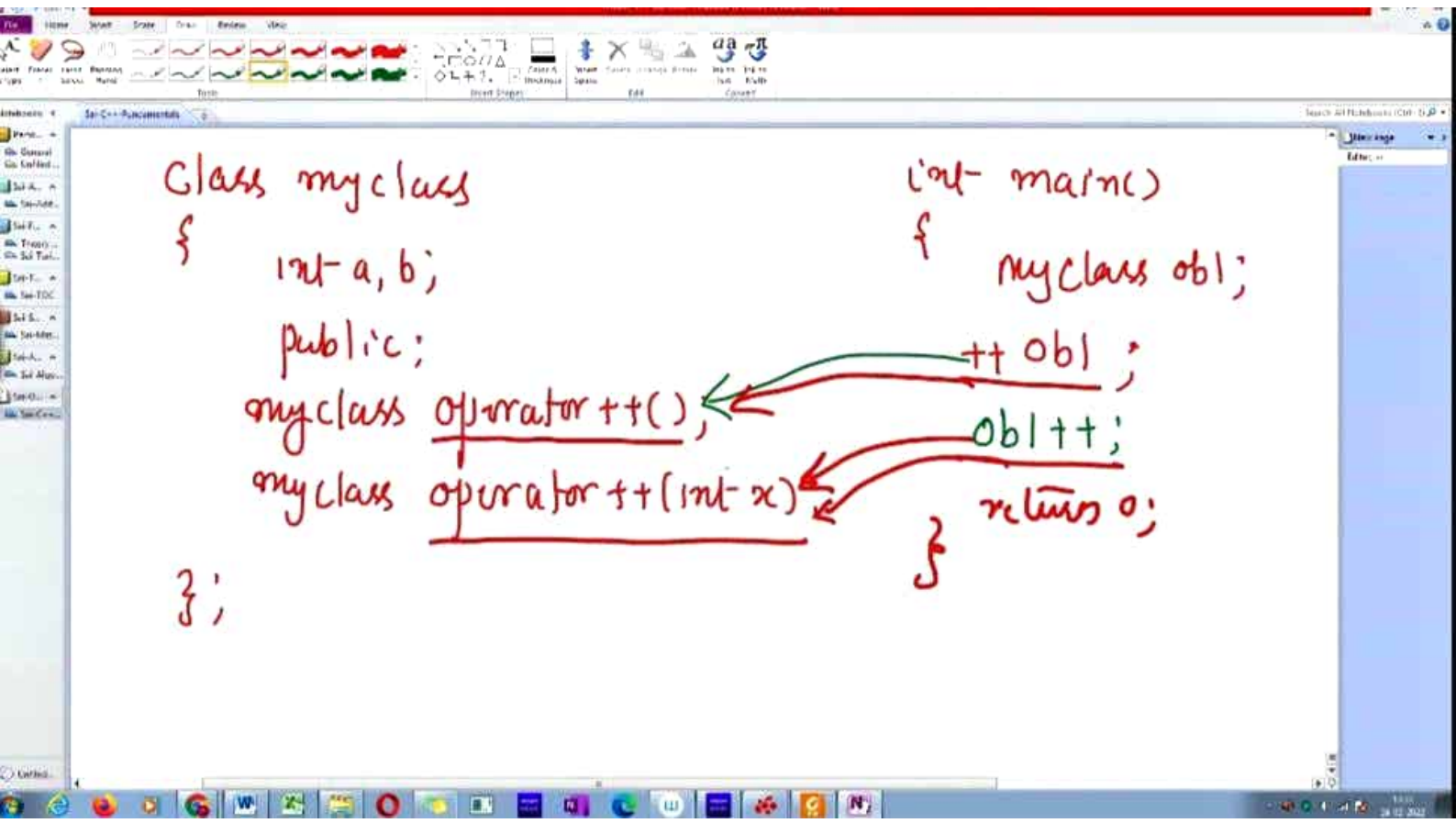
General form for prefix:

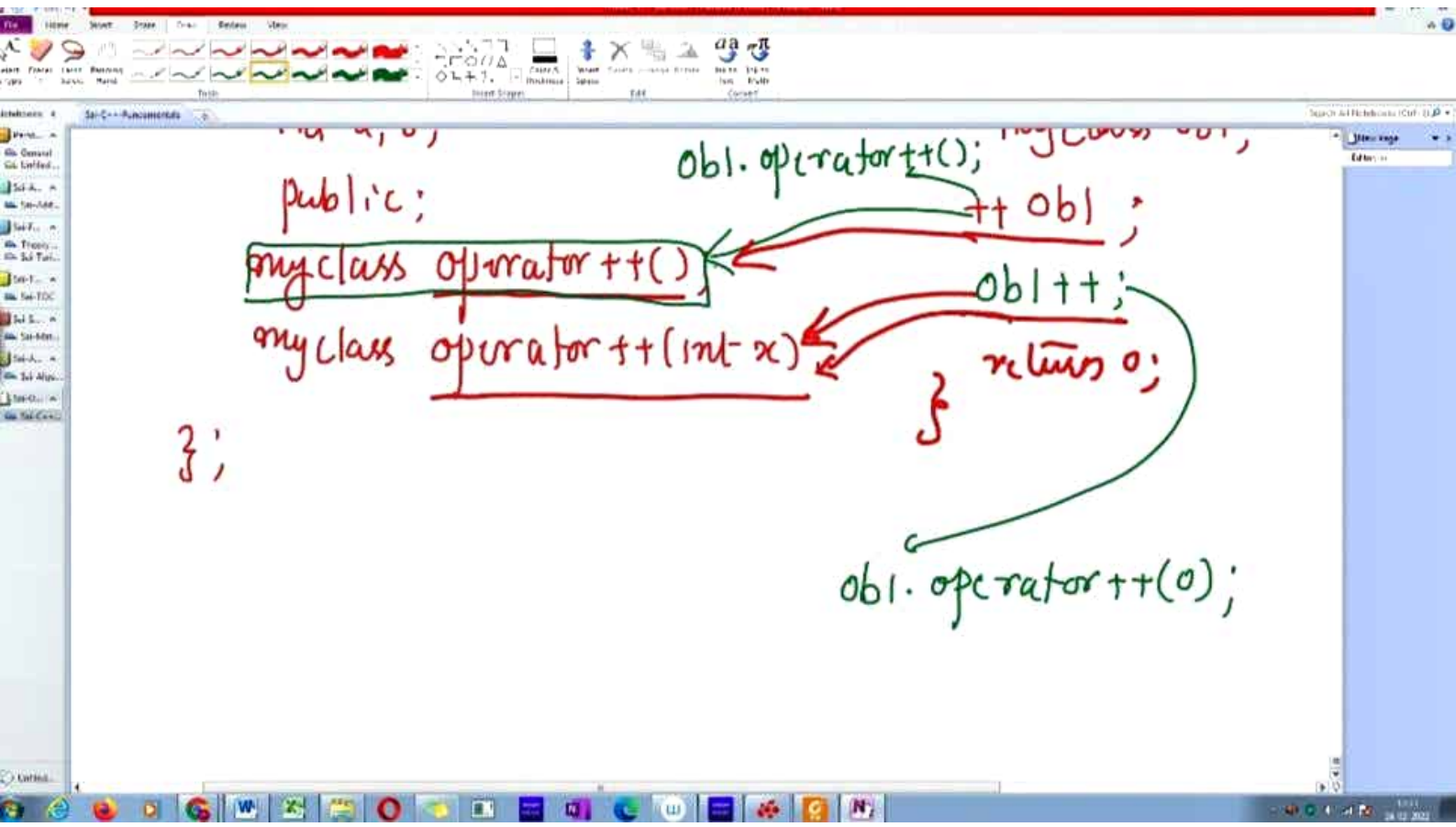
```
type operator++()
{
}
type operator--()
{
}
```

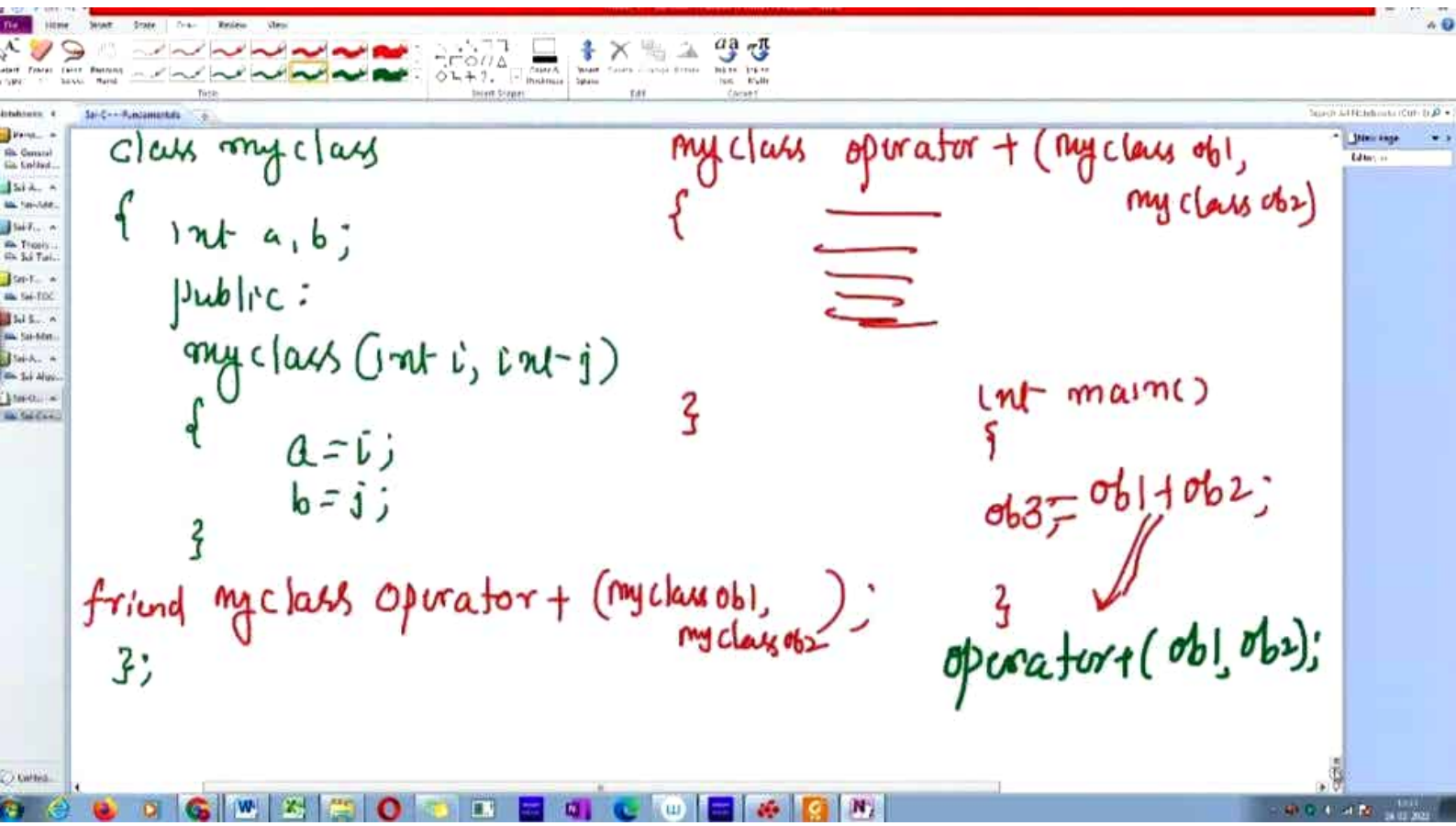
Postfix:

```
type operator++(int x)
{
}
type operator--(int x)
{
}
```


Class myclass
{
int a, b;
public;
myclass operator++();
myclass operator++(int x);
};
int main()
{
myclass obl;
++obl;
obl++;
return 0;
}


Class myclass
{
int a, b;
public;
myclass operator ++();
myclass operator ++(int x)
};
int main()
{
myclass obl;
++obl;
obl++;
return 0;
}


obl.operator++();
public:
++obl;
myclass operator++()
obl++;
myclass operator++(int x)
return 0;
};
obl.operator++(0);


class myclass
{ int a, b;
public:
myclass (int i, int j)
{
a = i;
b = j;
}
friend myclass operator + (myclass ob1, myclass ob2);
};
myclass operator + (myclass ob1, myclass ob2)
{
}
int main()
{
ob3 = ob1 + ob2;
}
operator + (ob1, ob2);

```
class myclass
{ int a,b;
  public:
  myclass(int i, int j)
  {
     a=i; b=j;
  }
myclass operator+(myclass ob1);
};
```

SAME

```
struct myclass
{
  myclass(int i, int j)
  {  a=i; b=j;
  }
myclass operator+(myclass ob);
  private:
     int a,b;
};
```