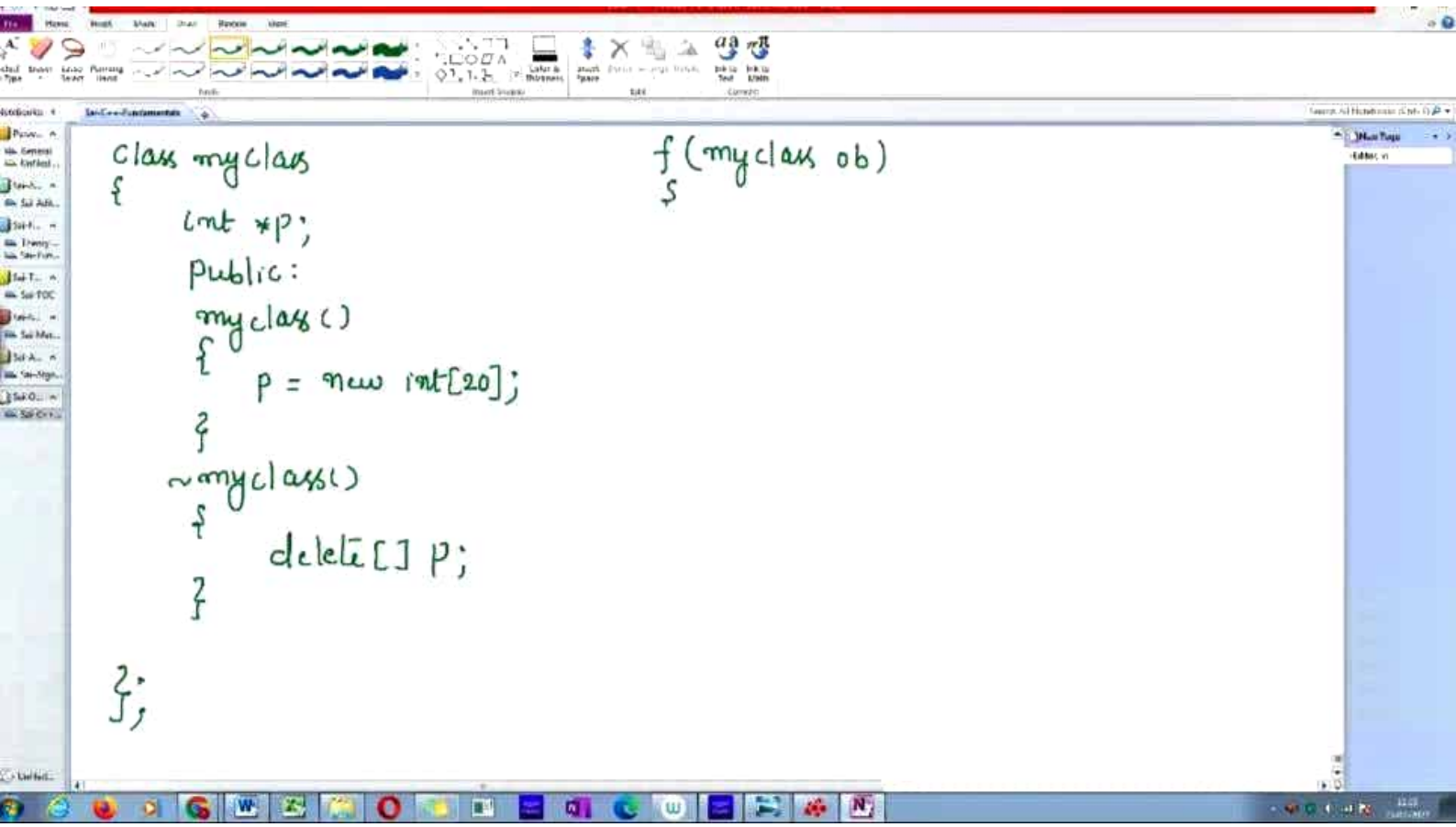
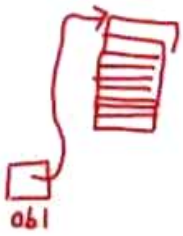

class myclass
{
int *p;
public:
myclass()
{
p = new int[20];
}
~myclass()
{
delete[] p;
}
};
f(myclass ob)
{

```
Class myclass
{
    int *p;
    Public:
     myclass()
     {
        p = new int[20];
     }
    ~myclass()
     {
        delete[] p;
     }

};
```

```
void f(myclass ob)
    {
        Cout<<"Hello";
    }

int main()
{
        myclass ob1;
        Cout<<"world";
         return 0;
}
```

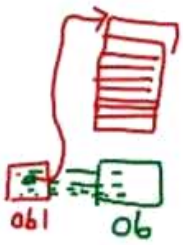
```
class myclass
{
    int *p;
  public:
    myclass()
    {
        p = new int[20];
    }
   ~myclass()
    {
        delete[] p;
    }
};
```

```
void f(myclass ob)
{
    cout<<"Hello";
}
int main()
{
    myclass ob1;
    f(ob1);
    cout<<"world";
    return 0;
}
```



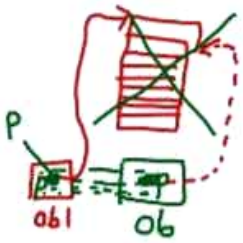
```
class myclass
{
    int *p;
  public:
    myclass()
    {
        p = new int[20];
    }
   ~myclass()
    {
        delete[] p;
    }
};
```

```
void f(myclass ob)
{
    cout<<"Hello";
}
int main()
{
    myclass ob1;
    f(ob1);
    cout<<"world";
    return 0;
}
```



```
class myclass
{
    int *p;
  public:
    myclass()
    {
        p = new int[20];
    }
   ~myclass()
    {
        delete[] p;
    }
};
```

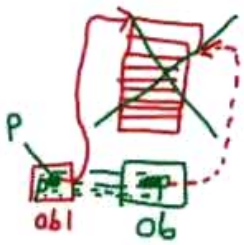
```
void f(myclass ob)
{
    cout<<"Hello";
}
int main()
{
    myclass ob1;
    f(ob1);
    cout<<"world";
    return 0;
}
```



Hello

```
class myclass
{
    int *p;
  public:
    myclass()
    {
        p = new int[20];
    }
   ~myclass()
    {
        delete[] p;
    }
};
```

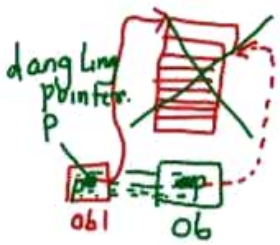
```
void f(myclass ob)
{
    cout<<"Hello";
}
int main()
{
    myclass ob1;
    f(ob1);
    cout<<"world";
    return 0;
}
```



Hello
world

```
class myclass
{
    int *p;
  public:
    myclass()
    {
        p = new int[20];
    }
   ~myclass()
    {
        delete[] p;
    }
};
```

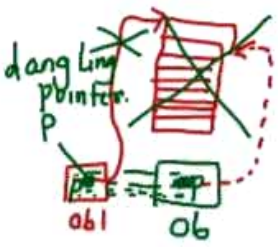
```
void f(myclass ob)
{
    cout<<"Hello";
}
int main()
{
    myclass ob1;
    f(ob1);
    cout<<"world";
    return 0;
}
```



Hello
world

```
class myclass
{
    int *p;
  public:
    myclass()
    {
        p = new int[20];
    }
   ~myclass()
    {
        delete[] p;
    }
};
```

```
void f(myclass ob)
{
    cout<<"Hello";
}
int main()
{
    myclass ob1;
    f(ob1);
    cout<<"world";
    return 0;
}
```



Hello
world

Problem: Default Copy Constructor.

Assume a class called Myclass, and let this class allocate memory for each object when object created.

Let ob1 be an object of class myclass.

Assume a class called Myclass, and let this class allocate memory for each object when object created.

Let ob1 be an object of class myclass.

↳ ob1 has memory already

Assume a class called Myclass, and let this class allocate memory for each object when object created.

Let ob1 be an object of class myclass.

↳ ob1 has memory already allocated.

```
Myclass ob = ob1;
```

If a bitwise Copy is performed.
then ob will have Exact Copy ob1.

```
f(myclass ob)
{
}
f(ob1)
```

Assume a class called myclass, and let this class allocate memory for each object when object created.

Let ob1 be an object of class myclass.

↳ ob1 has memory already allocated.

```
Myclass ob = ob1;
```

```
f(myclass ob)
{
}
f(ob1)
```

If a bitwise Copy is performed.

then ob will have Exact Copy ob1.

↳ ob will be using same allocated memory as that of ob1.

Let ob1 be an object of class myclass.

↳ ob1 has memory already allocated.

Myclass ob = ob1;

If a bitwise Copy is performed.

then ob will have Exact Copy ob1.

↳ ob will be using same allocated memory as that of ob1.

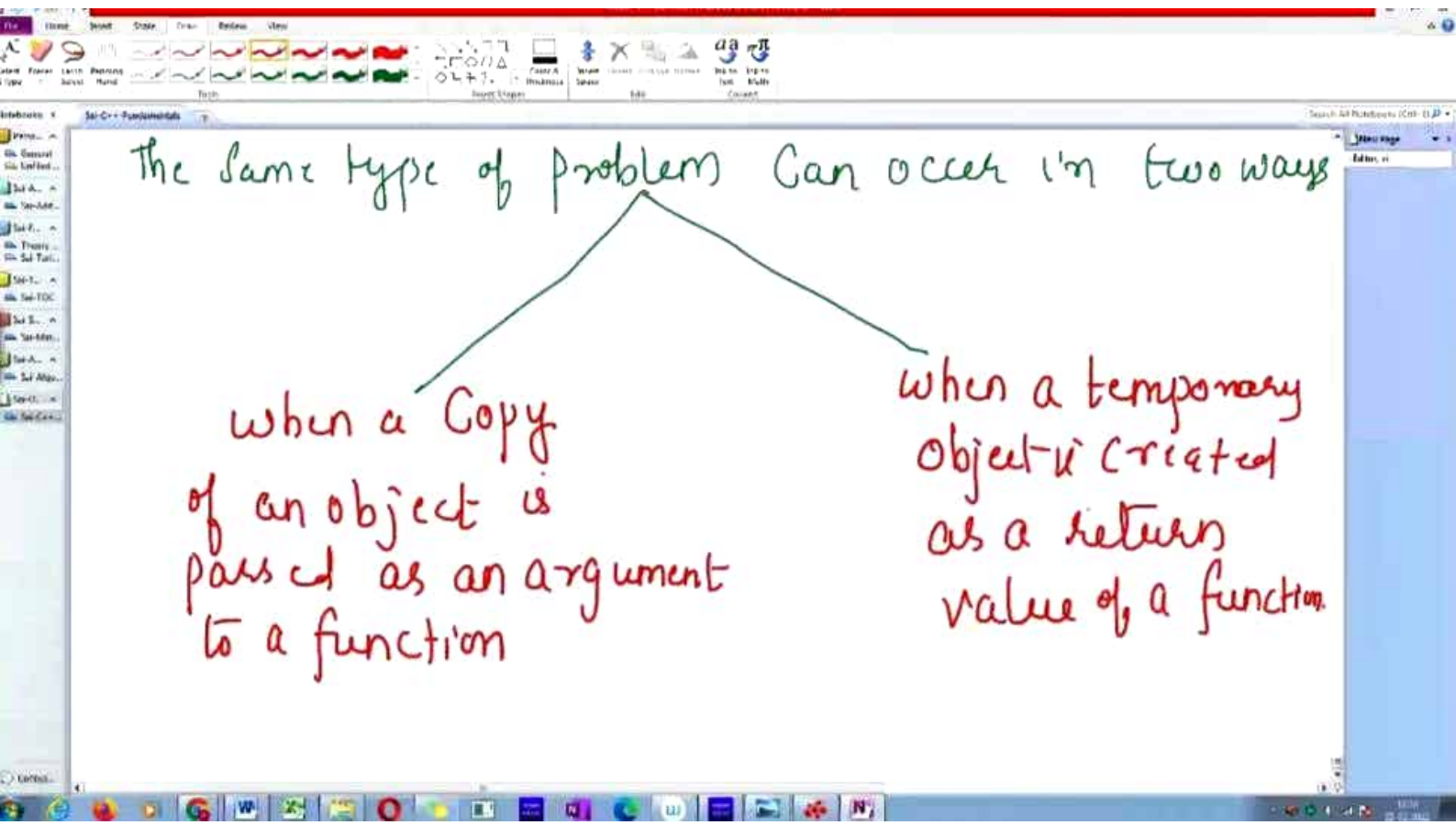
↳ Separate memory will not be allocated.

```
f(myclass ob)
{
}
f(ob1)
```

allocated.

If myclass has a destructor which deallocates memory

→ The same memory will be freed twice when ob & ob1 are destroyed.


The same type of problem can occur in two ways
when a copy of an object is passed as an argument to a function
when a temporary object is created as a return value of a function

Assume a class called Myclass, and let this class allocate memory for each object when object created.

Let ob1 be an object of class myclass.

↳ ob1 has memory already allocated.

```
Myclass ob = ob1;
```

If a bitwise Copy is performed.

If a bitwise Copy is performed.

```
f(myclass ob)
{
}
ff(ob1)
{
}
ff(ob1)
```

Let ob1 be an object of class myclass.

↳ ob1 has memory already allocated.

```
Myclass ob = ob1;
```

```
f(myclass ob)
{
}
f(ob1)
```

If a bitwise Copy is performed.

then ob will have Exact Copy ob1.

↳ ob will be using same allocated memory as that of ob1.

↳ Separate memory will not be allocated.

① when a Copy of an object is passed as an argument to a function

② when a temporary object is created as a return value of a function

```
myclass ob, ob1;
ob = ob1.
```

```
myclass ob1;
myclass ob = ob1.
```

assignment

```
int a, b;
a = b;
```

```
int b;
int a = b;
```

← initialization

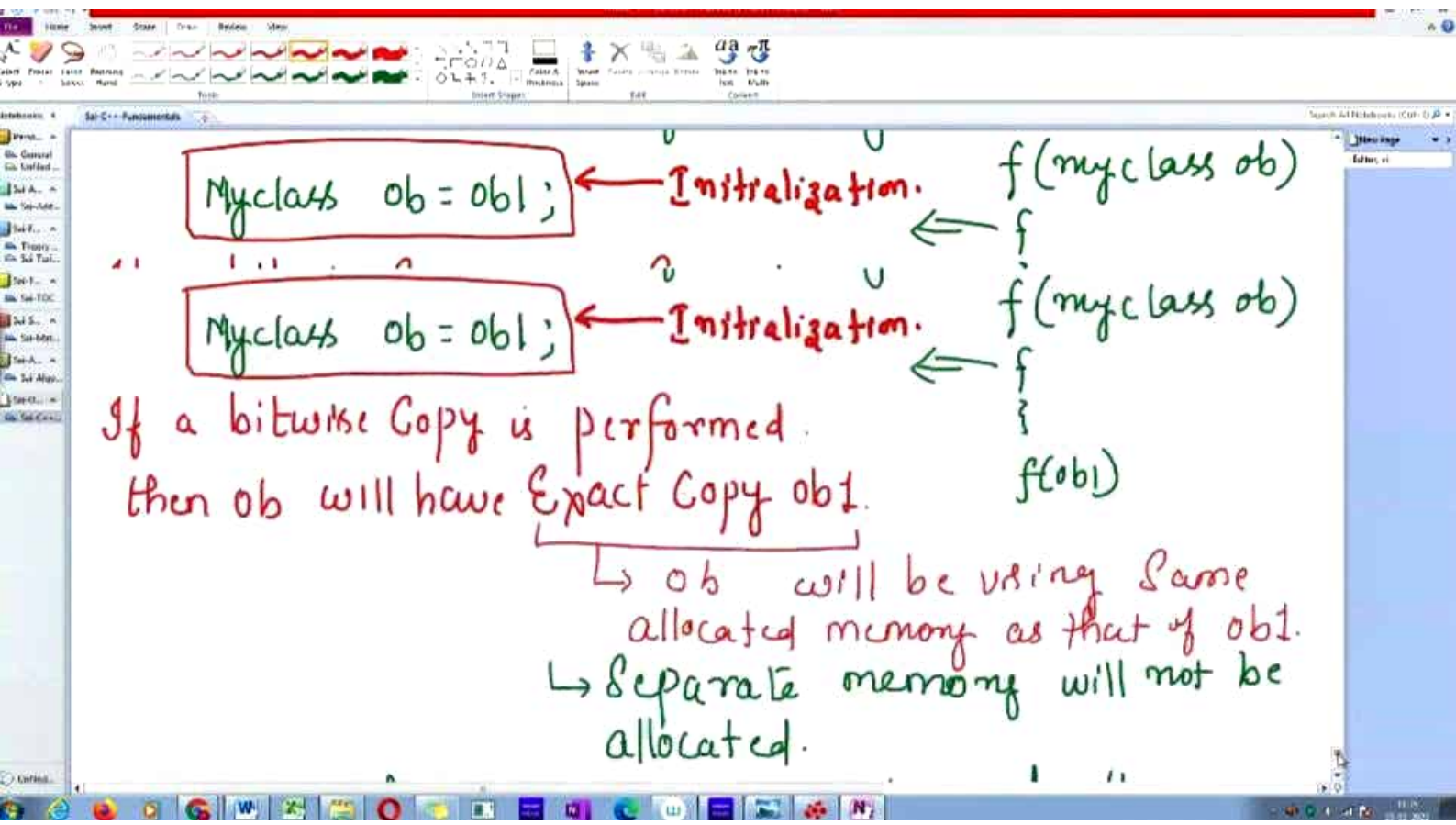
If a bitwise Copy is performed.
then ob will have Exact Copy ob1.

↳ ob will be using same allocated memory as that of ob1.
↳ Separate memory will not be allocated.

{
f(ob1)

If myclass has a destructor which deallocates memory
↳ The same memory will be freed twice when ob & ob1 are destroyed.

int main()


Myclass ob = ob1;
Initialization.
f(myclass ob)
{
}
Myclass ob = ob1;
Initialization.
f(myclass ob)
{
}
f(ob1)
If a bitwise Copy is performed.
then ob will have Exact Copy ob1.
↳ ob will be using same allocated memory as that of ob1.
↳ Separate memory will not be allocated.

a=b;

Initialization

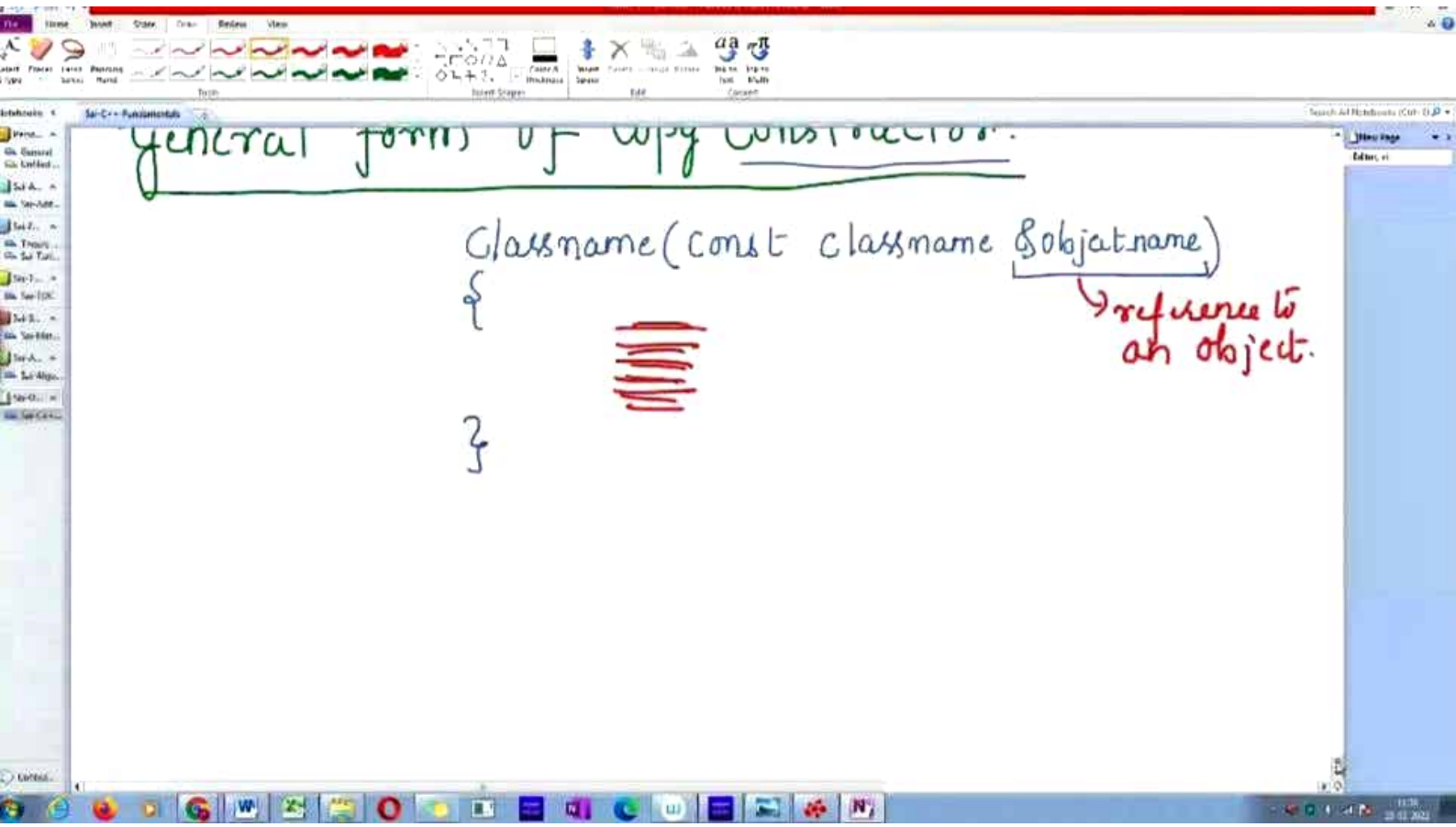
How To Solve this problem?

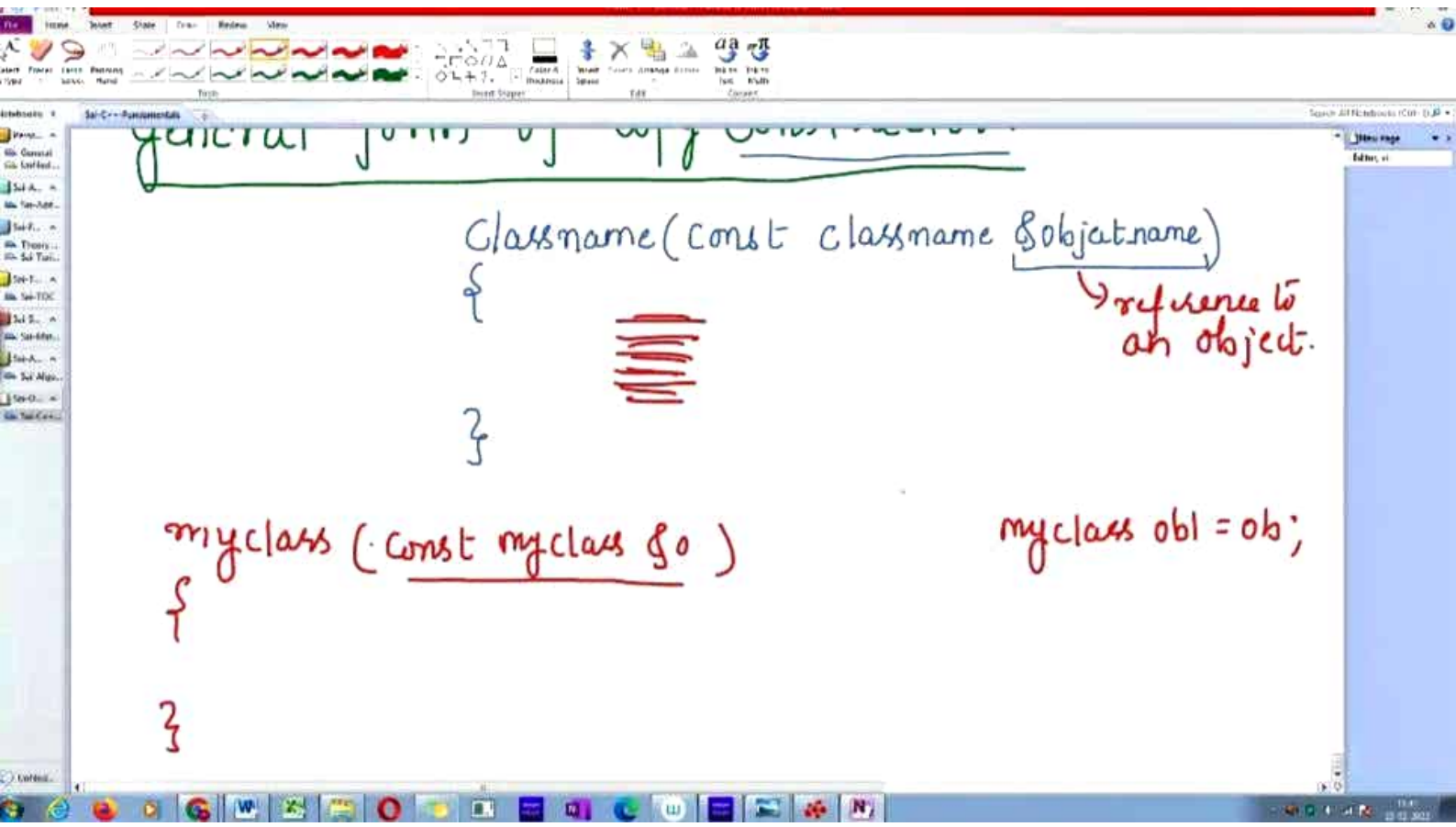
↳ How to avoid default BITWISE Copy?

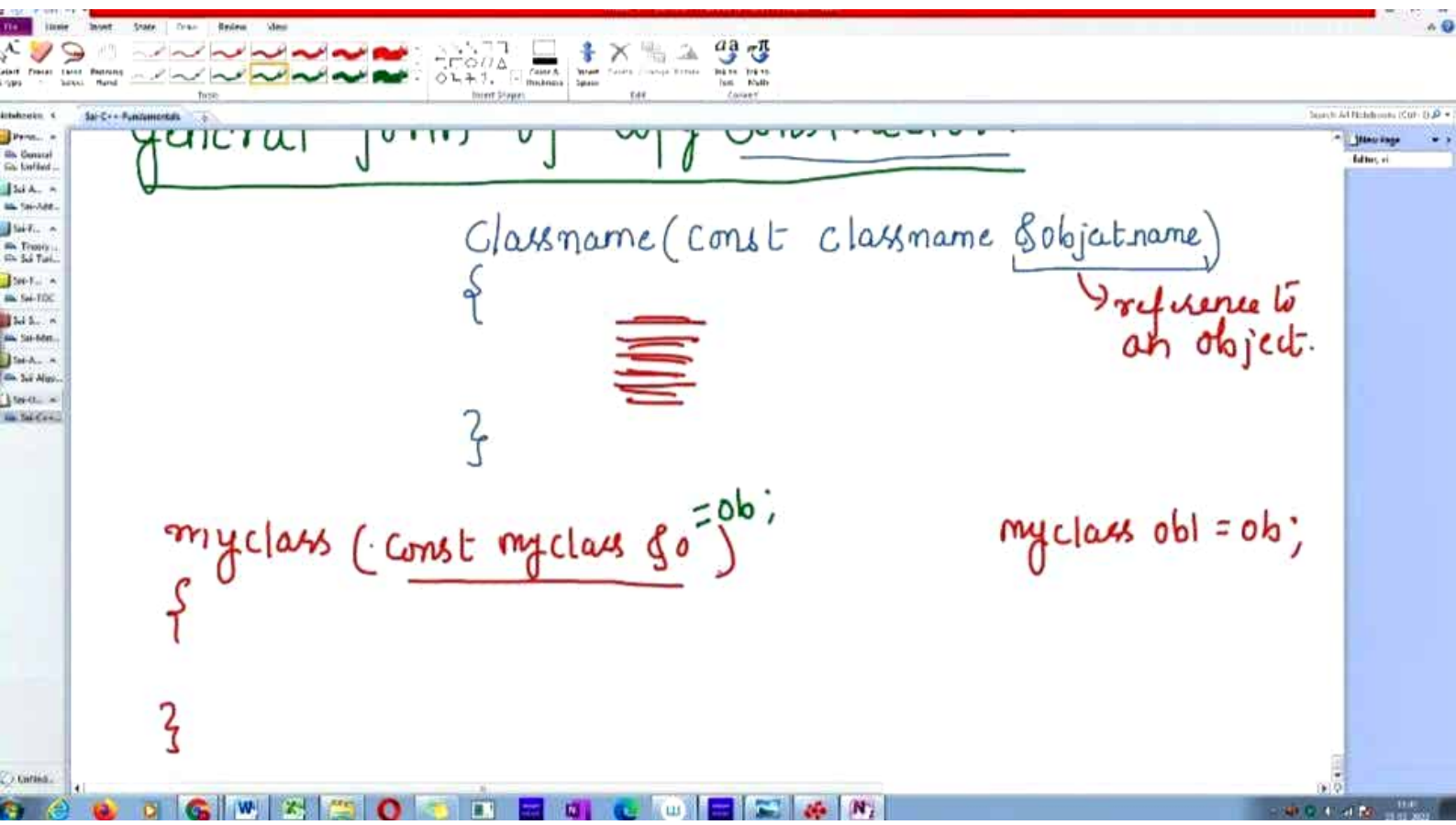
- C++ allows us to Create a Copy Constructor which the Compiler Can use when object initializes another object.

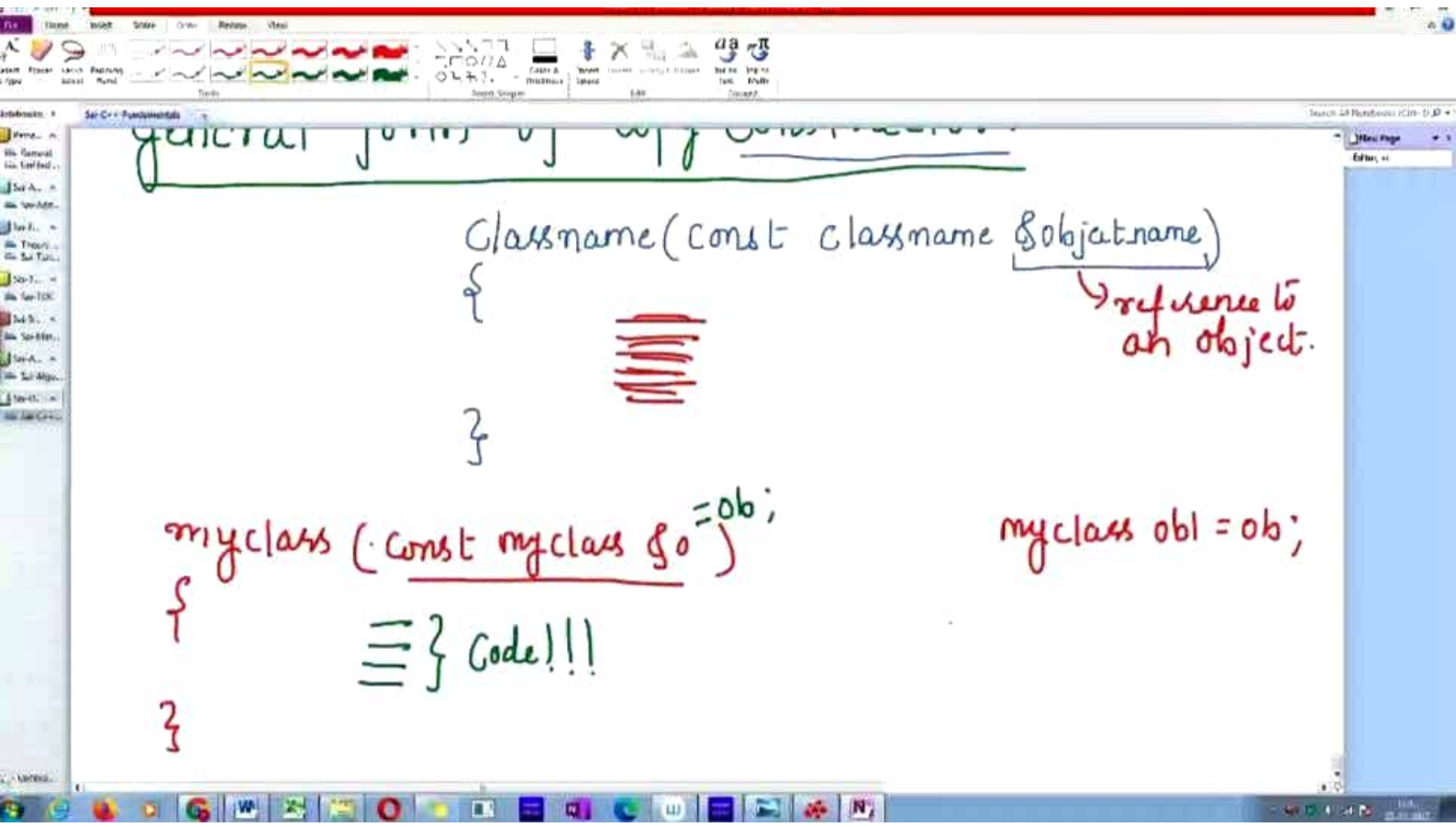
Copy?

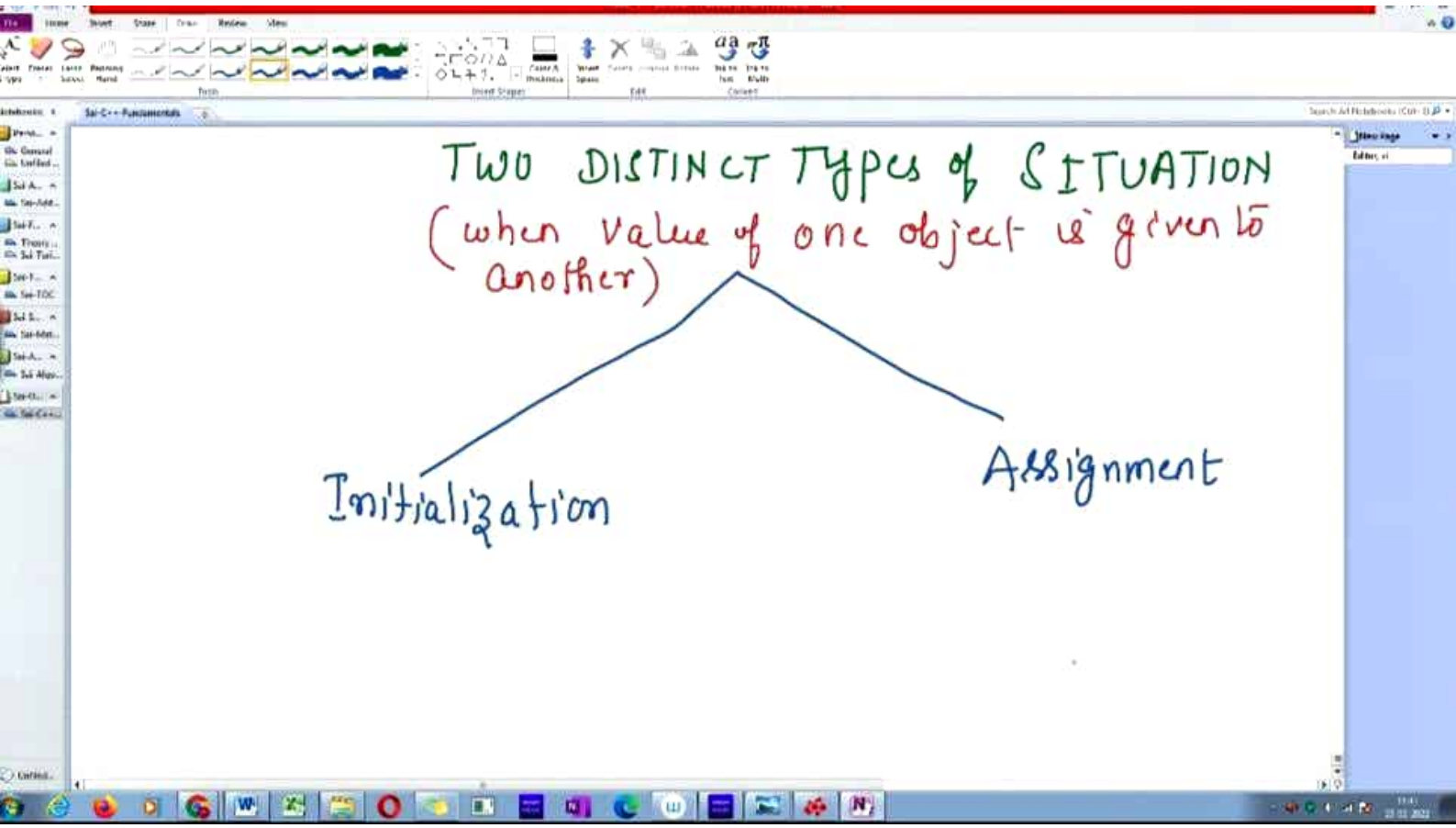
- C++ allows us to Create a Copy Constructor which the Compiler Can use when object initializes another object.
- By Creating a Copy Constructor.
  ↳ we bypass the default Bitwise Copy.


General form of copy constructor:
Classname(const classname &objatname)
{
}
reference to an object.

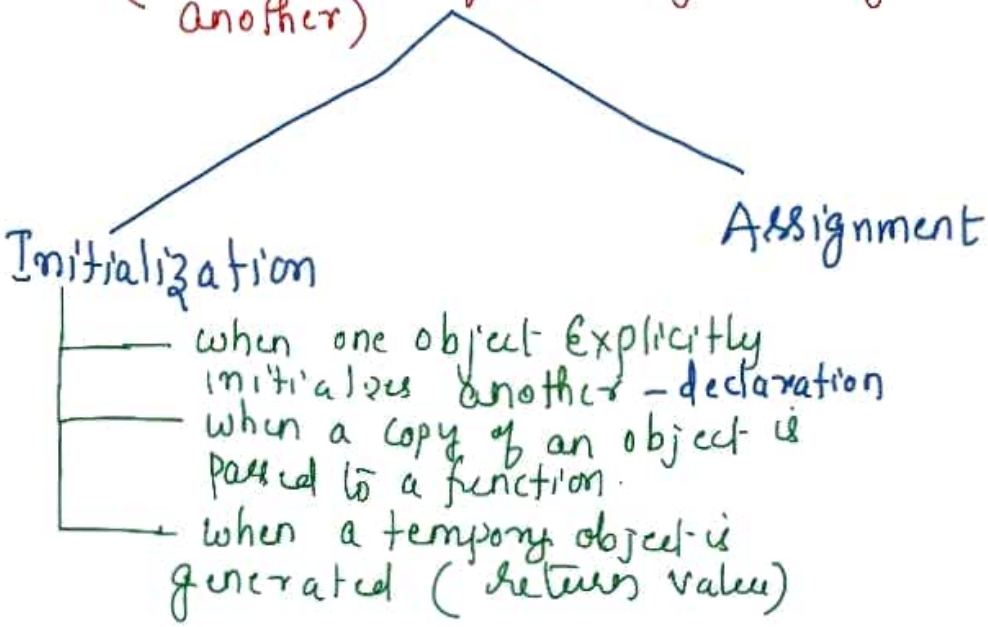

Classname(const classname &objatname)
{
}
reference to an object.
myclass (const myclass &o)
{
}
myclass obl = ob;


Classname(const classname &objatname)
{
}
reference to an object.
myclass (const myclass &o) =ob;
{
}
myclass obl = ob;


Classname(const classname &objatname)
{
}
↳ reference to an object.
myclass (const myclass &o) = ob;
myclass ob1 = ob;
{
} Code!!!
}


TWO DISTINCT Types of SITUATION
(when value of one object is given to another)
Initialization
Assignment

(when value of one object is given to another)



(when value of one object is given to another)

Assignment

Initialization

- when one object explicitly initializes another - declaration
- when a copy of an object is passed to a function.
- when a temporary object is generated (returns value)

Copy Constructor.

applies only to

Initialization

Initialization

- when one object explicitly initializes another - declaration
- when a copy of an object is passed to a function.
- when a temporary object is generated (returns value)

Assignment

generated (returns value)

QUIZ Suppose y is an object class myclass;

which of the following involve Initialization?

(A) myclass x = y;

(B) f(y);

(C) y = f(x);

(D) x = y;

generated (returns value)

QUIZ: Suppose y is an object class myclass;

which of the following involve Initialization?

(A) myclass x = y;

(B) f(y);

(C) y = f(x);

(D) x = y;

} Initialization

generated (returns value)

QUIZ: Suppose y is an object class myclass;

which of the following involve Initialization?

(A) myclass x = y;
(B) f(y);
(C) y = f(x);

} Initialization

(D) x = y; } Assignment.

generated (returns value)

QUIZ: Suppose y is an object class myclass;

which of the following involve Initialization?

Copy Constructor

(A) myclass x = y;

(B) f(y);

(C) y = f(x);

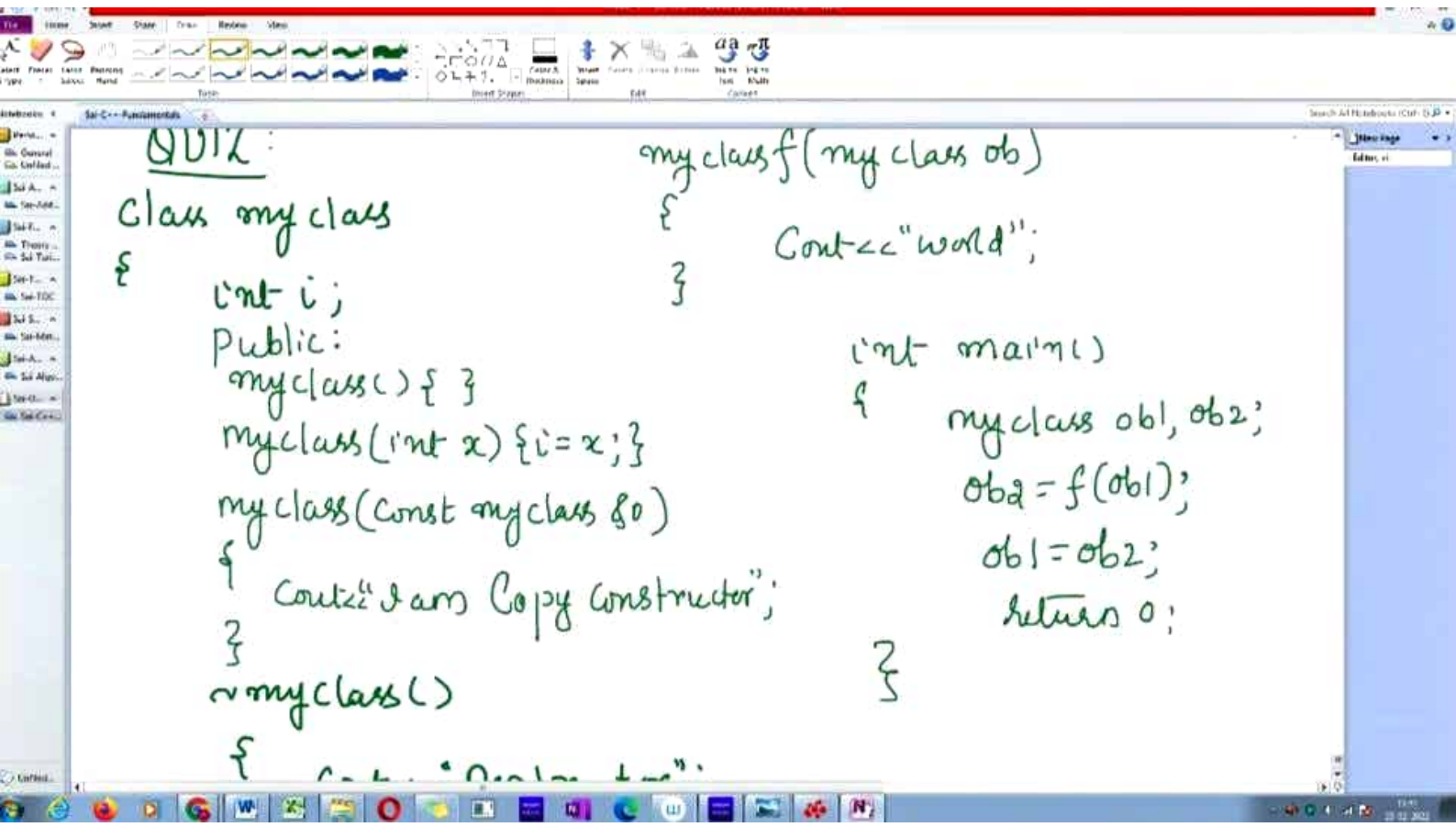
Initialization

(D) x = y; } Assignment.

QUIZ:

```
Class myclass
{
    int i;
    Public:
     myclass() { }
     myclass(int x) {i=x;}
     myclass(const myclass &o)
     {
       Cout<<"I am Copy Constructor";
     }
     ~myclass()
     {
```

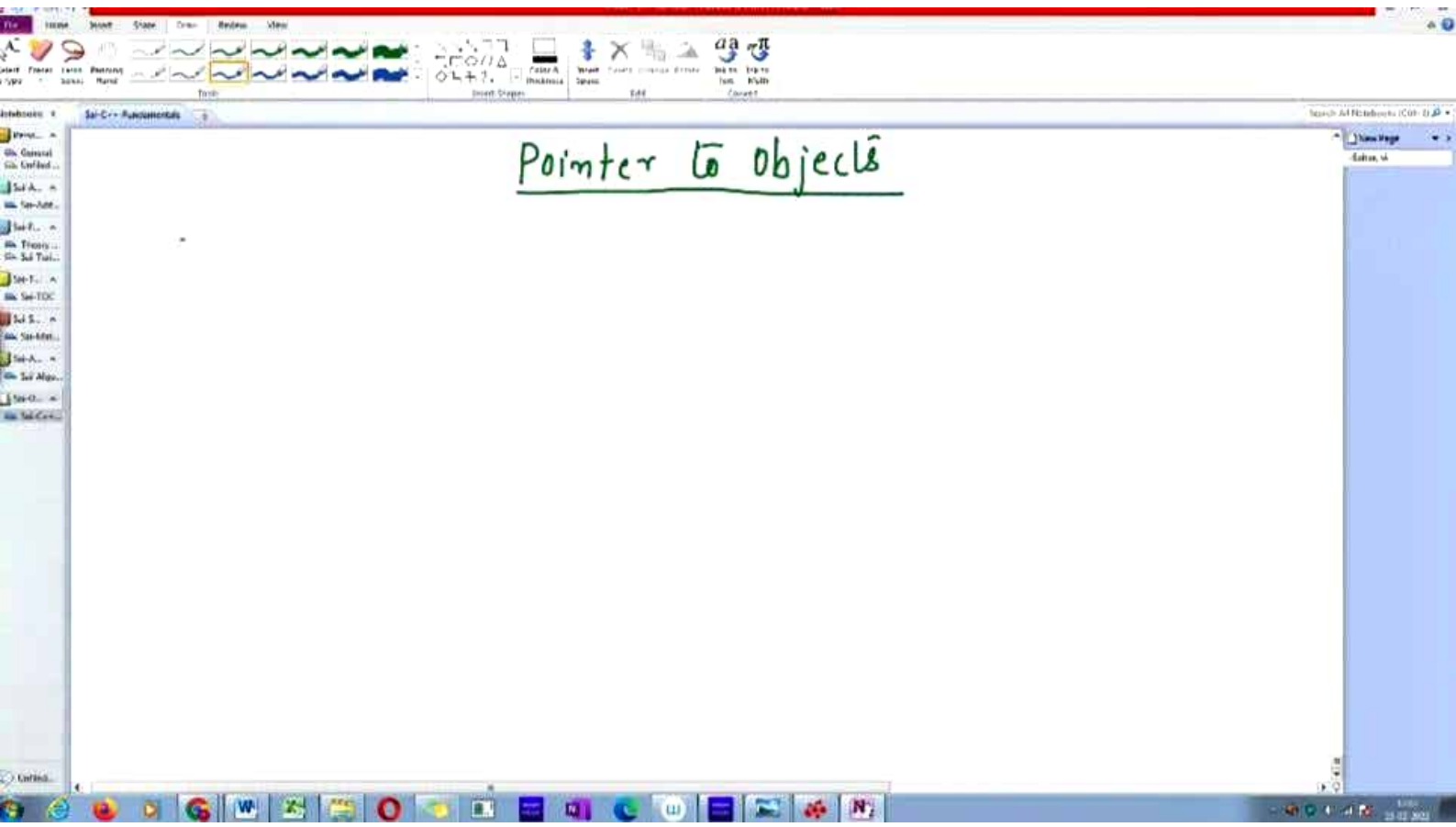
```
myclass f(myclass ob)
{
    Cout<<"world";
}

int main()
{
    myclass ob1, ob2;
    ob2 = f(ob1);
    ob1 = ob2;
    return 0;
}
```


QUIZ :
Class myclass
{
int i;
Public:
myclass() { }
myclass(int x) {i=x;}
myclass(const myclass &o)
{
Cout<<"I am Copy Constructor";
}
~myclass()
{
myclass f(myclass ob)
{
Cout<<"world";
}
int main()
{
myclass ob1, ob2;
ob2 = f(ob1);
ob1 = ob2;
return 0;
}

```
myclass() { }
myclass(int x) {i = x;}
myclass(const myclass &o)
{
  Cout<<"I am Copy Constructor";
}
~myclass()
{
  Cout<<"Destructor";
}
};
```

```
{
  myclass ob1, ob2;
  ob2 = f(ob1);
  ob1 = ob2;
  return 0;
}
```

```
I am copy constructor
word
destructor
destructor
destru
```


Pointer to objects

```
Class myclass
{ int i;
  public:
  myclass(int x) {i=x;}
  int get_i() {return i;
};
```

```
int main()
{
   myclass ob(99);
   myclass *p;
     p = &ob; //p points to ob
   Cout << p->get_i();
     return 0;
}
```

```
int main()
{
   myclass ob(99);
   Cout<< ob.get_i();
```

```
Class myclass
{ int i;
  public:
  myclass(int x) {i=x;}
  int get_i() {return i;
};
```

```
int main()
{
   myclass ob(99);
   myclass *p;
     p = &ob; //p points to ob
   Cout << p->get_i();
    return 0;
}
```

```
int main()
{
   myclass ob(99);
   Cout<< ob.get_i();
} return 0;
```

```
Class myclass
{ int i;
  public:
  myclass(int x) {i=x;}
  int get_i() {return i;
};
```

```
int main()
{
   myclass ob(99);
   myclass *p;
     p = &ob; //p points to ob
   Cout << p->get_i();
   return 0;
}
```

```
int main()
{
   myclass ob(99);
   Cout << ob.get_i();
} return 0;
```

What happens when a pointer is incremented?

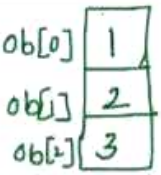
↳ It points to next element of its type.

All pointer arithmetic will be relative to the "BASE Type" of pointer.

```
Class myclass
{
    int i;
    public:
    myclass() { i=0; }
    myclass(int x) { i=x; }
    int get_i() { return i; }
};
```

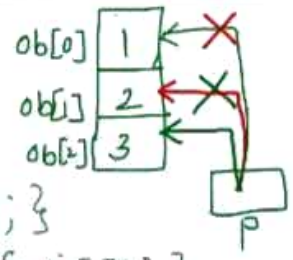
```
int main()
{
    myclass ob[3] = { 1, 2, 3 };
    myclass *p;
    p = ob;
    for(int i=0; i<3; i++)
    {
        cout << p->get_i();
        i++;
    }
    return 0;
}
```

```
Class myclass
{
    int i;
    public:
    myclass() { i=0; }
    myclass(int x) { i=x; }
    int get_i() { return i; }
};
```




```
int main()
{
    myclass ob[3] = { 1, 2, 3 };
    myclass *p;
    p = ob;
    for(int i=0; i<3; i++)
    {
        cout << p->get_i();
        i++;
    }
    return 0;
}
```

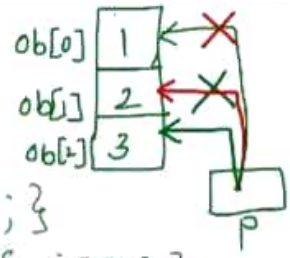
```
Class myclass
{
    int i,j;
    public:
    myclass() { i=0; }
    myclass(int x) { i=x; }
    int get_i() { return i; }
};
```

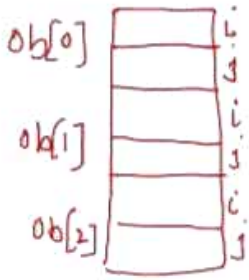



```
int main()
{
    myclass ob[3] = { 1, 2, 3 };
    myclass *p;
    p = ob; // p = &ob[0];
    for(int i=0; i<3; i++)
    {
        cout<< p->get_i();
        i++;
    }
    return 0;
}
```

1 2 3

```
class myclass
{
    int i,j;
public:
    myclass() { i=0; }
    myclass(int x) { i=x; }
    int get_i() { return i; }
};
```






```
int main()
{
    myclass ob[3] = {1, 2, 3};
    myclass *p;
    p = ob; // p = &ob[0];
    for(int i=0; i<3; i++)
    {
        cout << p->get_i();
        i++;
    }
    return 0;
}
```

1 2 3

```
class myclass
{
    public:
    int i;
    myclass (int x)
    {
      i=x;
    }
};
```

```
int main()
{
    myclass ob(20);
    int *p;
    p = &ob.i;
    cout << *p;
    return 0;
}
```

```
class myclass
{
    public:
    int i;
    myclass (int x)
    {
      i=x;
    }
};
```

i 20
ob

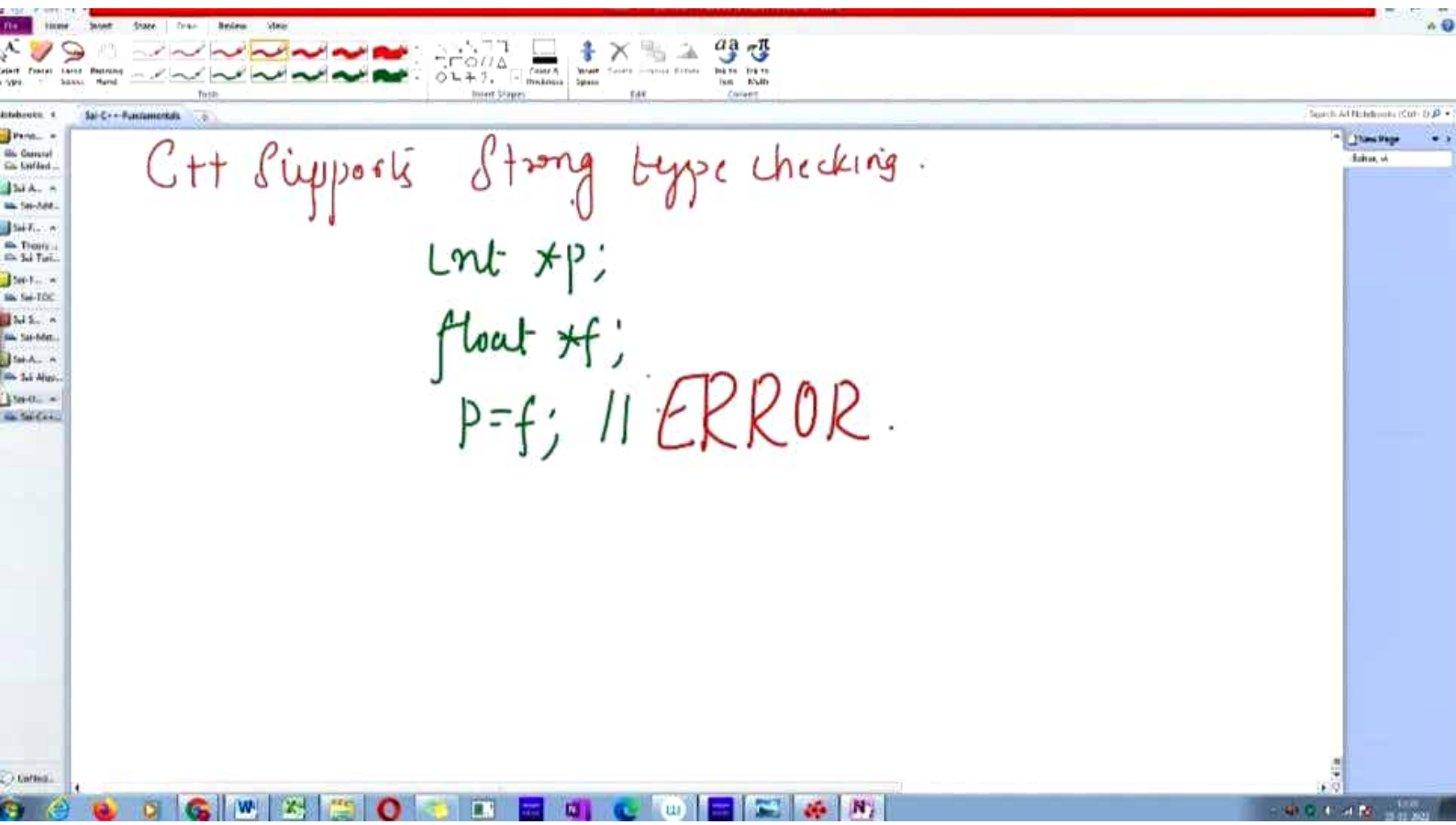
```
int main()
{
    myclass ob(20);
    int *p;
    p = &ob.i;
    cout<< *p;
    return 0;
}
```

Is it Valid?

```
class myclass
{
    public:
    int i;
    myclass (int x)
    {
        i=x;
    }
};
```


```
int main()
{
    myclass ob(20);
    int *p;
    p = &ob.i;
    cout<<*p; //20.
    return 0;
}
```

Is it valid?


C++ supports Strong type checking.
int *p;
float *f;
p=f; // ERROR.

this pointer: [Implicit this pointer]

↳ Keyword.

- Every member function of a class will be implicitly passed an argument which is a pointer to an invoking object which invokes or calls this function.

this pointer: [Implicit this pointer]

↳ keyword.

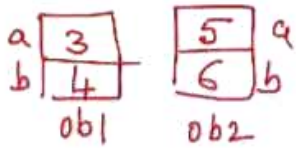
- Every member function of a class will be implicitly passed an argument which is a pointer to an invoking object which invokes or calls this function.
- This pointer is called this.

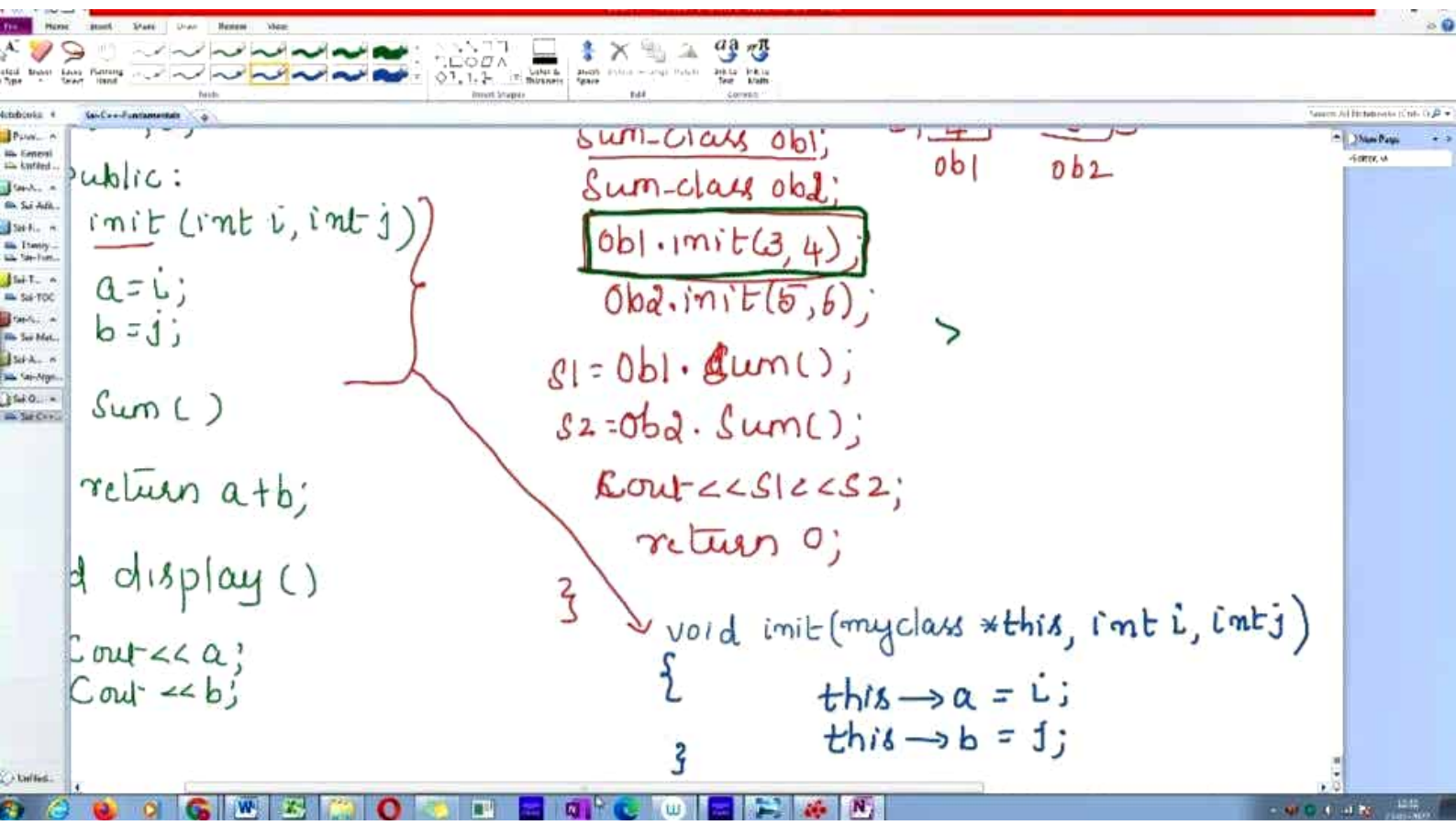
```
Class Sum_class
{
    int a,b;
    public:
void init (int i, int j)
{
    a=i;
    b=j;
}
int Sum ( )
{
    return a+b;
}
Void display ( )
{
    Cout<< a;
```

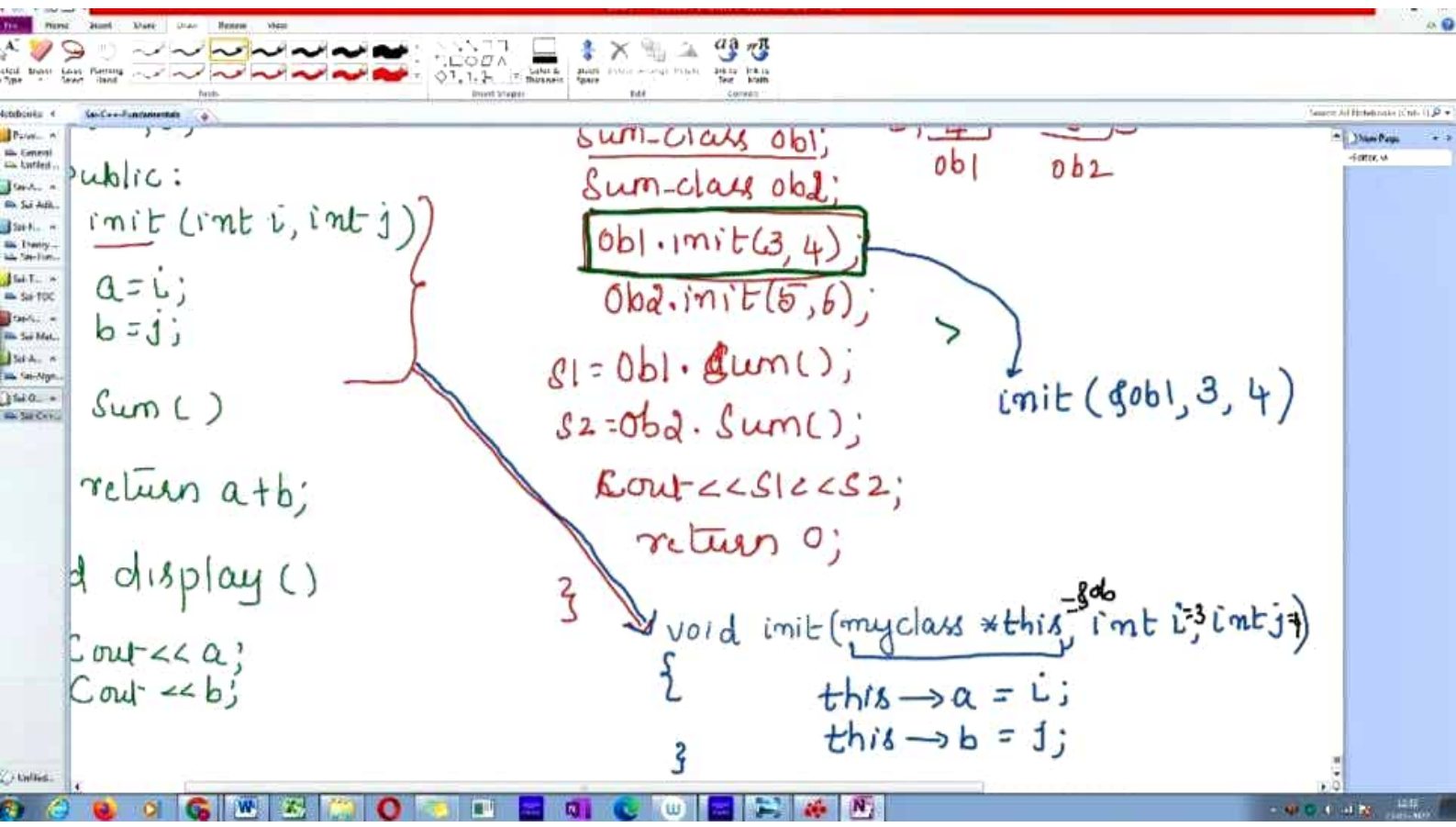
```
int main( )
{  int s1,s2;
   Sum_class ob1;
   Sum_class ob2;
    ob1.init(3, 4);
    ob2.init(5, 6);
  s1 = ob1.Sum( );
  s2 =ob2.Sum( );
    Cout<<s1<<s2;
      return 0;
}
```

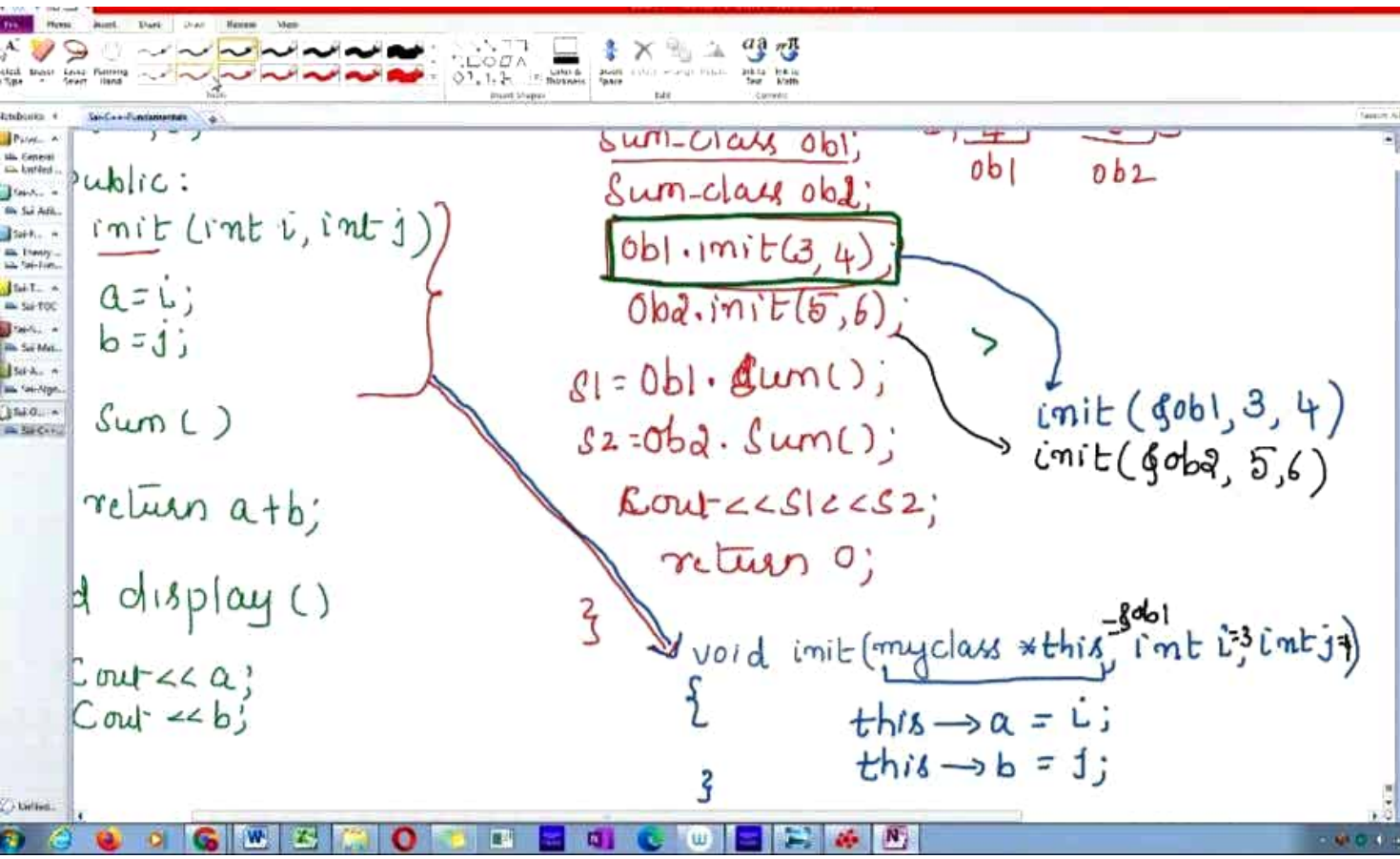
```
Class Sum_class
{
    int a,b;
    public:
void init (int i, int j)
{
    a=i;
    b=j;
}
int Sum ( )
{
    return a+b;
}
Void display ( )
{
    Cout<< a;
```

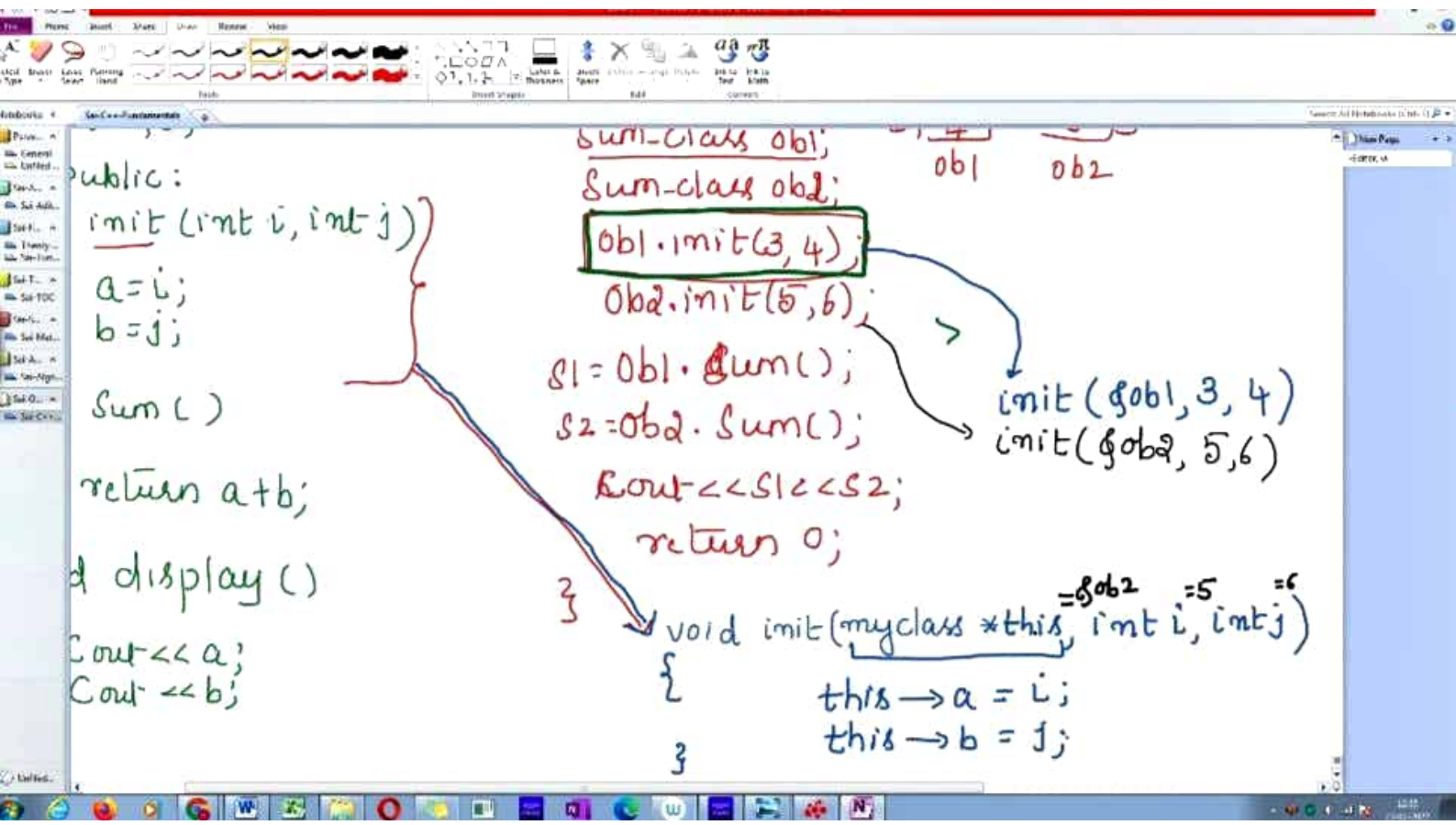
```
int main( )
{  int s1,s2;
   Sum_class ob1;
   Sum_class ob2;
   ob1.init(3, 4);
   ob2.init(5,6);
  s1 = ob1. Sum( );
  s2 =ob2. Sum( );
    Cout<<s1<<s2;
     return 0;
}
```

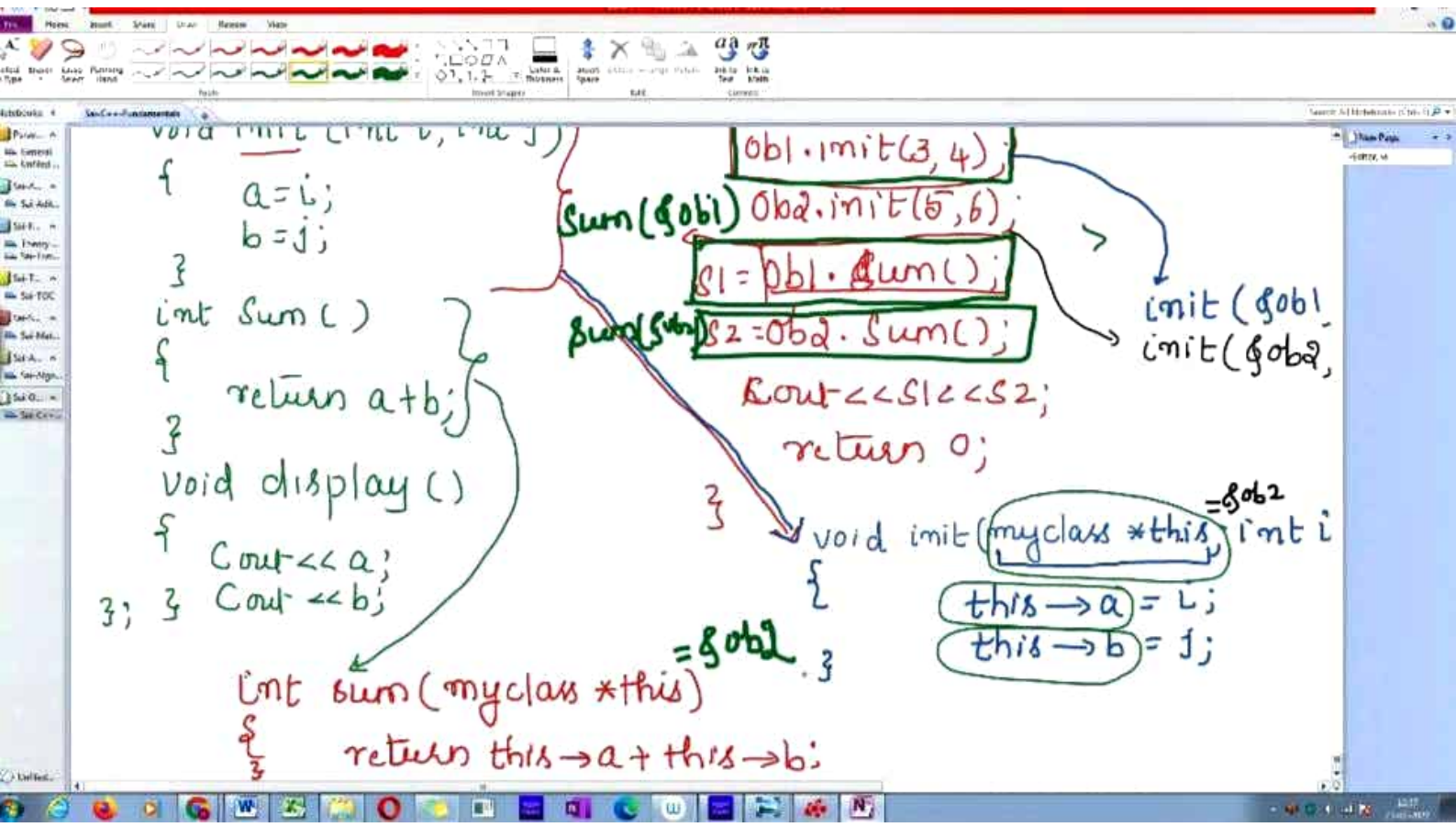


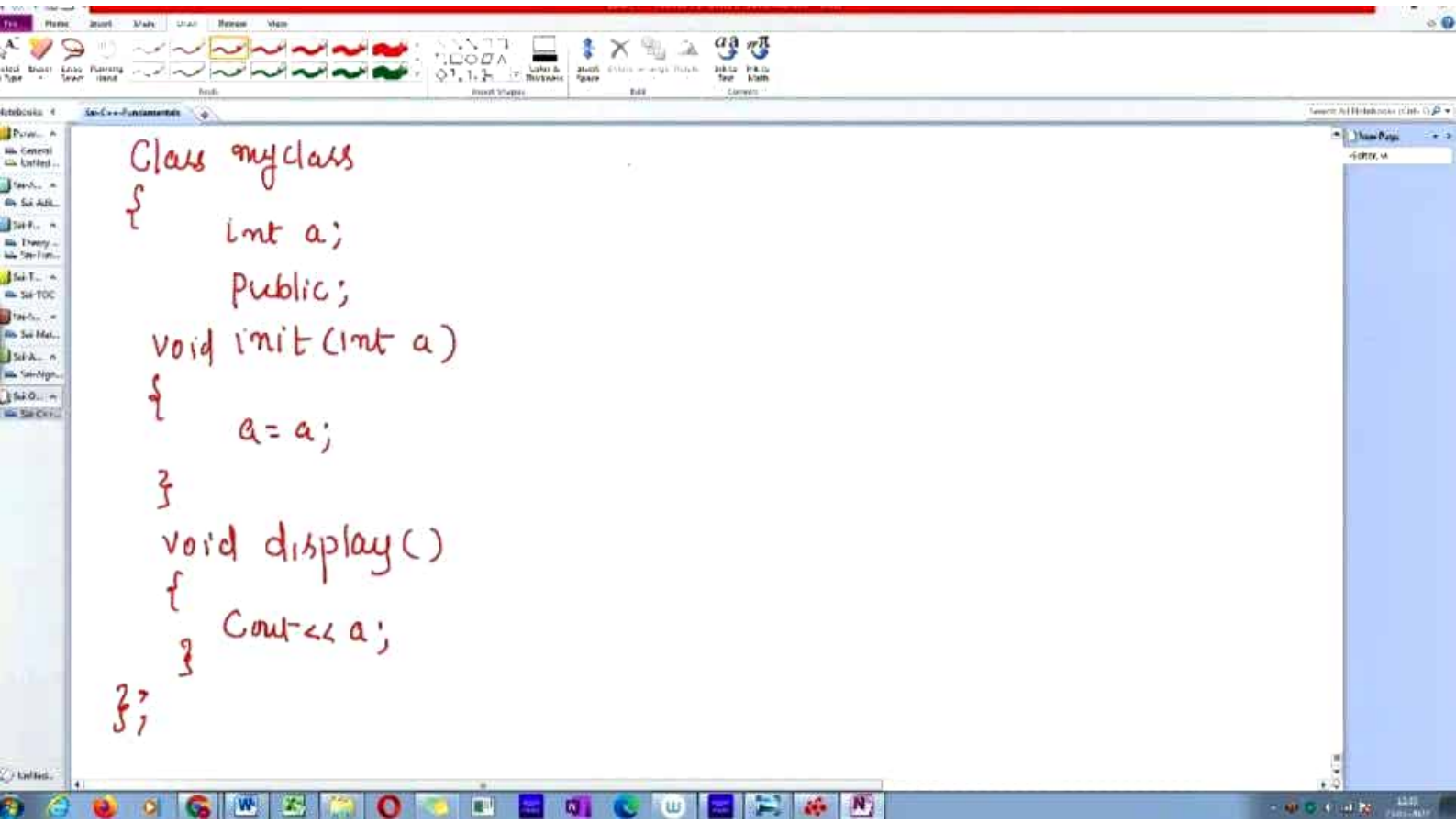

public:
init (int i, int j)
a=i;
b=j;
Sum ( )
return a+b;
display ( )
Cout<< a;
Cout << b;
Sum_class ob1;
Sum_class ob2;
ob1.init(3, 4);
ob2.init(5, 6);
s1 = ob1. Sum ( );
s2 = ob2. Sum ( );
Cout<<s1<<s2;
return 0;
}
void init(myclass *this, int i, int j)
{
this → a = i;
this → b = j;
}
ob1
ob2


Sum_class ob1;
Sum_class ob2;
ob1.init(3, 4);
ob2.init(5, 6);
s1 = ob1.Sum();
s2 = ob2.Sum();
cout<<s1<<s2;
return 0;
}
ob1
ob2
init(&ob1, 3, 4)
public:
init (int i, int j)
a = i;
b = j;
Sum ()
return a+b;
display ()
cout<< a;
cout << b;
void init(myclass *this, int i=3, int j=4)
{
this → a = i;
this → b = j;
}


public:
init (int i, int j)
a=i;
b=j;
Sum ( )
return a+b;
display ( )
Cout<< a;
Cout << b;
Sum-class obl;
Sum-class ob2;
obl.init(3, 4);
ob2.init(5,6);
s1 = obl.Sum();
s2 = ob2.Sum();
Cout<<s1<<s2;
return 0;
}
obl
ob2
init (&obl, 3, 4)
init(&ob2, 5,6)
void init(myclass *this, int i=3; int j=4)
{
this → a = i;
this → b = j;
}

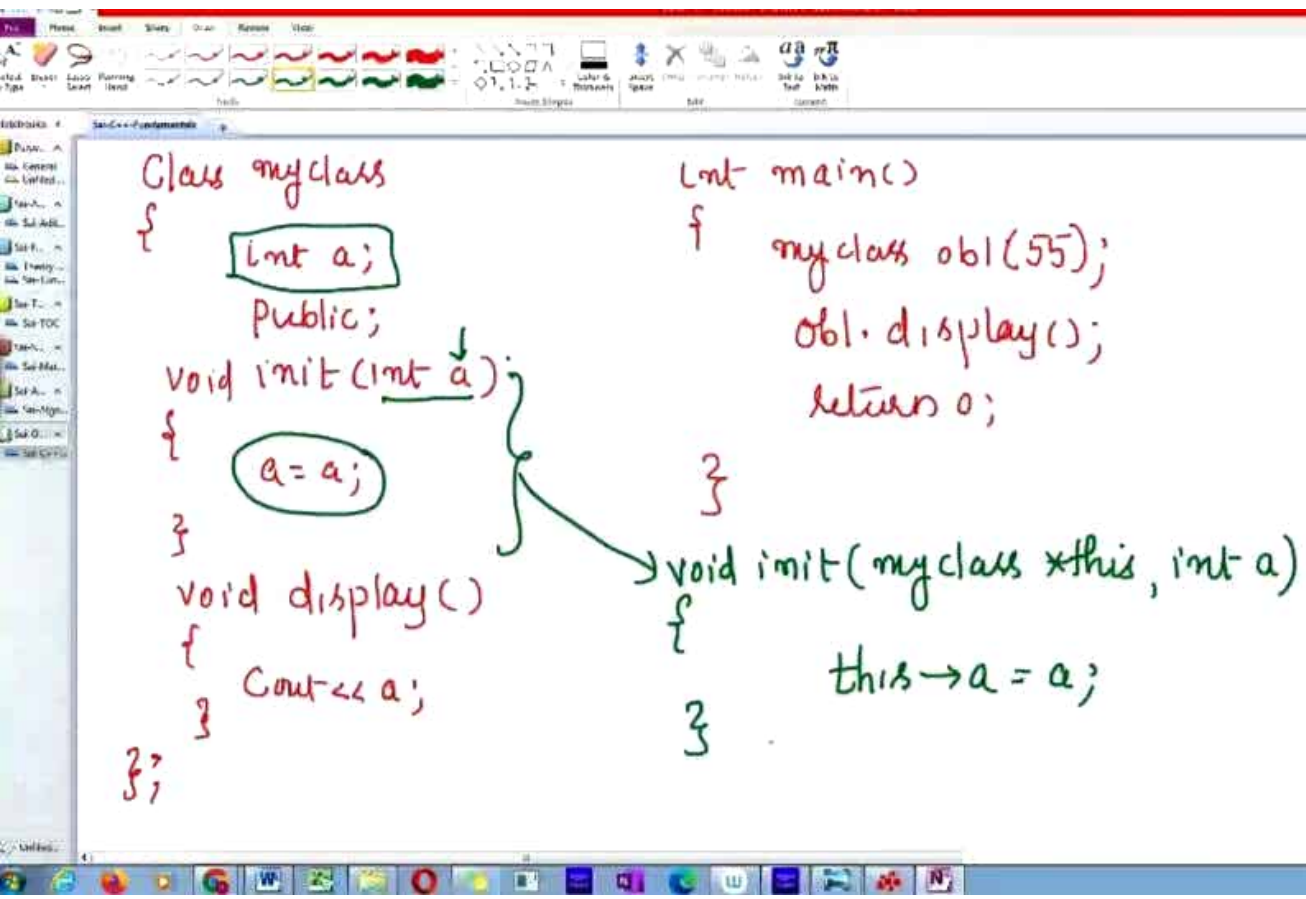

Sum-class ob1;
Sum-class ob2;
ob1.init(3, 4);
ob2.init(5, 6);
s1 = ob1.Sum();
s2 = ob2.Sum();
cout<<s1<<s2;
return 0;
}
ob1
ob2
init(&ob1, 3, 4)
init(&ob2, 5, 6)
public:
init(int i, int j)
a = i;
b = j;
Sum()
return a+b;
display()
cout<<a;
cout<<b;
void init(myclass *this, int i, int j)
=&ob2
=5
=6
{
this->a = i;
this->b = j;
}


void init (int i, int j)
{
a=i;
b=j;
}
int sum()
{
return a+b;
}
void display()
{
Cout<<a;
Cout<<b;
};
ob1.init(3,4);
sum(&ob1) ob2.init(5,6);
s1=ob1.sum();
sum(&ob2) s2=ob2.sum();
Cout<<s1<<s2;
return 0;
}
init(&ob1,
init(&ob2,
void init(myclass *this, int i
=&ob2
{
this→a = i;
this→b = j;
}
int sum(myclass *this)
=&ob2
{
return this→a+this→b;
}


Class myclass
{
int a;
Public;
void init(int a)
{
a=a;
}
void display()
{
Cout<< a;
}
};

```
Class myclass
{
    int a;
    Public;
  void init(int a)
  {
     a=a;
  }
  void display()
  {
    Cout<< a;
  }
};
```

```
int main()
{
   myclass obj(55);
    obj.display();
    return 0;
}
```


Class myclass
{
int a;
Public;
void init(int a);
{
a=a;
}
void display()
{
Cout<< a;
}
};
int main()
{
myclass obj(55);
Obj.display();
return 0;
}
void init(myclass *this, int a)
{
this->a = a;
}

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

myclass Sum(        )
{


}
```

```
int main()
{ myclass ob1(2,4);
  myclass ob2(5,6);
  myclass ob3;
    ob3 = [          ]


    return 0;
}
```

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

myclass Sum( myclass ob )
{
    myclass x;
      x.a = a + ob.a;
      x.b = b + ob.b;
    return x;
}
```

```
int main()
{ myclass ob1(2,4);
  myclass ob2(5,6);
  myclass ob3;
    ob3 = ob1.Sum(ob2);

    return 0;
}
```

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

myclass Sum( myclass ob )
{
    myclass x;
      x.a = a + ob.a;
      x.b = b + ob.b;
    return x;
 }

};
```

```
int main()
{ myclass ob1(2,4);
  myclass ob2(5,6);
  myclass ob3;
    ob3 = ob1.Sum(ob2);

    return 0;
}
```

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

myclass Sum( myclass ob )
{
    myclass x;
      x.a = a + ob.a;
      x.b = b + ob.b;
     return x;
 }
  void display(){ cout<<a <<b; }
};
```

```
int main()
{ myclass ob1(2,4);
  myclass ob2(5,6);
  myclass ob3;
    ob3 = ob1.Sum(ob2);

    return 0;
}
```

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

myclass Sum( myclass ob )
{
    myclass x;
      x.a = a + ob.a;
      x.b = b + ob.b;
    return x;
}
 void display(){ cout<<a <<b; }
};
```

```
int main()
{ myclass ob1(2,4);
  myclass ob2(5,6);
  myclass ob3;
    ob3 = ob1.sum(ob2
    return 0;
}
```

```
myclass Sum(myclass ob1, myclass ob2)
{
    myclass x;
    x.a = ob1.a + ob2.a;
    x.b = ob1.b + ob2.b;
    return x;
}
```

```
int main()
{
    ob3 = Sum(ob1, ob2);
}
```