Constructor→ construct object and assign value to it.

—----------

The purpose of a constructor is **to construct an object and assign values to the object's members**.

1. Name of the constructor → same as the name of the class.
2. It does not have any return type.
3. Constructor ek **instance member function** hai → instance member function meaning they are associated with instances (objects) of a class. They are responsible for initializing the state of an object and are called automatically when an object is created.

4. **constructor kabhi static nahi ho sakta hai** → Static methods or functions in a class are associated with the class itself rather than instances of the class. Constructors, on the other hand, are specific to instances and are used to initialize the properties of each individual object. Therefore, constructors are always instance methods and cannot be declared as static.

5. Member function⇒ 2 types ke ho sakte hai
    a. (non-static) or Instance member function →associated with object i.e
    These methods are associated with instances (objects) of a class. They have access to the instance-specific data and can be called on individual objects. Constructors, regular methods, and other member functions that operate on instance-specific data fall into this category.

    b. Static member function → related to class
    These methods are associated with the class itself rather than instances of the class. They don't have access to instance-specific data but can only operate on class-level data or perform operations that are independent of a specific instance. Static methods are called on the class rather than an instance of the class.

==========

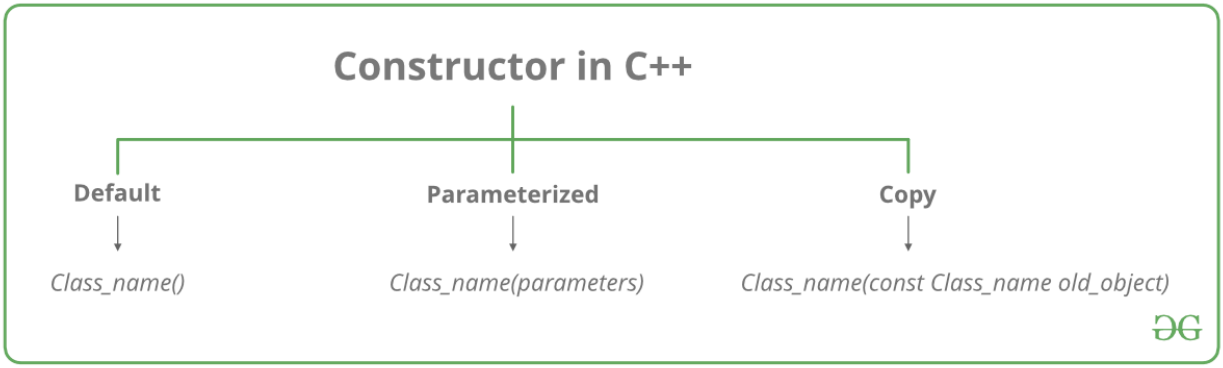
6. When we create the object → constructor will be called automatically.
Here we have created 2 objects → so 2 times the constructor will be called.

```cpp
#include<bits/stdc++.h>
#include <iostream>
using namespace std;
//------------------------------------
class A
{
    private:
        int a;
    public:
        A()//this is constructor
        {
            cout<<"constructor"<<endl;
        }
};

int main()
{
    A obja;
    A obja1;
    return 0;
}
```

```
constructor
constructor

...Program finished with exit code 0
Press ENTER to exit console.
```

==========

 # Types of the constructor => https://www.geeksforgeeks.org/constructors-c/



1. Default Constructor → no parameter

==========

1. **No Arguments:** The default constructor does not accept any arguments. It's a parameterless constructor.
2. **Automatic Generation:** If a class does not have any constructor defined explicitly, the compiler automatically generates a default constructor. This default constructor takes no parameters.

```cpp
#include <iostream>
using namespace std;
class A
{
```

```
    private:
        int a;
    public:
        A()//default constructor
        {
            a=10;
        }
        // Public member function to access the private member a
        int getVal() const
        {
            return a;
        }
        //Const Member Function: A const member function is a member
function that does not modify the object's state. It is a promise to
the compiler that the function will not change any member variables
of the class.
};

int main()
{
    A obja;
    // Accessing the value of a using the public member function
    cout <<obja.getVal()<<endl;//10
    return 0;
}
```

---

2. Parameterized constructor

==========

1. Constructor with parameters is called → parameterized constructor
2. Main work → to initialize the data member

Example:

```
#include <iostream>
using namespace std;
class A {
private:
    int a, b;
```

```
public:
    // Parameterized constructor
    A(int aVal, int bVal) {
        a = aVal;
        b = bVal;
       //this->a=a;//this.a=a; → is wrong
       //this->b=b;
    }

    // Public member function to access the sum of a and b
    int getAddition() const {
        return a + b;
    }
};

int main() {
    // Creating an object of class A using the parameterized
constructor
    A objA1(10, 20);
    A objA2(10, 30);
    // Accessing the sum of a and b using the public member function
    cout << "Sum: " << objA1.getAddition() << endl;//30
    cout << "Sum: " << objA2.getAddition() << endl;//40

    return 0;
}
```

3.copy constructor →

==========

1. Copy constructor → to copy content of one object to another.
2. It takes an object of the same class as a parameter and initializes the new object using the values of the existing object.
3. The copy constructor is invoked when a new object is created as a copy of an existing object, such as when passing objects by value, returning objects by value, or explicitly creating a new object as a copy.

Here are common situations in which a copy constructor is invoked:

1. Initialization of a New Object:

```
MyClass obj1(10, 20);//Parameterized constructor is called to create
obj1
MyClass obj2 = obj1;//Copy constructor is called to create obj2 as a
copy of obj1
```

In this case, when `obj2` is being created as a copy of `obj1`, the copy constructor is automatically called.

Example:

```
#include <iostream>
using namespace std;
class A {
private:
    int a, b;
public:
    // Parameterized constructor
    A(int aVal, int bVal) {
        this->a = aVal;
        this->b = bVal;
    }
    A(const A &obj)//copy constructor
    {
        // Initialize members using the corresponding members of
'obj'
        this->a=obj.a;
        this->b=obj.b;
    }
    // Public member function to access the sum of a and b
    int getAddition() const {
        return a + b;
    }
};

int main() {
    // Creating an object of class A using the parameterized
constructor
    A objA1(10, 20);
    // Creating another object as a copy of objA1 using the copy
constructor
```

```
    A objA2=objA1;//here copy constructor will be called

    // Accessing the sum of a and b using the public member function
    cout << "Sum: " << objA1.getAddition() << endl;//30
    cout << "Sum: " << objA2.getAddition() << endl;//30

    return 0;
}
```

—----------

---

2. Passing Objects by Value:

—----------

Example:

```
#include <iostream>
using namespace std;
class A {
private:
    int a, b;
public:
    // Parameterized constructor
    A(int aVal, int bVal) {
        this->a = aVal;
        this->b = bVal;
    }
    A(const A &obj)//copy constructor
    {
        // Initialize members using the corresponding members of
'obj'
        this->a=obj.a;
        this->b=obj.b;
    }
    // Public member function to access the sum of a and b
    int getAddition() const {
        return a + b;
    }
};

void someFunction(A obj)
{
```

```
    cout<<"copy constructor is called.."<<endl;
}

int main() {
    // Creating an object of class A using the parameterized
constructor
    A objA1(10, 20);

    // Calling someFunction and passing obj by value
    // Copy constructor is called when obj is passed by value
    someFunction(objA1); // Copy constructor is called here

    // Accessing the sum of a and b using the public member function
    cout << "Sum: " << objA1.getAddition() << endl;//30

    return 0;
}
//output
// copy constructor is called..
// Sum: 30
```

When an object is passed by value to a function, a copy of that object is created inside the function, and the copy constructor is called.

—----------

3. Returning Objects by Value:

—----------

When a function returns an object by value, a copy of the returned object is created, and the copy constructor is invoked.

```
#include <iostream>
using namespace std;
class A {
private:
    int a, b;
public:
    // Parameterized constructor
    A(int aVal, int bVal) {
        this->a = aVal;
        this->b = bVal;
    }
```

```
    A(const A &obj)//copy constructor
    {
        // Initialize members using the corresponding members of
'obj'
        this->a=obj.a;
        this->b=obj.b;
    }
    // Public member function to access the sum of a and b
    int getAddition() const {
        return a + b;
    }
};
A createObject()
{
    A newObj(50, 60);
    return newObj; // Copy constructor is called when newObj is
returned by value
}

int main() {
    A obja=createObject();// Copy constructor is called when
createObject() returns a value


    // Accessing the sum of a and b using the public member function
    cout << "Sum: " << obja.getAddition() << endl;//110
    return 0;
}
//output
// Sum: 110
```

—----------

4. Initializing an Object with Another Object

```
#include <iostream>
using namespace std;

class A {
private:
```

```
    int a, b;

public:
    // Parameterized constructor
    A(int aVal, int bVal) {
        this->a = aVal;
        this->b = bVal;
    }
    A(const A &obj)//copy constructor
    {
        // Initialize members using the corresponding members of
'obj'
        cout<<"copy constructor is called..."<<endl;
        this->a=obj.a;
        this->b=obj.b;
    }
    // Public member function to access the sum of a and b
    int getAddition() const {
        return a + b;
    }
};

int main() {
    A obja(10,20);
    // Initializing another object (obj3) with obj1 using the copy
constructor
    A objb(obja);//copy constructor is called Here...

    // Accessing the sum of a and b using the public member function
    cout << "Sum: " << obja.getAddition() << endl;//30
    cout << "Sum: " << objb.getAddition() << endl;//30

    return 0;
}
//output
// Sum: 30
```

==========

 # Constructor Overloading in C++ :

==========

1. In C++, We can have more than one constructor in a class with the same name, as long as each has a different list of arguments.
2. This concept is known as Constructor Overloading and is quite similar to function overloading.
3. Overloaded constructors essentially have the same name (exact name of the class) and different by number and type of arguments.
4. A constructor is called depending upon the number and type of arguments passed.
5. While creating the object, arguments must be passed to let the compiler know which constructor needs to be called.

```
 // Constructor overloading
#include <iostream>
using namespace std;
class construct
{
 public:
    float area;

    // Constructor with no parameters
    construct()
    {
        area = 0;
    }
    // Constructor with two parameters
    construct(int a, int b)
    {
        area = a * b;
    }
    void disp()
    {
        cout<< area<< endl;
    }
};
int main()
{
```

```
    // Constructor Overloading
    // with two different constructors
    // of class name
    construct o;
    construct o2( 10, 20);

    o.disp();
    o2.disp();
    return 1;
}
```

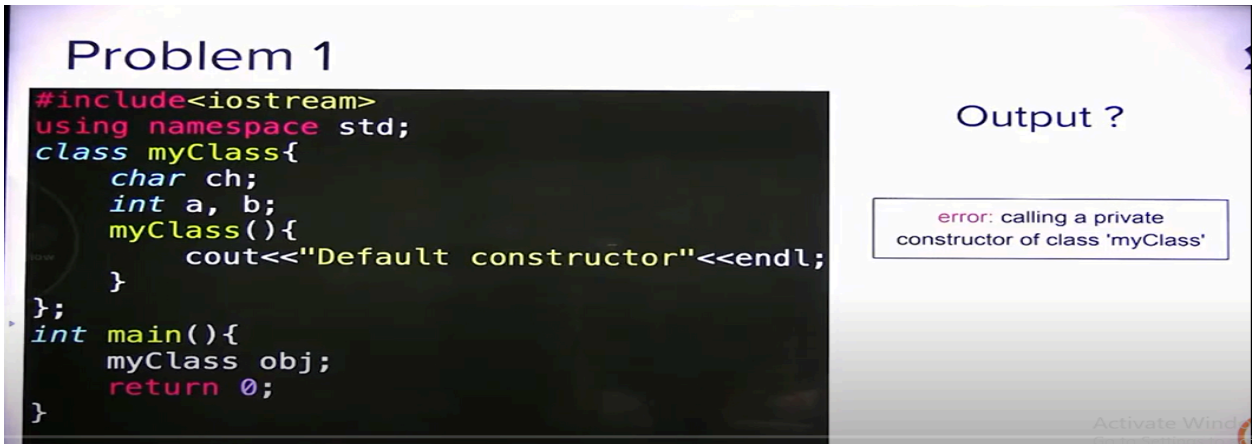
—----------

==========

Numerical on constructor:(jay bansal lecture)-LEC 19 and 20

—----------

Q1. **You cannot define a constructor as private** ⇒ you will get an error if you do the same.



So you have to declare the constructor as public.

```
 8  #include <iostream>
 9  using namespace std;
10  class A
11  {
12      int a,b;
13      A()
14      {
15          cout<<"hello"<<endl;
16      }
17  };
18  int main()
19  {
20      A obj;
21      return 0;
22  }
23
```

input

Compilation failed due to following error(s).

```
main.cpp: In function 'int main()':
main.cpp:20:7: error: 'A::A()' is private within this context
   20 |     A obj;
      |       ^~~
main.cpp:13:5: note: declared private here
   13 |     A()
      |     ^
```

```
 8  #include <iostream>
 9  using namespace std;
10  class A
11  {
12          int a,b;
13      public:
14          A()
15          {
16              cout<<"hello"<<endl;
17          }
18  };
19  int main()
20  {
21      A obj;
22      return 0;
23  }
```

```
hello


...Program finished with exit code 0
Press ENTER to exit console.
```

Q2: Here while creating obj a,the compiler will search for a constructor and the required constructor are public.so the private constructor are hidden so it does not create any problem.

Here object name,parameter name and attribute name are all the same ..it is possible.
Since we cant use a=a ,therefore we have to use **this** keyword.
this refers to the current object.

Problem 2

```
#include<iostream>
using namespace std;
class A{
private:
    A(){
        cout<<"Default Constructor"<<endl;
    }
public:
    int a;
    A(int a){
        this->a = a;
    }
};
int main(){
    A a(10);
    cout<<a.a<<endl;
    return 0;
}
```

8:54 / 1:16:05

```
 8  #include <iostream>
 9  using namespace std;
10  class A
11  {
12      private:
13          A()
14          {
15              cout<<"hello"<<endl;
16          }
17
18      public:
19          A(int a)
20          {
21              cout<<"helloa"<<endl;
22          }
23  };
24  int main()
25  {
26      A obj(10);
27      return 0;
28  }
29
```

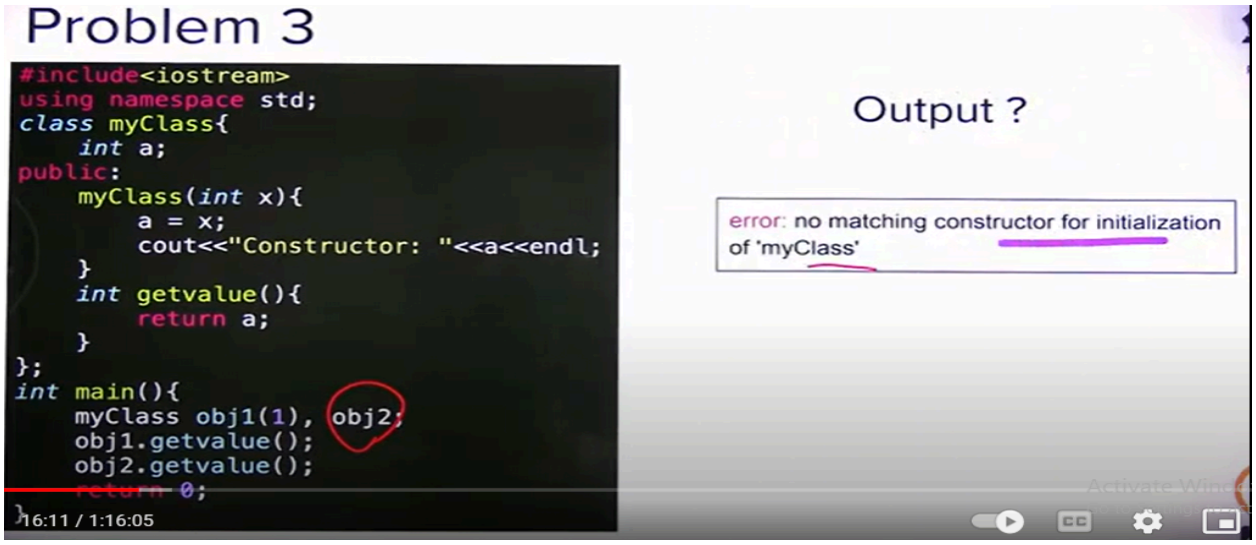
```
helloa
```

Output: 10

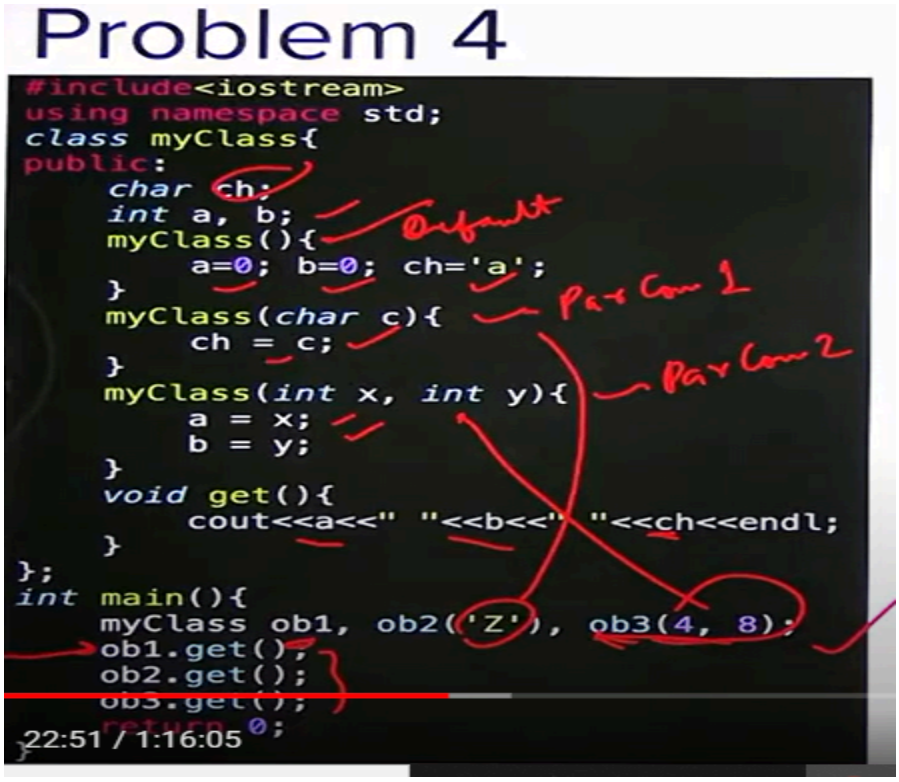
Q3:

1→ if we do not create any constructor then,the compiler will create the default constructor.

2→ BUt if we create any constructor the compiler will not create default constructor.



Q4:

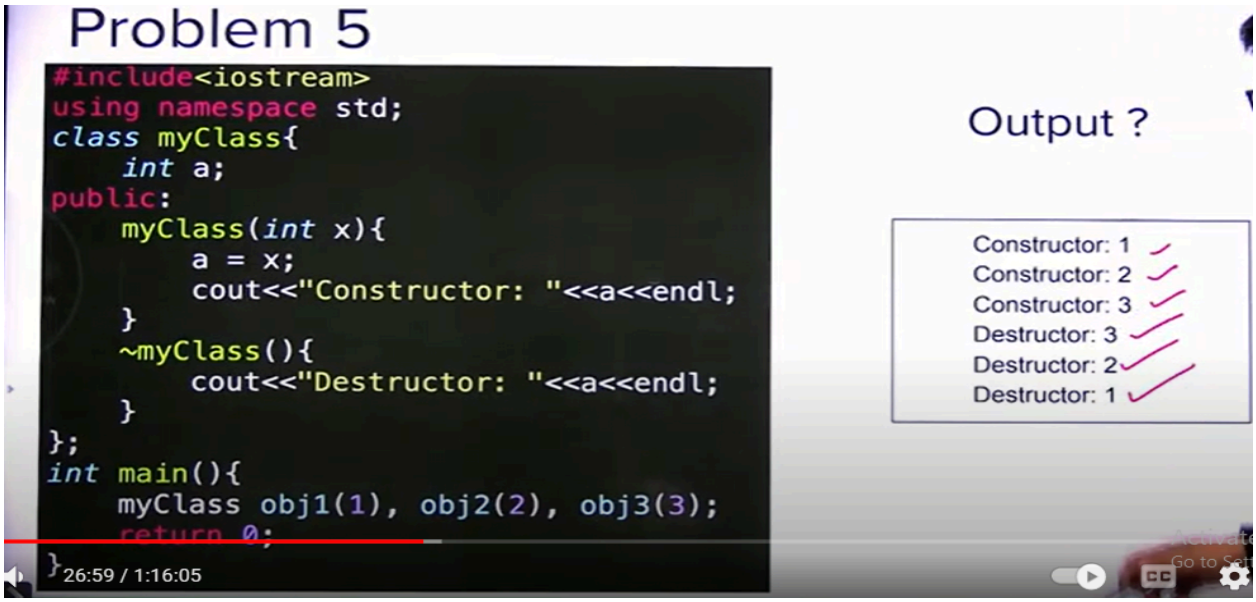


Op: 0 0 a

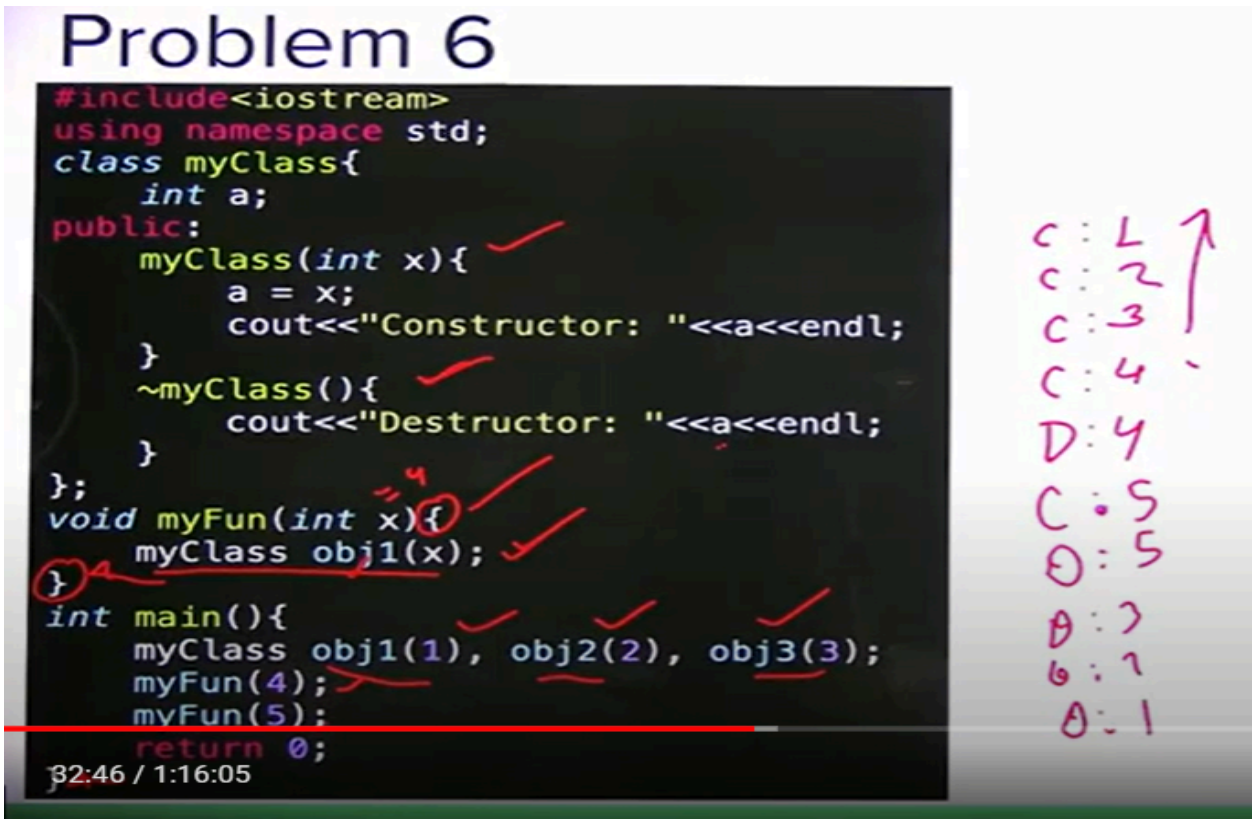
Gv gv z

4 8 gv

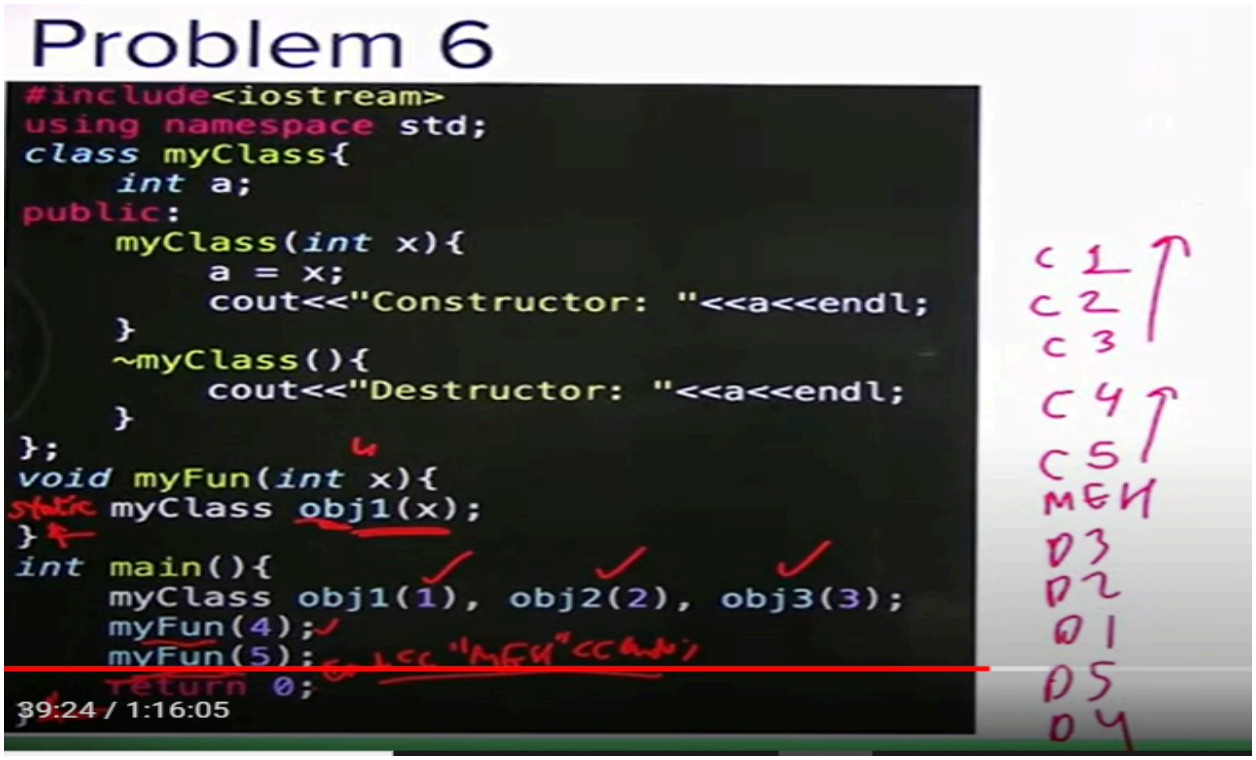
Q5 : Destructor will be called in the reverse order when objects die.
Jab bhi object destroy hoga tab destructor chalega.
Object ka scope end hone pr object destroy ho jaiga.



Q6. Destructor will be called when the scope of that particular object ends.

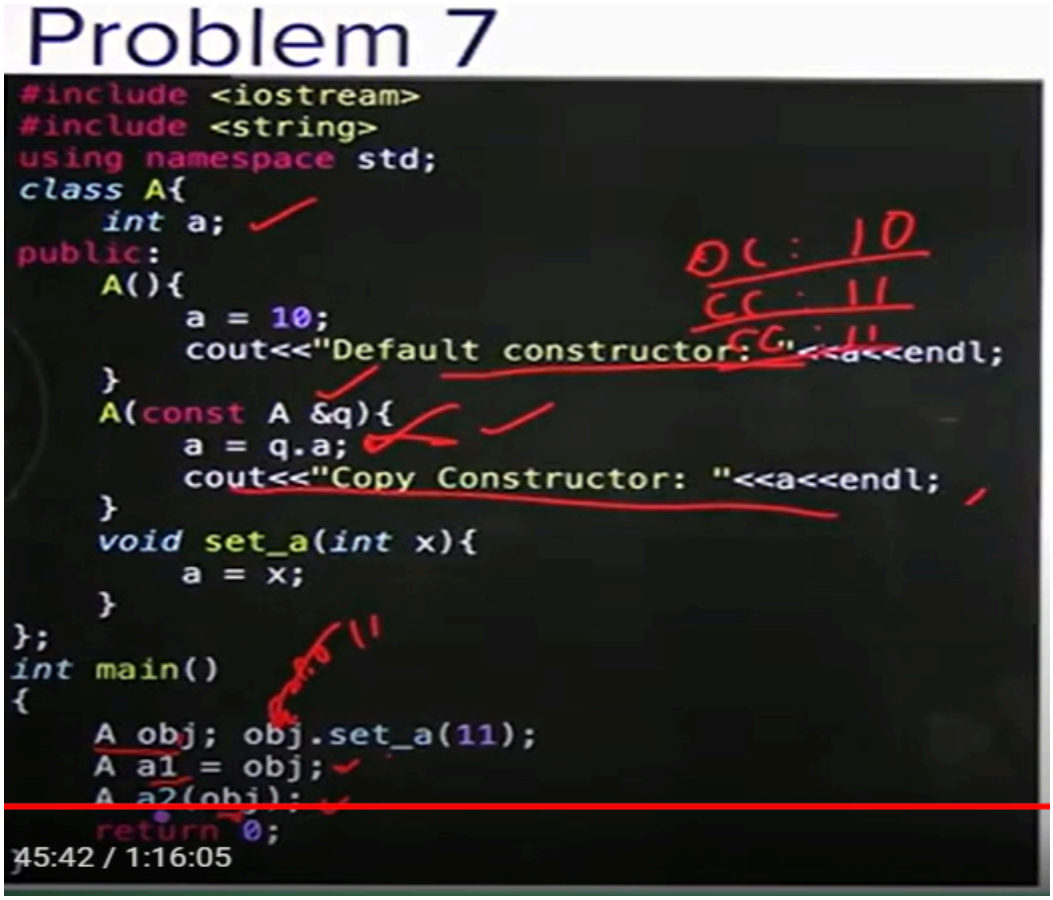


Q6 B: static variable dies when the program dies.



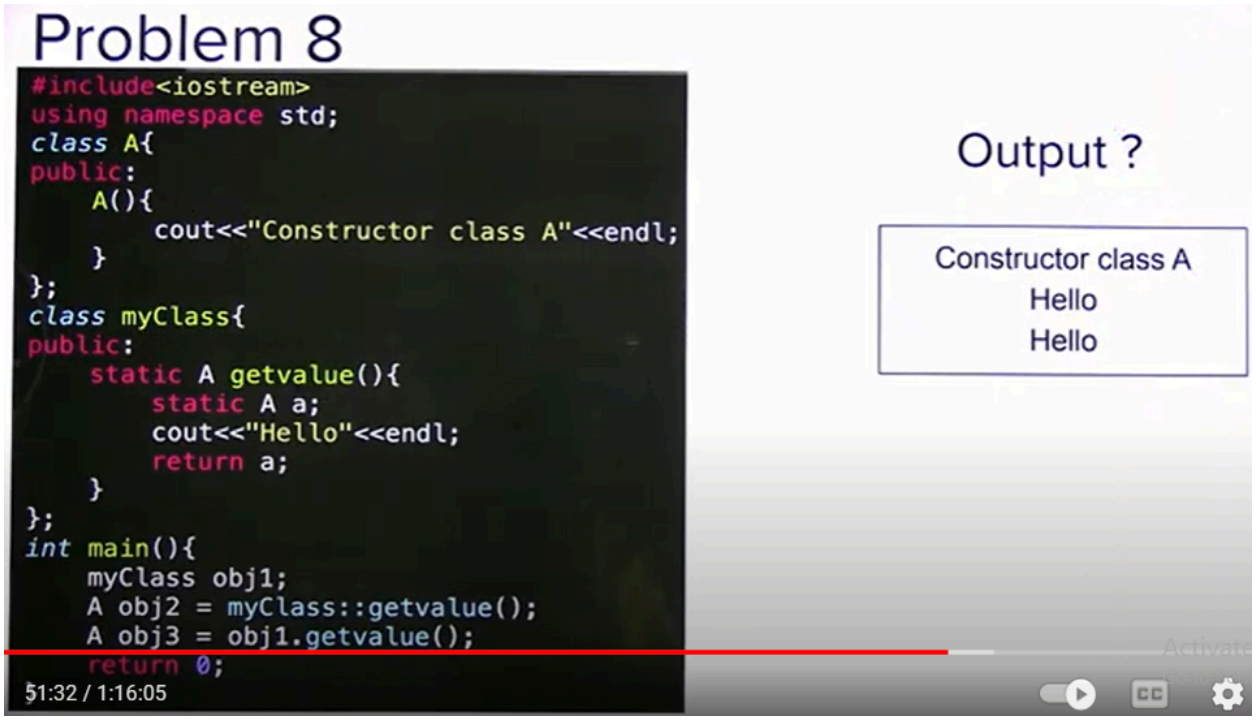
Here first main end and then program end.so D3,D2,D1 will be called first and then D5,D4 Destructor are called in the reverse order.

Q7: Copy constructor are uses to copy the content of one object into another.



A a1=obj;
A a2(obj);
These 2 are similar.

Q8:

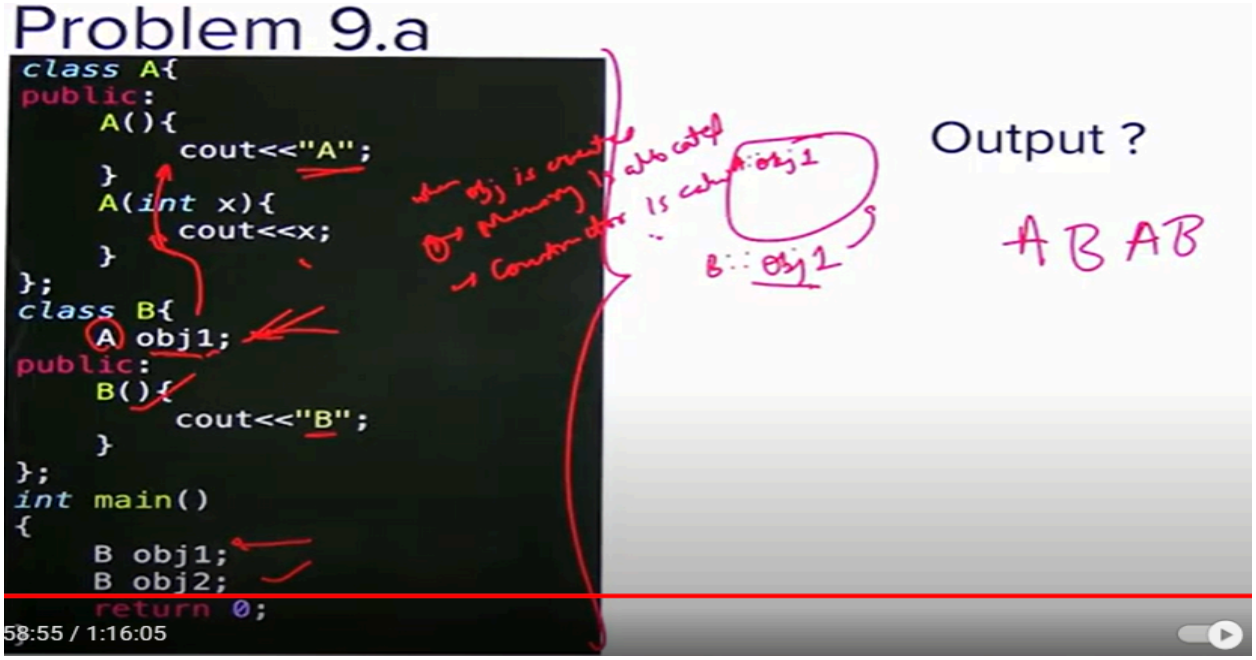


Q9:

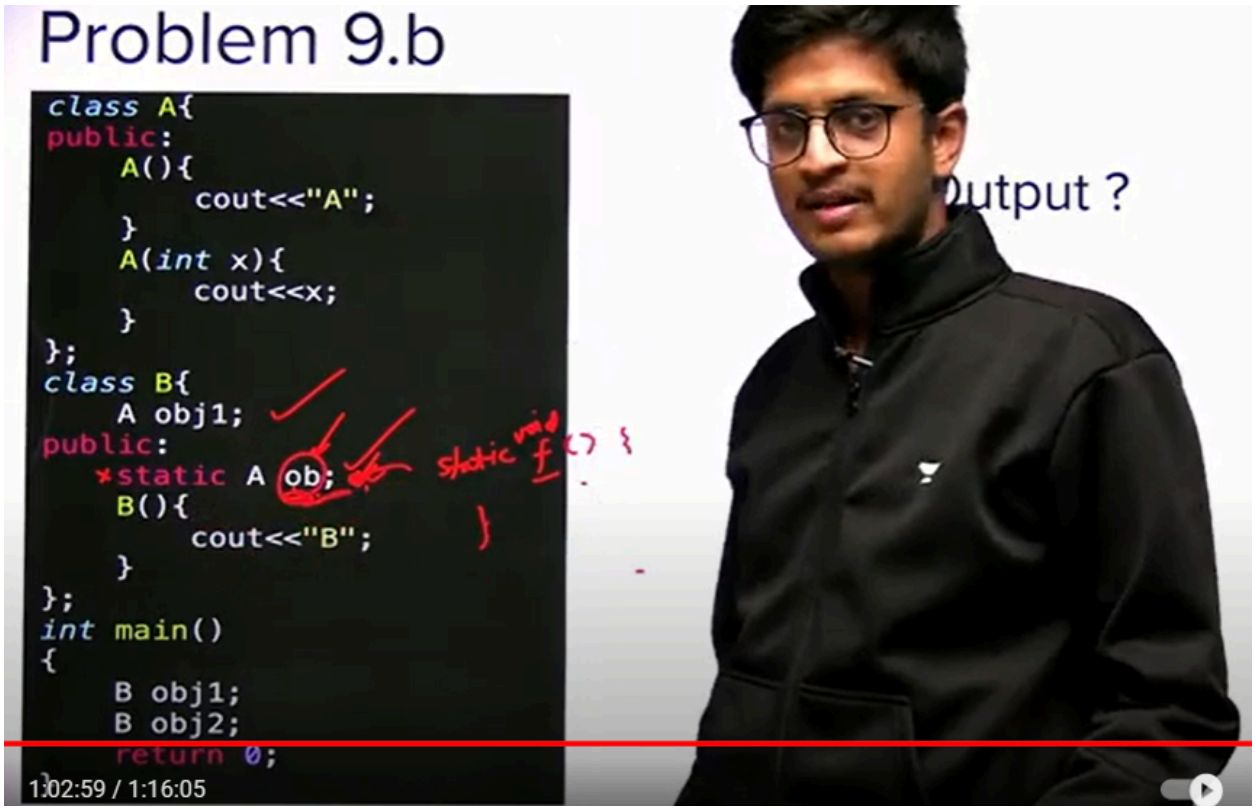
A: when we create the object then

1→ memory will be assigned to the data member(non static data member)
Non-static data members belong to the object.

2→ **and then the constructor will be called.**



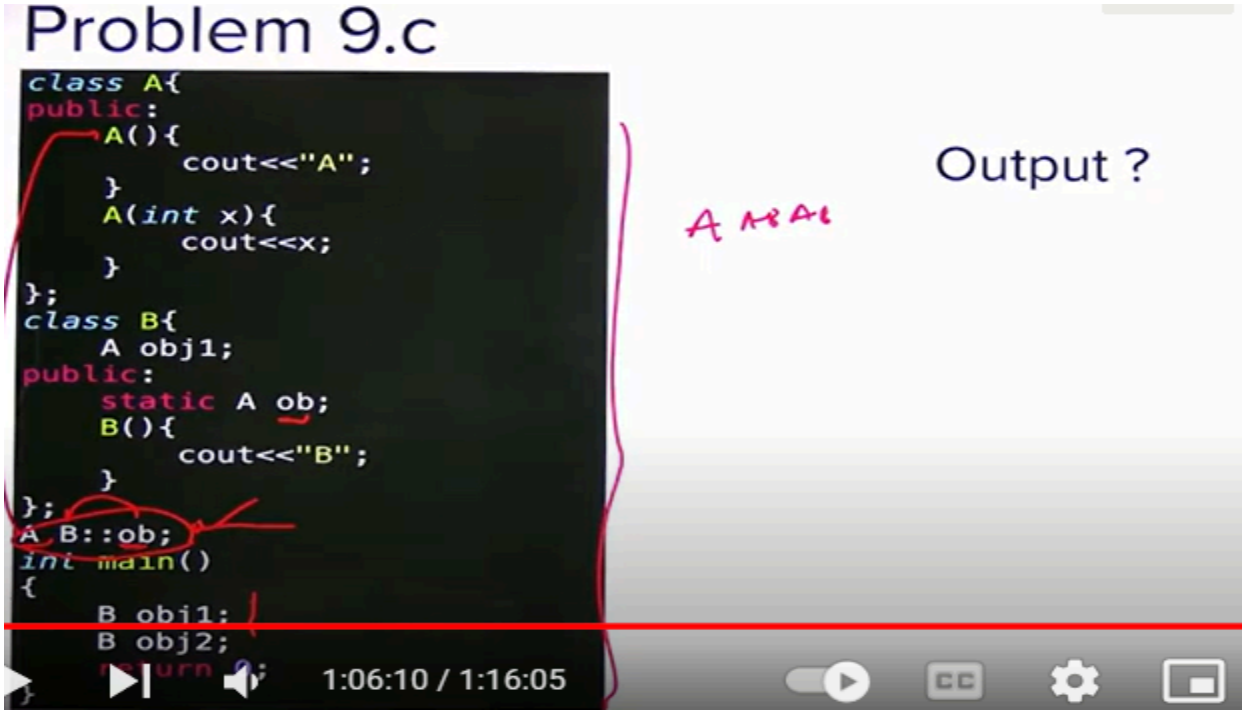
Q9 B:



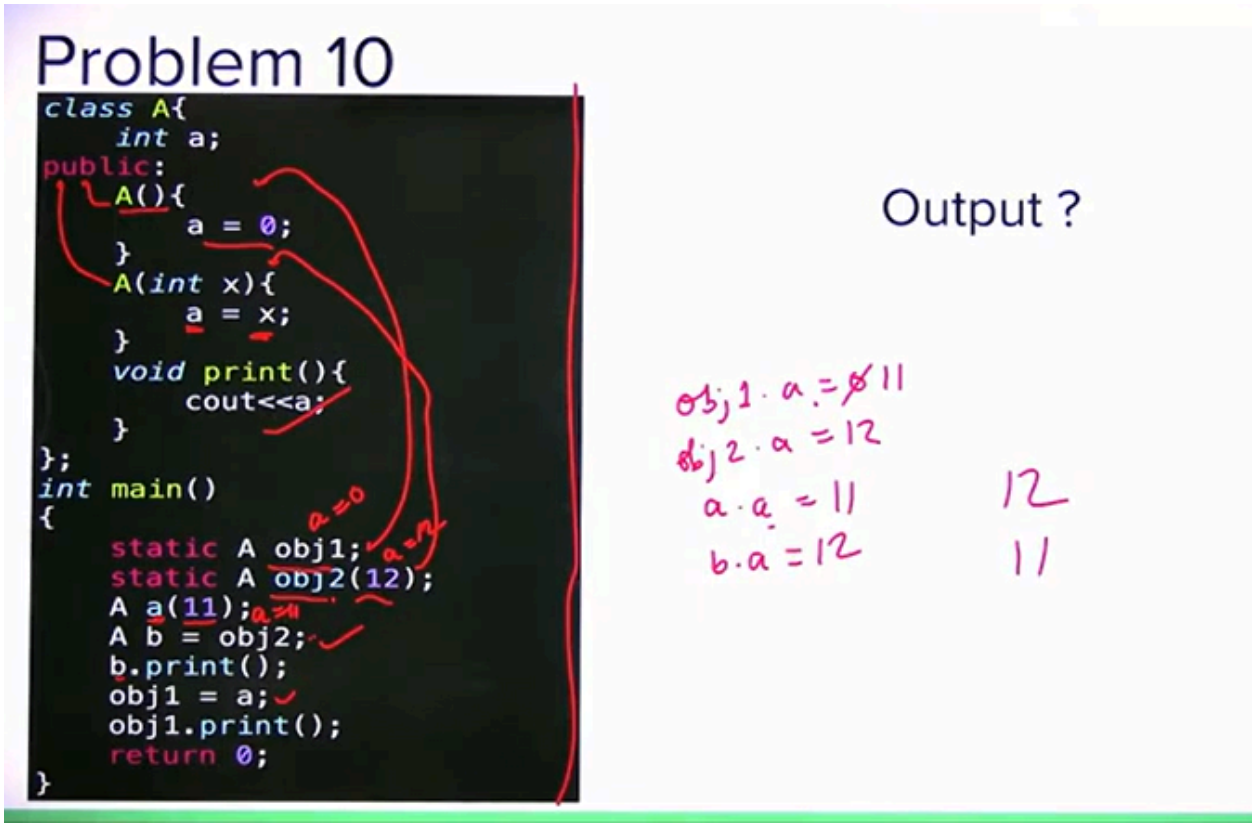
Op: ABAB
We know when the local variables are created inside the class→when we create the object.
But when the static variables are created?
—----------
Ans -> when we use the static variable outside of the class then memory will be allocated to the static member (both variable and function.)
Not at the time of creation.
So here we have not used the variable "obj" outside ofthe class then memory will not be assigned to the "obj".

Q9 C:
When we have any global variable then first global variable are created and then main is executed.
A B :: ob;==> ob belongs to class B and it is of type A.

Since ob is of type A hence constructor for A will be called.

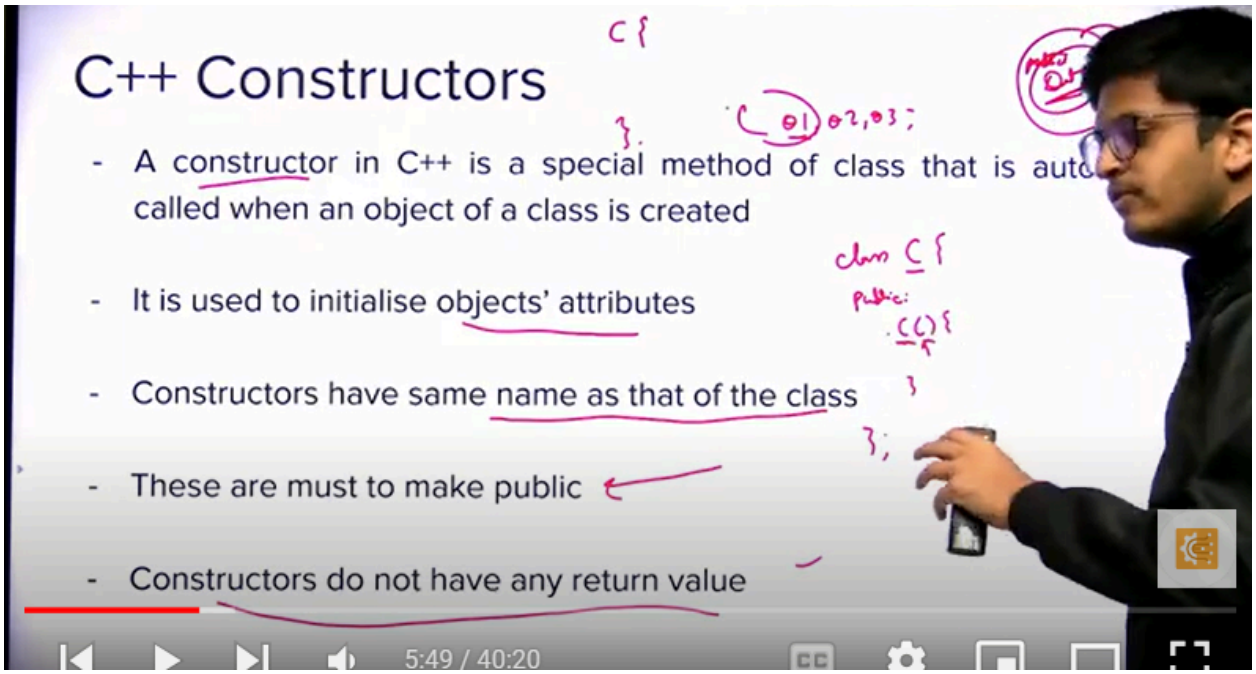

Problem 9.c
class A{
public:
A(){
cout<<"A";
}
A(int x){
cout<<x;
}
};
class B{
A obj1;
public:
static A ob;
B(){
cout<<"B";
}
};
A B::ob;
int main()
{
B obj1;
B obj2;
Output ?
1:06:10 / 1:16:05

Q10:

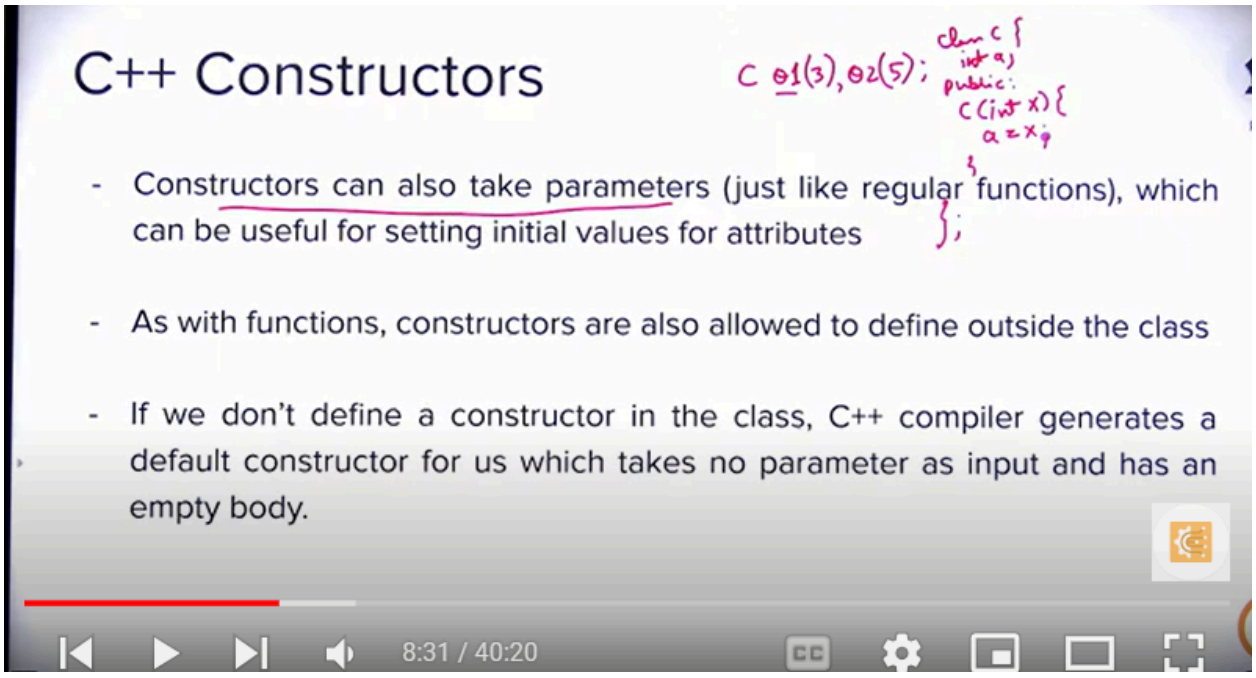

Problem 10
class A{
int a;
public:
A(){
a = 0;
}
A(int x){
a = x;
}
void print(){
cout<<a;
}
};
int main()
{
static A obj1;
static A obj2(12);
A a(11);
A b = obj2;
b.print();
obj1 = a;
obj1.print();
return 0;
}
Output ?

LEC -19 Constructors & Destructors | L:19 | C++ | Competitive Programming | Ravindrababu Ravula | Jay Bansal

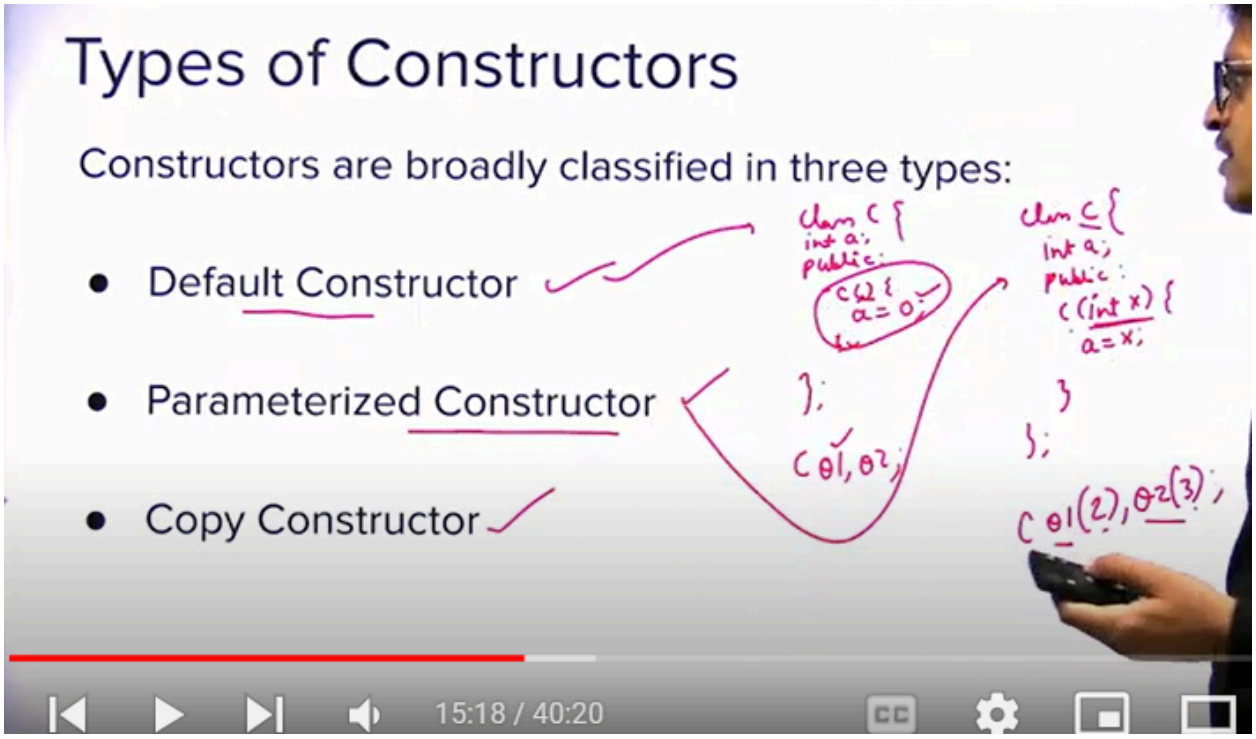
==========



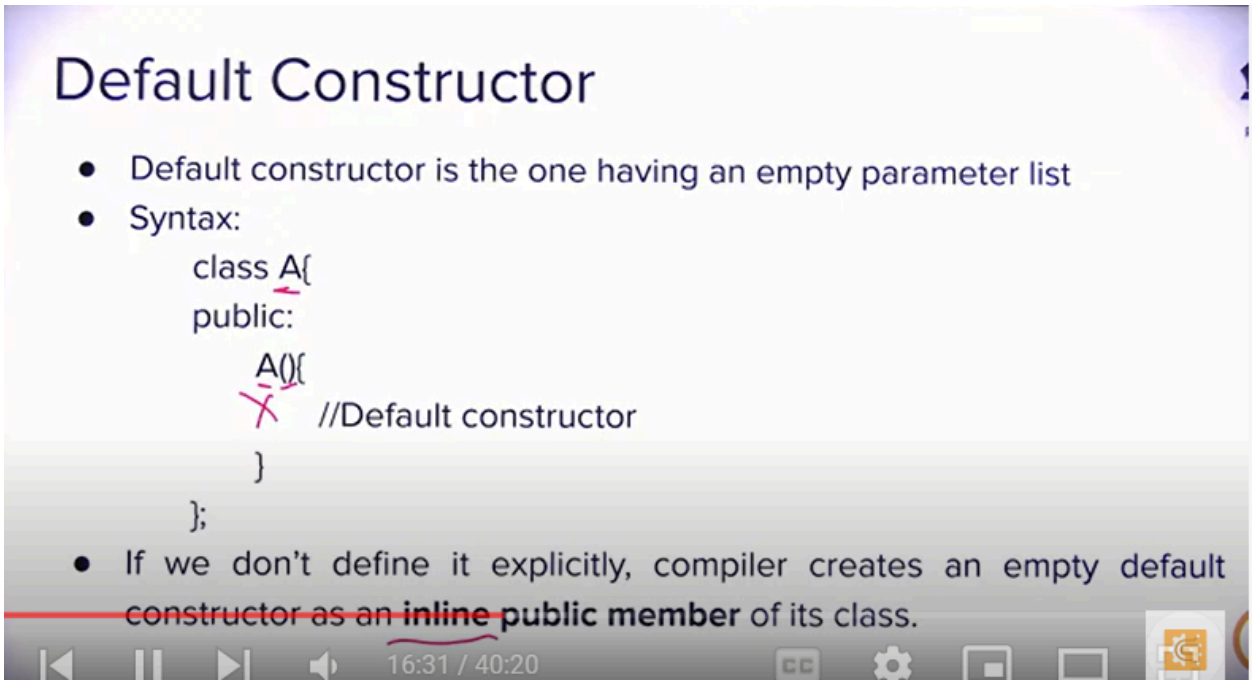
.....



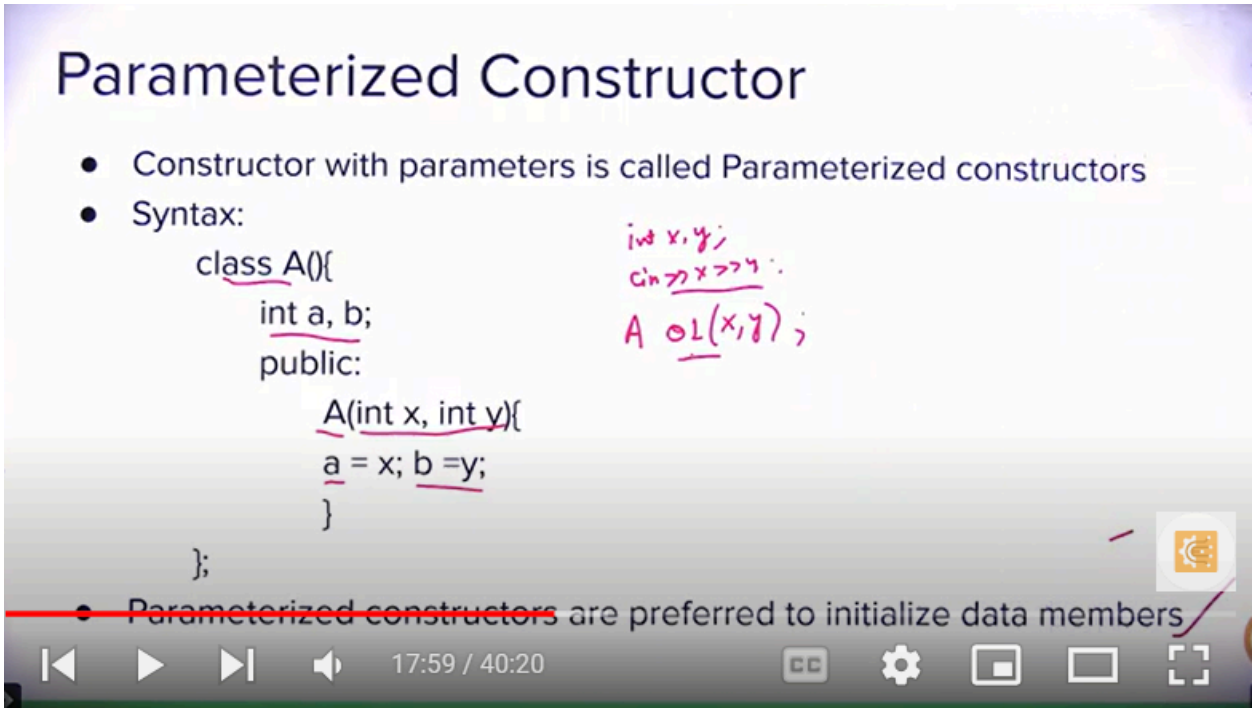
.....


Types of Constructors
Constructors are broadly classified in three types:
Default Constructor
Parameterized Constructor
Copy Constructor
15:18 / 40:20

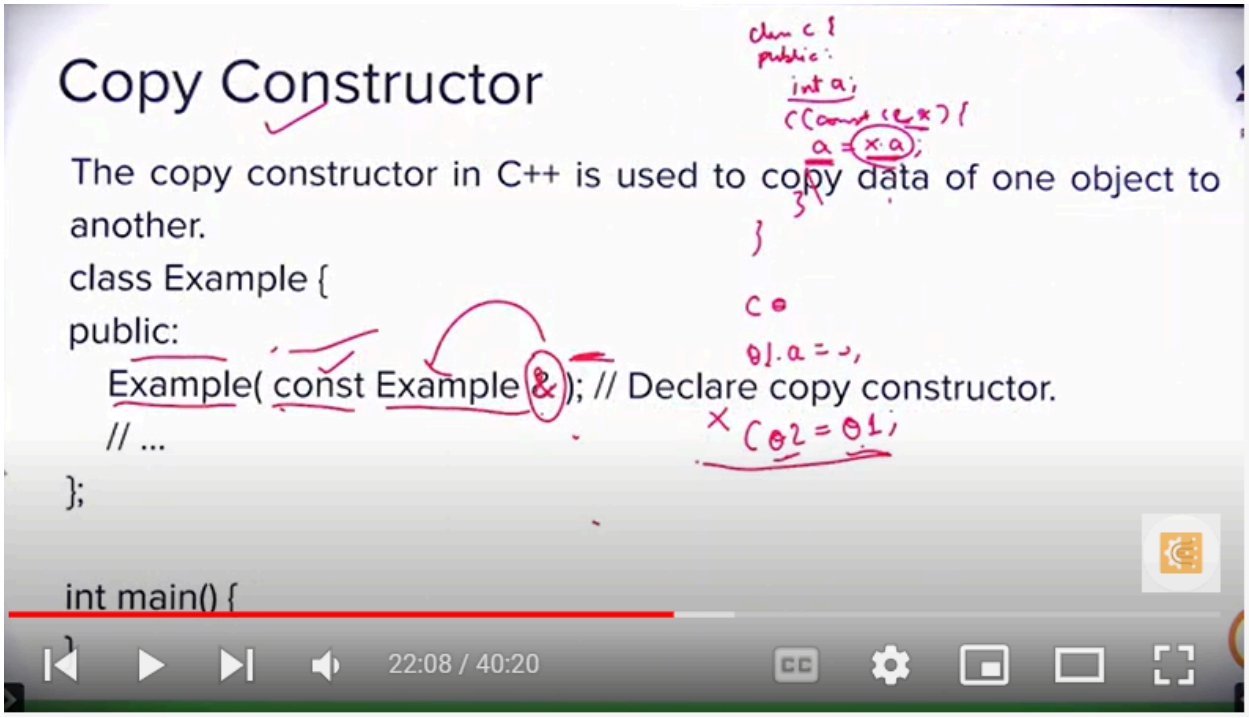
.....


Default Constructor
Default constructor is the one having an empty parameter list
Syntax:
class A{
public:
A(){
//Default constructor
}
};
If we don't define it explicitly, compiler creates an empty default constructor as an inline public member of its class.
16:31 / 40:20

.....



.....



→ Compiler by default 2 constructor banata hai..agar class me koi bhi constructor nhi bna ho to….
1.default constructor
2.Copy constructor

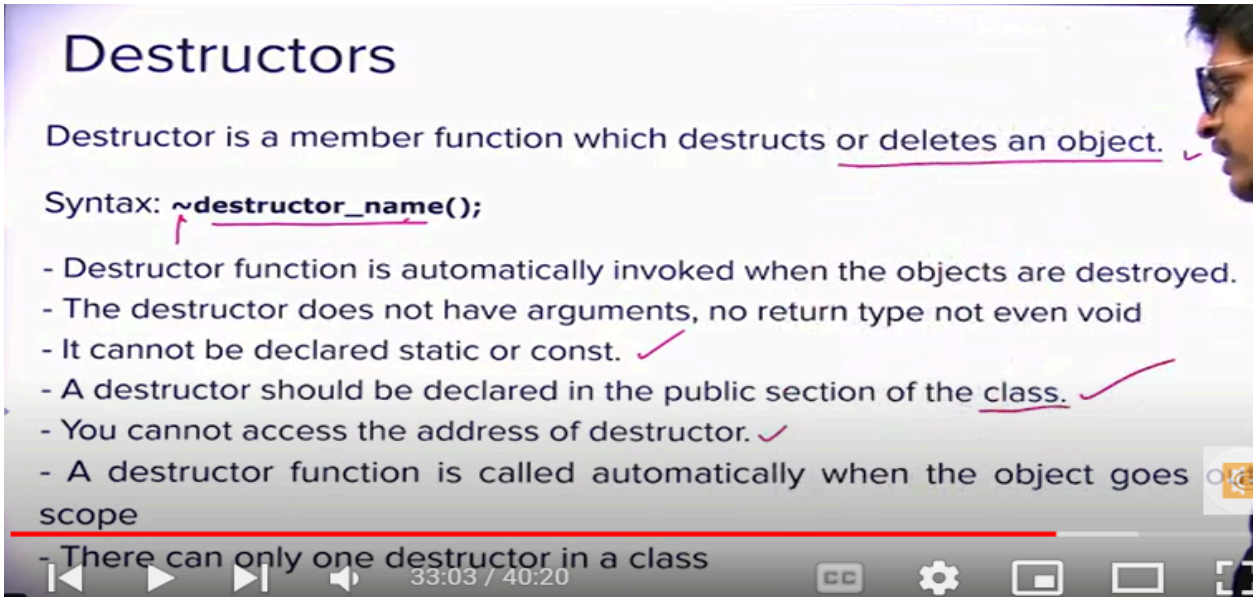
→ agar koi constructor bna diya hai..to compiler koi bhi default constructor nhi baniga.
But copy constructor to banaiga yadi hamne nhi banaya hai to…

→ agar hmne default and copy dono constructor banaya hai…to compiler koi bhi constructor nhi baniga.

—> copy constructor me reference variable banana compulsory hota hai,...nhi to arror aaiga..

==========

Dectrucor:



.....