Virtual class in CPP
==========
Practice Problems in Virtual Class and methods | L:27 | C++ | Ravindrababu Ravual | Jay Bansal
==========
Example 1:

```
#include <iostream>
using namespace std;
//--------------------------
class A{
    public:
        void fun()
        {
            cout<<"function in class A"<<endl;
        }
};
class B:public A{
    public:
        void fun()//overriding the fun() in class A
        {
            cout<<"function in class B"<<endl;
        }
};
int main()
{
    B b;//create the object of class b
    b.fun();//function in class B

    //to call the function in class A we have to resolve
    b.A::fun();//function in class A
    b.B::fun();//function in class B
    return 0;
}
```

—----------

Bez of the function overriding b.fun() will call the fun() present in class B.

To call the fun() present in A we have to resolve .
b.A::fun()===> call the function in class A
==========

Example 2:
==========

```
#include <iostream>
using namespace std;
//--------------------------
class A{
    public:
        void fun()
        {
            cout<<"function in class A"<<endl;
        }
};
class B{
    public:
        void fun()
        {
            cout<<"function in class B"<<endl;
        }
};
class C: public A, public B{
    public:
        void fun()
        {
            cout<<"function in class c"<<endl;
        }
};

int main()
{
    C c;//create the object of class c
    c.fun();//function in class c
    //All the functions will be overridden by the child class. So the
fun() in class C will be executed.

    //to call the function in class A and class B we have to resolve
    c.A::fun();//function in class A
    c.B::fun();//function in class B
    return 0;
}
```

Example 3:

```
#include <iostream>
using namespace std;
//--------------------------
class A{
    public:
        void fun()
        {
            cout<<"function in class A"<<endl;
        }
};
class B{
    public:
        void fun()
        {
            cout<<"function in class B"<<endl;
        }
};
class C: public A, public B{
    public:
        void fun2()
        {
            // fun();//error: reference to 'fun' is ambiguous
            //to resolve the ambiuity problem
            A::fun();//function in class A
            B::fun();//function in class B
            cout<<"function in class C"<<endl;
        }
};
int main()
{
    C c;
    c.fun2();//function in class c
    return 0;
}
```

Example 4:
==========

```
#include <iostream>
using namespace std;
//--------------------------
class A{
    public:
        void fun()
        {
            cout<<"function in class A"<<endl;
        }
};
class B{
    public:
        void fun()
        {
            cout<<"function in class B"<<endl;
        }
};
class C: public A, public B{
    public:
        void fun()
        {
            fun();
            cout<<"function in class C"<<endl;
        }
};
int main()
{
    C c;
    c.fun();//function in class c will be called
    return 0;
}
```

This will be an infinite recursion.
fun() in class C will call fun() in class C bez of the function overriding….
So no output will be printed and it will be an infinite recursion.
==========
==========

Example 5:
==========

```
#include <iostream>
using namespace std;
//--------------------------
class B{
    private:
        int a;
    public:
        void fun()
        {
            cout<<"Base class B"<<endl;
        }
};
class D:public B{
    private:
        int a;
    public:
        void fun()
        {
            cout<<"Derived class D"<<endl;
        }
};

int main()
{
    B b = B();
    b.fun();//Base class B
    return 0;
}
```

——---

B obj =B();==> we are creating the object of class B and initializing it with the object of class B.
B obj =D();==>we are creating the object of class B and initializing it with the object of class D.

Note: any object of the base class can hold the object of the derived class.
==========

Example 6:

```
#include <iostream>
using namespace std;
//--------------------------
class B{
    private:
        int a;
    public:
        void fun()
        {
            cout<<"Base class B"<<endl;
        }
};
class D:public B{
    private:
        int a;
    public:
        void fun()
        {
            cout<<"Derived class D"<<endl;
        }
};

int main()
{
    B b = D();//Base class object is initialized with object of derived
class
    b.fun();//Base class B
    return 0;
}
```

—----------

B obj =D();==>we are creating the object of class B and initializing it with the object of class D.

Obj base ka hia===>therefore base ka fun() call hoga.,yadi hum derived ka obj store kr rhe hai fir bhi.

Therefore we have to use the virtual function to do the run time bunding of the fun() with the object.

==========

Example 7:

```
#include <iostream>
using namespace std;
//--------------------------
class B{
    private:
        int a;
    public:
        virtual void fun()
        {
            cout<<"Base class B"<<endl;
        }
};
class D:public B{
    private:
        int a;
    public:
        void fun()
        {
            cout<<"Derived class D"<<endl;
        }
};

int main()
{
    B objb;
    D objd;
    B *b = &objb;
    b->fun();//Base class B
    return 0;
}
```

——————

Virtual fun()==> fun() ka late binding karna hai.

Here pointer⇒ Base ka hai and obj bhi base ka hai…base ka fun() call hoga…

==========

Example 8

```
#include <iostream>
using namespace std;
//--------------------------
class B{
    private:
        int a;
    public:
        virtual void fun()
        {
            cout<<"Base class B"<<endl;
        }
};
class D:public B{
    private:
        int a;
    public:
        void fun()
        {
            cout<<"Derived class D"<<endl;
        }
};

int main()
{
    B objb;
    D objd;
    B *b = &objd;
    b->fun();//Derived class D
    return 0;
}
```

——---

Virtual fun()==> fun() ka late binding karna hai..

Here pointer⇒ Base ka hai and obj child ka hai…child ka fun() call hoga…

==========

Example 9:

```
#include <iostream>
using namespace std;
//--------------------------
class B{
    private:
        int a;
    public:
        virtual void fun()
        {
            cout<<"Base class B"<<endl;
        }
};
class D:public B{
    private:
        int a;
    public:
        void fun()
        {
            cout<<"Derived class D"<<endl;
        }
};

int main()
{
    B objb;
    D objd;
    // D *p = &objb;//Child class pointer can't hold the parent class
object address.
    D *p = &objd;
    p->fun();//Derived class D
    return 0;
}
```

——----------

Here , pointer derived class ka hai..and it can hold the address of object of the derived class only.

==========

Example 10:

```
#include <iostream>
using namespace std;
//--------------------------
class B{
    private:
        int a;
    public:
        static virtual void fun()//error: member ‘fun’ cannot be declared
both ‘virtual’ and ‘static’
        {
            cout<<"Base class B"<<endl;
        }
};
class D:public B{
    private:
        int a;
    public:
        void fun()
        {
            cout<<"Derived class D"<<endl;
        }
};
int main()
{
    B objb;
    D objd;
    B *p = &objb;
    p->fun();
    return 0;
}
```

If some method is described as static ⇒ this method is the property of that class, not the object. Static functions are resolved at compile time, while virtual functions are resolved at runtime based on the object's type.

Virtual functions can not be static ⇒ bez, virtual functions are made so that they can be inherited in child class and their object can use them. Virtual functions need an object to operate on.

So it does not make any sense to make the virtual function static.

Example 11

```
#include <iostream>
using namespace std;
//--------------------------
class B //class B is an abstract class because it contains pure virtual
fn.
{
    private:
        int a;
    public:
        virtual void fun1()=0;//pure virtual function
        void fun2()
        {
            cout<<"this is the fun2 in the base class"<<endl;
        }
};
class D:public B{
    private:
        int a;
    public:
        D()
        {
            cout<<"Derived class Constructor"<<endl;
        }
};
int main()
{
    B objb;//error: cannot declare variable 'objb' to be of abstract type
'B'
    D objd;//error: cannot declare variable 'objd' to be of abstract type
'D'
    objb.fun1();
    return 0;
}
```

Virtual void fun()=0; ⇒ this is pure virtual function.
Any class which contains the pure virtual function is called an **abstract class.**

Any class inheriting the abstract class **should contain** the definition of pure virtual function.
Else that child class will also become the abstract class.
We can not create the object of the abstract class.

Example 12

```
#include <iostream>
using namespace std;
//--------------------------
class B{
    private:
        int a;
    public:
        virtual void fun1(){
            cout<<"virtual fun1 in Base"<<endl;
        }
        void fun2()
        {
            cout<<"fun2 in the base class"<<endl;
        }
};
class D:public B{
    private:
        int a;
    public:
        void fun1(){
            cout<<"override the virtual fun1 in the Derived class"<<endl;
        }
        void fun2()
        {
            cout<<"Derived class fun2"<<endl;
        }
};
int main()
{
    B *b;
    D objd;
    b=&objd;

    b->fun2();//fun2 in the base class
    b->fun1();//override the virtual fun1 in the Derived class
    return 0;
}
```

Example 12.1

```
#include <iostream>
using namespace std;
//--------------------------
class B{
    private:
        int a;
    public:
        virtual void fun1(){
            cout<<"virtual fun1 in Base"<<endl;
        }
        void fun2()
        {
            cout<<"fun2 in the base class"<<endl;
        }

};
class D:public B{
    private:
        int a;
    public:
        void fun2()
        {
            cout<<"Derived class fun2"<<endl;
        }
};
int main()
{
    B *b;
    D objd;
    b=&objd;
    b->fun1();//virtual fun1 in Base
    return 0;
}
```

If you have any virtual function,it does not mean that again you have to provide the definition in child class.
So it does not give any error if we do not provide the defination.

Example 13

```
#include <iostream>
using namespace std;
//--------------------------
class B{
    private:
        int a;
    public:
        virtual void fun1()=0;
        void fun2()
        {
            cout<<"fun2 in the base class"<<endl;
        }

};
class D:public B{
    private:
        int a;
    public:
        void fun1()
        {
            cout<<"Implementing the pure virtual function in Derived
class"<<endl;
        }
};
int main()
{
    //B objb;//error: cannot declare variable 'objb' to be of abstract
type 'B'
    D objd;
    objd.fun2();//fun2 in the base class
    return 0;
}
```

We can not create the object of abstract class.
But we can create the pointer of the abstract class.

Example 13.1

```
#include <iostream>
using namespace std;
//--------------------------
class B //B is an abstract class
{
    private:
        int a;
    public:
        virtual void fun1()=0;//pure virtual function
        void fun2()
        {
            cout<<"fun2 in the base class"<<endl;
        }
};
class D:public B{
    private:
        int a;
    public:
        void fun1()
        {
            cout<<"Implementing the pure virtual function in Derived
class"<<endl;
        }
};
int main()
{
    //B objb;//We can not create the object of abstract class.
    B *b;//But we can create the pointer of the abstract class.
    D objd;
    b=&objd;
    b->fun2();//fun2 in the base class
    return 0;
}
```

Example 14

```
#include <iostream>
using namespace std;
class A//class A is abstract class, because it contain the pure virtual
function
{
    public:
        virtual void fun1()=0;//this is the pure virtual function in A
        void fun2(){
            cout<<"fun2 body in class A"<<endl;
        }
};
class B:public A
{
    public:
        virtual void fun1(){
            cout<<"Implementing the PVF of class A in class B"<<endl;
        }
        virtual void fun3()=0;
};
class C:public B
{
    public:
        virtual void fun3(){
            cout<<"Implementing the PVF of class B in class C"<<endl;
        }
};
int main()
{
    C objc;
    objc.fun2();//fun2 body in class A
    return 0;
}
```

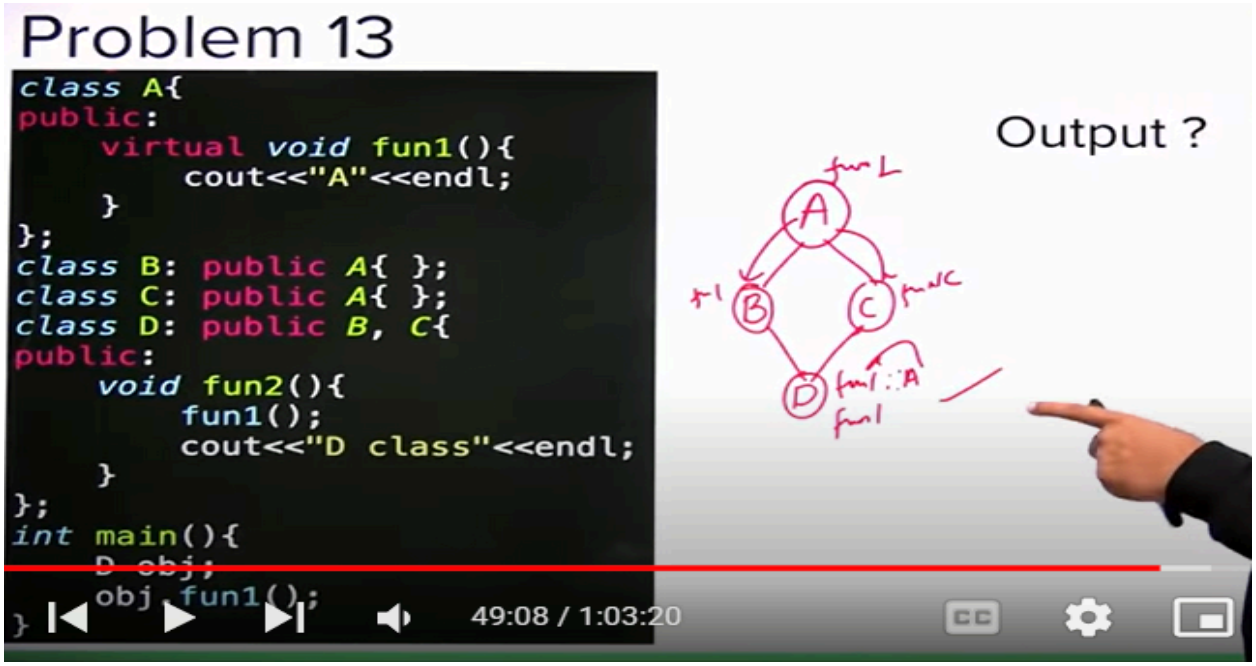
Class B inherits the class A (abstract class) so it must contain the definition of all the pure virtual functions. If it will not override the PVF then it will also become an abstract class.
Class C inherits the class B (abstract class) so it must contain the definition of all the pure virtual functions else it will also become the abstract class.

Example 14.1

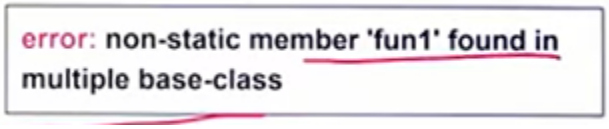
```
#include <iostream>
using namespace std;
class A//class A is abstract class, because it contain the pure virtual
function
{
    public:
        virtual void fun1()=0;//this is the pure virtual function in A
        void fun2(){
            cout<<"fun2 body in class A"<<endl;
        }
};
class B:public A
{
    public:
        virtual void fun1(){
            cout<<"Implementing the PVF of class A in class B"<<endl;
        }
        virtual void fun3()=0;
};
class C:public B,public A
{
    public:
        virtual void fun3()
        {
            cout<<"Implementing the PVF of class B in class C"<<endl;
        }
};
int main()
{
    C objc;//error: cannot declare variable 'objc' to be of abstract type
'C'
//unimplemented PVF fun1 in class C
    return 0;
}
```

Since class c is inheriting the class A directly, it should give the definition of the fun1() which is PVF in class A, else it will also becomes the abstract class.

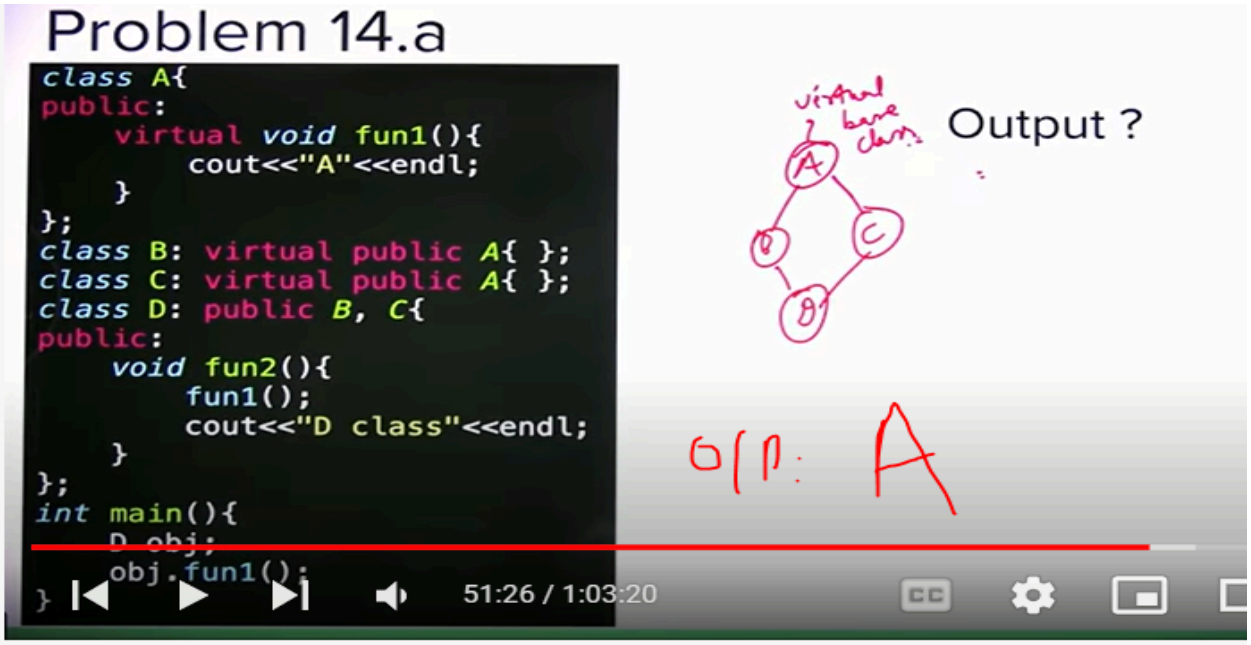
Example 15



In class D there are 2 fun1(),so compiler is unable to distinguish ===>error



Example 15.1 ⇒ **Virtual Base Class**



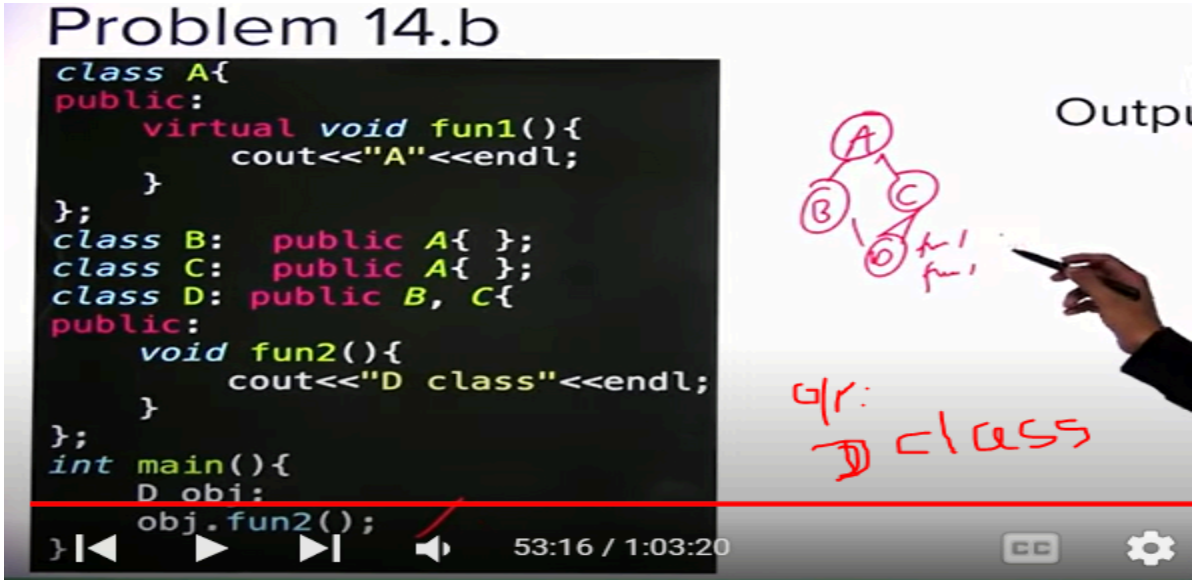
Here we have inherited class A virtually,so A acts as virtual base class.
whenever you inherit virtual base class,it is like all the member of base class are static,
So only one of them is used.

Here are the key points about virtual base classes in C++:

1. **Diamond Problem:** Multiple inheritance can lead to the diamond problem, where a class inherits from two classes that both inherit from a common base class. This creates ambiguity in the inheritance hierarchy, especially when there are member variables or functions with the same name in the intermediate base classes.
2. **Virtual Base Class:** To solve the diamond problem, you can use virtual base classes. When a base class is declared as virtual, it means that only one instance of the base class will be present in the inheritance hierarchy, even if multiple derived classes inherit from it indirectly.
3. **Single Instance:** By using a virtual base class, the derived classes that inherit from it indirectly share a single instance of the virtual base class. This ensures that there are no duplicate subobjects of the virtual base class in the final derived class.
4. **Object Layout:** The use of virtual base classes affects the object layout in memory. It introduces additional mechanisms, such as a **virtual base pointer**, to ensure that the correct instance of the virtual base class is accessed and that there are no duplicate subobjects.
5. **Initialization:** When constructing objects that involve virtual base classes, the virtual base class part of the object is initialized before any non-virtual base classes. This ensures that the virtual base class is properly constructed and avoids issues with uninitialized data.

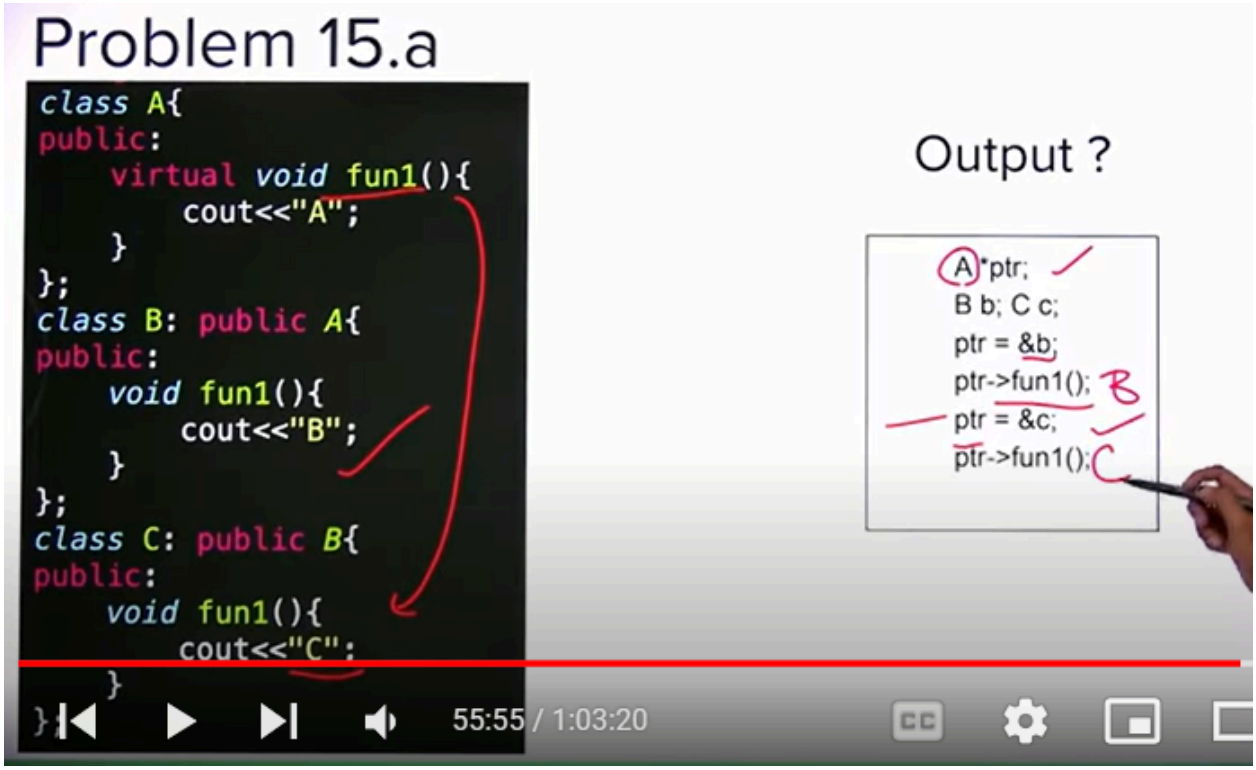
In summary, virtual base classes in C++ are used to handle multiple inheritance scenarios where the diamond problem could arise. They allow for a single shared instance of the base class in the inheritance hierarchy, ensuring proper object layout and avoiding ambiguities in member access.

Example 16

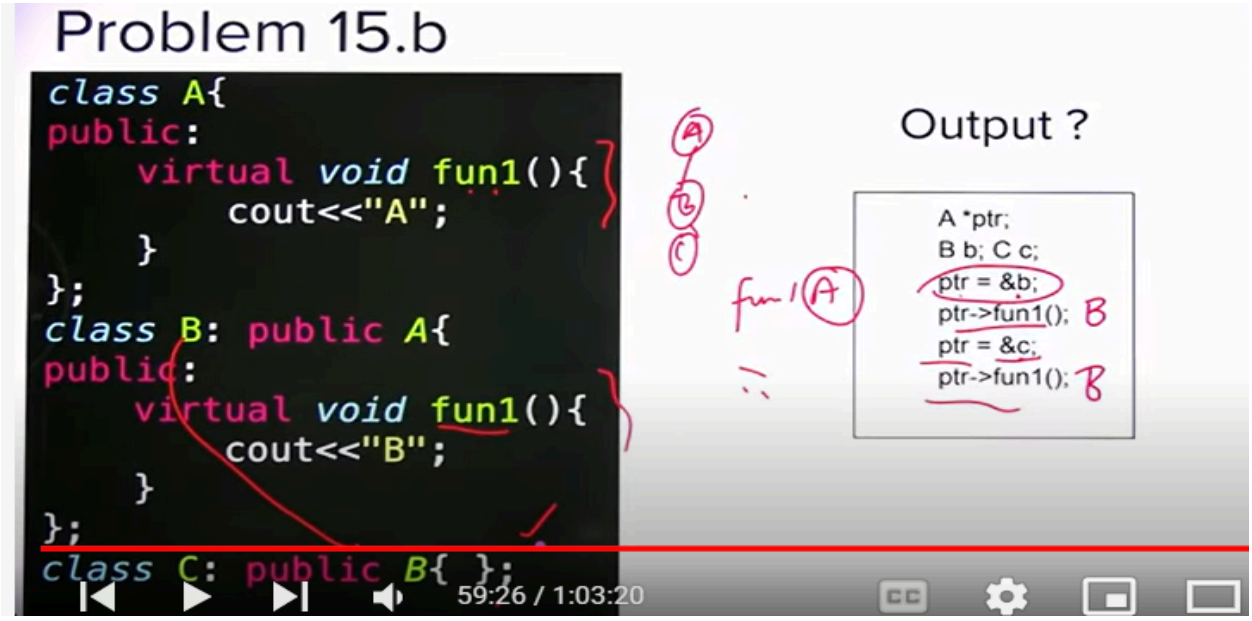


Here there are 2 fun() function in class D,but jab tak usko call nhi karenge tab tk koi bhi error nahi aayega,
Compiler will give an error only when we call the fun1().

Example 17



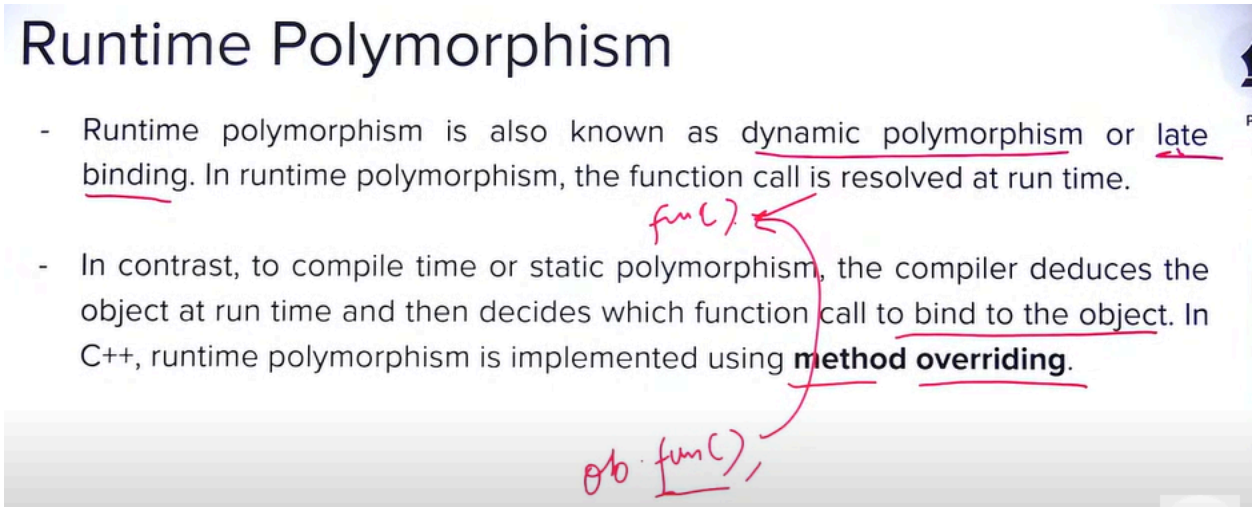
Example 18



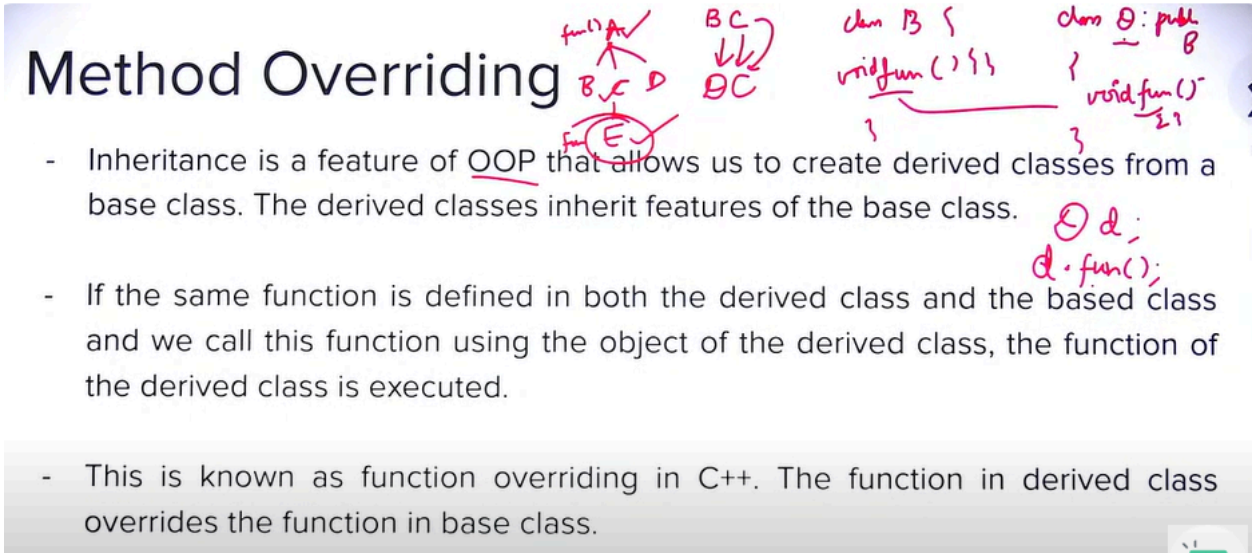
Example 18 ⇒ Virtual Base Pointer

Virtual Class, Virtual Methods, Abstract Classes | L:26 | C++

Runtime polymorphism ⇒



—----------



—----------

Virtual Functions

- A virtual function is a member function in the base class that we **expect to redefine** in derived classes.

- A virtual function is used in the base class in order to ensure that the function is overridden. This especially applies to cases where a pointer of base class points to an object of a derived class.
