

Practice Problems on Inheritance | L:22 | C++ | Competitive Programming

Q1: when we create the child class object **then the constructor of both parent and child class** will be called.,But

- a—> parent class constructor will be called first.
- b—> then the child class constructor will be called.

The image shows a video player interface with a C++ code editor and a terminal window below it.

Code Editor (Top Left):

```
#include<iostream>
using namespace std;
class A{
public:
    A(){
        cout<<"A";
    }
};
class B: public A{
public:
    B(){
        cout<<"B";
    }
};
int main(){
    B obj;
    return 0;
}
```

Terminal (Bottom Left):

```
10  using namespace std;
11  class A
12  {
13      public:
14          A()
15          {
16              cout<<"A";
17          }
18      };
19  class B : public A
20  {
21      public:
22          B()
23          {
24              cout<<"B";
25          }
26      };
27  class C : public B
28  {
29      public:
30          C()
31          {
32              cout<<"C";
33          }
34      };
35  int main()
36  {
37      C obj;
38      return 0;
39  }
```

Output (Top Right):

A B

Terminal Output (Bottom Right):

```
ABC
...Program finished with exit code 0
```

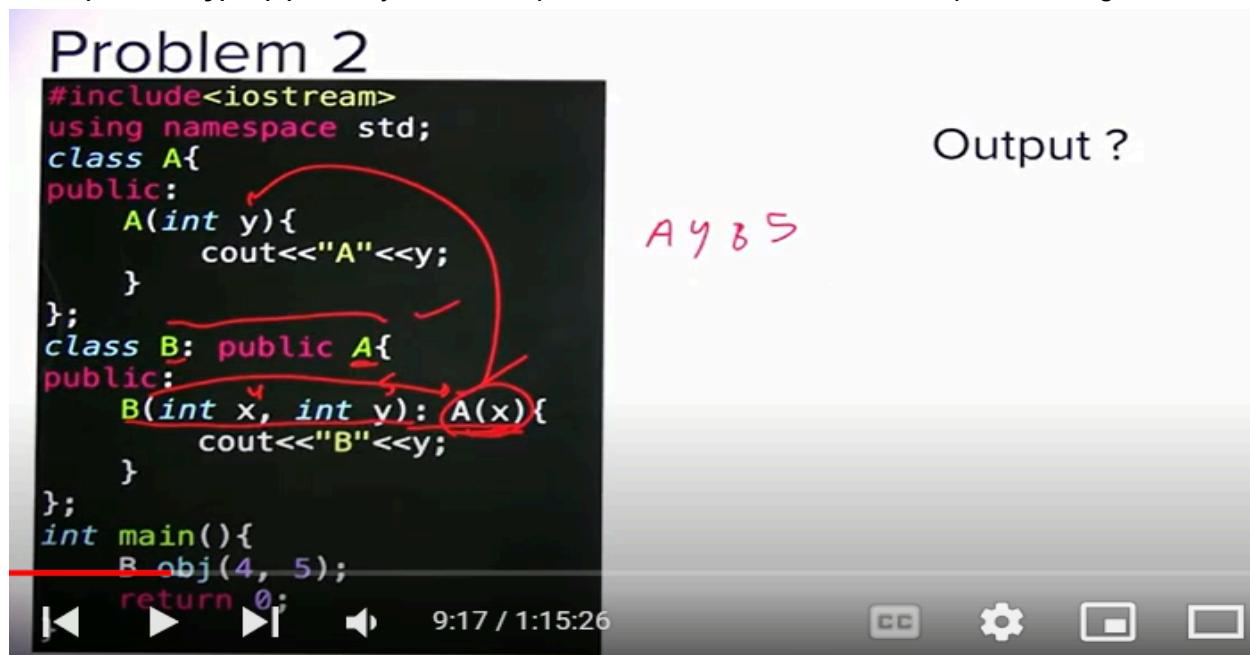
Q2: **B(int x,int y):A(x)** ⇒ way to call the parent constructor if we have to pass the argument.

Problem 2

```
#include<iostream>
using namespace std;
class A{
public:
    A(int y){
        cout<<"A"<<y;
    }
};
class B: public A{
public:
    B(int x, int y): A(x){
        cout<<"B"<<y;
    }
};
int main(){
    B obj(4, 5);
    return 0;
}
```

Output ?

A 4 B 5

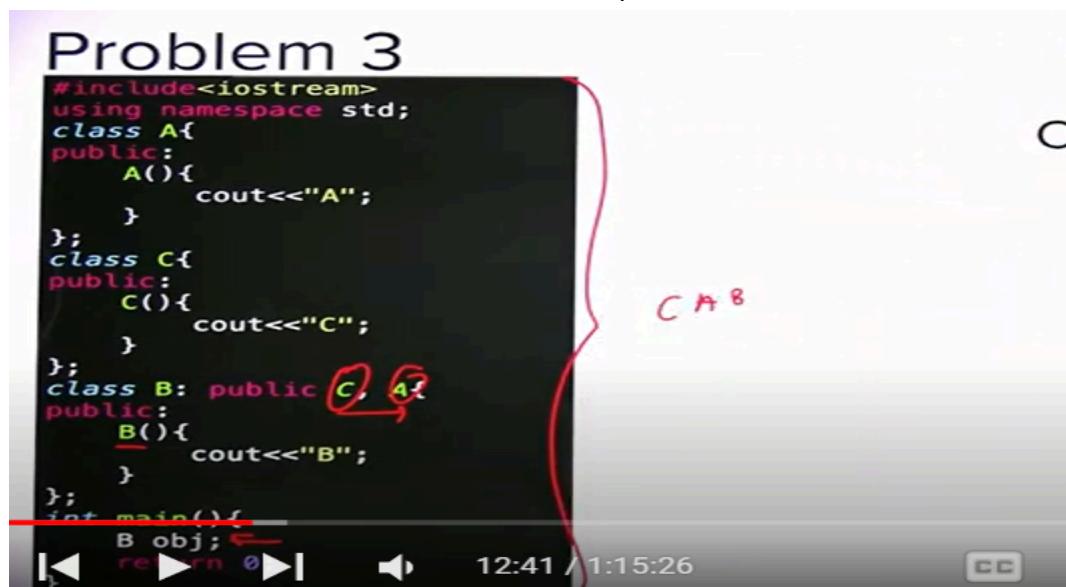


Q3: constructor are called in the order in which parent classes are written.

Problem 3

```
#include<iostream>
using namespace std;
class A{
public:
    A(){
        cout<<"A";
    }
};
class C{
public:
    C(){
        cout<<"C";
    }
};
class B: public C, A{
public:
    B(){
        cout<<"B";
    }
};
int main(){
    B obj;
    return 0;
}
```

C A B



Q4:

constructor of parents are always called in the order in which the classes are inherited,no matter how you have written it in the constructor.

If the parent class does not have any constructor then the default constructor of the parent class will be called.

Problem 4

```
#include<iostream>
using namespace std;
class A{
public:
    A(int x){
        cout<<"A"<<x;
    }
};
class C{
public:
    C(int x){
        cout<<"C"<<x;
    }
};
class B: public A, C{
public:
    B(int x, int y, int z): C(x), A(y){
        cout<<"B"<<z;
    }
};
int main(){
    B obj(2, 5, 9);
    return 0;
}
```

Output ?

A5C2B9 ✓



Q5-A :

1. when we create any constructor inside the class then the compiler will not create the default constructor.

2. When the **obj** of the child is created , first constructor of the parent class will be called.

Here we have not called the parent class constructor explicitly in the child , then default constructor of parent class will be called.

But in the child default constructor, are not present and compiler also not create the default constructor because we have created the parameterized constructor.

So it will give **error**.

Problem 5.a

```
#include<iostream>
using namespace std;
class A{
public:
    A(int x){ cout<<"A"<<x; }
};
class B: public A{
public:
    B(int x){ cout<<"B"<<x; }
};
int main(){
    B obj(12);
    return 0;
}
```

Output ?

error: constructor for 'B' must explicitly initialize the base class 'A' which does not have a default constructor

```
9 #include <iostream>
10 using namespace std;
11 class A
12 {
13     public:
14         A()
15     {
16         cout<<"A";
17     }
18 };
19 class B : public A
20 {
21     public:
22         B(int x)
23     {
24         cout<<"B"<<x;
25     }
26 };
27 int main()
28 {
29     B obj(10);
30     return 0;
31 }
32
```

AB10
...Program finished with exit code 0

Q5-B:

Problem 5.b

```
#include<iostream>
using namespace std;
class A{
public:
    A(int x){
        cout<<"A"<<x;
    }
};
class B: public A{
public:
    B(int x): A(x){
        cout<<"B"<<x;
    }
};
int main(){
    B obj(12);
    return 0;
}
```

Output

A 12 B 12

It will not give any error

Op: A 12 B 12

Q6 -A :

A child class can only call the constructor of its direct(immediate) parent class , not any other parent class.

Problem 6.a

```
#include<iostream>
using namespace std;
class A{
public:
    A(int x){
        cout<<"A"<<x;
    }
};
class B: public A{
public:
    B(int x){
        cout<<"B"<<x;
    }
};
class C: public B{
public:
    C(int x, int y, int z): A(y), B(z){
        cout<<"C"<<x;
    }
};
int main(){
    C obj(2, 5, 9);
    return 0;
}
```

Output

Error!

C is trying to call the constructor of class A but it is not inheriting class A.==> therefore Error

error: type 'A' is not a direct or virtual
base of 'C'

Q6-B :

Problem 6.b

```
#include<iostream>
using namespace std;
class A{
public:
    A(int x){
        cout<<"A"<<x;
    }
};
class B: public A{
public:
    B(int x, int y): A(y){
        cout<<"B"<<x;
    }
};
class C: public B{
public:
    C(int x, int y, int z): B(y, z){
        cout<<"C"<<x;
    }
};
int main(){
    C obj(2, 5, 9);
    return 0;
}
```

A 9
B 5
C 2

Q6-C :

Problem 6.c

```
class A{
public:
    A(int x){
        cout<<"A"<<x;
    }
    void get(){cout<<" Hello world ";}
};
class B: public A{
public:
    B(int x, int y): A(y){
        cout<<"B"<<x;
    }
};
class C: public B{
public:
    C(int x, int y, int z): B(y, z){
        get();
        cout<<"C"<<x;
    }
};
int main(){
    C obj(2, 5, 9);
    return 0;
}
```

A 9 B 5
Hello world
C 2

Output ?

Q7 -A :

Protected members can be inherited.

But it can not be accessed outside of the class. Inside the class it can be accessed.

Problem 7.a

```
class A{  
public:  
    A(int x){  
        cout<<"A"<<x;  
    }  
    void get(){cout<<" Hello world ";}  
};  
class B: protected A{  
public:  
    B(int x, int y): A(y){  
        cout<<"B"<<x;  
    }  
};  
class C: public B{  
public:  
    C(int x, int y, int z): B(y, z){  
        get();  
        cout<<"C"<<x;  
    }  
};
```

P ↗✓
Ans ✓ Output ?

C obj(2, 5, 9);

Op:

A9B5 Hello world C2

Class B is derived from the A in protected mode, so get() will become the protected in the class B.

Again class C is derived from the class B in public mode, so get() will remain protected inside the class C, bez maximum security will be taken.

Q7 B :

Problem 7.b

```
class A{  
public:  
    A(int x){  
        cout<<"A"<<x;  
    }  
    void get(){cout<<" Hello world ";}  
};  
class B: protected A{  
public:  
    B(int x, int y): A(y){  
        cout<<"B"<<x;  
    }  
};  
class C: public B{  
public:  
    C(int x, int y, int z): B(y, z){  
        get();  
        cout<<"C"<<x;  
    }  
}
```

Output ?

A 20
H W
A a(20);
a.get();

Q7 C:

Problem 7.c

```
class A{  
public:  
    A(int x){  
        cout<<"A"<<x;  
    }  
    void get(){cout<<" Hello world ";}  
};  
class B: protected A{  
public:  
    B(int x, int y): A(y){  
        cout<<"B"<<x;  
    }  
};  
class C: public B{  
public:  
    C(int x, int y, int z): B(y, z){  
        get();  
        cout<<"C"<<x;  
    }  
}
```

→ Public ✓
→ protected X
→ private X
Access can be inherited ✓
Output ?

main()
 B obj2(4,5);
 obj2.get(); X
}

error: 'get' is a protected member
of 'A'

get() will be inherited but it will become protected in the class B.
Protected members can not be accessed outside of the class ==> therefore it will give the error.

	can be accessed outside of the class	can be inherited
public:	yes	yes
protected	No	yes
private	NO	No

Q8:

Problem 8

```
class A{
public:
    ~A(){
        cout<<"A";
    }
};
class B{
public:
    ~B(){
        cout<<"B";
    }
};
class C: public A, B{
public:
    ~C(){
        cout<<"C";
    }
}
```

Output ?

C c;

CBA

The screenshot shows a video player interface with a dark theme. On the left is the C++ code for classes A, B, and C. On the right, there are two text boxes: one labeled 'Output ?' containing 'C c;', and another below it containing 'CBA'. At the bottom of the screen, there are standard video control buttons (rewind, fast forward, play, volume) and a timestamp '56:22 / 1:15:26'.

Destructor are called in the reverse order in which constructor are called.

Q9 :

Problem 9

```
class base {
public int arr[10]; // 40B
};

class b1: public base { };

class b2: public base { };

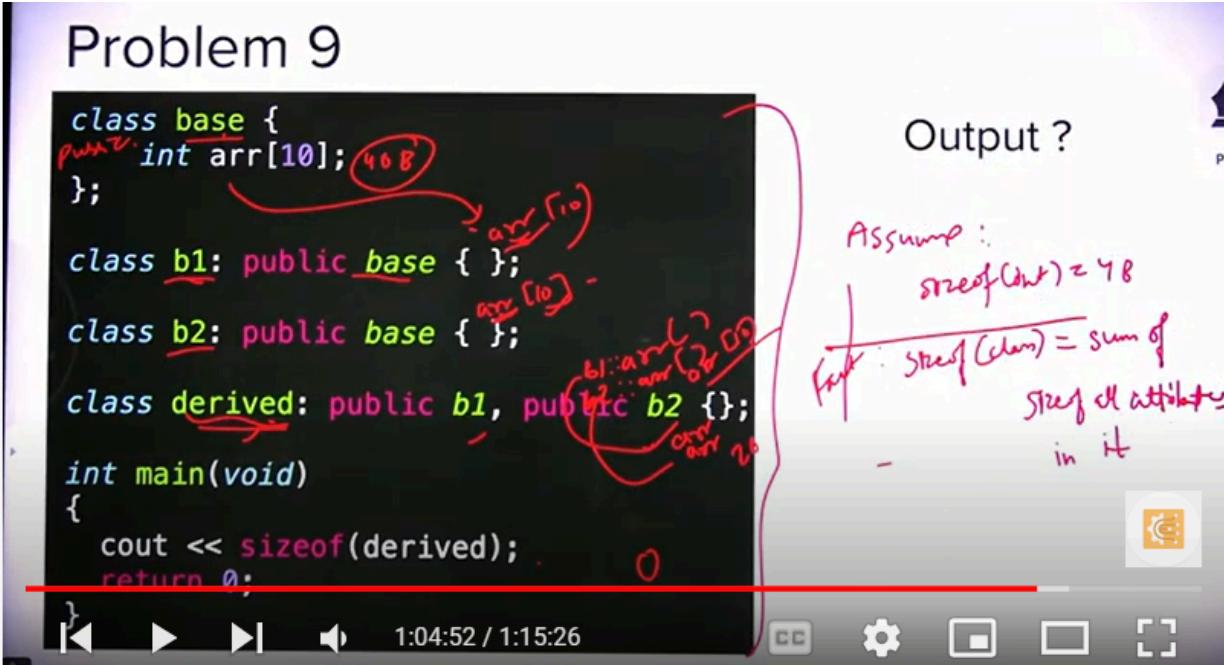
class derived: public b1, public b2 {};

int main(void)
{
    cout << sizeof(derived);
    return 0;
}
```

Output ?

Assume :
 $\text{sizeof}(\text{arr}) = 40$

~~fact : $\text{sizeof}(\text{class}) = \text{sum of}$
 $\text{sizeof all attributes}$
in it~~



2 arr will be present in the derived class.

There name will be same, hence we will use the class name to resolve the conflict.

```
9 #include <iostream>
10 using namespace std;
11 class A
12 {
13     public:
14         int arr[10];
15 };
16 class B1 : public A
17 {};
18 class B2 : public A
19 {};
20 class derived : public B1, public B2
21 {};
22
23 int main()
24 {
25     cout << sizeof(derived);
26     return 0;
27 }
```

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

Q10 :

Problem 10.a

Which specifier makes all the data members and functions of base class inaccessible by the derived class?

- a. private ✓
- b. protected
- c. public
- d. both private and protected



Problem 10.b

```
class A{  
public:  
    int a;  
    void change(int i){  
        a = i;  
    }  
    void value_of_a(){  
        cout<<a;  
    }  
};  
class B: public A{  
    int a = 15;  
public:  
    void print(){  
        cout<<a;  
    }  
}
```

Output ?

```
int main() {  
    B b;  
    b.change(10);  
    b.print();  
    b.value_of_a();  
}
```

Op: 15 10

Note: the base class method which is inherited, will access the base class variable only.

So when b.change() will be called, variable a which are present in the class A will be accessed by the change method.

b.print()--> it will access the variable a which are present in the class B

b.value_of_a()--> it will access the variable in class A.

Inheritance | L:21 | C++ | Competitive Programming |

Ravindrababu Ravula | Jay Bansal

C++ Inheritance

```
class vehicle { public: void set(string milage); };
class A : public vehicle {
public: void print();
};
class B : public A {
public: void print();
};
```

Set extra milage
class vehicle
class A
class B
B obj;

- In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.
- In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.
- In C++, the class which inherits the members of another class is called **derived class** and the class whose members are inherited is called **base class**. The derived class is the **specialized class** for the base class.

Derived Classes

A Derived class is defined as the class derived from the base class:

The Syntax of Derived class:

```
class derived_class_name :: visibility-mode base_class_name {  
    // body of the derived class.
```

}

derived_class_name: It is the name of the derived class.

visibility mode: The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

The screenshot shows a video player interface with a background image of a presentation slide. Handwritten notes are overlaid on the slide, explaining C++ access specifiers. On the left, there's a 'New Batch Announced' banner for 'GENIUS 2021'. The handwritten notes include:

- Class A definition: `class A {
public:
int a;
protected:
int b;
private:
X int c;
};`
- Class B definition: `class B : public A {
int fun();
};`
- Diagram illustrating inheritance: `B obj;` is shown above `obj.a`, with arrows indicating the inheritance path from public to protected to private members.

At the bottom of the slide, there's a banner with 'ENROLL NOW' and a timestamp '19:54 / 50:40'.

We have to take the maximum security.

Public, Protected and Private inheritance

- public inheritance makes public members of the base class public in the derived class, and the protected members of the base class remain protected in the derived class.
- protected inheritance makes the public and protected members of the base class protected in the derived class.
- private inheritance makes the public and protected members of the base class private in the derived class.



Visibility of Inherited Members

Base class visibility		Derived class visibility	
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	NOT inherited	NOT inherited	NOT inherited



Types of Inheritance

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance



24:27 / 50:40



Single Inheritance



Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.

```
class Base {  
public:  
    float salary = 50000;  
};  
class Derived: public Base {  
public:  
    float bonus = 2000;  
};
```

```
int main(void) {  
    Derived p1;  
    cout<<"Salary: "<<p1.salary<<endl;  
    cout<<"Bonus: "<<p1.bonus<<endl;  
    return 0;  
}  
Output:  
Salary: 50000  
Bonus: 2000
```



34:38 / 50:40



Single Inheritance

```
class Base {  
public:  
    int a, b;  
    int mul() {  
        return a * b;  
    }  
};  
  
class Derived: private Base {  
public:  
    void f() {  
        int a, b;  
        return mul(a * b);  
    }  
};
```

```
int main(void) {  
    Derived b;  
    cout << b.f(2, 3) << endl;  
    return 0;  
} b.mul(2,3) X
```

Output:

6



Multilevel Inheritance

Multilevel inheritance is a process of deriving a class from another derived class. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

```
class Base {  
public:  
    float salary = 50000;  
};  
  
class Derived1: public Base {  
};  
  
class Derived2: public Derived1 {  
public:  
    float bonus = 2000;  
};
```



```
int main(void) {  
    Derived2 p1;  
    cout << "Salary: " << p1.salary << endl;  
    cout << "Bonus: " << p1.bonus << endl;  
    return 0;  
}
```

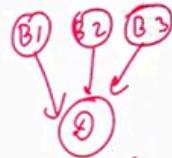
Output:

Salary: 50000
Bonus: 2000



Multiple Inheritance

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.



Syntax of the Derived class:

```
class D : visibility B-1, visibility B-2, ... {  
    // Body of the class;  
}
```

38:57 / 50:40



Ambiguity Resolution in Inheritance

Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base classes.

```
class A {  
public:  
    void fun() {  
        cout << "Class A" << endl;  
    }  
};  
class B {  
public:  
    void fun() {  
        cout << "Class B" << endl;  
    }  
};  
class C : public A, public B {  
public:  
    void fun2() {  
        fun();  
    }  
};
```

error: reference to 'display' is ambiguous
display();

41:30 / 50:40



In class C there are 2 fun() which are derived from the class A and B, so it will be ambiguous.

So we have to resolve the ambiguous by using the class name.

Ambiguity Resolution in Inheritance

Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base classes.

```
class A {
```

```
public:
```

```
void fun() {
```

```
cout << "Class A" << endl;
```

```
}
```

```
};
```

```
class B {
```

```
public:
```

```
void fun() {
```

```
cout << "Class B" << endl;
```

```
}
```

```
class C : public A, public B {
```

```
public:
```

```
void fun2() {
```

```
A :: fun();
```

```
B :: fun();
```

```
}
```

C obj;

obj.fun2();

Class A
Class B



42:47 / 50:40



Ambiguity Resolution in Inheritance

Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base classes.

```
class A {
```

```
public:
```

```
void fun() {
```

```
cout << "Class A" << endl;
```

```
}
```

```
};
```

```
class B {
```

```
public:
```

```
void fun() {
```

```
cout << "Class B" << endl;
```

```
}
```

```
class C : public A, public B {
```

```
public:
```

```
void fun() {
```

```
A :: fun();
```

```
B :: fun();
```

```
}
```

C obj;
A :: fun();
B :: fun();
- Unresolved
- Class B

3 fun in class C



43:56 / 50:40



Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.

