```
#include <iostream>
using namespace std;
class A
{
   public:
   void show()
   {
      cout<<"\nI am in base class A";
   }
};
class B
{
   public:
   void show()
   {
      cout<<"\nI am in base class B";
   }
};
class C:public A,public B
{
   public:
  C()
  {
     cout<<"\nI am default constructor of Class C";
  }
};
int main()
{
   C obj;
   obj.show();
   return 0;
}
---------------------------------------
//Solution-->Ambiguity problem in multiple inheritance
#include <iostream>
using namespace std;
class A
{
   public:
   void show()
   {
      cout<<"\nI am in base class A";
   }
};
class B
{
   public:
   void show()
```

```
    {
       cout<<"\nI am in base class B";
    }
};
class C:public A,public B
{
    public:
  C()
  {
      cout<<"\nI am default constructor of Class C";
  }
};
int main()
{
    C obj;
    obj.A::show();
    obj.B::show();
    return 0;
}
----------------------------------------
#include<iostream>
using namespace std;
class B
{
protected:
int x;
public:
void get_dataB()
{
cout<<"\n Enter value of x:";
cin>>x;
}
};
class DB1:virtual public B
{
protected:
int y;
public:
void get_dataDB1()
{
cout<<"\n Enter value of y:";
cin>>y;
}
};
class DB2: public virtual  B
{
protected:
int z;
public:
```

```
void get_dataDB2()
{
cout<<"\n Enter value of z:";
cin>>z;
}
};
class D:public DB1,public DB2
{
public:
void sum()
{
int result;
result=x+y+z;
cout<<"\n Result is:"<<result;
}
};
int main()
{
D obj1;
obj1.get_dataB();
obj1.get_dataDB1();
obj1.get_dataDB2();
obj1.sum();
return 0;
}
```

```
//Function overriding
#include <iostream>
using namespace std;
class A
{
   public:
   void show()
   {
      cout<<"\nI am in base class A";
   }
};
class B:public A
{
   public:
   void show()
   {
      cout<<"\nI am in derived class B";
   }
};

int main()
{
   B obj;
```

```
   obj.show();
   obj.A::show();
   return 0;
}
---------------------------------------
#include<iostream>
using namespace std;

class A
{
public:
void show()
{
cout<<"This is the base class A"<<endl;
}
};

class B: public A
{
public:
void show()
{
cout<<"This is the derived class B"<<endl;
}
};

class C: public B
{
public:
void show()
{
cout<<"This is the derived class C"<<endl;
}
};

class D: public C
{
public:

};

int main()
{
D obj1;
obj1.show();
return 0;
}
---------------------------------------
#include<iostream>
```

```
using namespace std;

class A
{
public:
void show()
{
cout<<"This is the base class A"<<endl;
}
};

class B: public A
{
public:
void show()
{
cout<<"This is the derived class B"<<endl;
}
};

class C: public A
{
public:
void show()
{
cout<<"This is the derived class C"<<endl;
}
};


int main()
{
B obj1;
obj1.show();
C obj2;
obj2.show();
return 0;
}
-------------------------------------
#include<iostream>
using namespace std;

class A
{
public:
void show(int a)
{
   cout<<"\nValue of a is:"<<a;
}
```

```
void show()
{
cout<<"\nNormal show() in A";
}
};

class B: public A
{
public:
void show()
{
cout<<"\nshow in B"<<endl;
}
};
int main()
{
B obj;
obj.show(5);//Error will come
return 0;
}
```

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

class A
{
public:
void show(int a)
{
   cout<<"\nValue of a is:"<<a;
}
void show()
{
cout<<"\nNormal show() in A";
}
};

class B: public A
{
public:
void show()
{
cout<<"\nshow in B"<<endl;
}
};
int main()
{
B obj;
obj.A::show(5);
```

```
return 0;
}
---------------------------------------
#include<iostream>
using namespace std;
class A
{
public:
A()
{
cout<<"\nCalling default base class constructor";
}
~A()
{
cout<<"\nCalling base class destructor";
}
};
class B:public A
{
public:
B()
{
cout<<"\n Calling default derived class constructor";
}
~B()
{
cout<<"\nCalling derived class destructor";
}
};
int main()
{
B obj1;
return 0;
}
---------------------------------------
#include<iostream>
using namespace std;
class A
{
public:
A()
{
cout<<"\nCalling default base class constructor";
}
~A()
{
cout<<"\nCalling base class destructor";
}
};
```

```
class B:public A
{
public:
/*B()
{
cout<<"\n Calling default derived class constructor";
}*/
~B()
{
cout<<"\nCalling derived class destructor";
}
};
int main()
{
B obj1;
return 0;
}
---------------------------------------
#include<iostream>
using namespace std;
class A
{
public:
/*A()
{
cout<<"\nCalling default base class constructor";
}*/
~A()
{
cout<<"\nCalling base class destructor";
}
};
class B:public A
{
public:
/*B()
{
cout<<"\n Calling default derived class constructor";
}*/
~B()
{
cout<<"\nCalling derived class destructor";
}
};
int main()
{
B obj1;
return 0;
}
```

```
---------------------------------------
#include<iostream>
using namespace std;
class A
{
int x;
public:
/*A()
{
cout<<"\nCalling base class default";
}*/
A(int a)
{
   x=a;
   cout<<"\nCalling base class parameterized "<<x;
}
~A()
{
cout<<"\nCalling base class destructor";
}
};
class B:public A
{
int l;
public:
/*B()
{
   cout<<"\nCalling derived class default";
}*/
B(int p):A(p)
{
l=p;
cout<<"\nCalling derived class parameterized:"<<l;
}
~B()
{
cout<<"\nCalling derived class destructor";
}
};
int main()
{
B obj1(12);
//B obj2;
return 0;
}
---------------------------------------
#include<iostream>
using namespace std;
class A
```

```
{
int x;
public:
A()
{
cout<<"\nCalling base class default";
}
A(int a)
{
   x=a;
   cout<<"\nCalling base class parameterized "<<x;
}
~A()
{
cout<<"\nCalling base class destructor";
}
};
class B:public A
{
int l;
public:
B()
{
   cout<<"\nCalling derived class default";
}
B(int p):A(p)
{
l=p;
cout<<"\nCalling derived class parameterized:"<<l;
}
~B()
{
cout<<"\nCalling derived class destructor";
}
};
int main()
{
B obj1(12);
B obj2;
return 0;
}
```