

**Solution 1**

```

class Figure
{
protected:
    int l,b;
public:
    void get()
    {
        cout<<"Enter l,b:";
        cin>>l>>b;
    }
    void show()
    {
        cout<<l<<" "<<b<<endl;
    }
};

class Printdata
{
public:
    void print(int a)
    {
        cout<<"Area is "<<a;
    }
};

class Rectangle:public Figure,public Printdata
{
    int area;
public:
    void cal_area()
    {
        area=l*b;
        print(area);
    }
};

int main()
{
    Rectangle R;
    R.get();
    R.show();
    R.cal_area();
    return 0;
}

```

Handwritten annotations for Solution 1:

- Arrows from `l` and `b` in `Figure::get()` point to the input values `10` and `20` in the `main()` function.
- An arrow from `print(area)` in `Rectangle::cal_area()` points to the `print` method in the `Printdata` class.
- A diagram of a rectangle is shown with dimensions `l=10` and `b=20`, and area `area=200`.
- Checkmarks are placed next to the `main()` function calls: `R.get();`, `R.show();`, and `R.cal_area();`.

**Solution 2**

```

class Figure
{
protected:
    int l,b;
public:
    void get()
    {
        cout<<"Enter l,b:";
        cin>>l>>b;
    }
    void show()
    {
        cout<<l<<" "<<b<<endl;
    }
};

class Printdata
{
public:
    void print(int a)
    {
        cout<<"Area is "<<a;
    }
};

class Rectangle:public Figure,public Printdata
{
    int area;
public:
    int cal_area()
    {
        area=l*b;
        return area;
    }
};

int main()
{
    Rectangle R;
    R.get();
    R.show();
    int x=R.cal_area();
    R.print(x);
    return 0;
}

```

Handwritten annotations for Solution 2:

- An arrow from `int x=R.cal_area();` in `main()` points to the `cal_area()` method in the `Rectangle` class.
- An arrow from `R.print(x);` in `main()` points to the `print` method in the `Printdata` class.
- The `cal_area()` method in the `Rectangle` class is enclosed in a dashed box, and the `return area;` statement is highlighted.
- Checkmarks are placed next to the `main()` function calls: `R.get();`, `R.show();`, and `int x=R.cal_area();`.

```
class Figure
```

```
{
protected:
```

```
int l,b;
```

```
public:
```

```
void get()
```

```
{
cout<<"Enter l,b:";
cin>>l>>b;
```

① Fig dj

② Fig x

③ Fig &

④ Dn dj

```
}
void show()
```

```
{
cout<<l<<" "<<b<<endl;
```

```
};
class Printdata
```

```
{
public:
```

```
void show(int a)
```

```
{
cout<<"Area is "<<a;
```

① Pnd -

② P - -

③ P

④ Dn dj

```
};
```

```
class Rectangle:public Figure,public Printdata
```

```
{
```

```
int area;
```

```
public:
```

```
int cal_area()
```

```
{
area=l*b;
return area;
```

```
}
```

```
};
int main()
```

```
{
Rectangle R;
```

```
R.get();
```

```
R.show();
```

```
int x=R.cal_area();
```

```
R.show(x);
```

```
return 0;
```

```
}
```

?

Error of Ambiguity

```
class Figure
```

```
{
protected:
```

```
int l,b;
```

```
public:
```

```
void get()
```

```
{
cout<<"Enter l,b:";
cin>>l>>b;
```

```
}
void show()
```

```
{
cout<<l<<" "<<b<<endl;
```

```
};
```

```
class Printdata
```

```
{
```

```
public:
```

```
void show(int a)
```

```
{
cout<<"Area is "<<a;
```

```
};
```

```
class Rectangle:public Figure,public Printdata
```

```
{
```

```
int area;
```

```
public:
```

```
int cal_area()
```

```
{
area=l*b;
return area;
```

```
}
```

```
};
int main()
```

```
{
Rectangle R;
```

```
R.get();
```

```
R.Figure::show();
```

```
int x=R.cal_area();
```

```
R.Printdata::show(x);
```

```
return 0;
```

```
}
```

Solving Ambiguity problem in multiple inh

## Behaviour Of Constructor And Destructor In Inheritance

When we inherit a class and suppose the base and derived classes , both , have a constructor and destructor, then when we will create an object of derived class , the compiler will call BOTH THE CONSTRUCTORS but the order will be , from base to derived .

Similarly when the object of the derived class will be destroyed , then compiler will call the DESTRUCTOR of both the classes but the order will be from derived to base

```
#include <iostream>
#include <stdlib.h>
using namespace std;
class A
{
public:
    A()
    {
        cout<<"In constructor of base class A"<<endl;
    }
    ~A()
    {
        cout<<"In destructor of base class A"<<endl;
    }
};
class B:public A
{
public:
    B()
    {
        cout<<"In constructor of derived class B"<<endl;
    }
    ~B()
    {
        cout<<"In destructor of derived class B"<<endl;
    }
};
```

```
int main()
{
    B obj;
    return 0;
}
```

OUTPUT  
=====

In const of base class A  
In const of derived class B  
In dest of derived class B  
In dest of base class A