Inheritance in C++

Inheritance is the capability of one class to acquire properties and characteristics from another class. The class whose properties are inherited by other class is called the **Parent** or **Base** or **Super** class. And, the class which inherits properties of other class is called **Child** or **Derived** or **Sub** class.

Inheritance makes the code reusable. When we inherit an existing class, all its methods and fields become available in the new class, hence code is reused.

NOTE: All members of a class except Private, are inherited

Purpose of Inheritance in C++

1. Code Reusability
2. Method Overriding (Hence, Runtime Polymorphism.)
3. Use of Virtual Keyword

What is child class?

A class that inherits another class is known as child class, it is also known as derived class or subclass.

What is parent class?

The class that is being inherited by other class is known as parent class, super class or base class.

Types of Inheritance in C++

1) Single inheritance
2) Multilevel inheritance
3) Multiple inheritance
4) Hierarchical inheritance
5) Hybrid inheritance

Basic Syntax of Inheritance

```
class Subclass_name : access_mode Superclass_name
```

While defining a subclass like this, the super class must be already defined or atleast declared before the subclass declaration.

Access Mode is used to specify, the mode in which the properties of superclass will be inherited into subclass**, public, privtate or protected**.

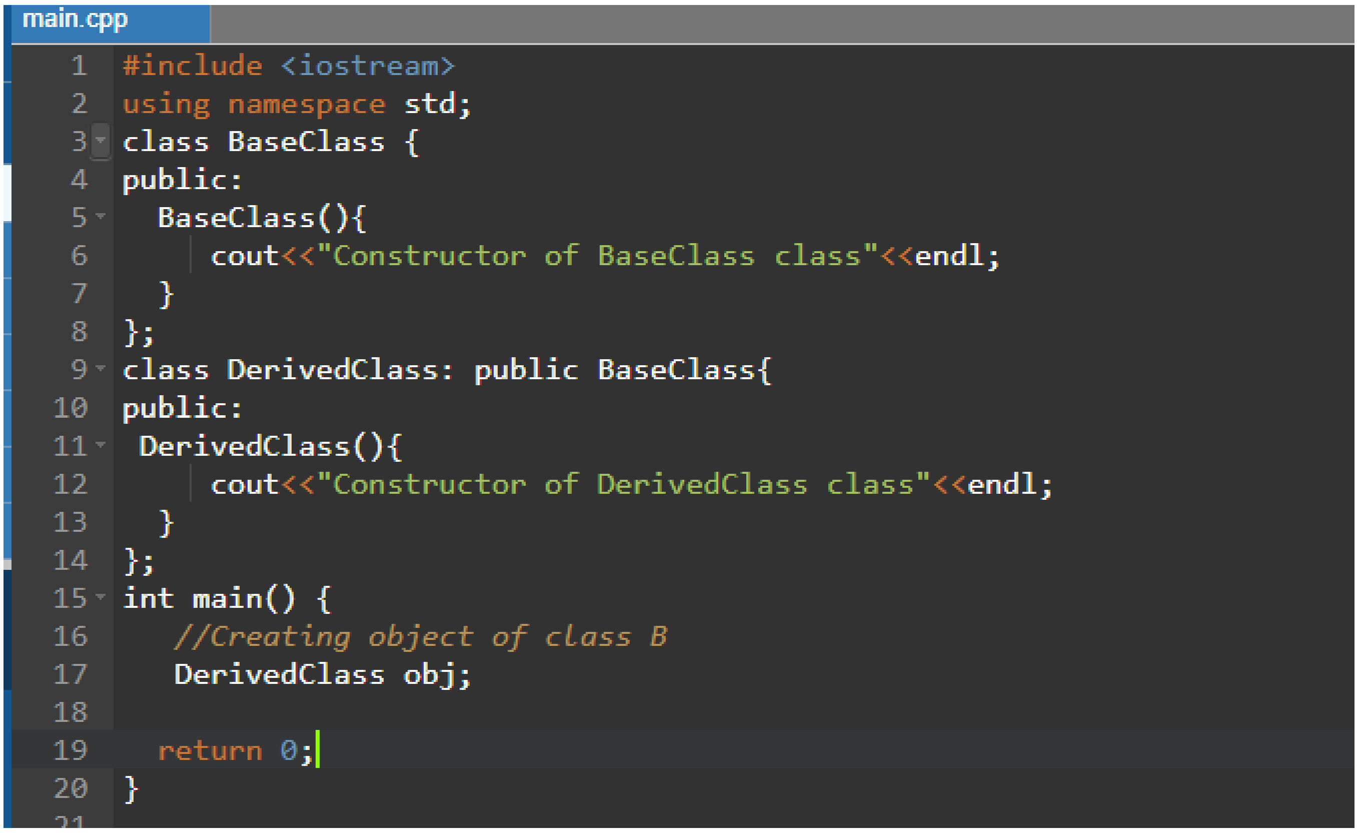
Visibility of Inherited Members

Base class visibility	**Derived class visibility**		
	Public	**Private**	**Protected**
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

- **Public**: When the member is declared as public, it is accessible to all the functions of the program.
- **Private**: When the member is declared as private, it is accessible within the class only.
- **Protected**: When the member is declared as protected, it is accessible within its own class as well as the class immediately derived from it.

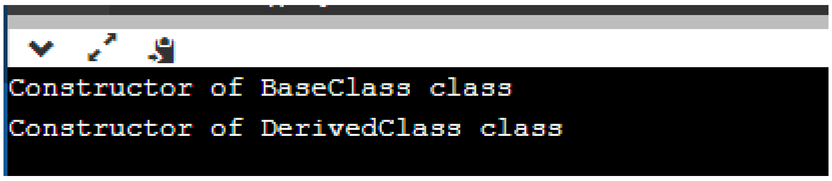
Single inheritance

In Single inheritance one class inherits one class exactly.
For example: Lets say we have class A and B



main.cpp

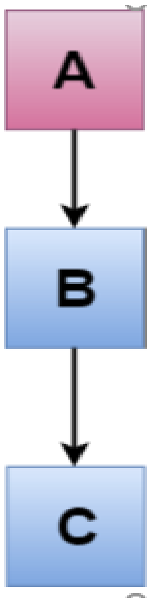
```cpp
#include <iostream>
using namespace std;
class BaseClass {
public:
   BaseClass(){
      cout<<"Constructor of BaseClass class"<<endl;
   }
};
class DerivedClass: public BaseClass{
public:
 DerivedClass(){
      cout<<"Constructor of DerivedClass class"<<endl;
   }
};
int main() {
    //Creating object of class B
    DerivedClass obj;

   return 0;
}
```



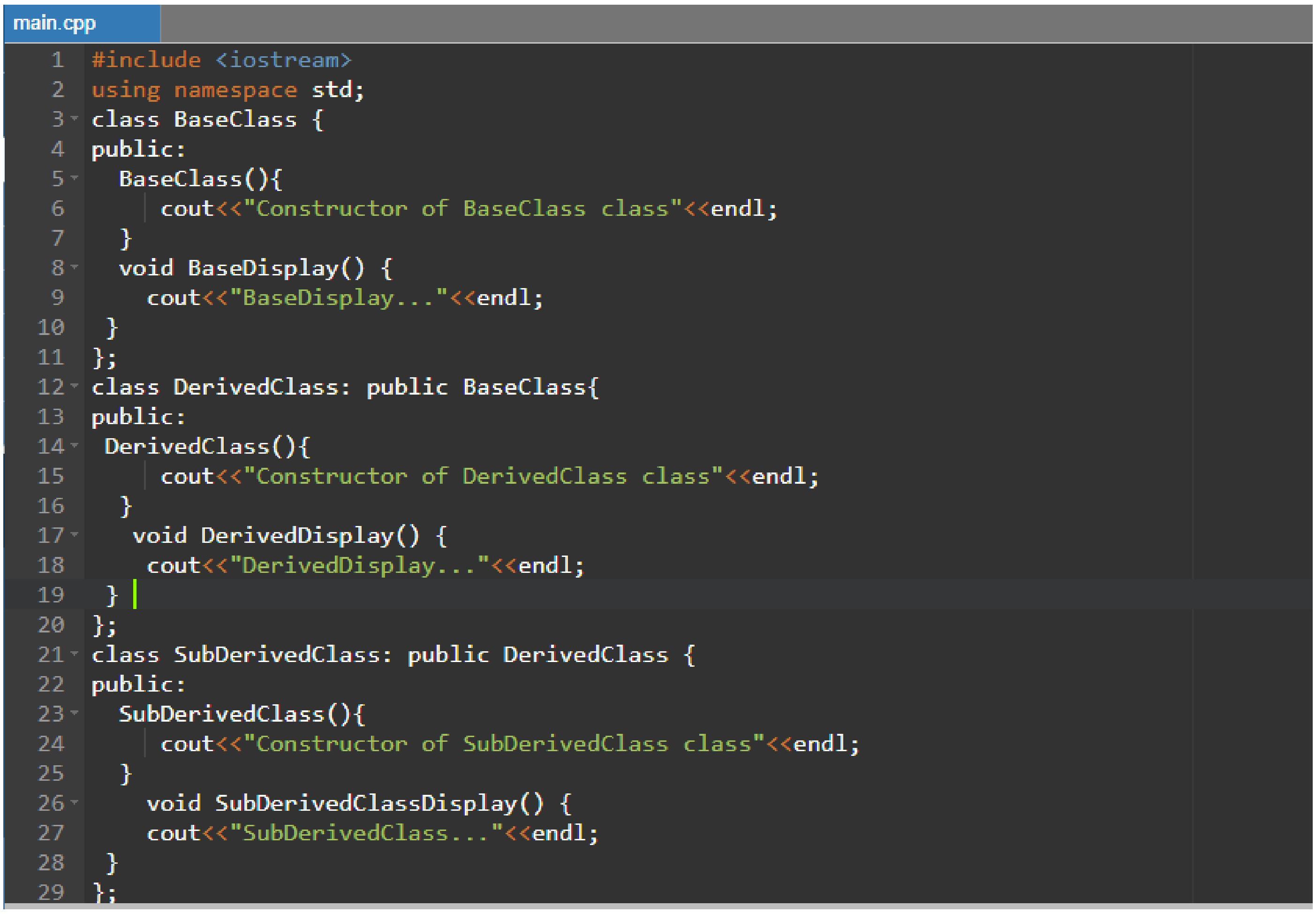
```
Constructor of BaseClass class
Constructor of DerivedClass class
```

C++ Multilevel Inheritance

When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

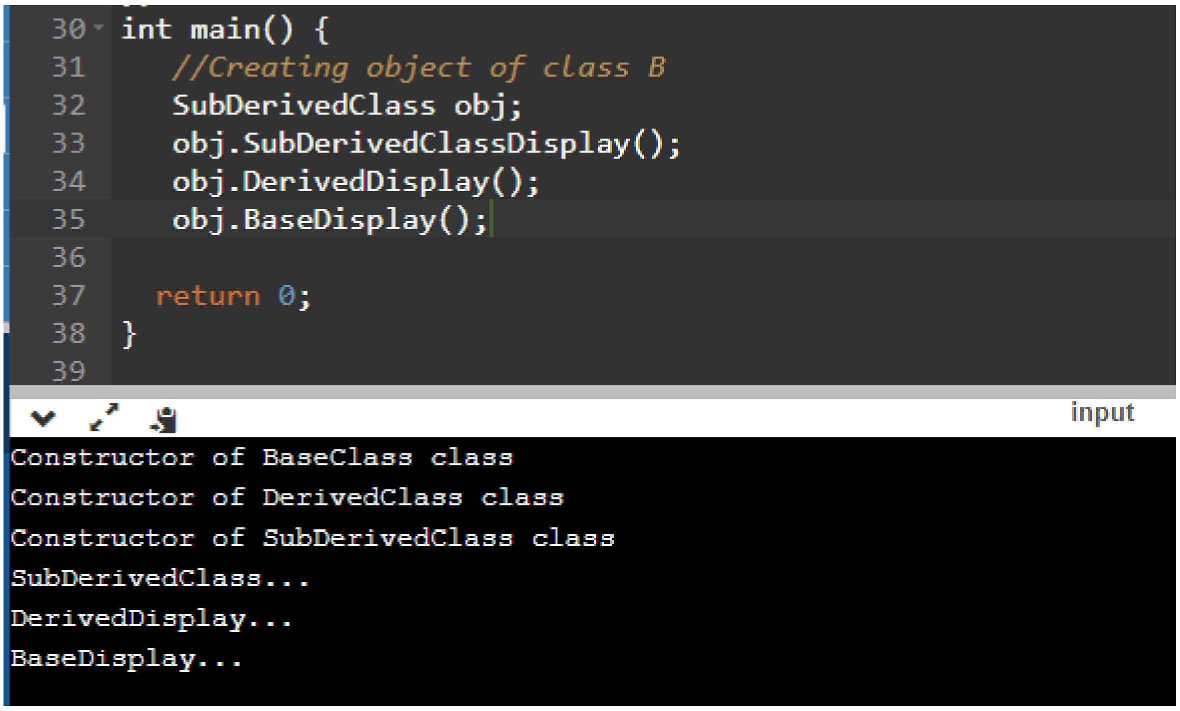


Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz: https://t.me/ccatpreparations **Visit:** https://ccatpreparation.com



main.cpp

```cpp
#include <iostream>
using namespace std;
class BaseClass {
public:
  BaseClass(){
     cout<<"Constructor of BaseClass class"<<endl;
  }
  void BaseDisplay() {
    cout<<"BaseDisplay..."<<endl;
 }
};
class DerivedClass: public BaseClass{
public:
 DerivedClass(){
     cout<<"Constructor of DerivedClass class"<<endl;
  }
   void DerivedDisplay() {
    cout<<"DerivedDisplay..."<<endl;
 }
};
class SubDerivedClass: public DerivedClass {
public:
  SubDerivedClass(){
     cout<<"Constructor of SubDerivedClass class"<<endl;
  }
    void SubDerivedClassDisplay() {
    cout<<"SubDerivedClass..."<<endl;
 }
};
```



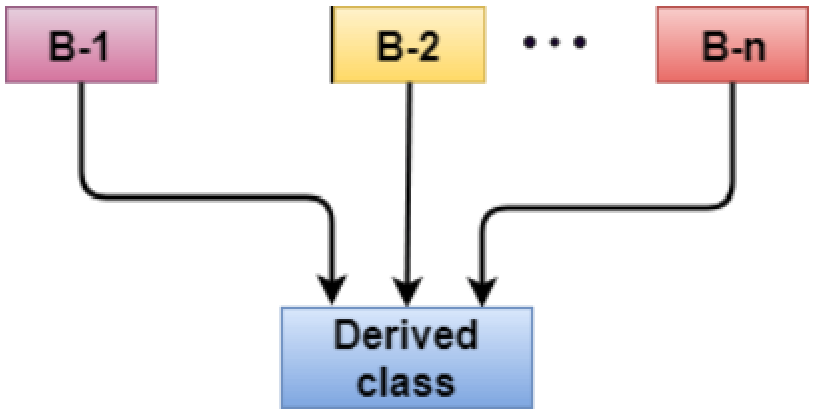
```cpp
int main() {
   //Creating object of class B
   SubDerivedClass obj;
   obj.SubDerivedClassDisplay();
   obj.DerivedDisplay();
   obj.BaseDisplay();

  return 0;
}
```

input

```
Constructor of BaseClass class
Constructor of DerivedClass class
Constructor of SubDerivedClass class
SubDerivedClass...
DerivedDisplay...
BaseDisplay...
```

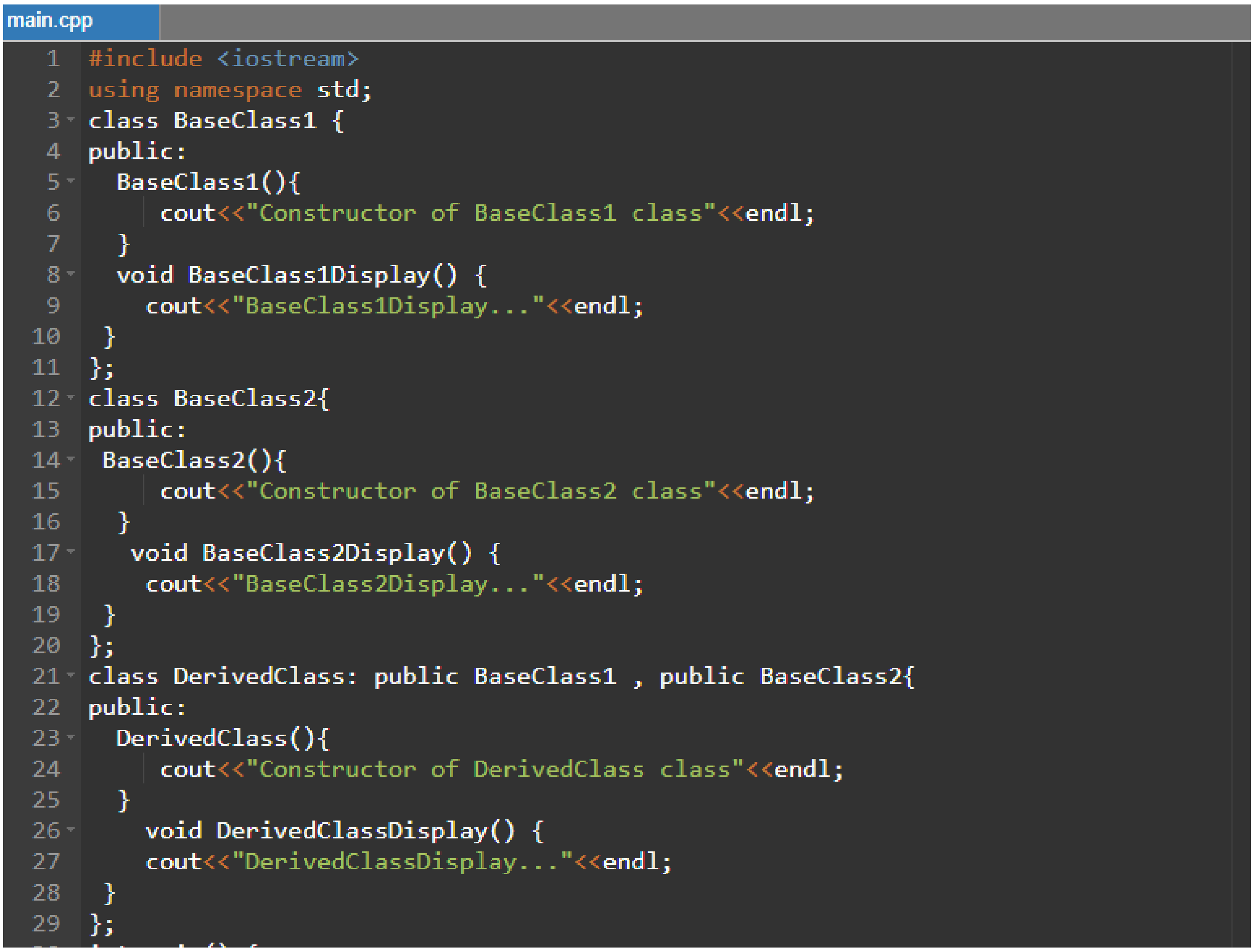
C++ Multiple Inheritance

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



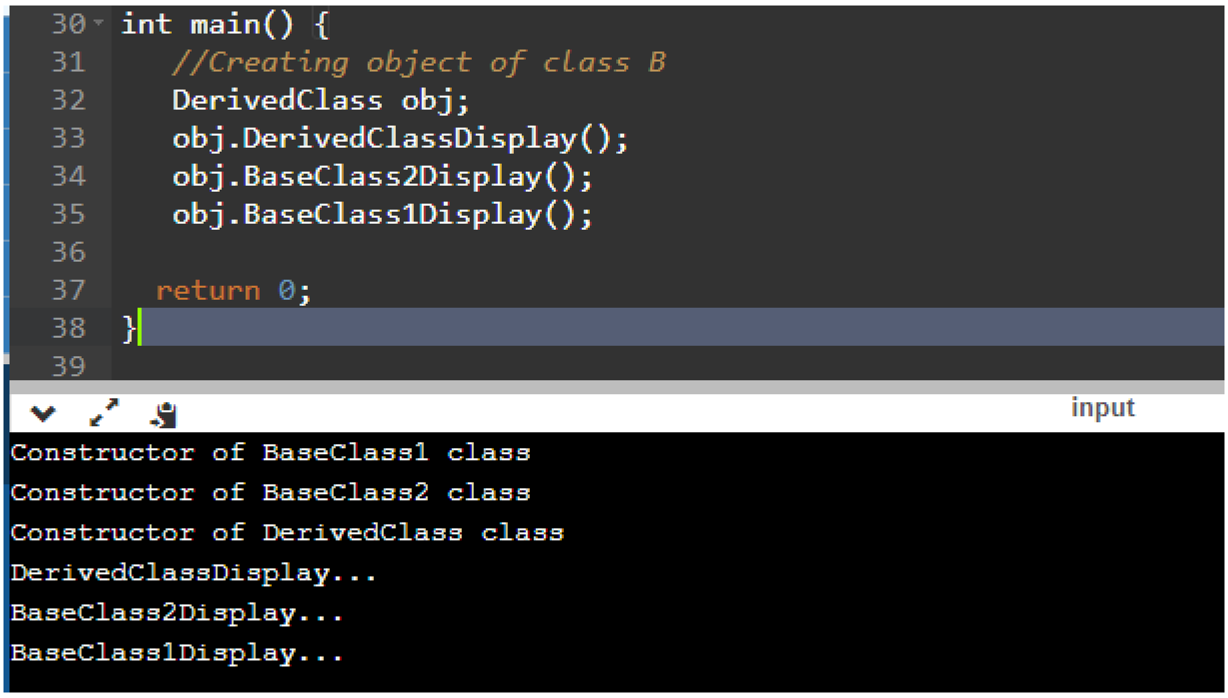
Syntax of the Derived class:

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



main.cpp

```cpp
#include <iostream>
using namespace std;
class BaseClass1 {
public:
  BaseClass1(){
     cout<<"Constructor of BaseClass1 class"<<endl;
  }
  void BaseClass1Display() {
    cout<<"BaseClass1Display..."<<endl;
 }
};
class BaseClass2{
public:
 BaseClass2(){
     cout<<"Constructor of BaseClass2 class"<<endl;
  }
   void BaseClass2Display() {
    cout<<"BaseClass2Display..."<<endl;
 }
};
class DerivedClass: public BaseClass1 , public BaseClass2{
public:
  DerivedClass(){
     cout<<"Constructor of DerivedClass class"<<endl;
  }
    void DerivedClassDisplay() {
    cout<<"DerivedClassDisplay..."<<endl;
 }
};
```



```cpp
int main() {
   //Creating object of class B
   DerivedClass obj;
   obj.DerivedClassDisplay();
   obj.BaseClass2Display();
   obj.BaseClass1Display();

  return 0;
}
```

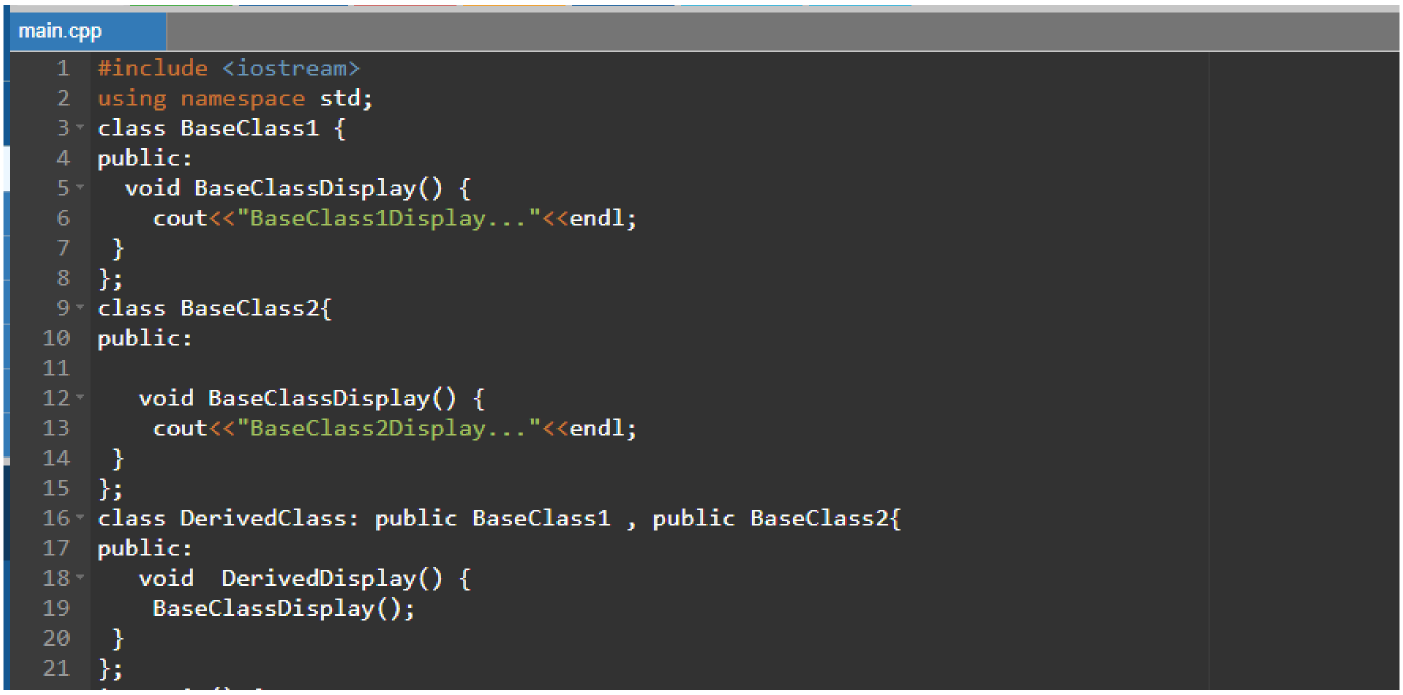
input

```
Constructor of BaseClass1 class
Constructor of BaseClass2 class
Constructor of DerivedClass class
DerivedClassDisplay...
BaseClass2Display...
BaseClass1Display...
```

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz: https://t.me/ccatpreparations **Visit:** https://ccatpreparation.com

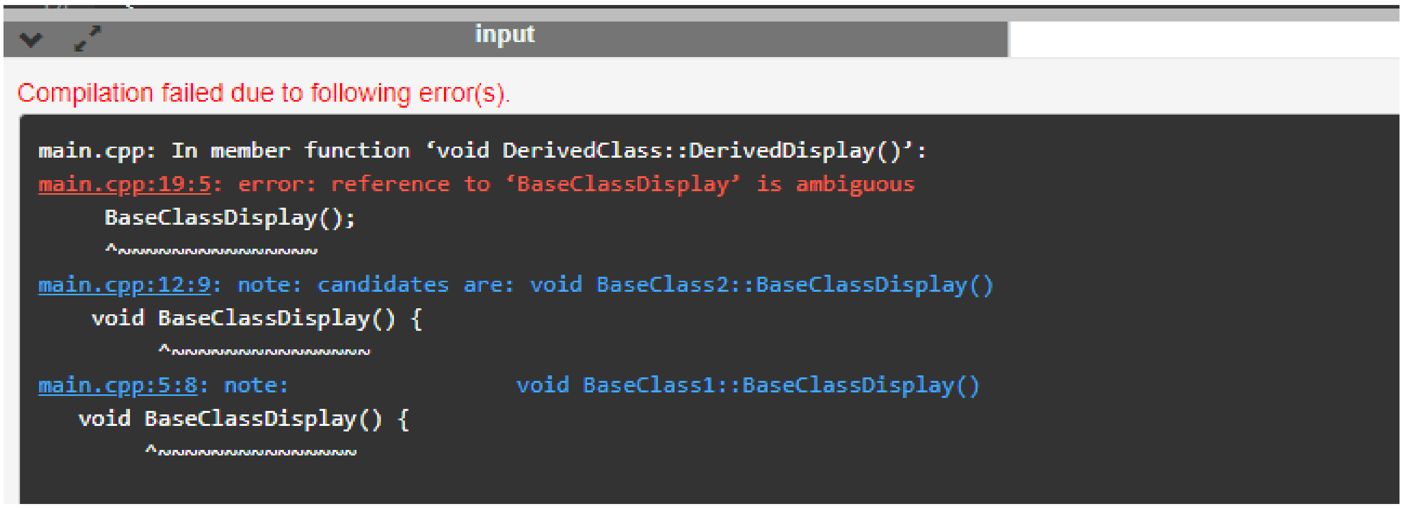
Ambiquity Resolution in Inheritance.

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



main.cpp

```
 1  #include <iostream>
 2  using namespace std;
 3  class BaseClass1 {
 4  public:
 5    void BaseClassDisplay() {
 6      cout<<"BaseClass1Display..."<<endl;
 7   }
 8  };
 9  class BaseClass2{
10  public:
11
12     void BaseClassDisplay() {
13      cout<<"BaseClass2Display..."<<endl;
14   }
15  };
16  class DerivedClass: public BaseClass1 , public BaseClass2{
17  public:
18     void  DerivedDisplay() {
19      BaseClassDisplay();
20   }
21  };
```

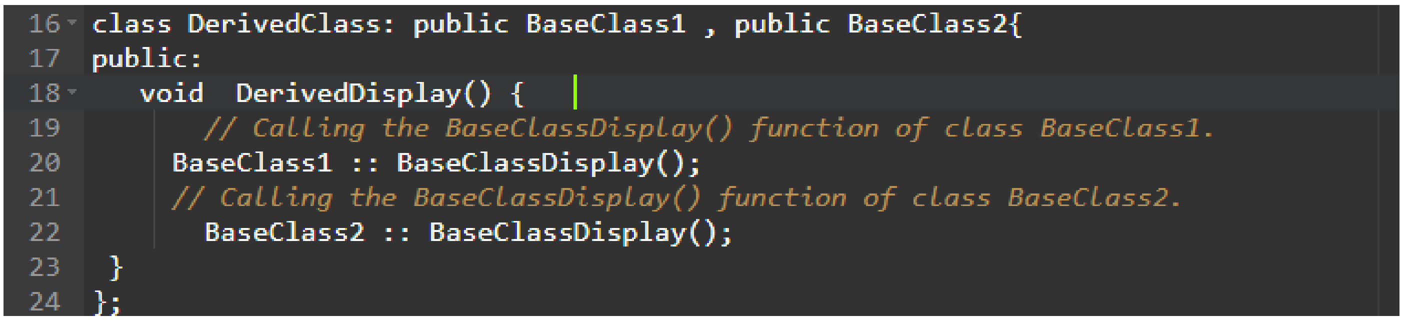


input

Compilation failed due to following error(s).

```
main.cpp: In member function ‘void DerivedClass::DerivedDisplay()’:
main.cpp:19:5: error: reference to ‘BaseClassDisplay’ is ambiguous
     BaseClassDisplay();
     ^~~~~~~~~~~~~~~~
main.cpp:12:9: note: candidates are: void BaseClass2::BaseClassDisplay()
    void BaseClassDisplay() {
         ^~~~~~~~~~~~~~~~
main.cpp:5:8: note:                 void BaseClass1::BaseClassDisplay()
   void BaseClassDisplay() {
        ^~~~~~~~~~~~~~~~
```

The above issue can be resolved by using the class resolution operator with the function. In the above example, the derived class code can be rewritten as:

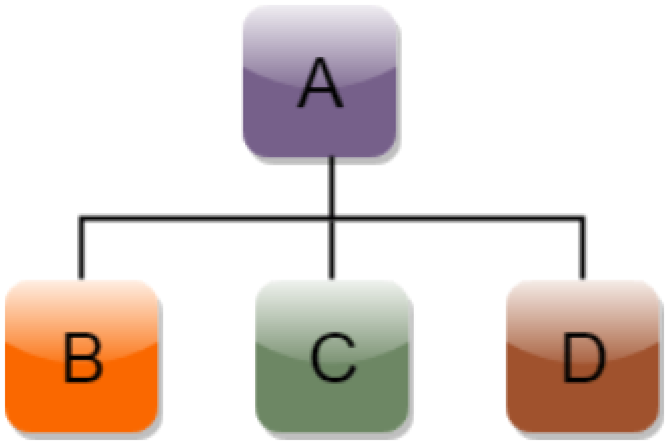


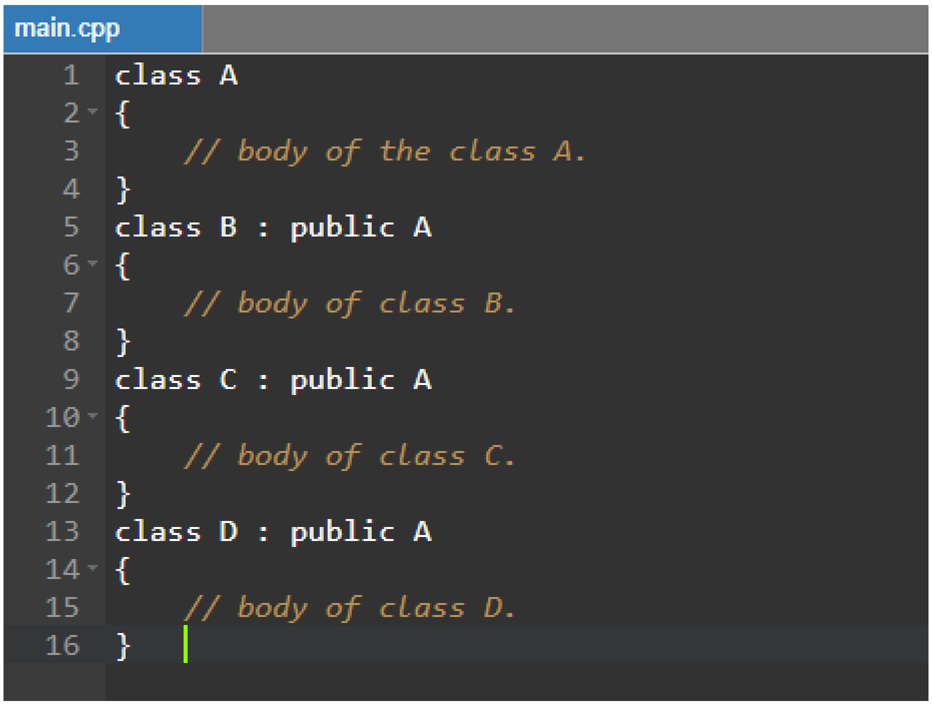
```
16  class DerivedClass: public BaseClass1 , public BaseClass2{
17  public:
18     void  DerivedDisplay() {
19        // Calling the BaseClassDisplay() function of class BaseClass1.
20      BaseClass1 :: BaseClassDisplay();
21      // Calling the BaseClassDisplay() function of class BaseClass2.
22        BaseClass2 :: BaseClassDisplay();
23   }
24  };
```

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:
https://t.me/ccatpreparations Visit: https://ccatpreparation.com

4)Hierarchical Inheritance

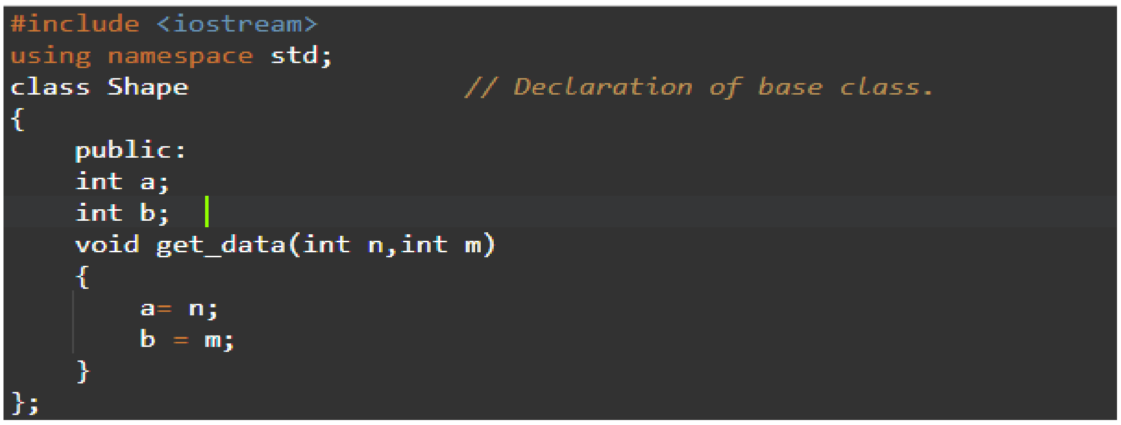
In this type of inheritance, one parent class has more than one child class. For example:




main.cpp

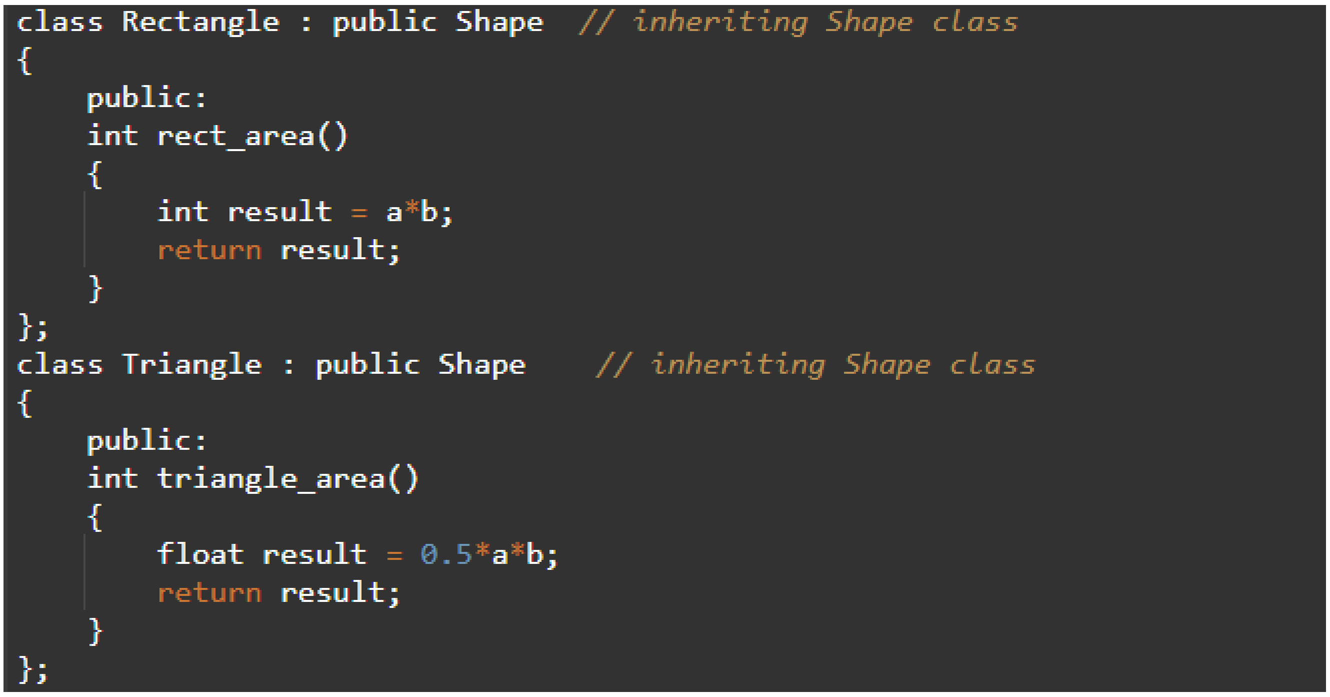
```cpp
class A
{
    // body of the class A.
}
class B : public A
{
    // body of class B.
}
class C : public A
{
    // body of class C.
}
class D : public A
{
    // body of class D.
}
```

Lets Take Shape Example

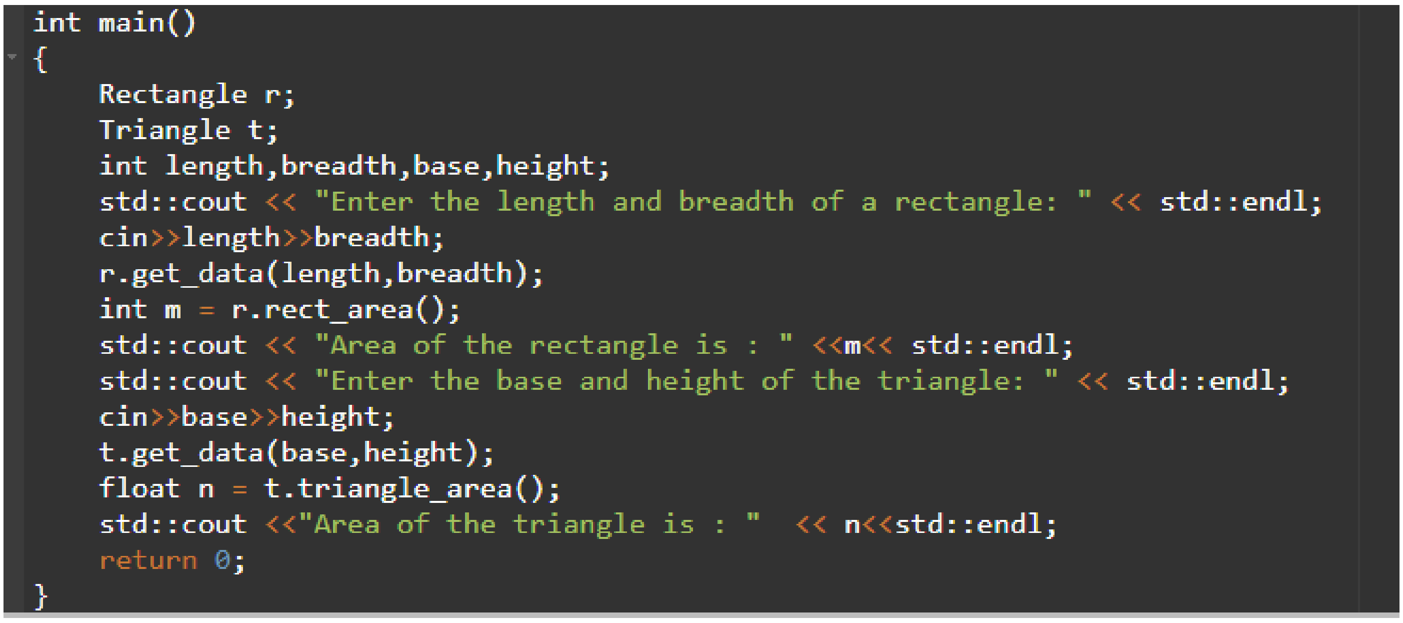


```cpp
#include <iostream>
using namespace std;
class Shape                 // Declaration of base class.
{
    public:
    int a;
    int b;
    void get_data(int n,int m)
    {
        a= n;
        b = m;
    }
};
```

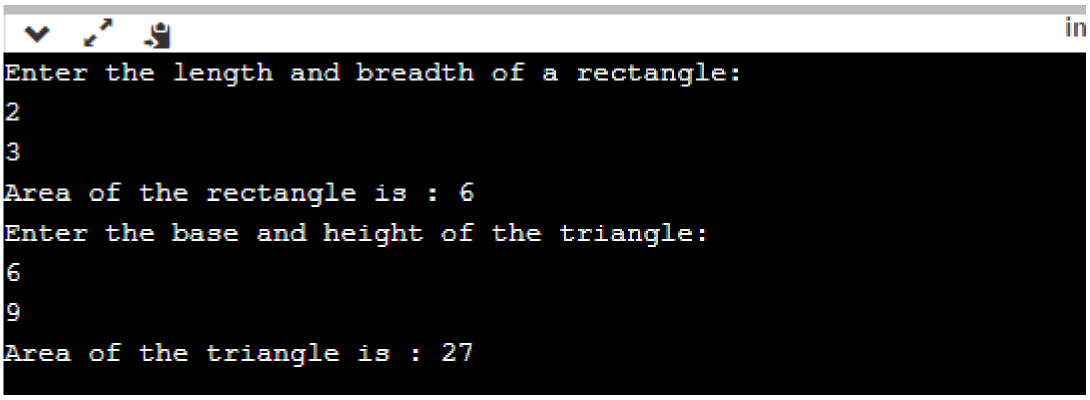
Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:
https://t.me/ccatpreparations **Visit:** https://ccatpreparation.com



```
class Rectangle : public Shape  // inheriting Shape class
{
    public:
    int rect_area()
    {
        int result = a*b;
        return result;
    }
};
class Triangle : public Shape    // inheriting Shape class
{
    public:
    int triangle_area()
    {
        float result = 0.5*a*b;
        return result;
    }
};
```



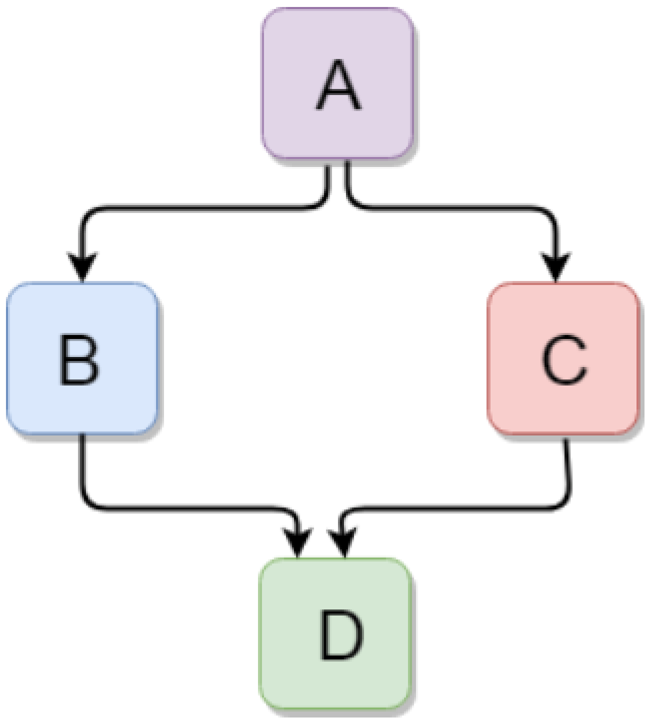
```
int main()
{
    Rectangle r;
    Triangle t;
    int length,breadth,base,height;
    std::cout << "Enter the length and breadth of a rectangle: " << std::endl;
    cin>>length>>breadth;
    r.get_data(length,breadth);
    int m = r.rect_area();
    std::cout << "Area of the rectangle is : " <<m<< std::endl;
    std::cout << "Enter the base and height of the triangle: " << std::endl;
    cin>>base>>height;
    t.get_data(base,height);
    float n = t.triangle_area();
    std::cout <<"Area of the triangle is : "  << n<<std::endl;
    return 0;
}
```



```
Enter the length and breadth of a rectangle:
2
3
Area of the rectangle is : 6
Enter the base and height of the triangle:
6
9
Area of the triangle is : 27
```

5) Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance. For example, A child and parent class relationship that follows multiple and hierarchical inheritance both can be called hybrid inheritance.



Check Source Code in practice Section