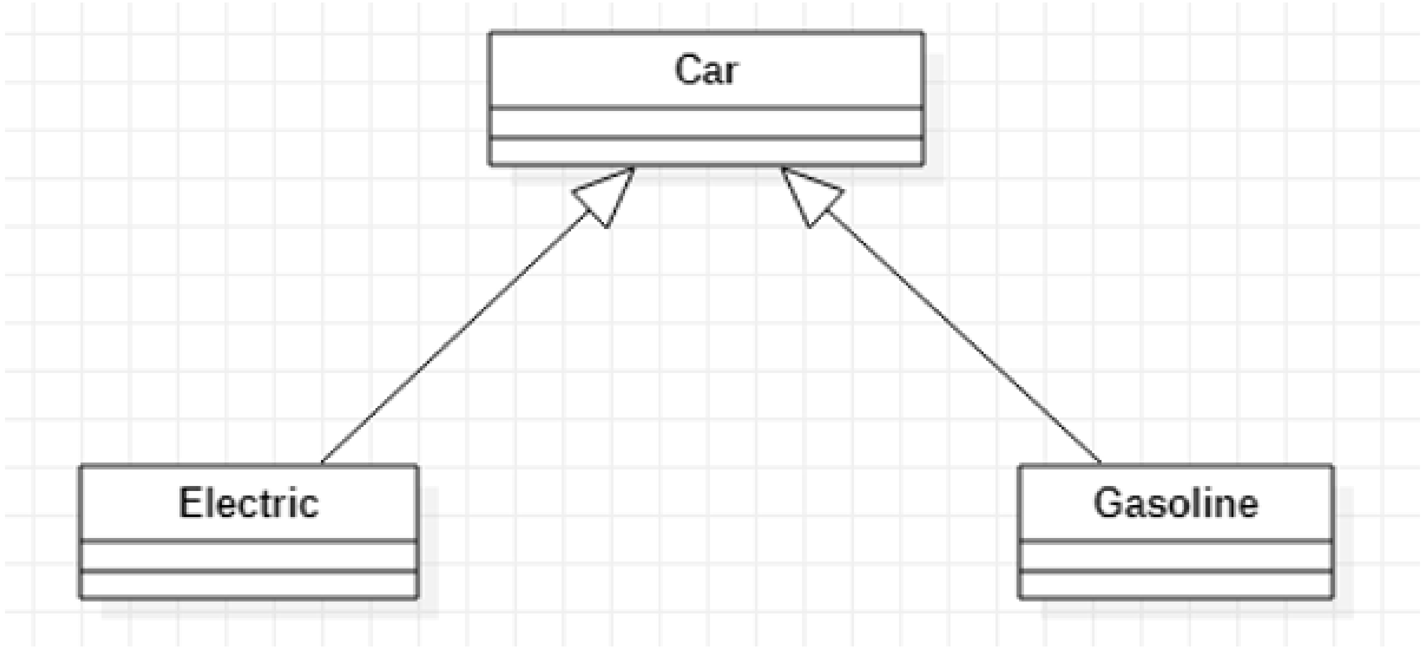
Virtual Inheritance in OOP with C++

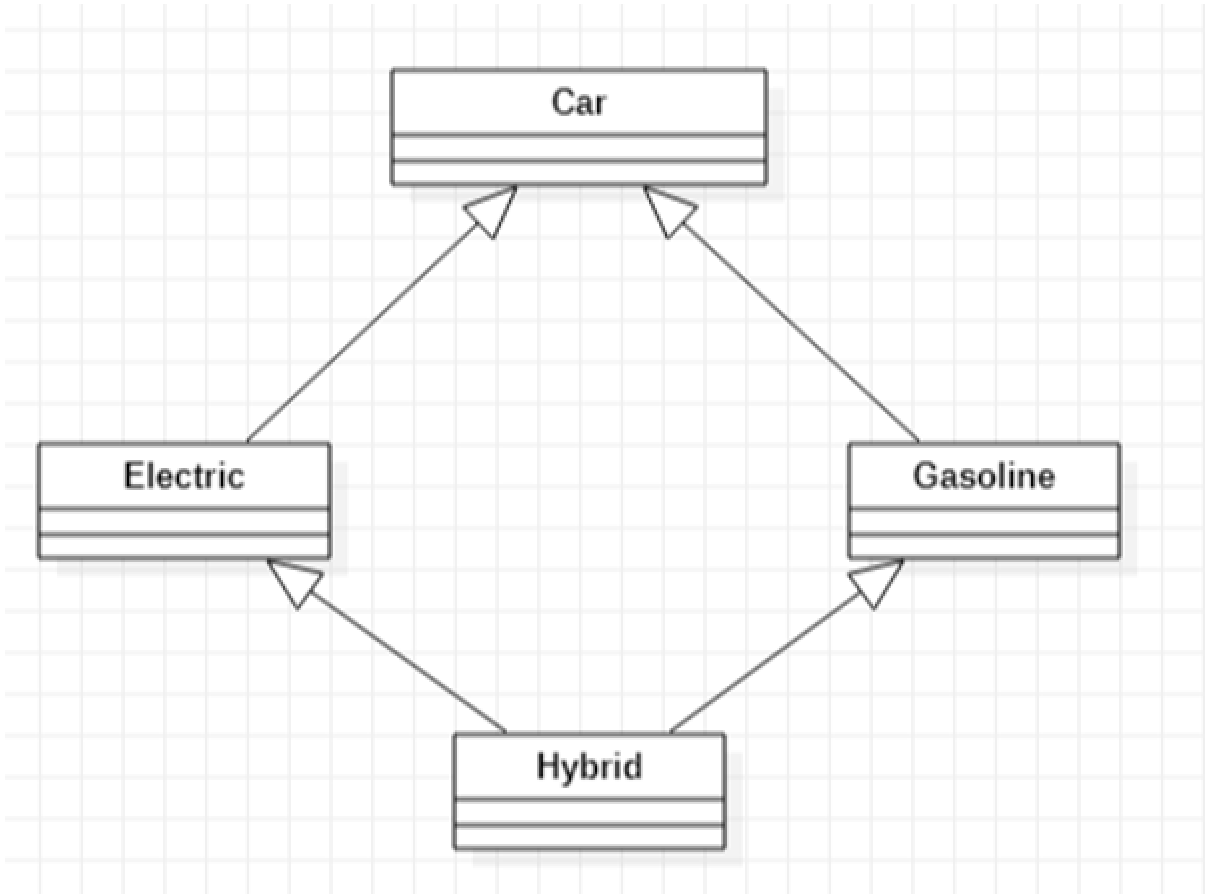
Diamond Problem in Inheritance

- Consider the example of a car.
- Both electric and gasoline cars inherit the properties of a car.



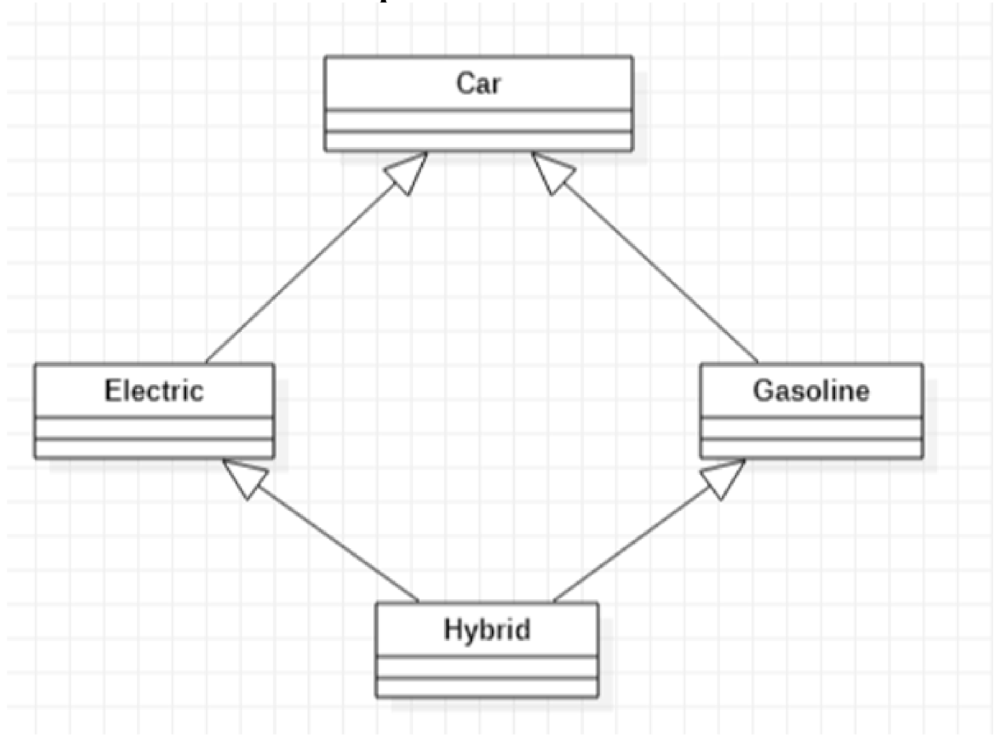
Diamond Problem in Inheritance

- The hybrid car is both an electric car and a gasoline car. These kinds of special cases will result in a diamond problem.

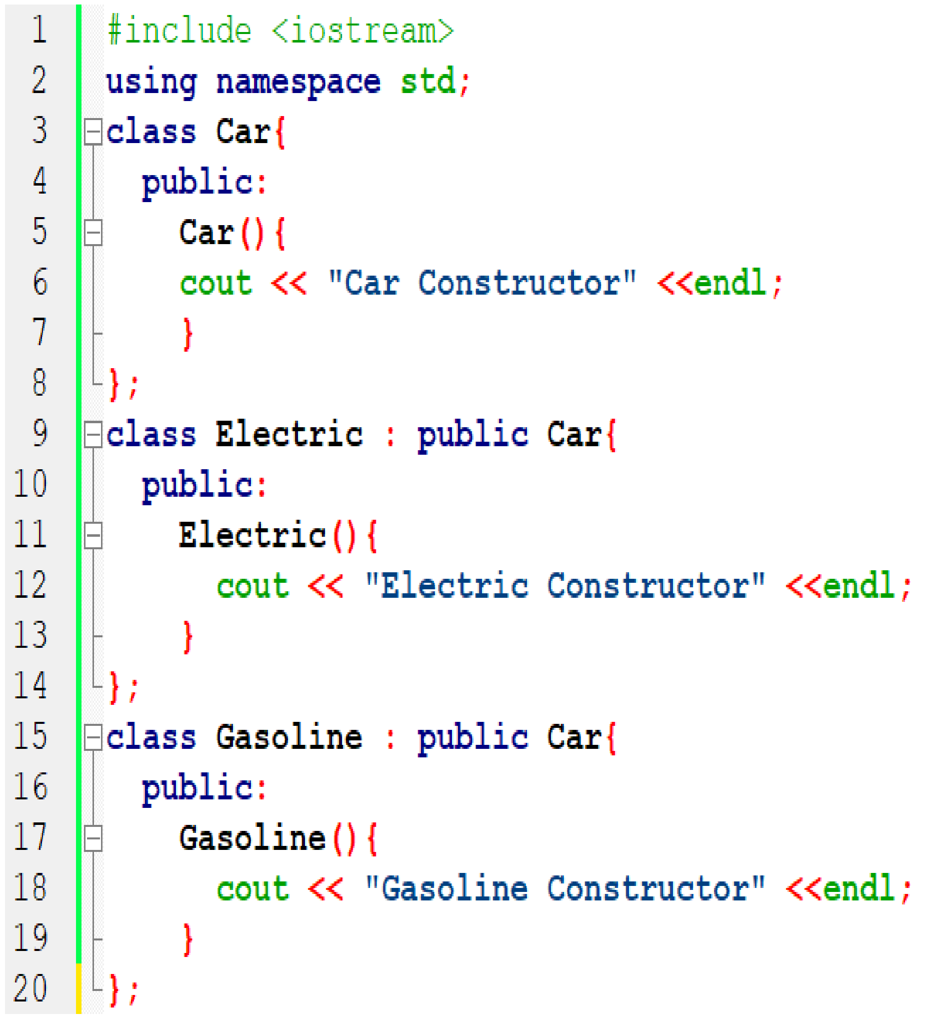


Diamond Problem in Inheritance

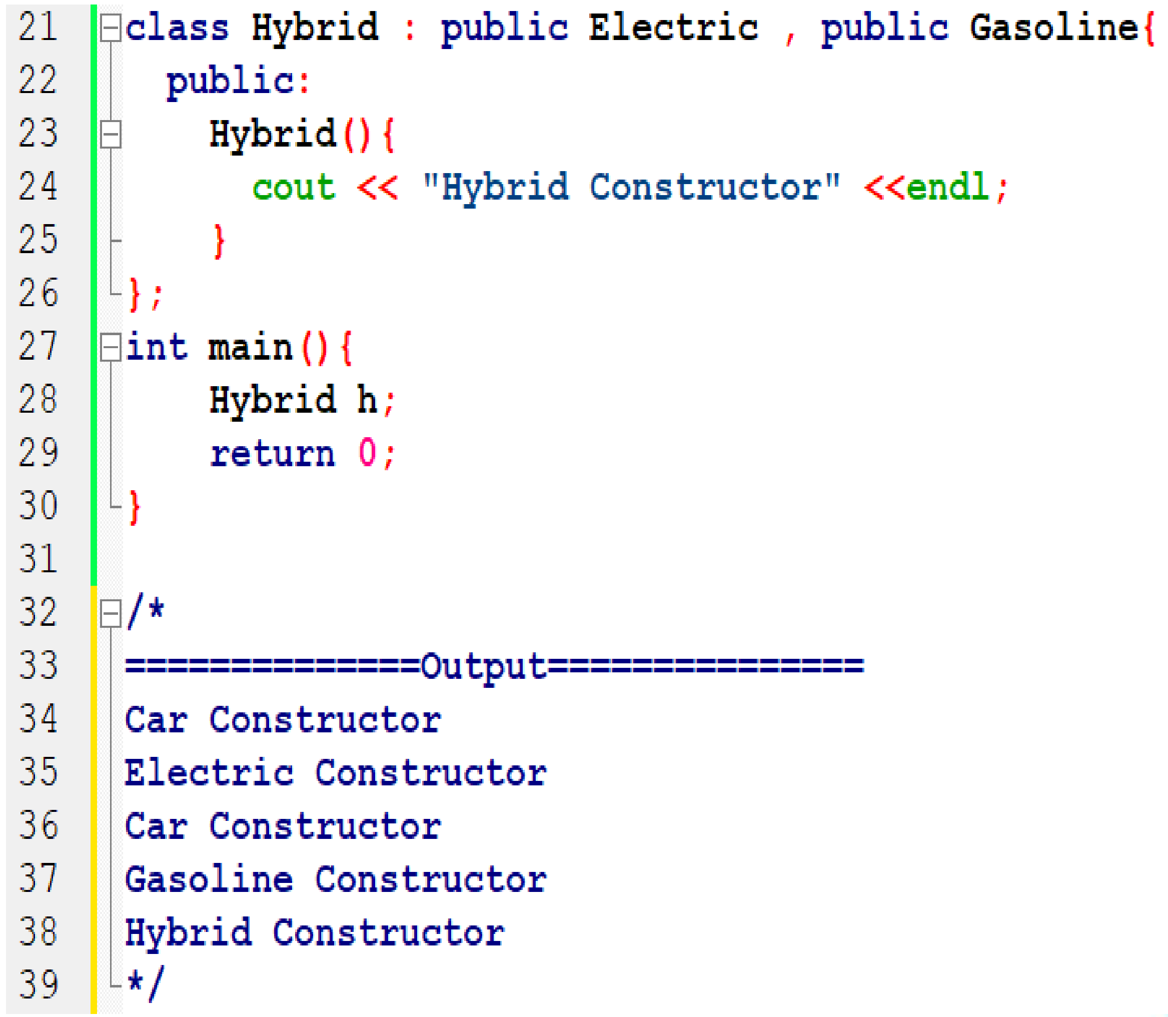
- The hybrid car is both an electric car and a gasoline car. These kinds of special cases will result in a diamond problem.
- This diamond creates a problem, because now the Hybrid class has two copies of the Car class for each path.



Diamond Problem Implementation



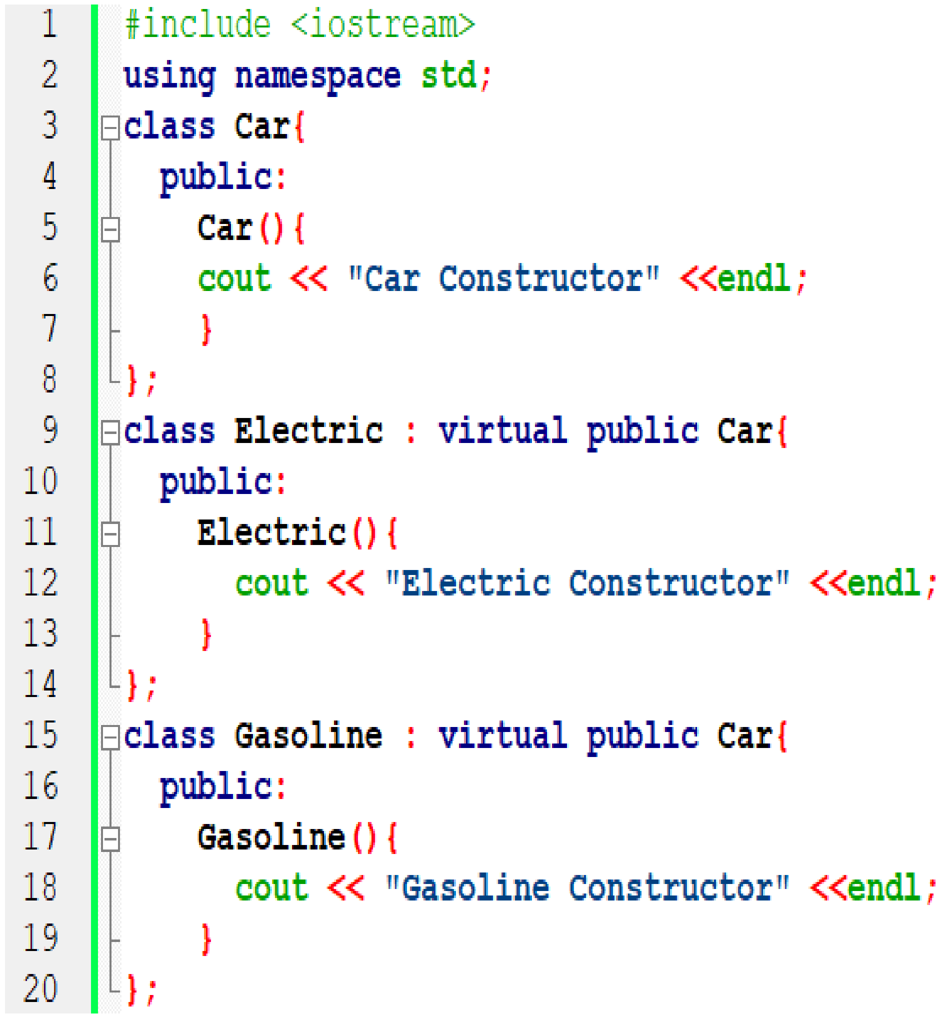
```cpp
#include <iostream>
using namespace std;
class Car{
  public:
    Car(){
    cout << "Car Constructor" <<endl;
    }
};
class Electric : public Car{
  public:
    Electric(){
      cout << "Electric Constructor" <<endl;
    }
};
class Gasoline : public Car{
  public:
    Gasoline(){
      cout << "Gasoline Constructor" <<endl;
    }
};
```



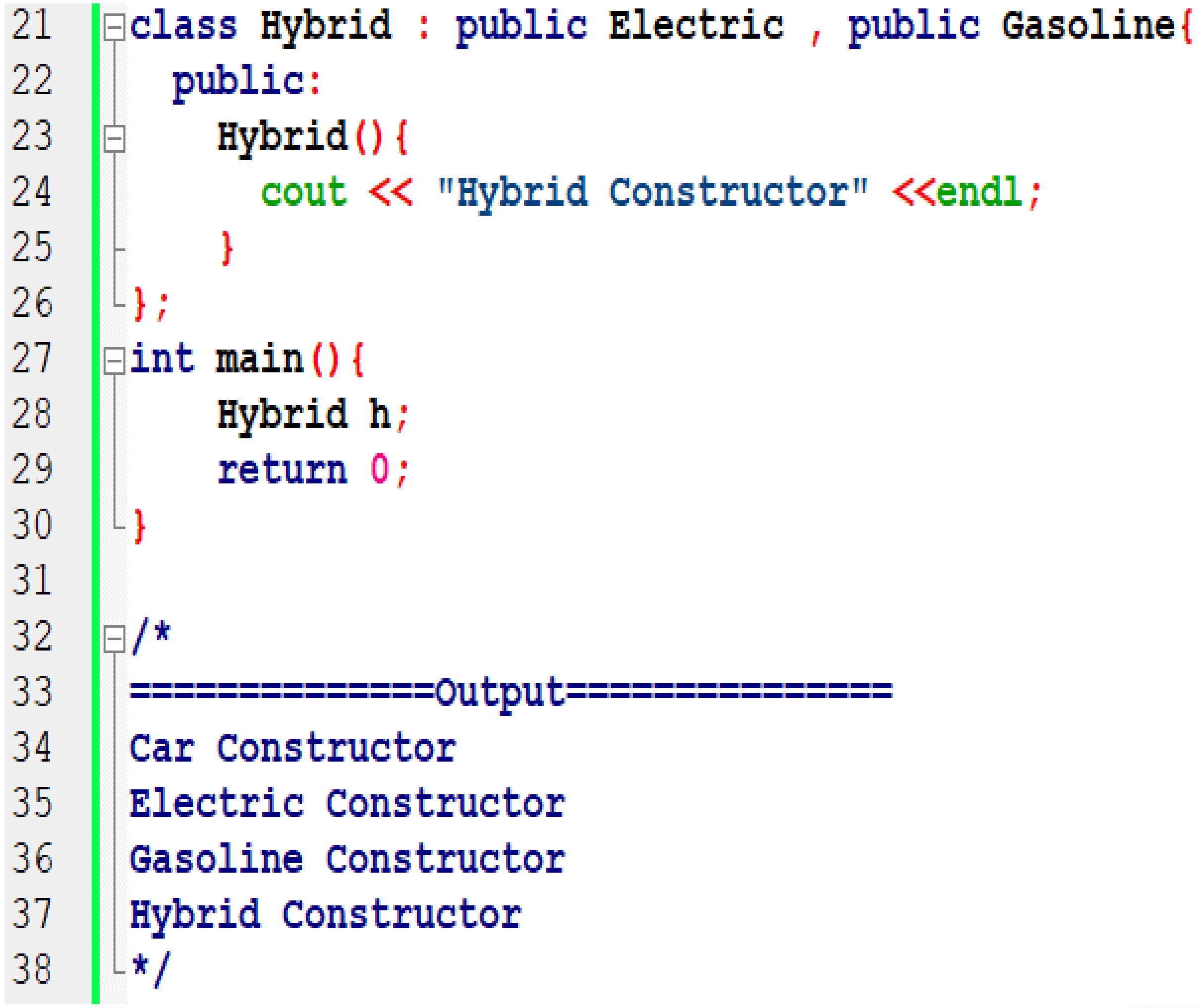
```cpp
class Hybrid : public Electric , public Gasoline{
  public:
    Hybrid(){
      cout << "Hybrid Constructor" <<endl;
    }
};
int main(){
    Hybrid h;
    return 0;
}

/*
==============Output===============
Car Constructor
Electric Constructor
Car Constructor
Gasoline Constructor
Hybrid Constructor
*/
```

Solution of Diamond Problem



```cpp
#include <iostream>
using namespace std;
class Car{
  public:
    Car(){
    cout << "Car Constructor" <<endl;
    }
};
class Electric : virtual public Car{
  public:
    Electric(){
      cout << "Electric Constructor" <<endl;
    }
};
class Gasoline : virtual public Car{
  public:
    Gasoline(){
      cout << "Gasoline Constructor" <<endl;
    }
};
```



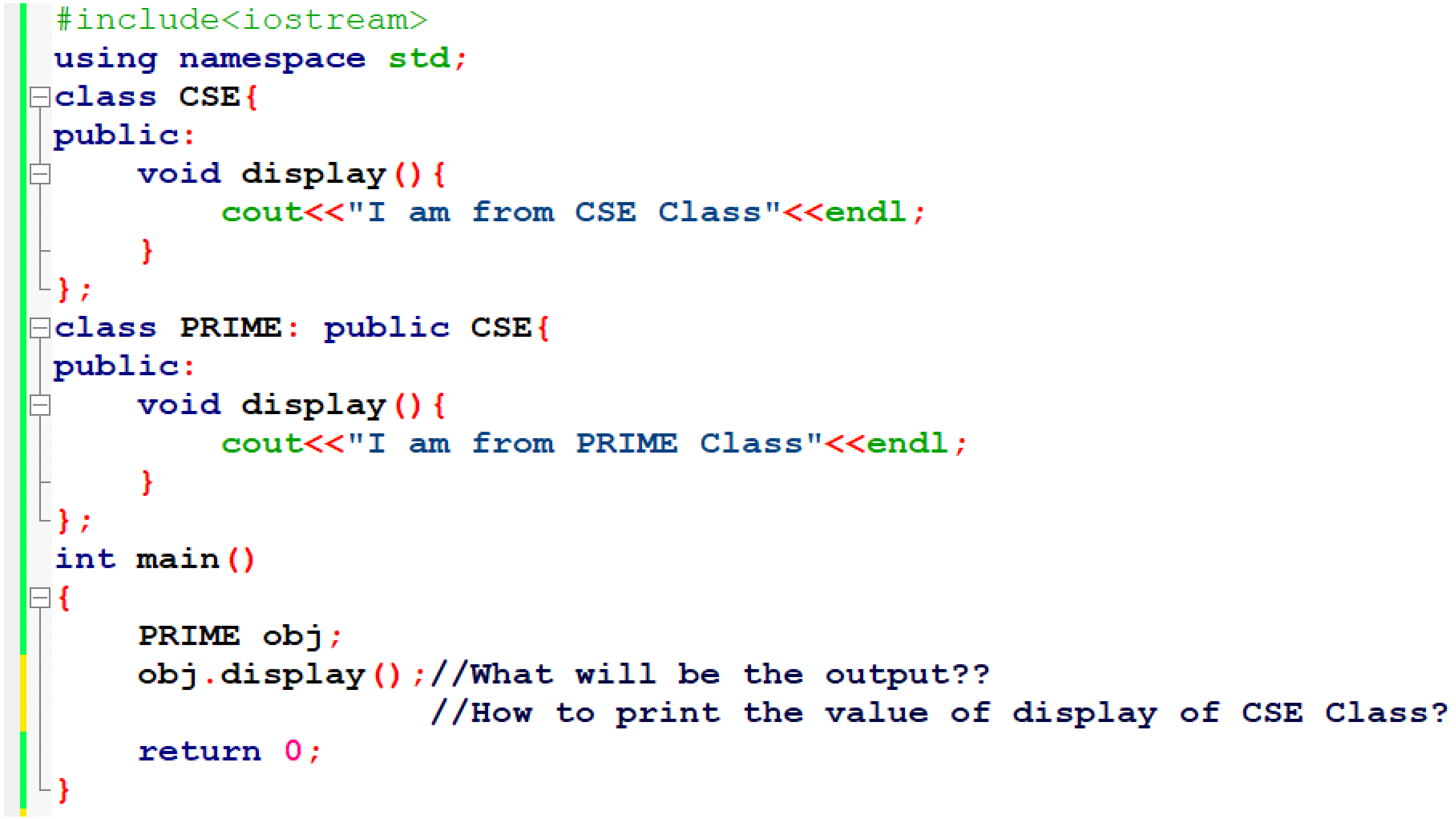
```cpp
class Hybrid : public Electric , public Gasoline{
  public:
    Hybrid(){
      cout << "Hybrid Constructor" <<endl;
    }
};
int main(){
    Hybrid h;
    return 0;
}

/*
==============Output===============
Car Constructor
Electric Constructor
Gasoline Constructor
Hybrid Constructor
*/
```

Solution of Diamond Problem

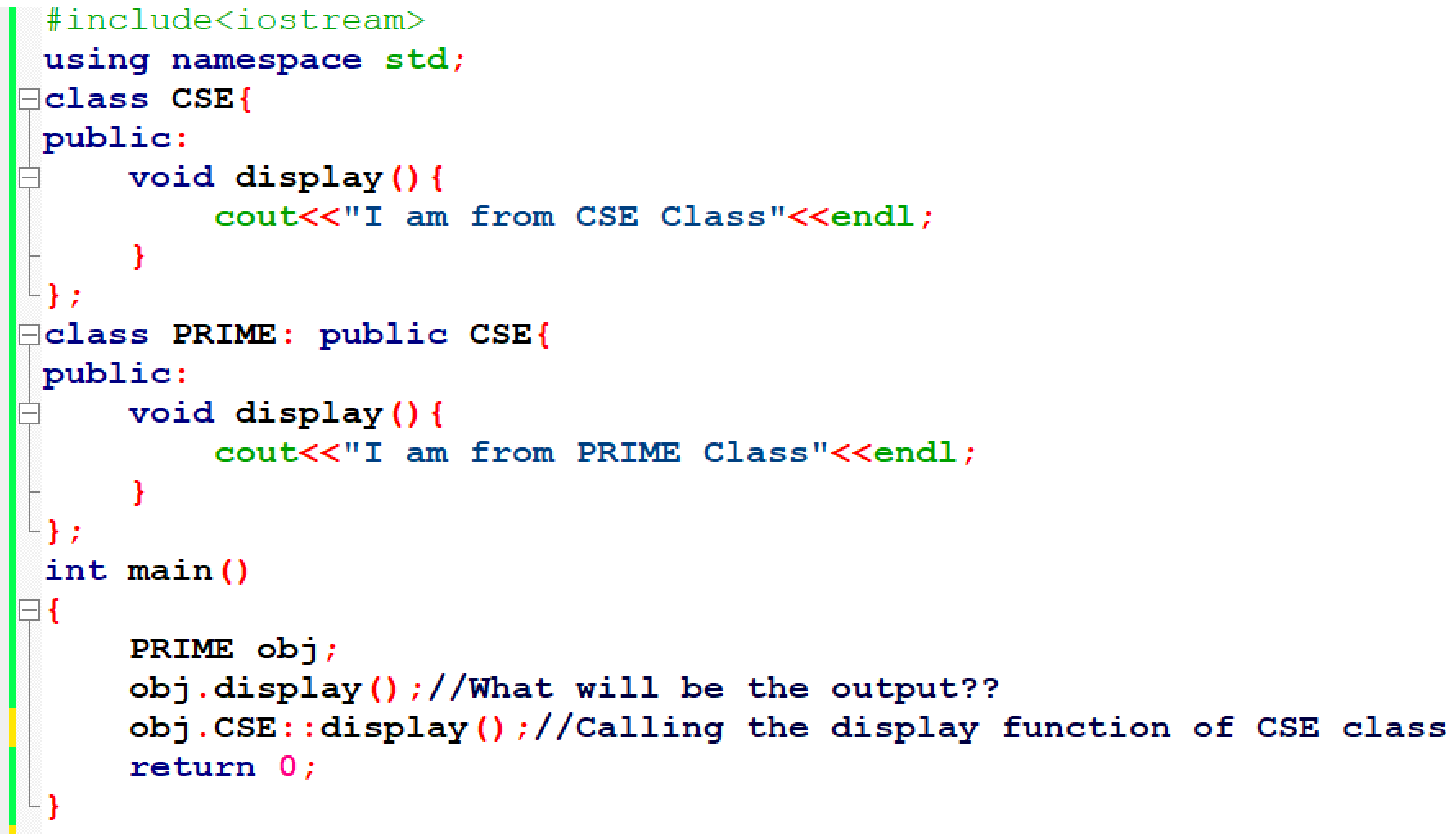
- Virtual can be written before or after the public.
- Now only one copy of data/function member will be copied to class Electric and class Gasoline and class Hybrid becomes the virtual base class.
- Virtual base classes offer a way to save space and avoid ambiguities in class hierarchies that use multiple inheritances. When a base class is specified as a virtual base, it can act as an indirect base more than once without duplication of its data members. A single copy of its data members is shared by all the base classes that use virtual base.

Special Case of Inheritance



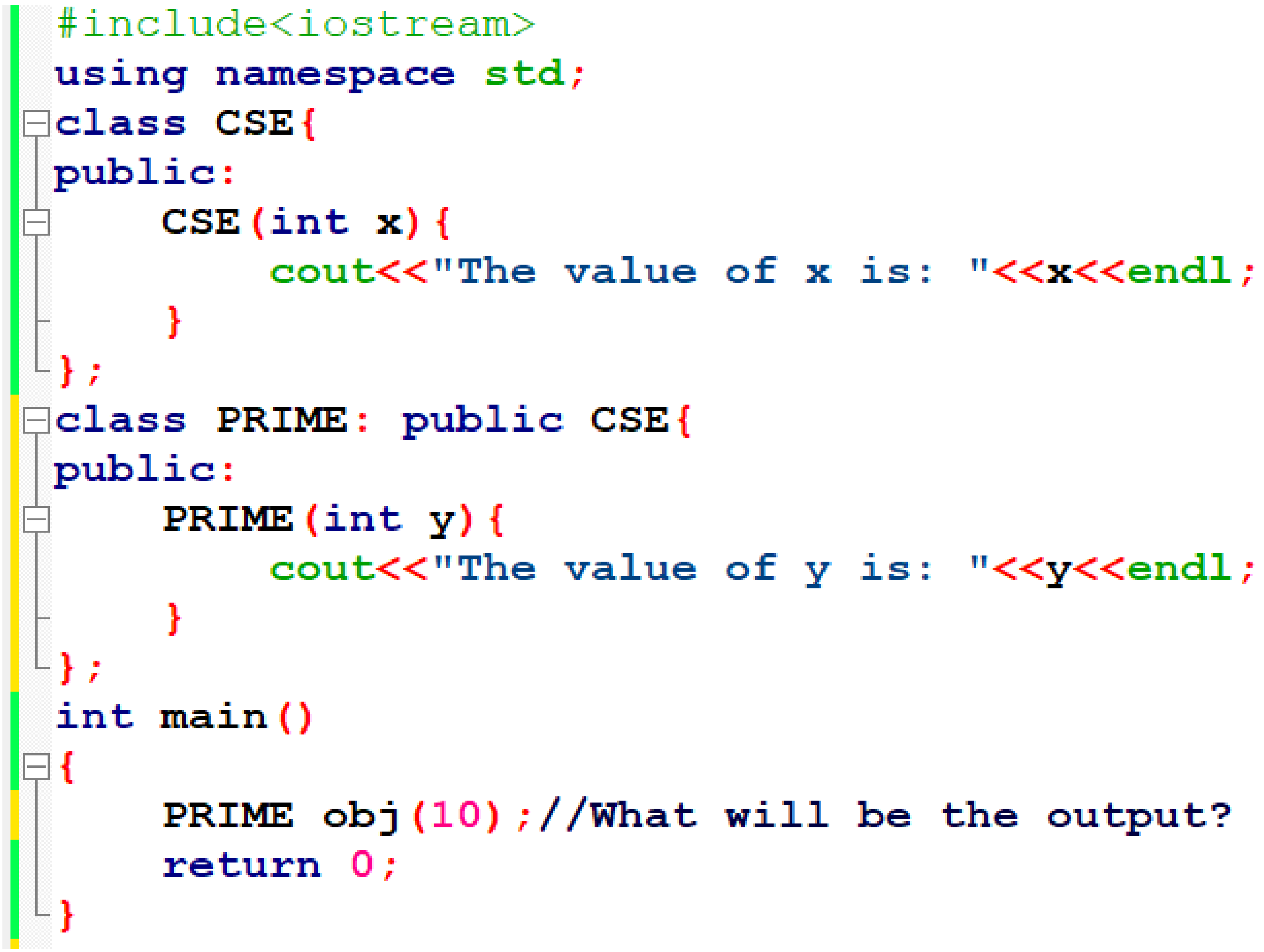
```
#include<iostream>
using namespace std;
class CSE{
public:
    void display(){
        cout<<"I am from CSE Class"<<endl;
    }
};
class PRIME: public CSE{
public:
    void display(){
        cout<<"I am from PRIME Class"<<endl;
    }
};
int main()
{
    PRIME obj;
    obj.display();//What will be the output??
                  //How to print the value of display of CSE Class?
    return 0;
}
```

Special Case of Inheritance



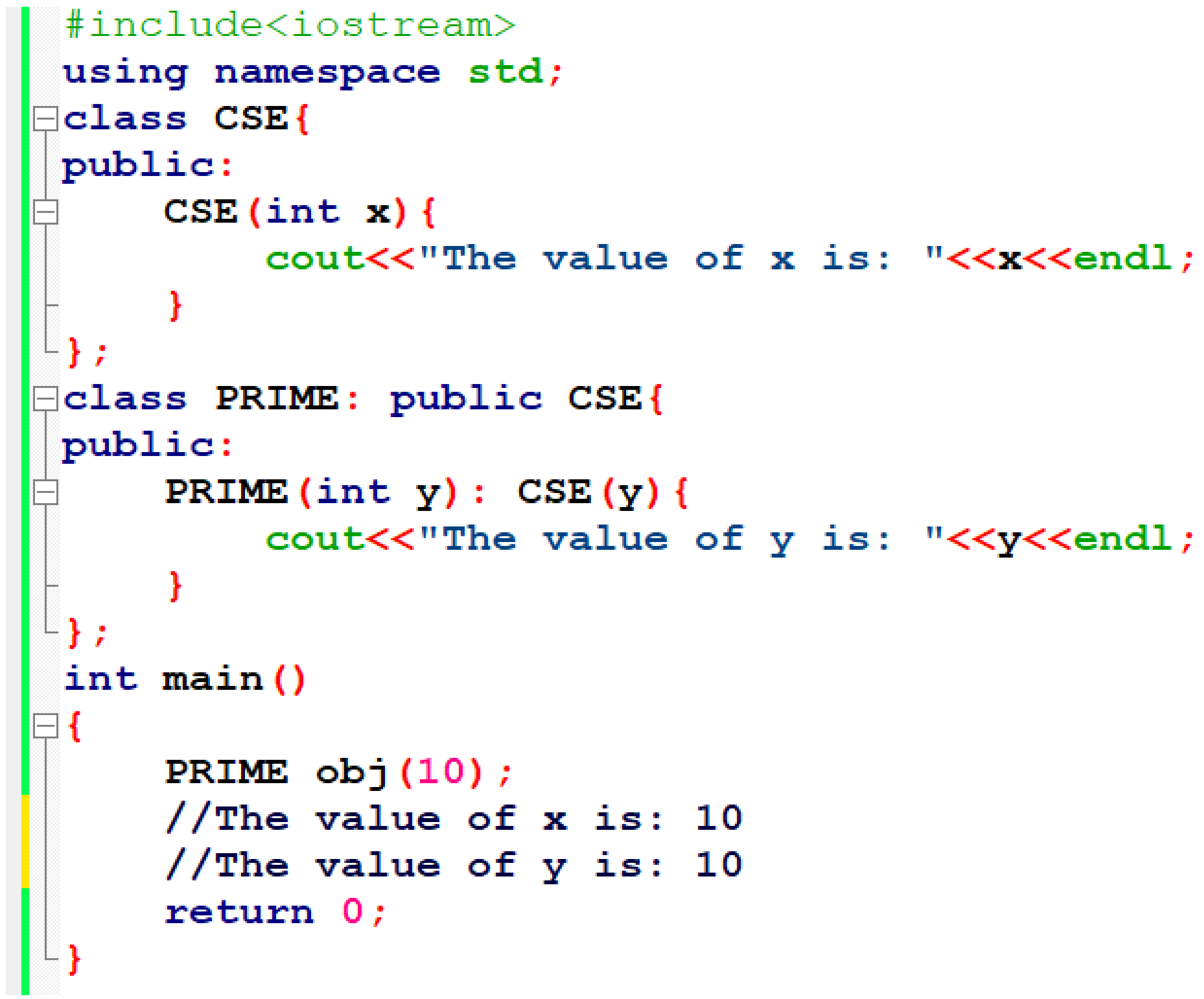
```cpp
#include<iostream>
using namespace std;
class CSE{
public:
    void display(){
        cout<<"I am from CSE Class"<<endl;
    }
};
class PRIME: public CSE{
public:
    void display(){
        cout<<"I am from PRIME Class"<<endl;
    }
};
int main()
{
    PRIME obj;
    obj.display();//What will be the output??
    obj.CSE::display();//Calling the display function of CSE class
    return 0;
}
```

Special Case of Constructor in Inheritance



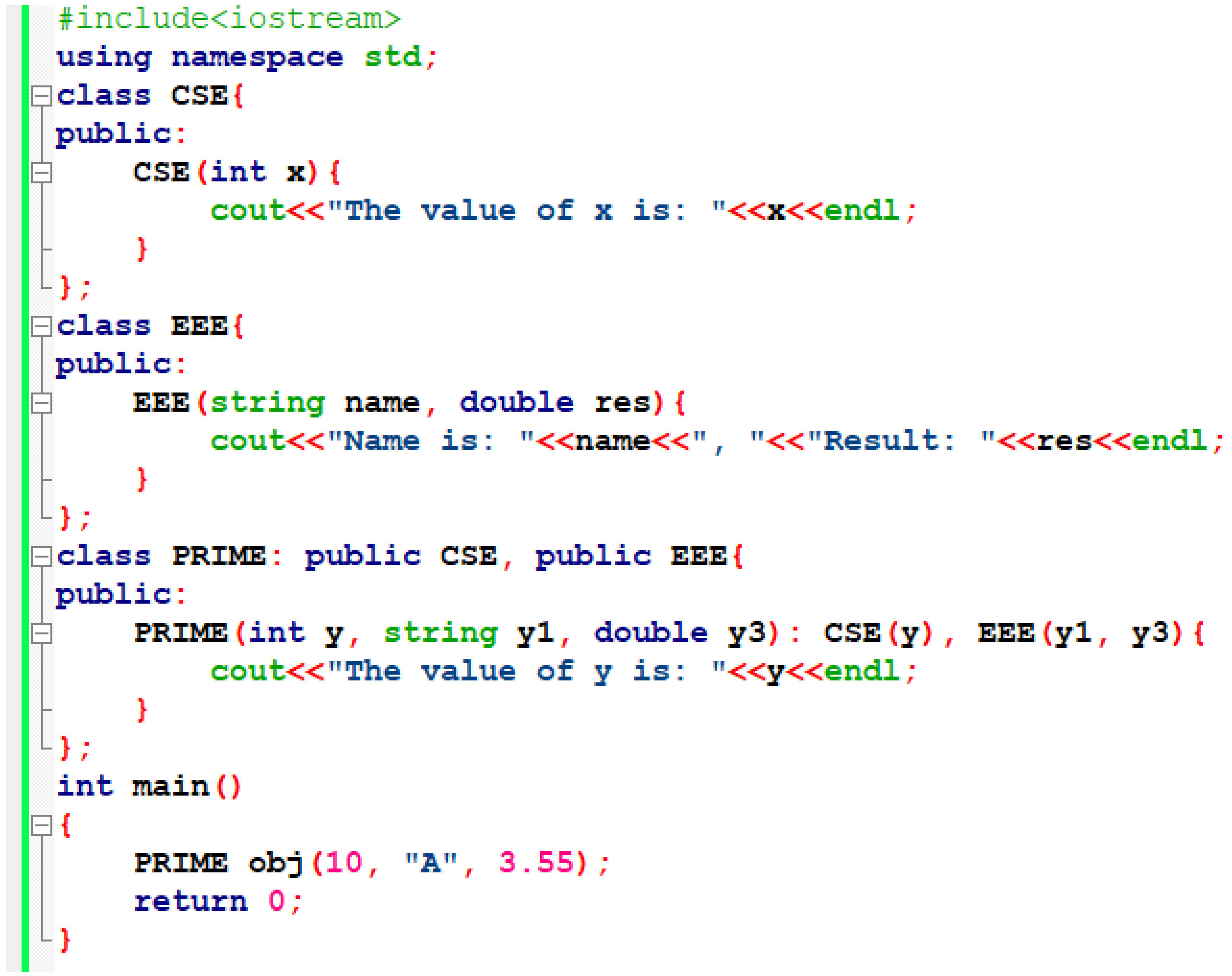
```
#include<iostream>
using namespace std;
class CSE{
public:
    CSE(int x){
        cout<<"The value of x is: "<<x<<endl;
    }
};
class PRIME: public CSE{
public:
    PRIME(int y){
        cout<<"The value of y is: "<<y<<endl;
    }
};
int main()
{
    PRIME obj(10);//What will be the output?
    return 0;
}
```

Special Case of Constructor in Inheritance (Solution)



```
#include<iostream>
using namespace std;
class CSE{
public:
    CSE(int x){
        cout<<"The value of x is: "<<x<<endl;
    }
};
class PRIME: public CSE{
public:
    PRIME(int y): CSE(y){
        cout<<"The value of y is: "<<y<<endl;
    }
};
int main()
{
    PRIME obj(10);
    //The value of x is: 10
    //The value of y is: 10
    return 0;
}
```

Special Case of Constructor in Inheritance (Solution)



```
#include<iostream>
using namespace std;
class CSE{
public:
    CSE(int x){
        cout<<"The value of x is: "<<x<<endl;
    }
};
class EEE{
public:
    EEE(string name, double res){
        cout<<"Name is: "<<name<<", "<<"Result: "<<res<<endl;
    }
};
class PRIME: public CSE, public EEE{
public:
    PRIME(int y, string y1, double y3): CSE(y), EEE(y1, y3){
        cout<<"The value of y is: "<<y<<endl;
    }
};
int main()
{
    PRIME obj(10, "A", 3.55);
    return 0;
}
```

Practice Example (1)

Implement the following description using C++ language:

Create a class “Bank”. Declare private variables like BankID, BankName, and Location and a class constructor that can take any two of declared parameters. And write a function Print_Values() that can print the constructor's values.

Create another class “Customer”. Declare class constructor that can take two parameters like Customer_ID and Amount. And write a function Print_Values() that can print the constructor's values. Inherit the Bank class from the Customer class.

By creating the main function call the Print_Values function of the Bank class.

Practice Example(02)

Write a program in C++ with the following description:

Create three classes “CPP”, “PYTHON”, and “JAVA” with Private Data Members: Invention_Year, Invented_Country. Public Member Functions: A parameterized constructor to take and assign values to the data members and print those values.

Inherit the “CPP”, and “PYTHON” classes from the “JAVA” class.

Now test the program by creating an object of the “JAVA” class.

Thank You