C++

OOP (Inheritance)

Inheritance in OOP

The capability of a class to derive properties and characteristics from another class is called Inheritance.

Inheritance is one of the most important feature of Object Oriented Programming.

Sub Class :

The class that inherits properties from another class is called Sub class or Derived class.

Super Class :

The class whose properties are inherited by sub class is called Base Class or Super class.

The article is divided into following subtopics :

- Why and when to use inheritance ?
- Modes of Inheritance
- Types of Inheritance

Why and when to use inheritance?

Consider a group of vehicles. You need to create classes for Bus, Car and Truck.
The methods fuelAmount(), capacity(), applyBrakes() will be same for all of the three classes. If we create these classes avoiding inheritance then we have to write all of these functions in each of the three classes as shown in below figure:

Class Car

fuelAmount()

Capacity()

applyBrakes()

Class Bus

fuelAmount()

Capacity()

applyBrakes()

Class Truck

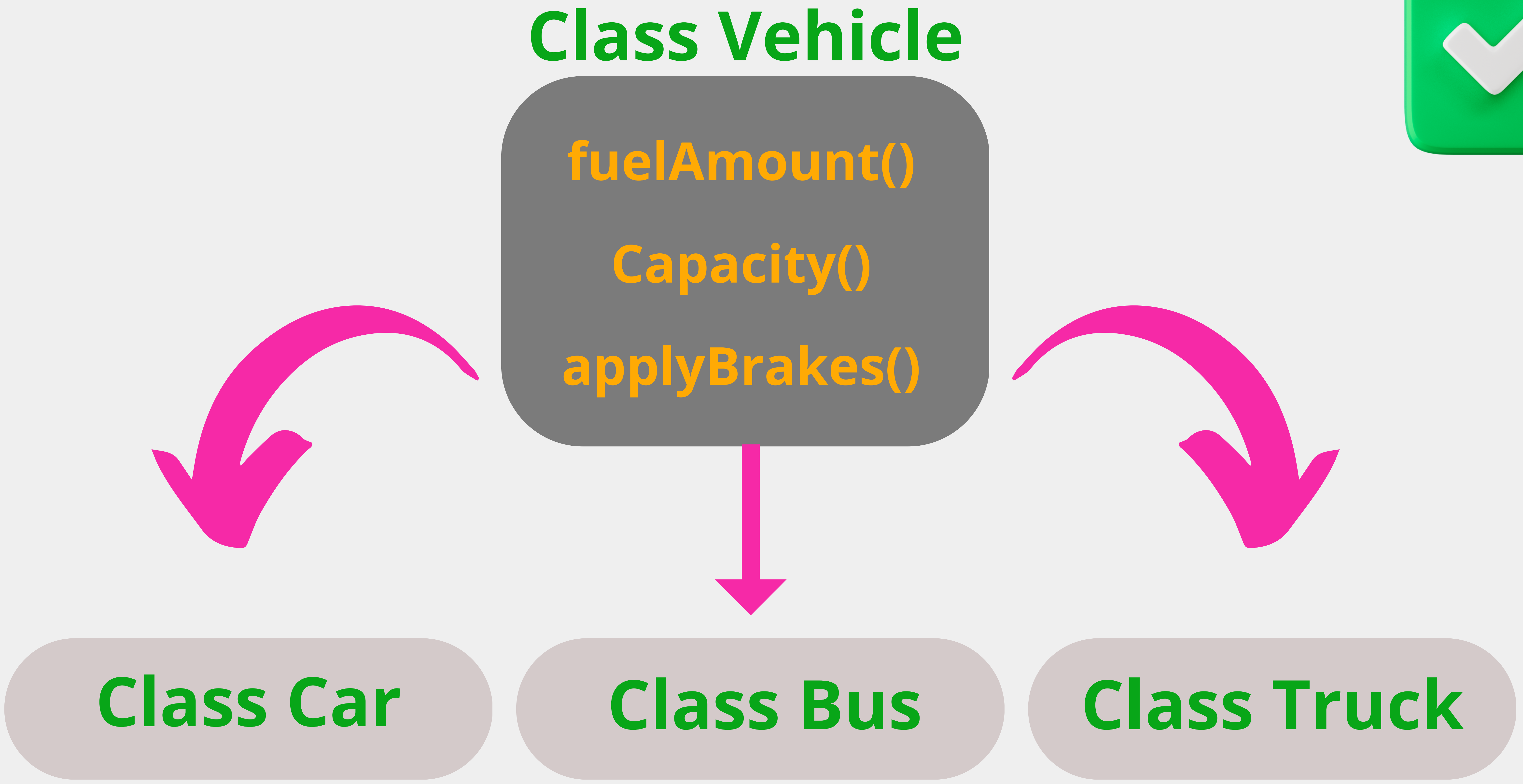
fuelAmount()

Capacity()

applyBrakes()

You can clearly see that above process results in duplication of same code 3 times. This increases the chances of error and data redundancy.

To avoid this type of situation, inheritance is used. If we create a class Vehicle and write these three functions in it and inherit the rest of the classes from the vehicle class, then we can simply avoid the duplication of data and increase re-usability.
Look at the below diagram in which the three classes are inherited from vehicle class :


Class Vehicle
fuelAmount()
Capacity()
applyBrakes()
Class Car
Class Bus
Class Truck

Implementing Inheritance in C++

Syntax

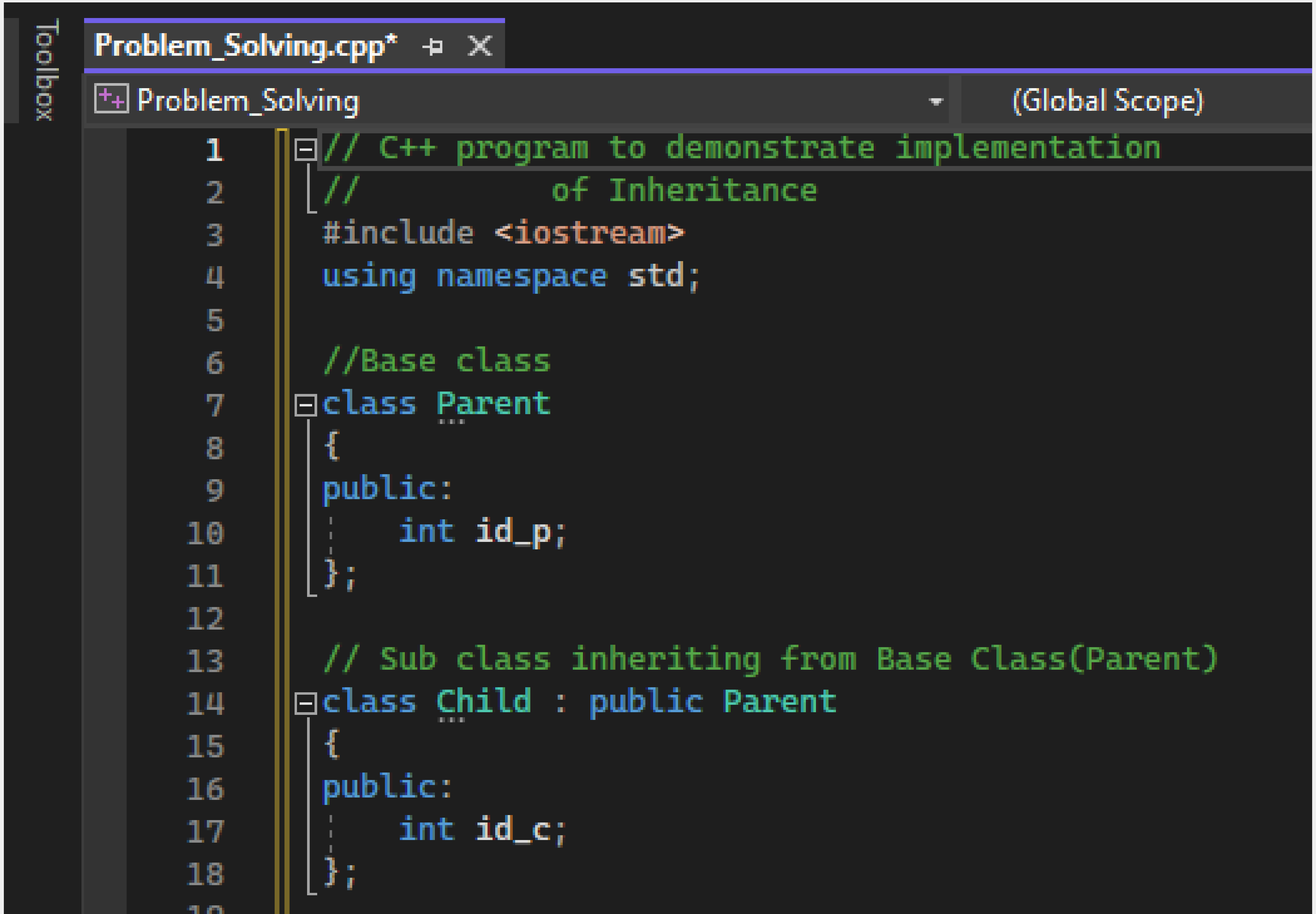
```
class subClass_name : access_mode baseClass_name
{
      //body of subClass
};
```

Here, subClass_name is the name of the sub class, access_mode is the mode in which you want to inherit this sub class for example: public, private etc. and baseClass_name is the name of the base class from which you want to inherit the sub class.

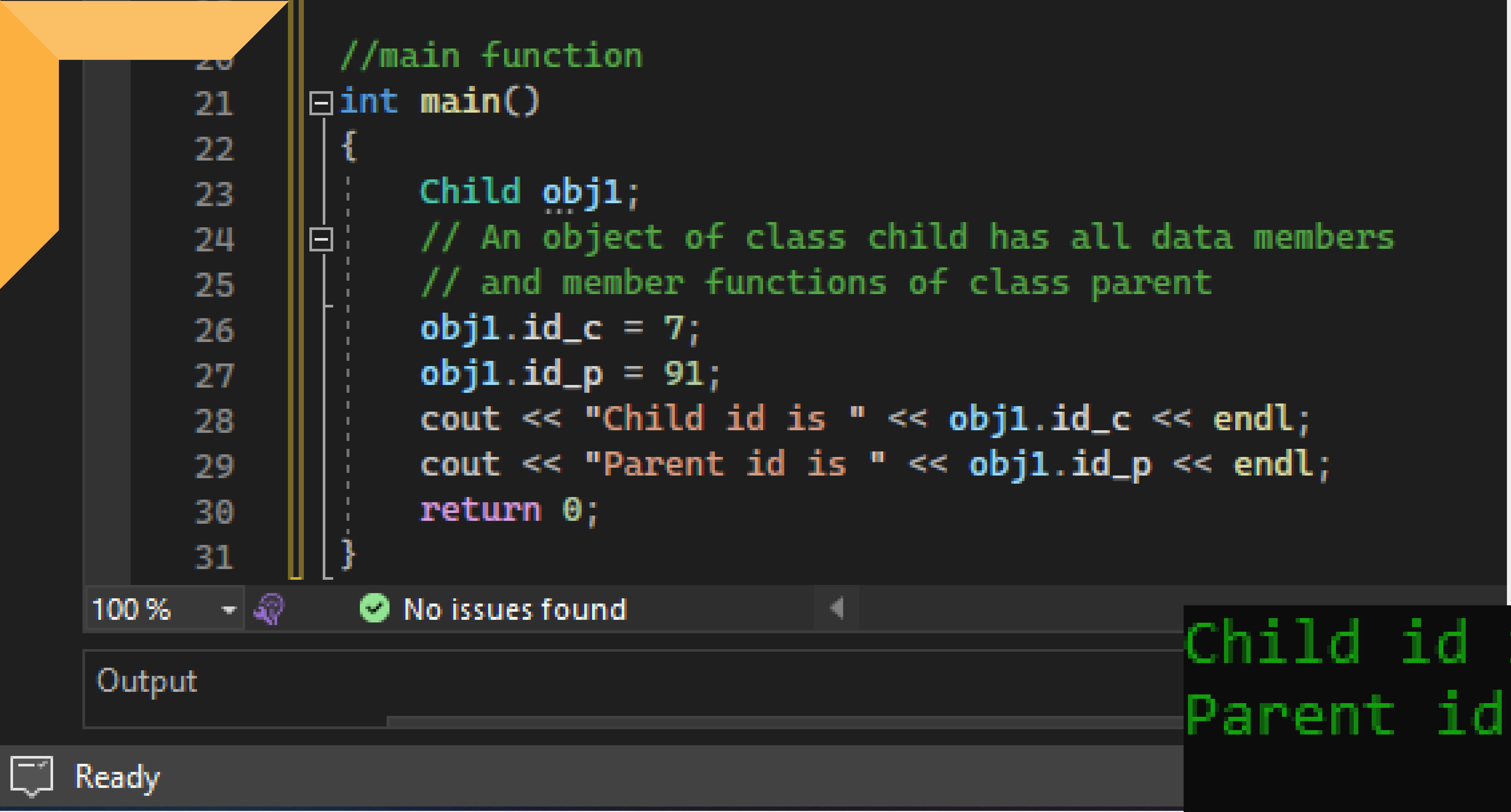
NOTE :

A derived class doesn’t inherit access to private data members. However, it does inherit a full parent object, which contains any private members which that class declares.

Example

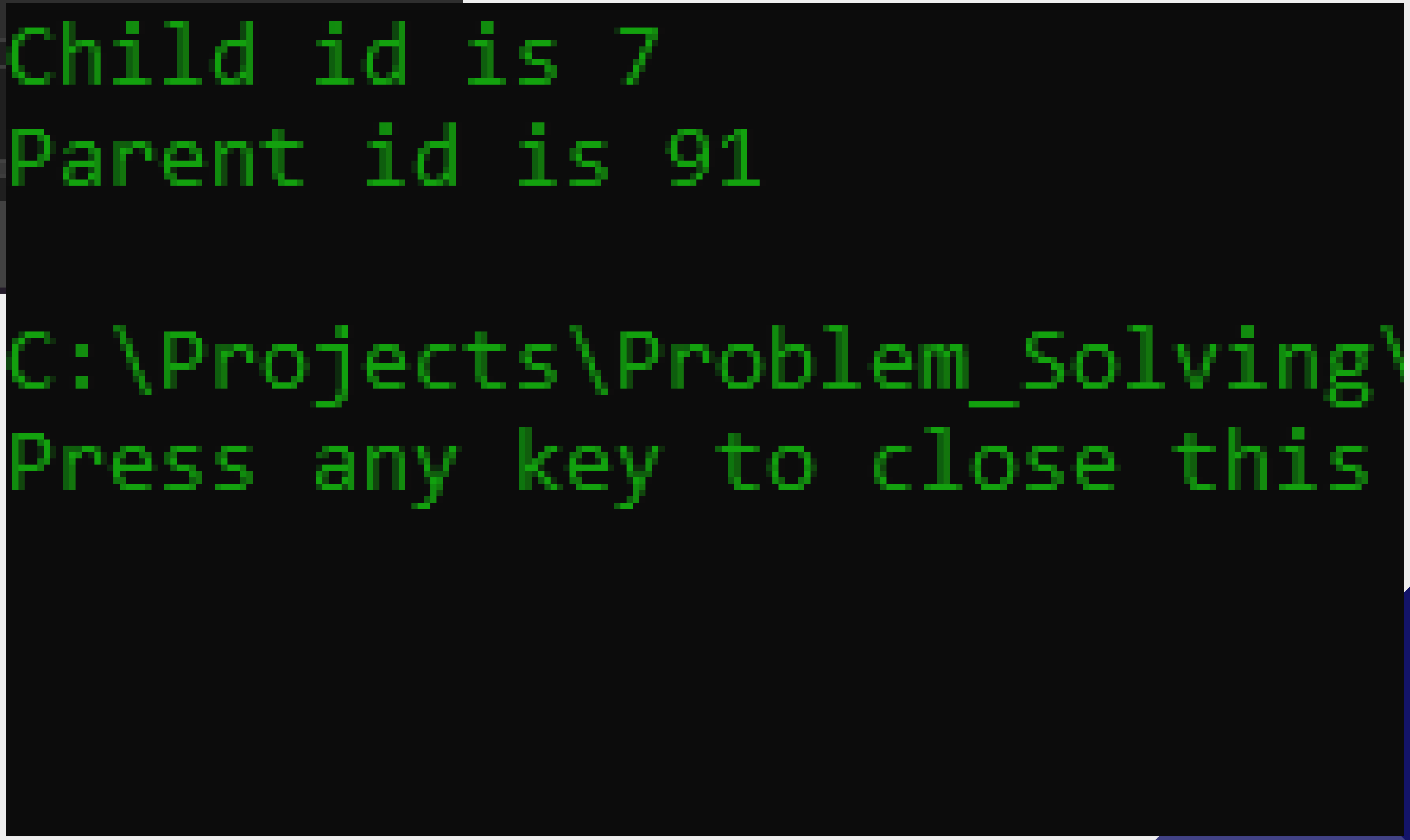


```cpp
// C++ program to demonstrate implementation
//              of Inheritance
#include <iostream>
using namespace std;

//Base class
class Parent
{
public:
    int id_p;
};

// Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
public:
    int id_c;
};
```



```
    //main function
int main()
{
    Child obj1;
    // An object of class child has all data members
    // and member functions of class parent
    obj1.id_c = 7;
    obj1.id_p = 91;
    cout << "Child id is " << obj1.id_c << endl;
    cout << "Parent id is " << obj1.id_p << endl;
    return 0;
}
```

OUTPUT



```
Child id is 7
Parent id is 91

C:\Projects\Problem_Solving\
Press any key to close this
```

Modes of Inheritance

Public mode: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.

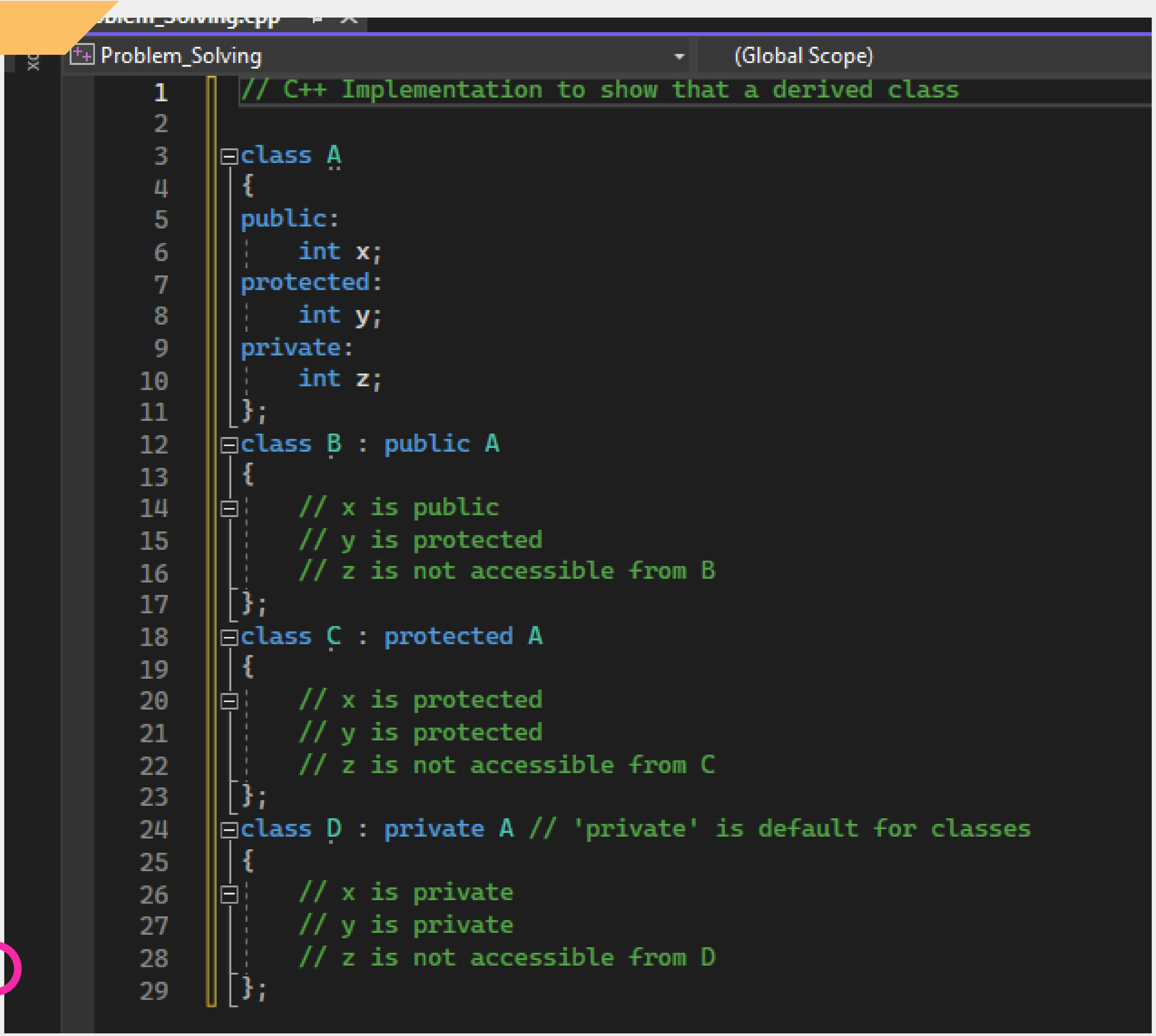
Protected mode: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.

Private mode: If we derive a sub class from a Private base class. Then both public members and protected members of the base class will become Private in derived class.

NOTE :

The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed.

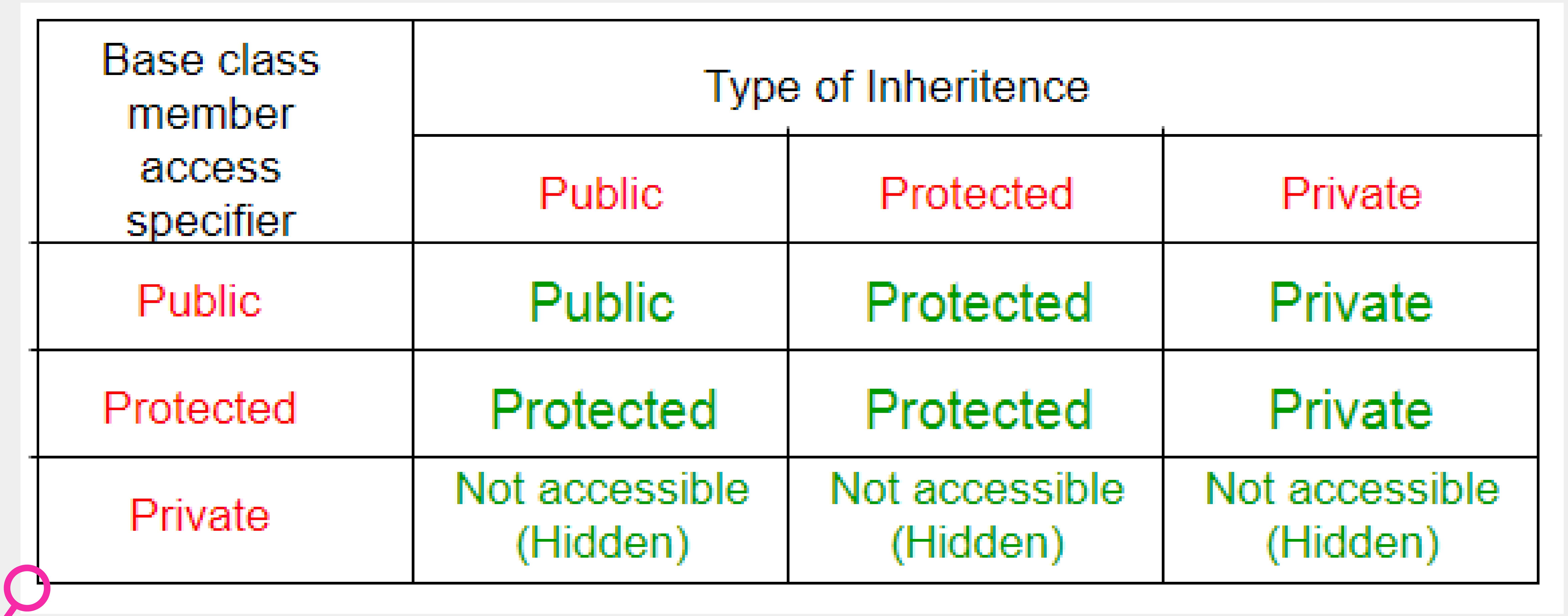
For example, We have Class A is base class and Classes B, C and D all contain the variables x, y and z in below example. It is just question of access.



```cpp
// C++ Implementation to show that a derived class

class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};
class B : public A
{
    // x is public
    // y is protected
    // z is not accessible from B
};
class C : protected A
{
    // x is protected
    // y is protected
    // z is not accessible from C
};
class D : private A // 'private' is default for classes
{
    // x is private
    // y is private
    // z is not accessible from D
};
```

Explain Access Mode

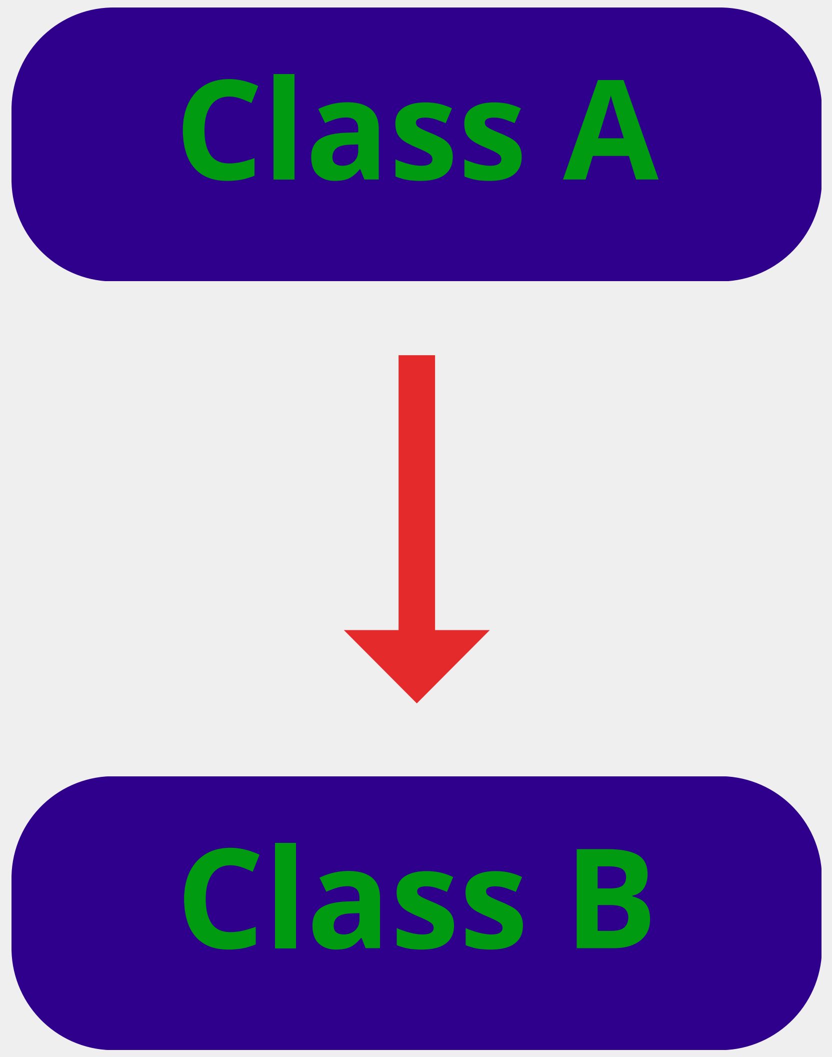
Inheritance Access Mode Table



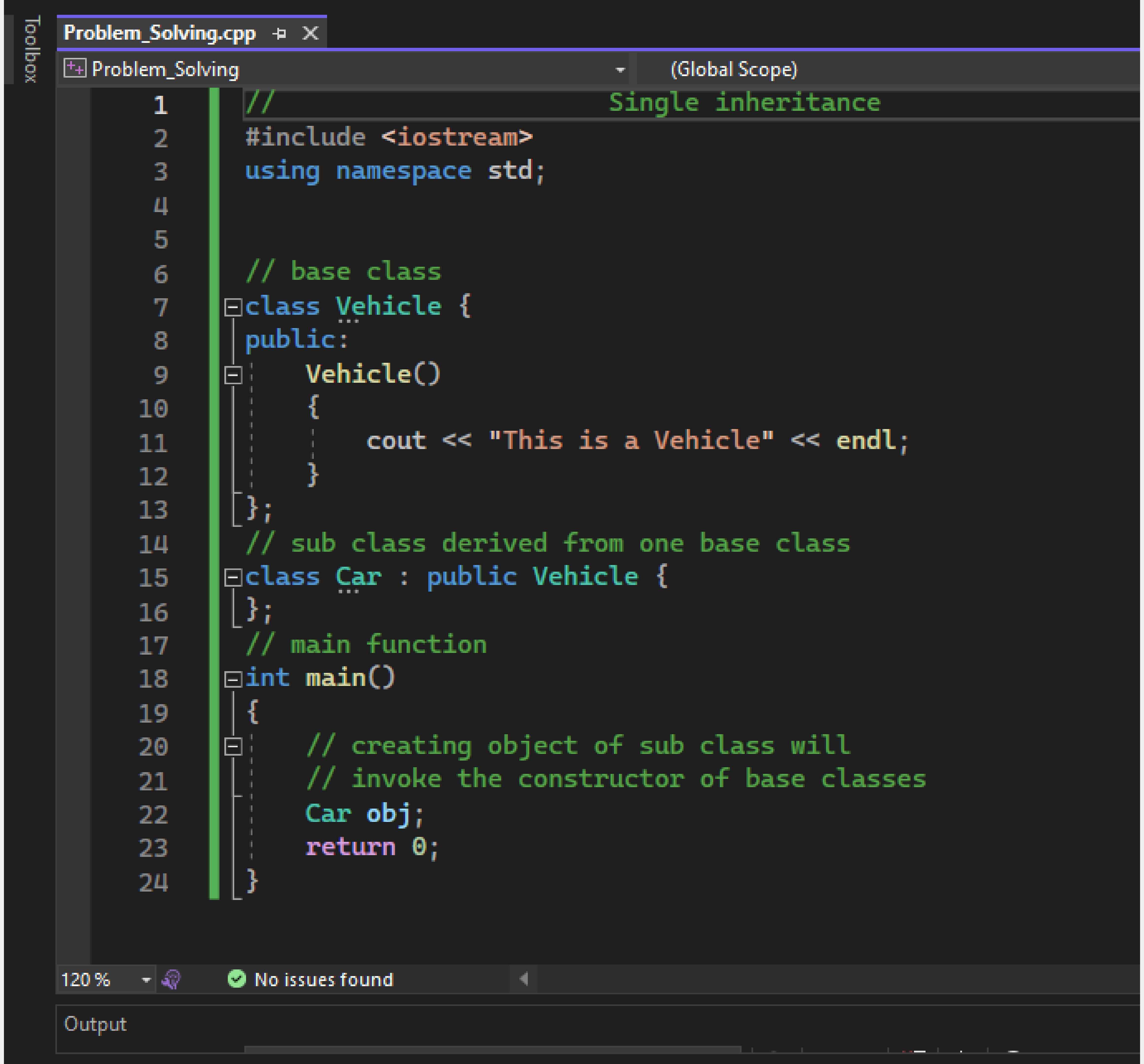
Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

Types of Inheritance Classes in C++

1 Single Inheritance



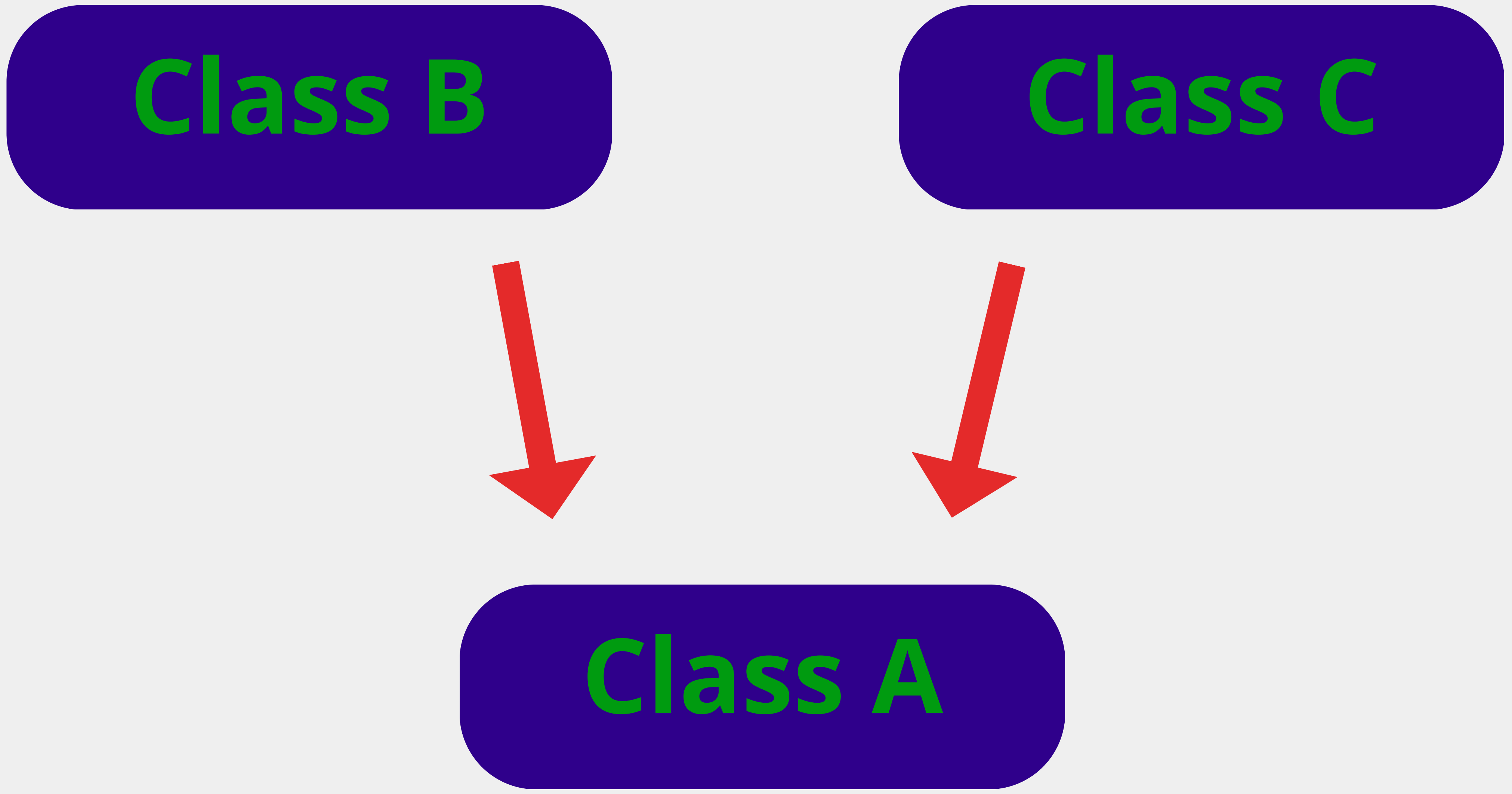
Syntax :



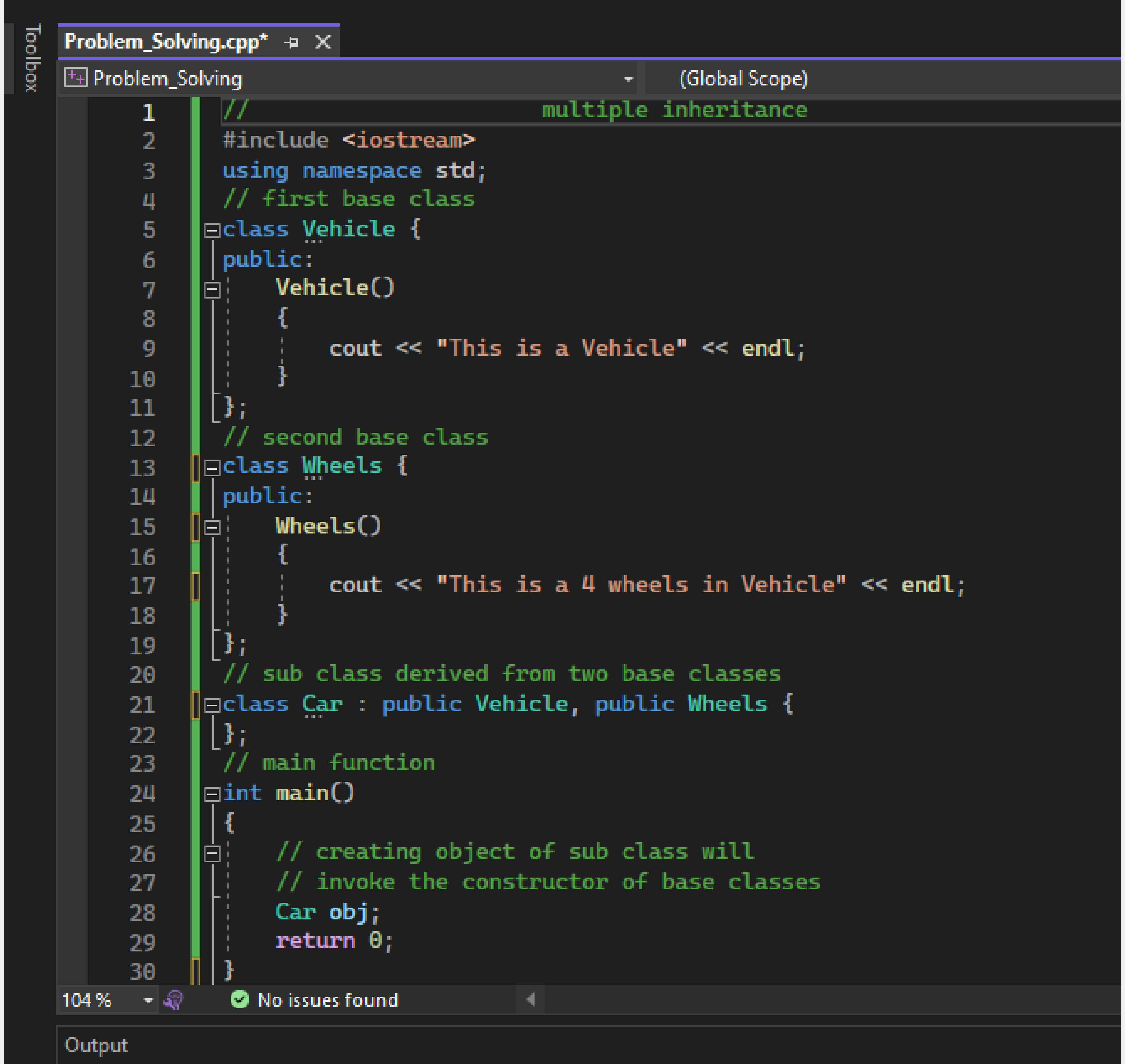
```
//                    Single inheritance
#include <iostream>
using namespace std;


// base class
class Vehicle {
public:
    Vehicle()
    {
        cout << "This is a Vehicle" << endl;
    }
};
// sub class derived from one base class
class Car : public Vehicle {
};
// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0;
}
```

Types of Inheritance Classes in C++

2 Multiple Inheritance



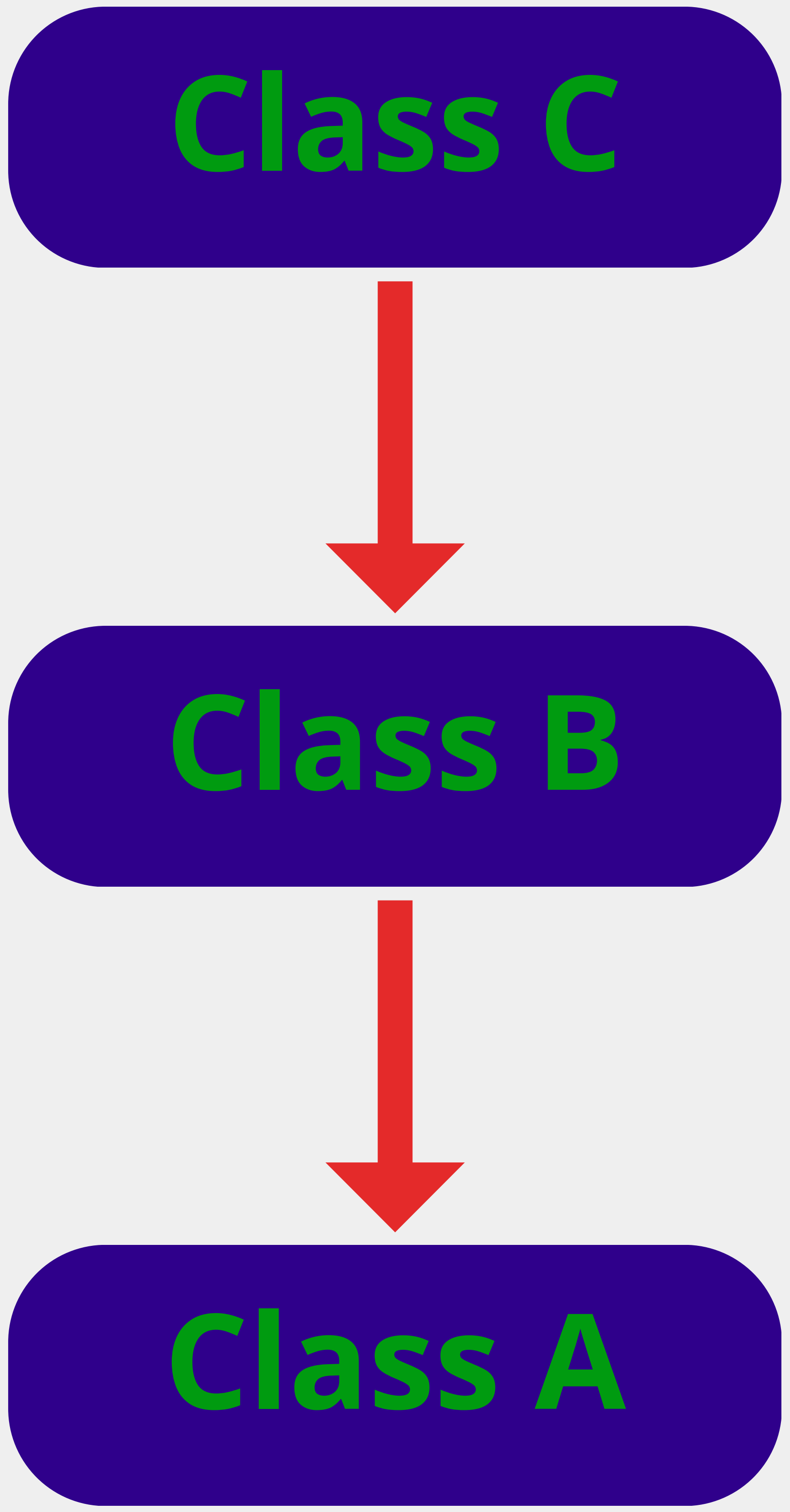
Syntax :



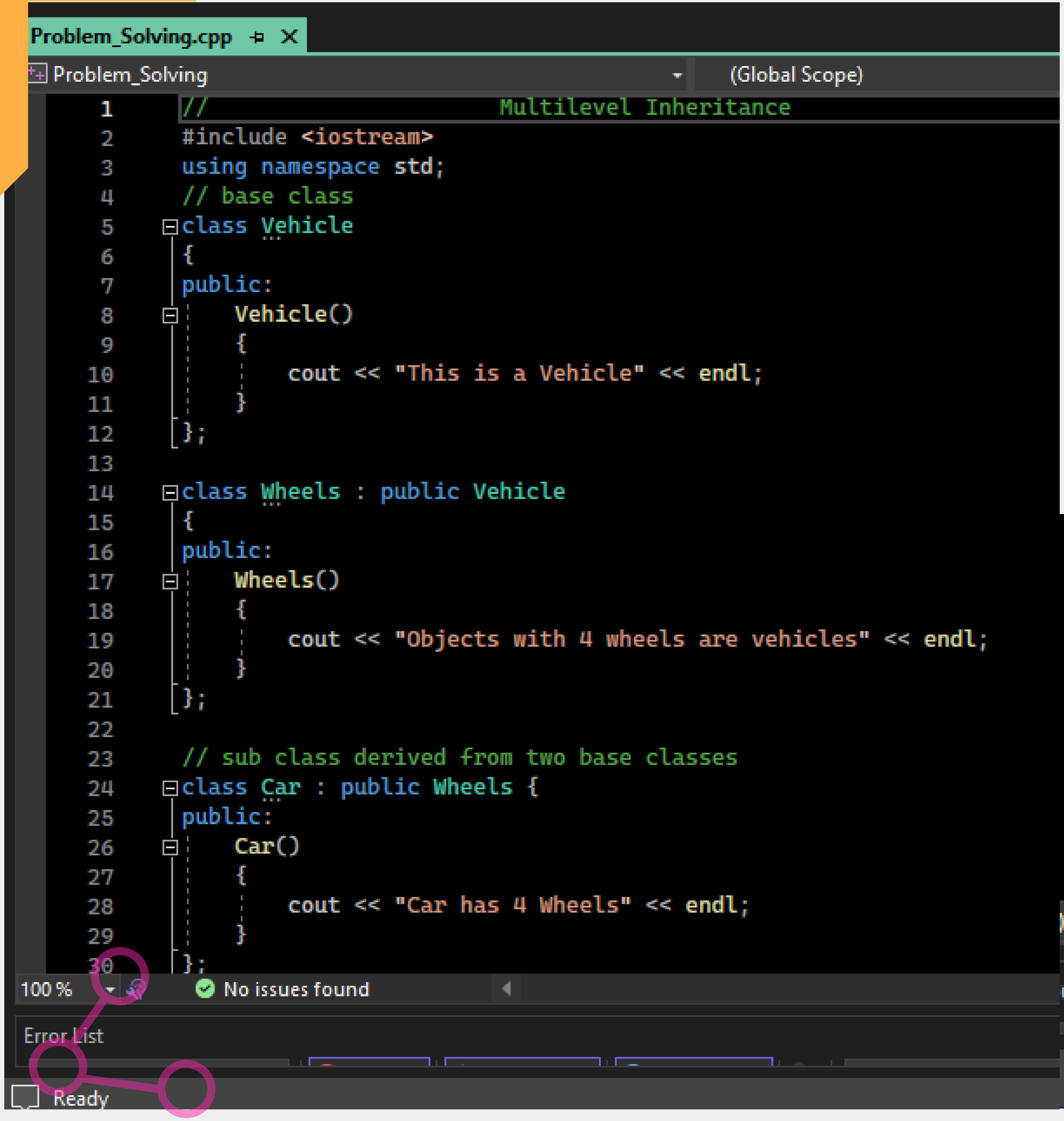
```cpp
//                     multiple inheritance
#include <iostream>
using namespace std;
// first base class
class Vehicle {
public:
    Vehicle()
    {
        cout << "This is a Vehicle" << endl;
    }
};
// second base class
class Wheels {
public:
    Wheels()
    {
        cout << "This is a 4 wheels in Vehicle" << endl;
    }
};
// sub class derived from two base classes
class Car : public Vehicle, public Wheels {
};
// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0;
}
```

Types of Inheritance Classes in C++

3 Multilevel Inheritance



Syntax



Problem_Solving.cpp

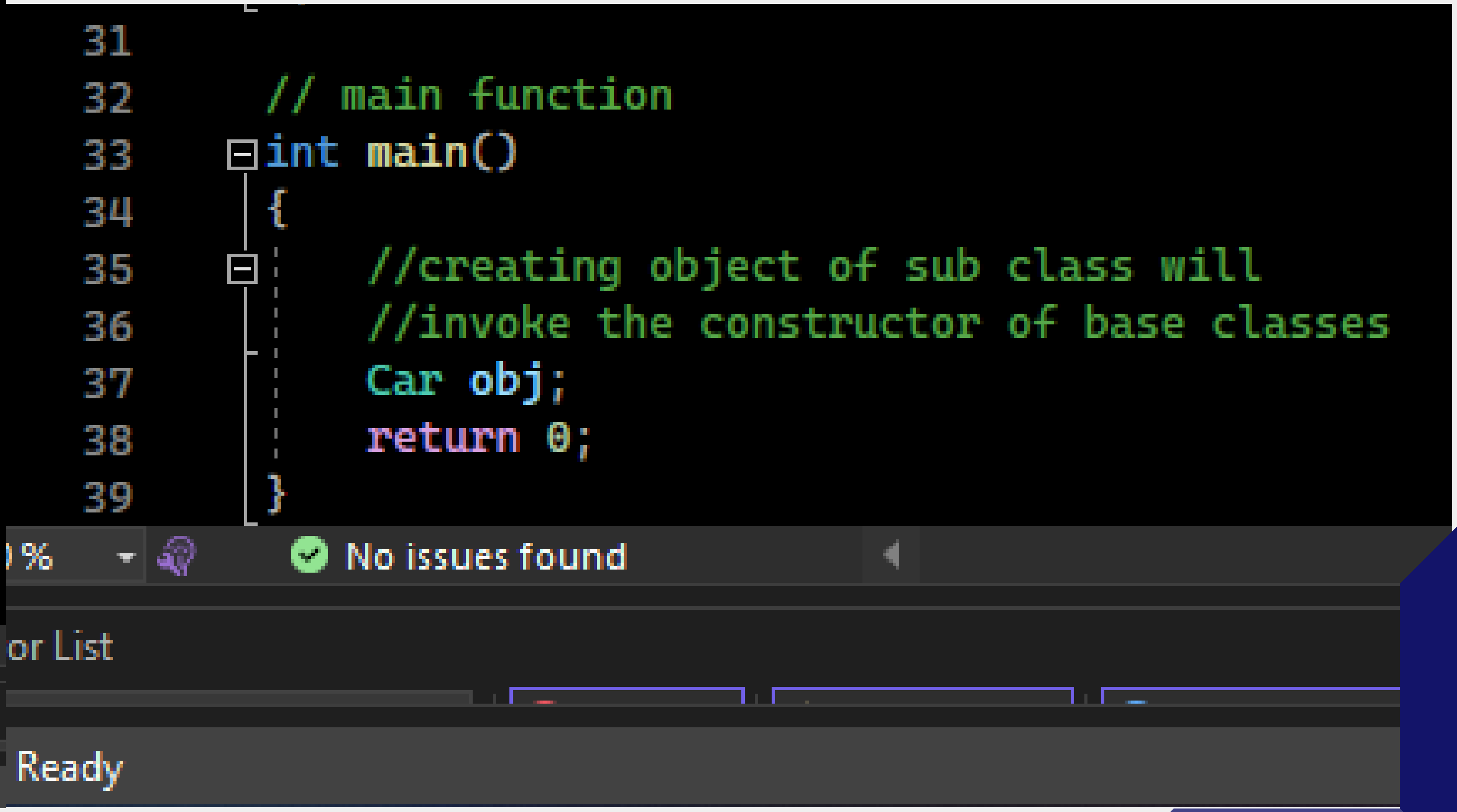
Problem_Solving (Global Scope)

```
//                     Multilevel Inheritance
#include <iostream>
using namespace std;
// base class
class Vehicle
{
public:
    Vehicle()
    {
        cout << "This is a Vehicle" << endl;
    }
};

class Wheels : public Vehicle
{
public:
    Wheels()
    {
        cout << "Objects with 4 wheels are vehicles" << endl;
    }
};

// sub class derived from two base classes
class Car : public Wheels {
public:
    Car()
    {
        cout << "Car has 4 Wheels" << endl;
    }
};
```

100 % No issues found

Error List

Ready



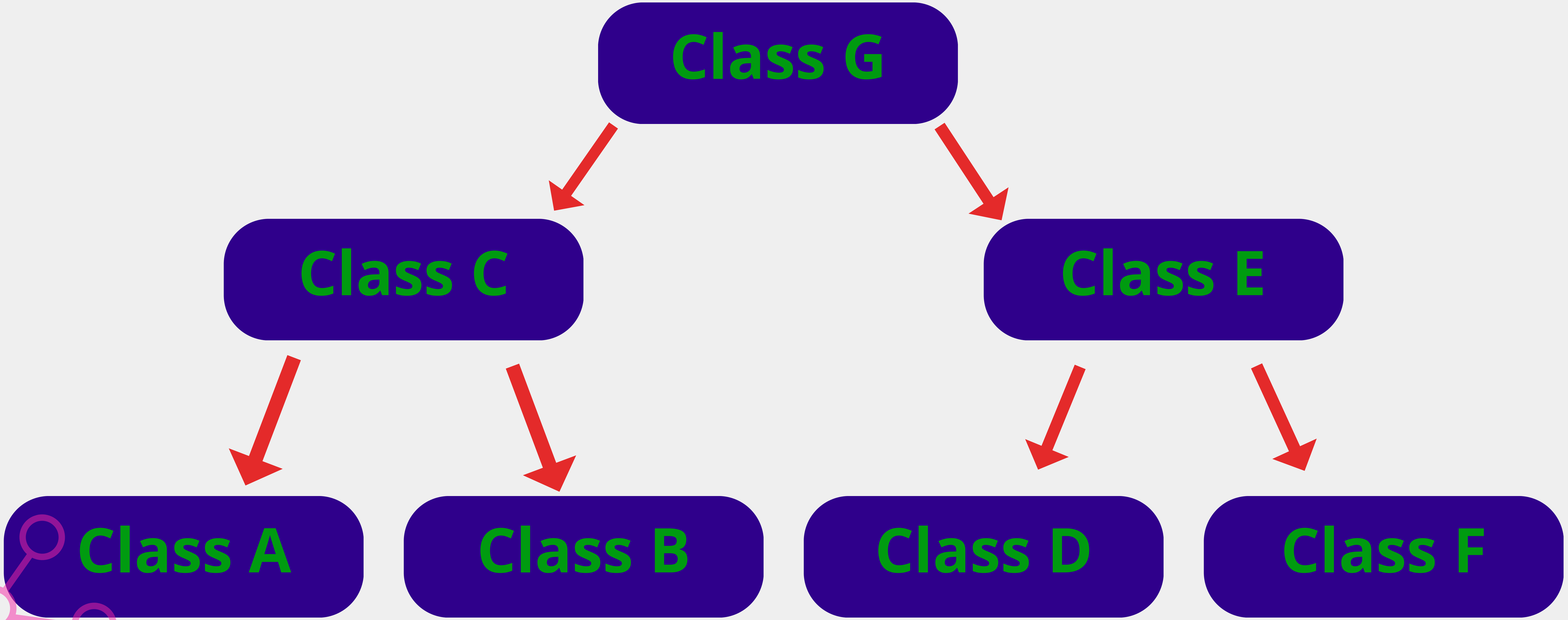
```

// main function
int main()
{
    //creating object of sub class will
    //invoke the constructor of base classes
    Car obj;
    return 0;
}
```

No issues found

Ready

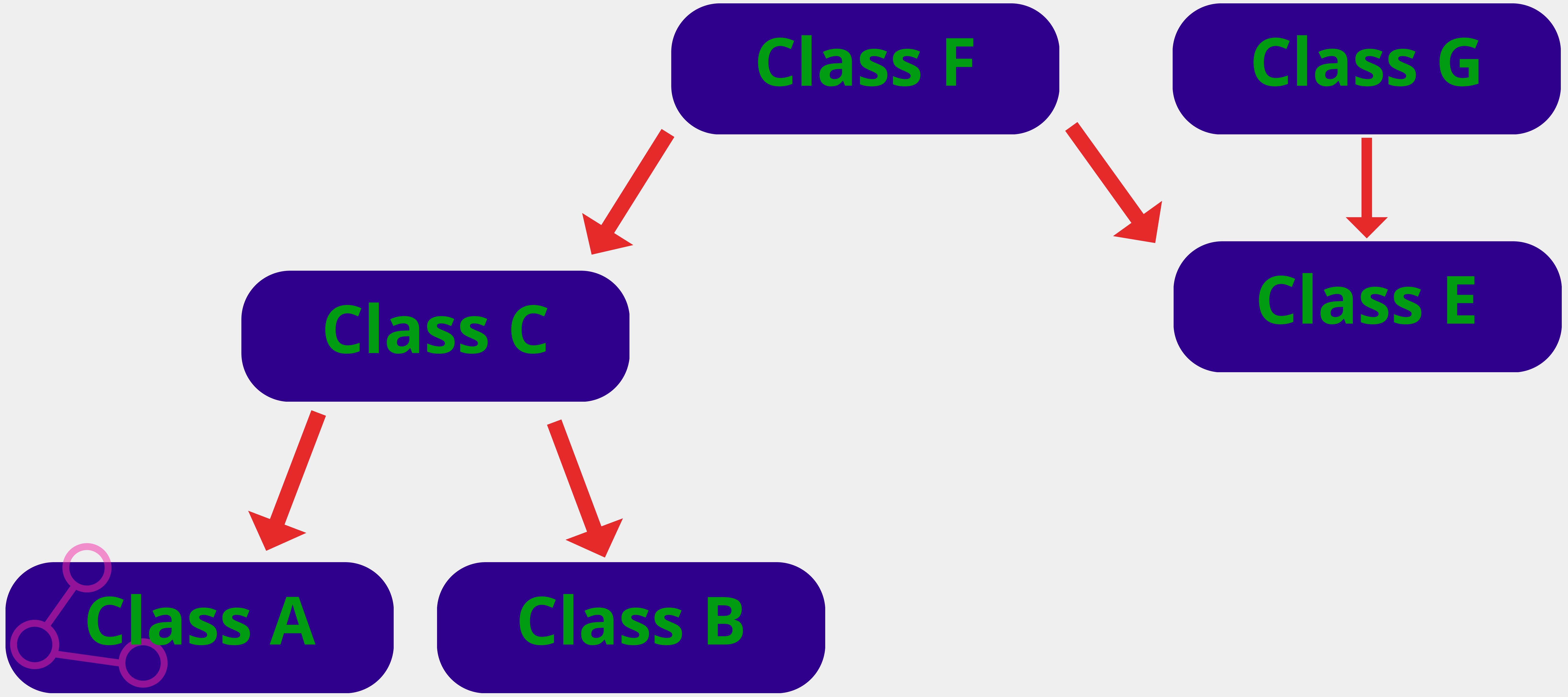
Types of Inheritance Classes in C++

4 Hierarchical Inheritance



Types of Inheritance Classes in C++

4 Hybrid (Virtual) Inheritance



Created By

Mohamed Hafez

Nada Hamada

THANK YOU FOR LISTENING!