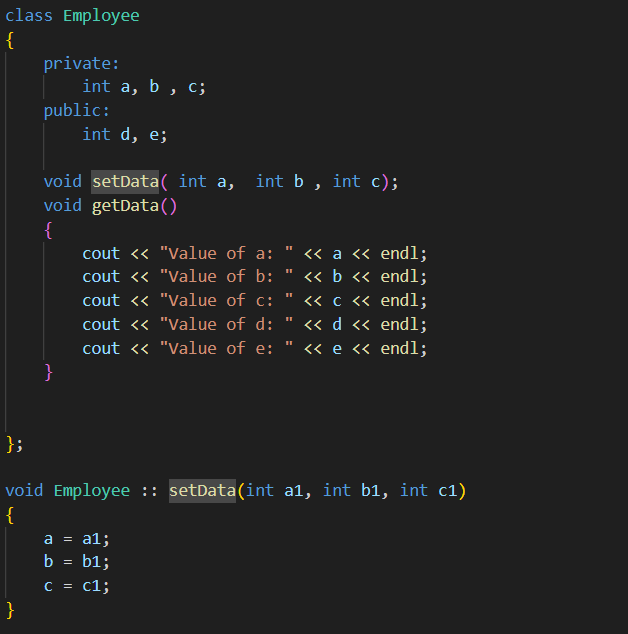
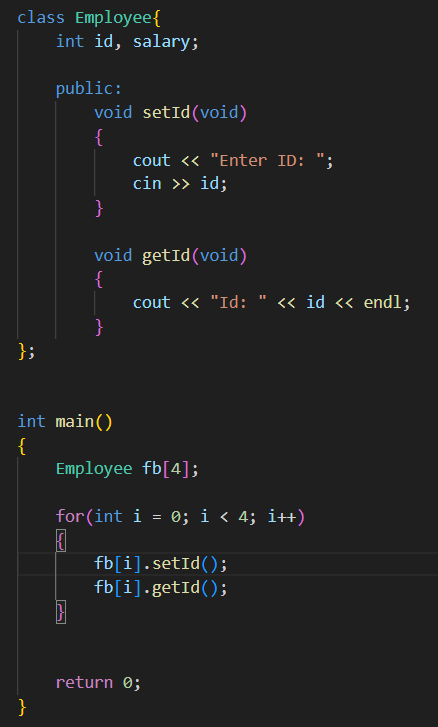
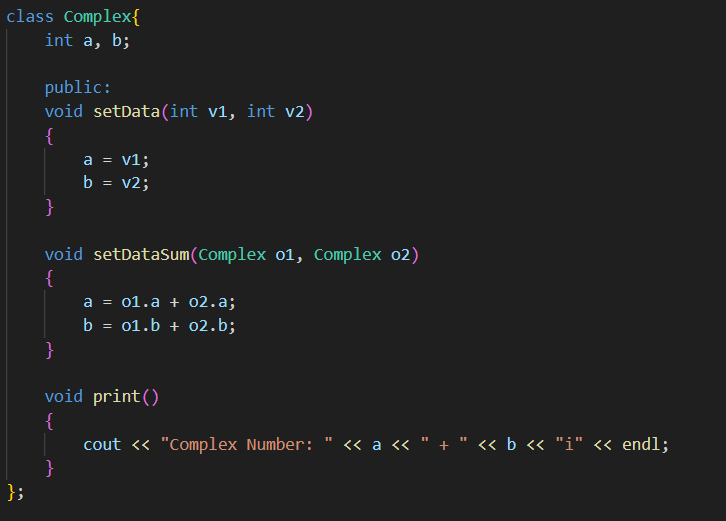
**Declaring Functions outside of class:**

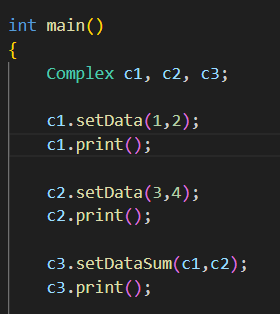
****

**Array of Objects:**

****

**Passing Objects to Functions as Arguments:**

****

****

**Friend Functions:**

Used to access private members of a class outside the class.

Not in the scope of the class [It only allows access and does not make it a function within the class]

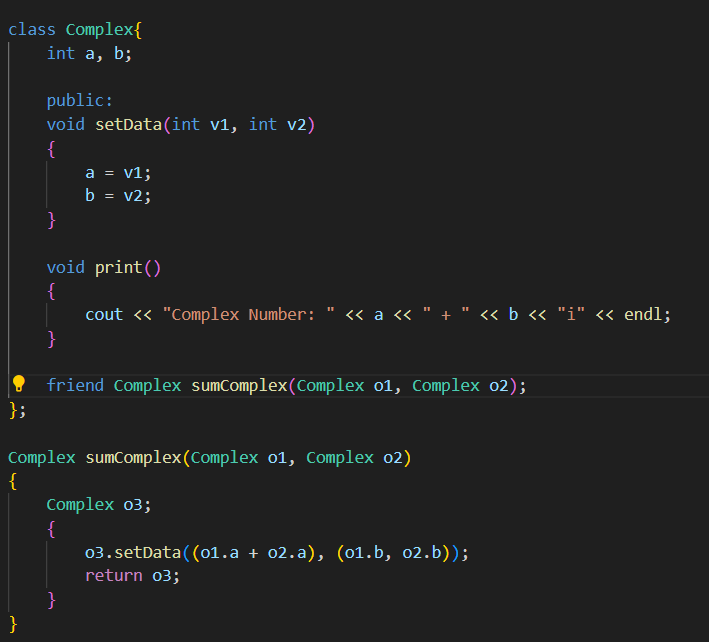
It cannot be called from objects of the class as it is not in scope of the class.

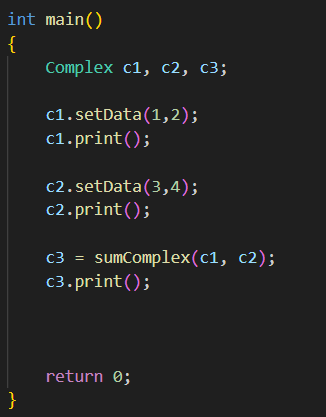
Can be invoked with objects

Usually contains objects as arguments

Can be declared inside public or private sections of the class

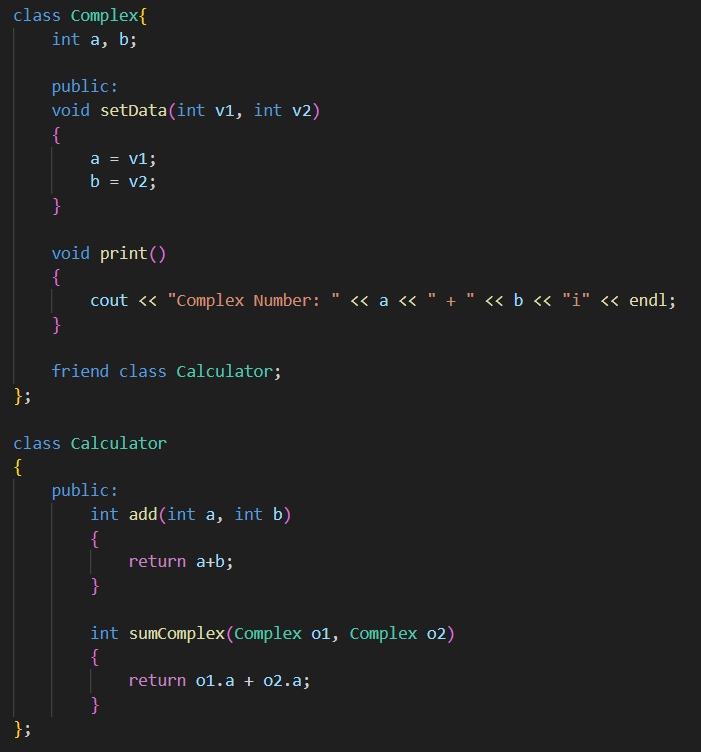
Cannot access the members directly by their names and need object and require object\_name.member\_name to access the members of the class.





**Friend Classes:**

Process of Declaring an entire Class as a friend



**Constructors:**

Special member function with the same name of the class to initialize its members

It is automatically Invoked whenever an object is created.

It has no return type.

Should be declared in the public section of the class.

It can have default arguments.

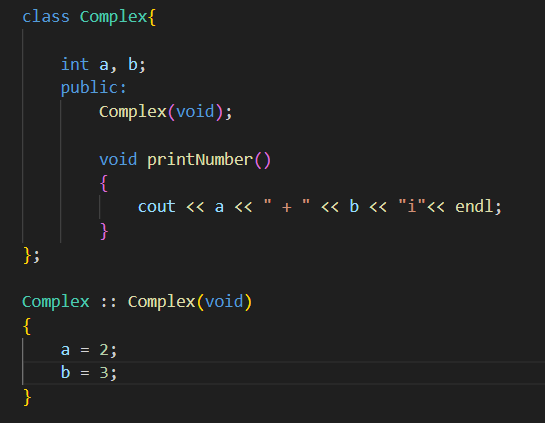
We cannot refer to their address.

We can Overload Constructors

We can give default arguments to Constructors.

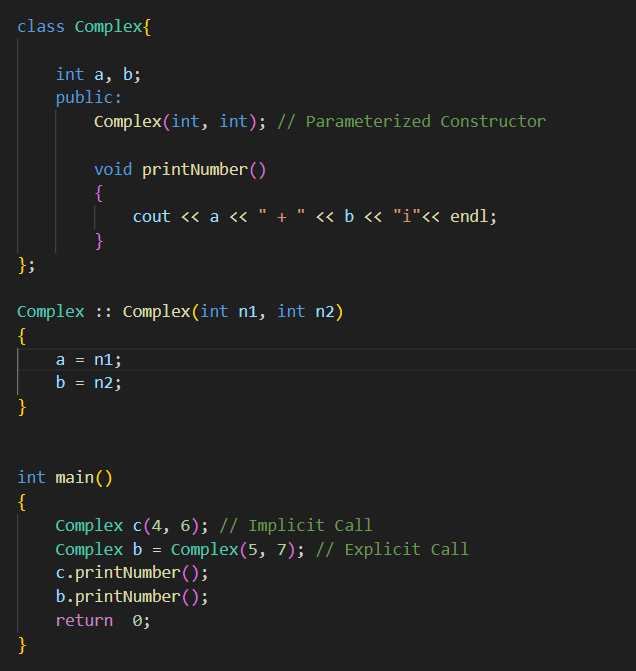
If we manually create a constructor then default constructors are not initialized.

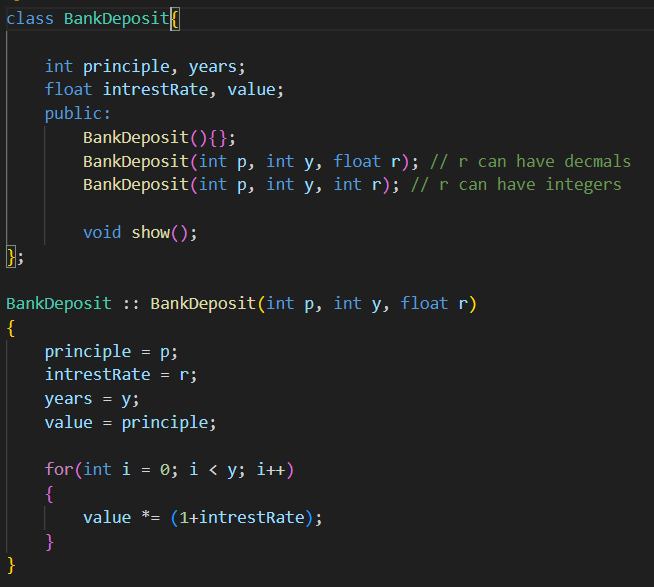
It is best practice to create an empty constructor if we are creating parameterized constructors.

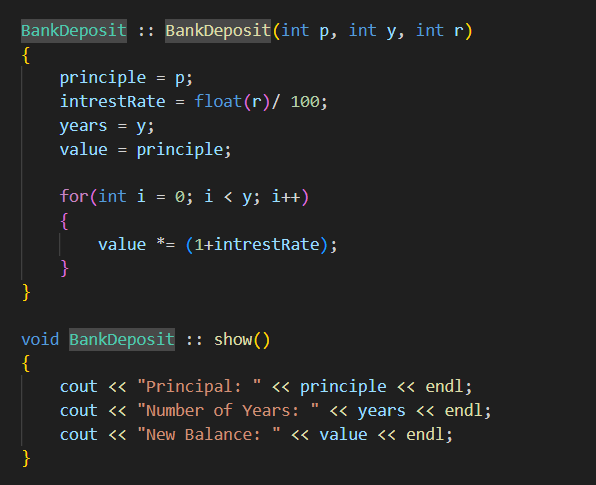


**Parameterized Constructor:**

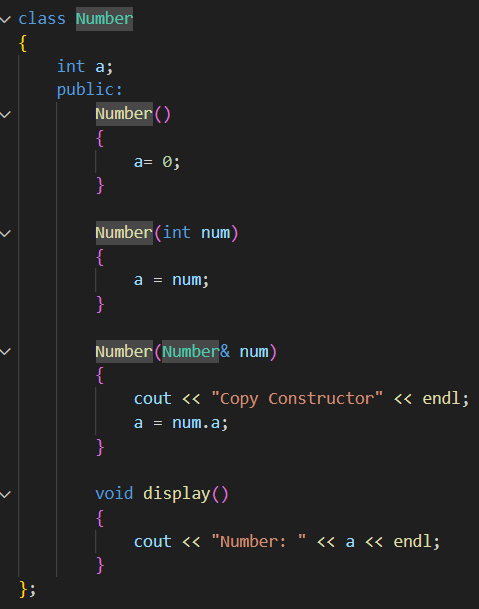
It takes arguments as parameters.

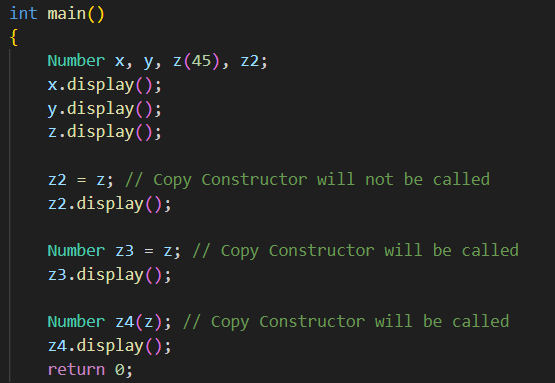


****

****

**Copy Constructors:**





**Destructors:**

Used to free the dynamic memory created when using a class.

Is created automatically if it is not defined manually.

It does not take any argument or returns any value

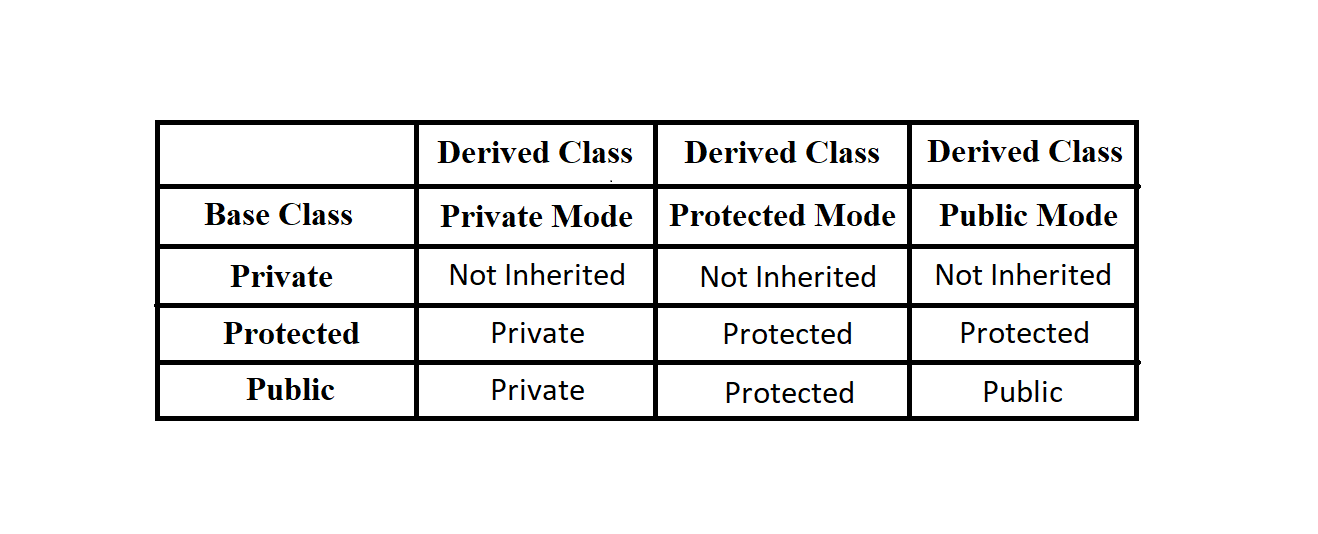
**Inheritance:**

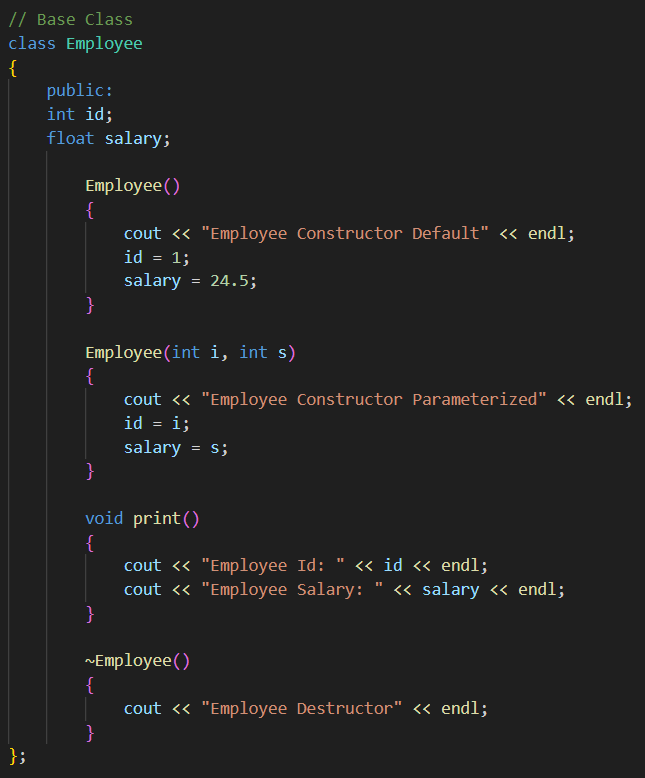
Existing class is called Base Class

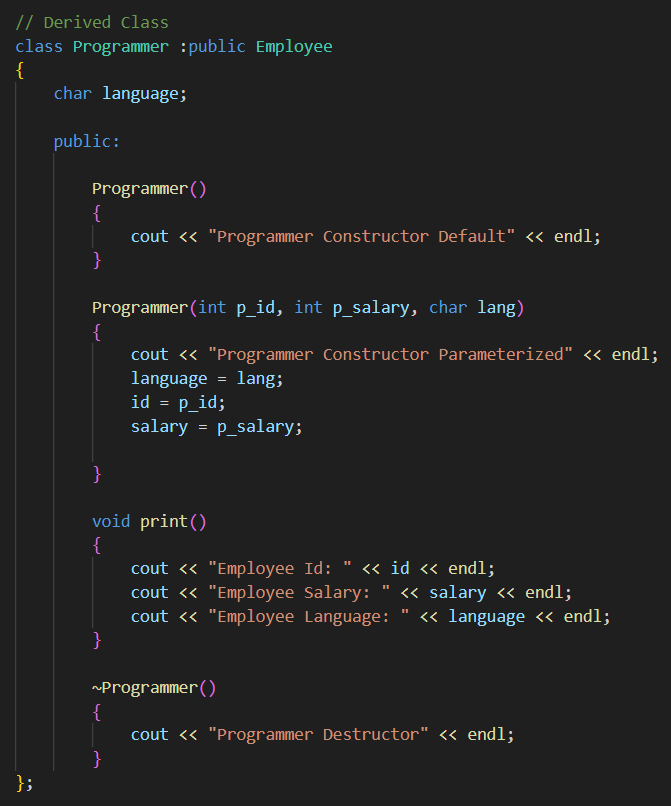
New class that is inherited is called Derived Class

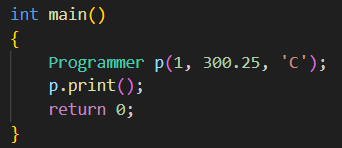
Method to Perform inheritance

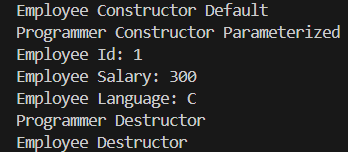
class {{Derived Class Name} : {visibility} {Base Class Name}}







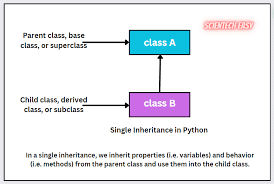


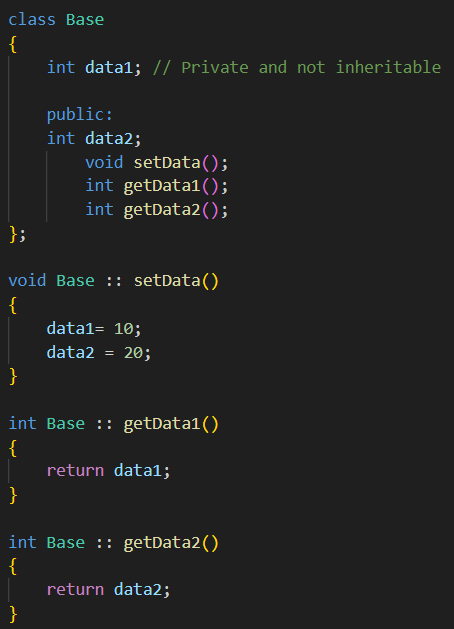


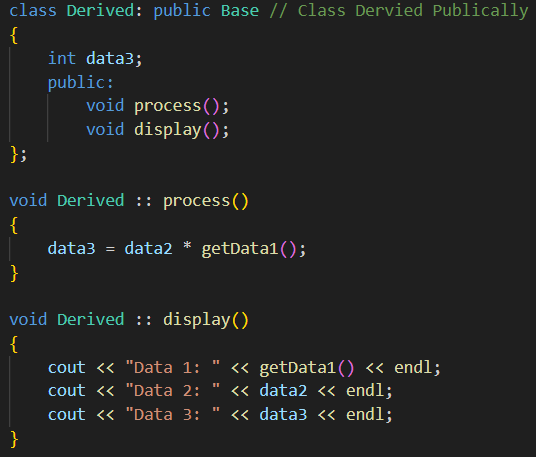
**Different Types of Inheritance:**

**Single Inheritence:**

A derived class with only one Base Class

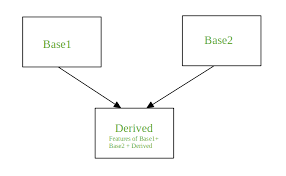


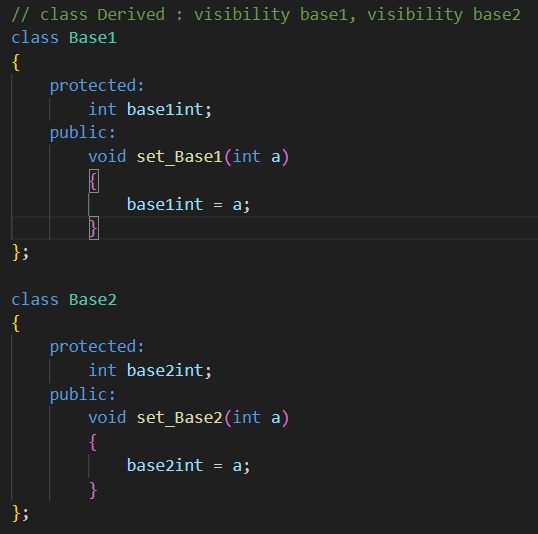


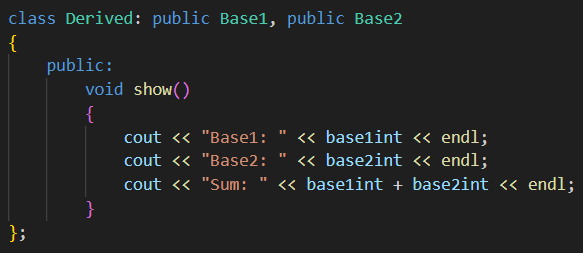


Multiple Inheritance:

A derived class with more than one Base Class

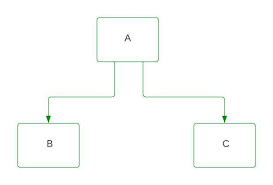






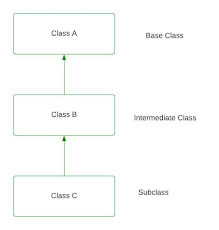
Hierarchical Inheritance:

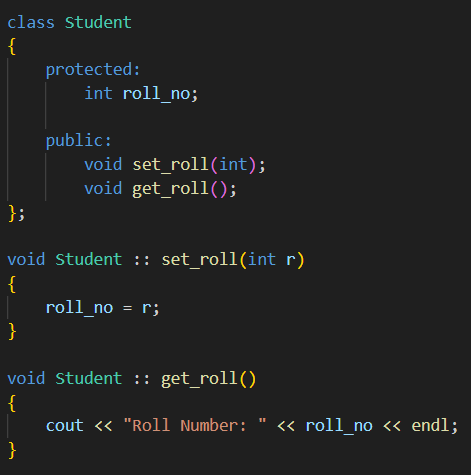
Several Derived classes form a single base class

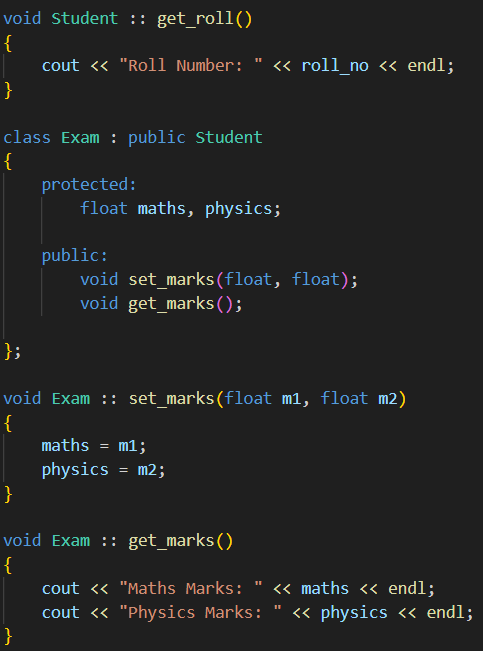


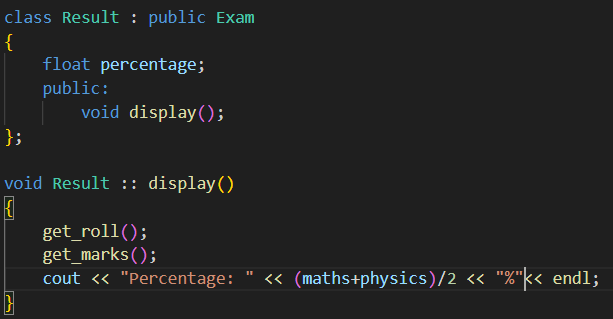
Multilevel Inheritance:

Derving a class from already derived class







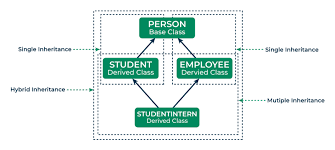


Hybrid Inheritance:

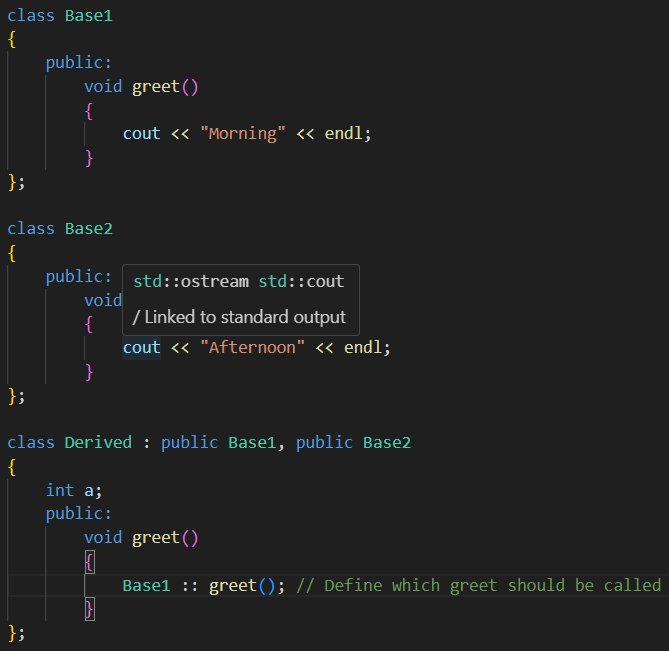
Combination of Multiple inheritance and multilevel inheritance

A class is derived from 2 classes as in multiple inheritance

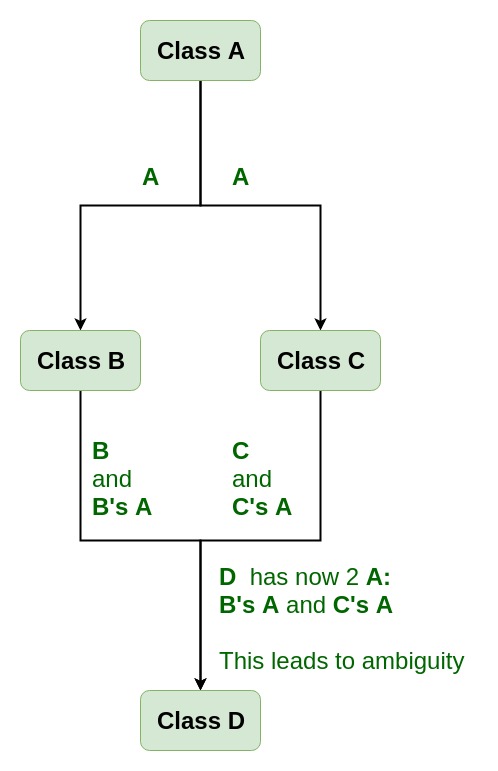
One parent class is not a base class

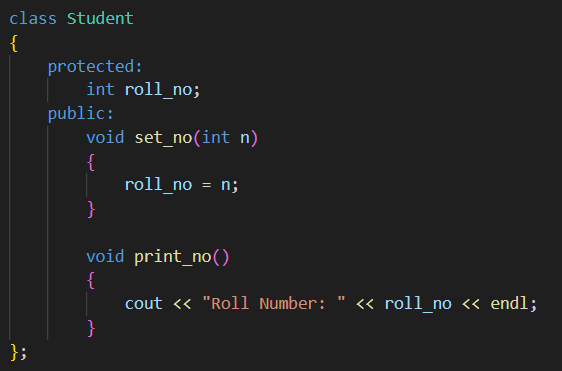


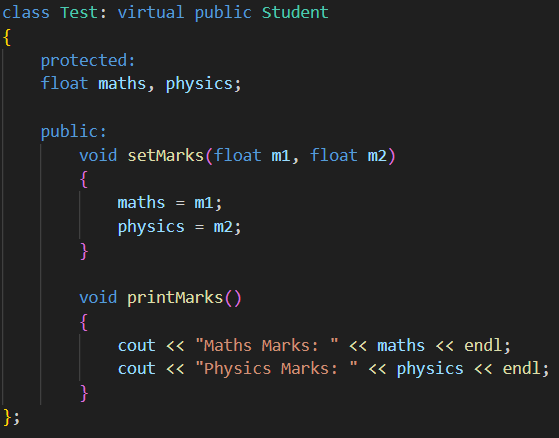
**Ambiguity Resolution:**

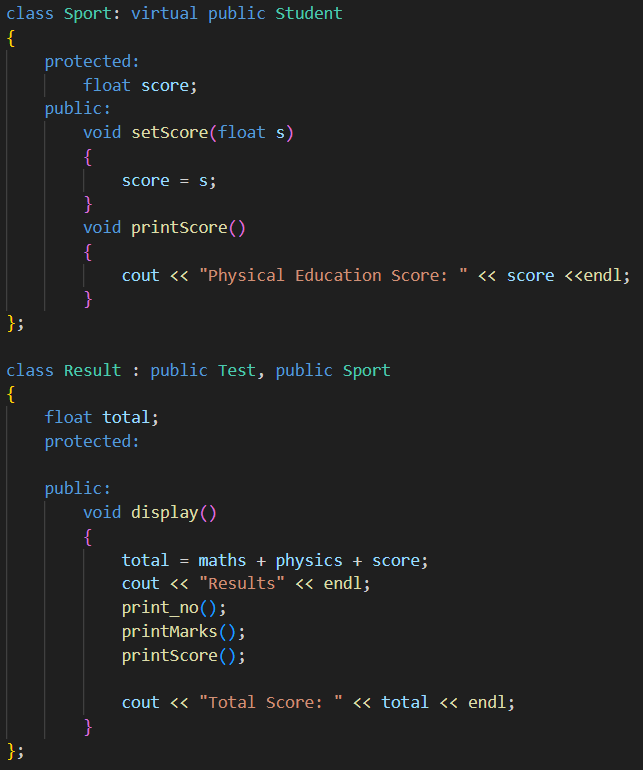
****

**Virtual Base Class:**



****

****

****

**Constructors in Derived Class:**

We can use constructors in derived class

If base class constructor does not have any arguments, then there is no need for any constructor in derived class

If there are one or more arguments in the base class constructor, derived class need to pass arguments to the base class constructor.

If both and derived class have constructor then base class constructor is called first

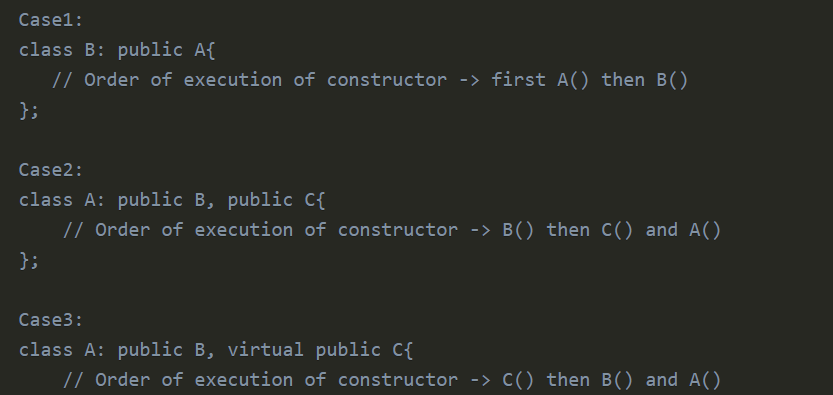
In multiple inheritance, base classes are constructed in the order in which they appear in the class declaration.

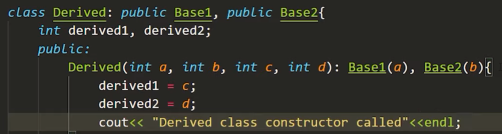
In multilevel inheritance, the constructors are executed in the order of inheritance.

Virtual base classes are invoked before any non virtual base class

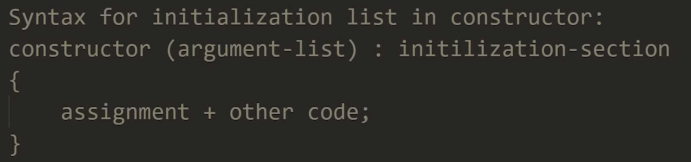
If tehre are multiple virtual base class, they are invoked in the order declared.

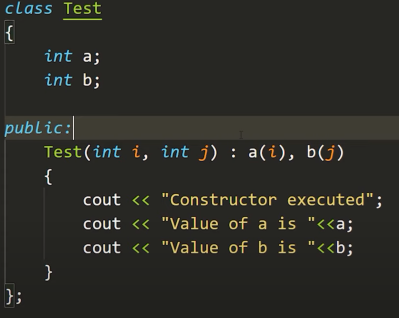
Any non-virtual base class are then executed before the derived class constructor is executed



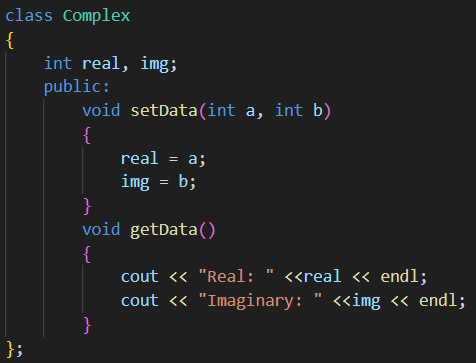


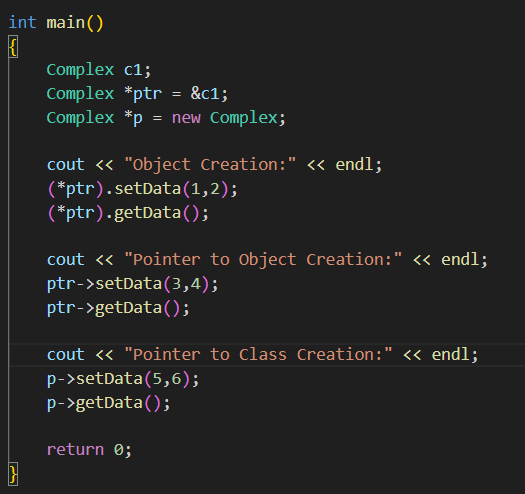
**Initialization List in Constructors:**





**Pointer to Class:**

****

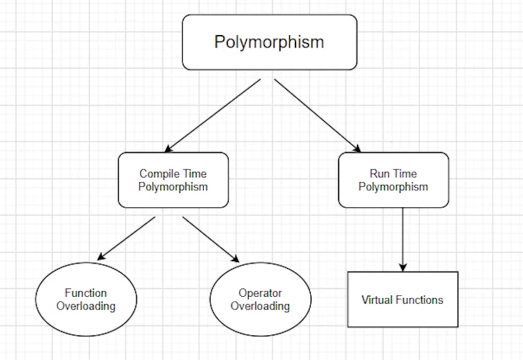
****

**Polymorphism:**

One name with Multiple forms

Function Overloading, Operator Overloading is an example of polymorphism

There are two types of Polymorphism



**Compile Time Polymorphism:**

Compiler Knows which function will be utilized and is known as early binding or static binding

**Run Time Polymorphism:**

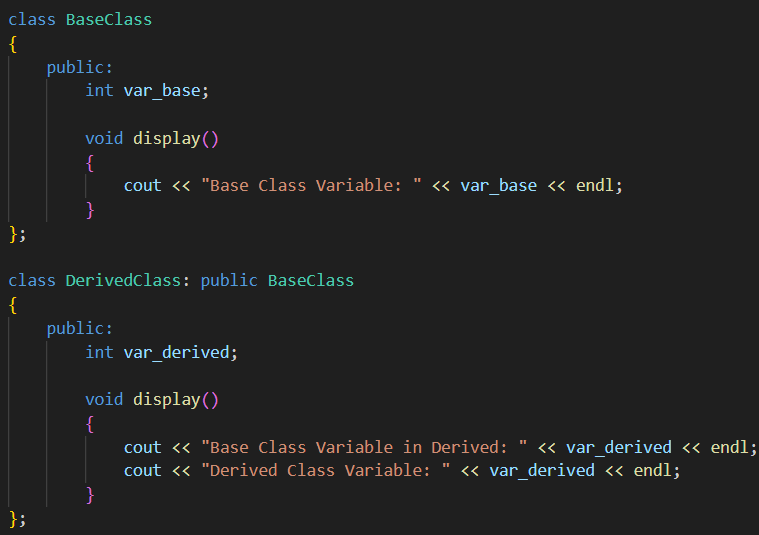
Function which will be utilized will be decided during runtime using the return type of pointers.

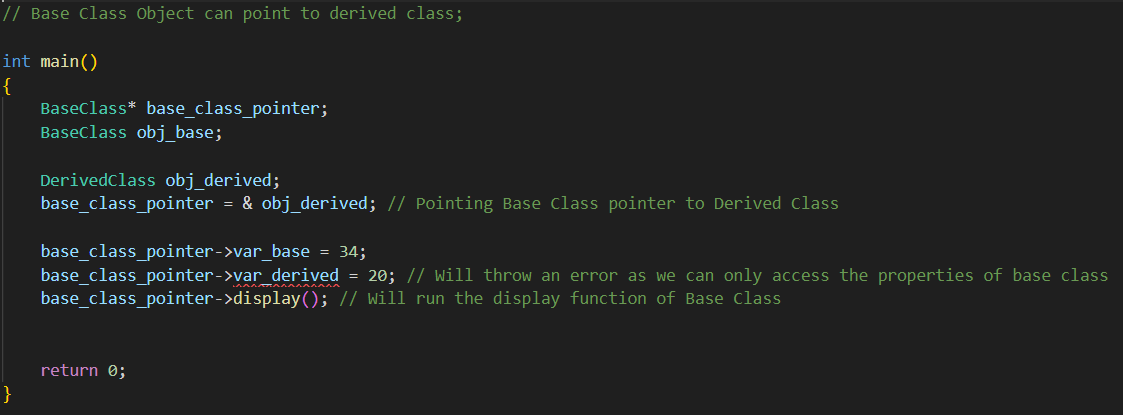
**Pointers to Derived Class:**

Base Class Object can point to derived class

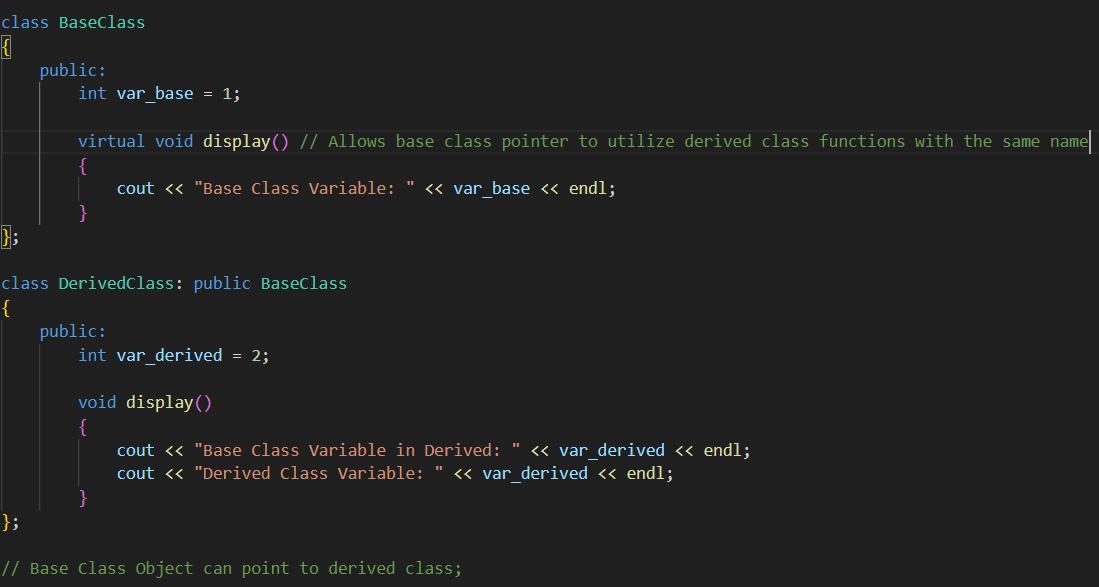
But we cannot access the functions of derived class

Base class object can access the features of only base class even if it is pointed to a derived class





**Virtual Functions:**

****

They cannot be static

They are accessed by object pointers

Virtual Functions can be a friend of another class

A virtual function in base class may not be used

If a virtual function is defined in the base class, then there is no necessity to redefine it in derived class.

**Abstract Base Class and Pure Virtual Functions:**

Abstract Base Class can be defined as a class that contains a virtual function that is required to be redefined in Derived Class.

Abstract Base class objects cannot be defined but Abstract base class pointers can be created.

