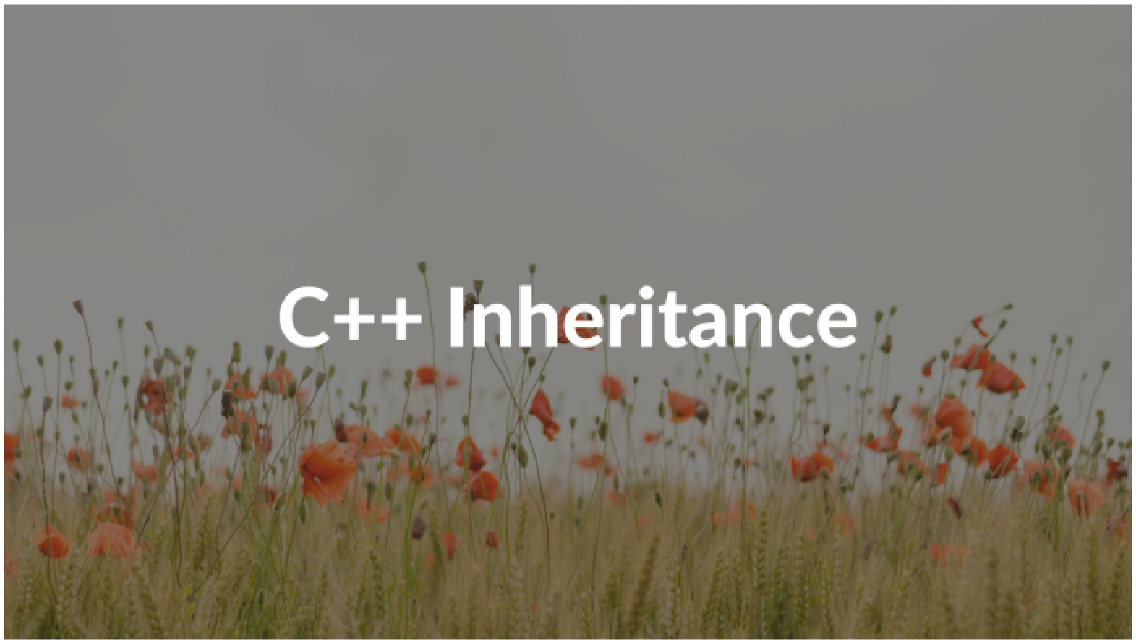
C++ Inheritance

By **Rahul** - August 27, 2019



Definition

Inherit Definition – Derive quality and characteristics from parents or ancestors. Like you inherit features of your parents.

Example: “She had inherited the beauty of her mother”

Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes.

New classes inherit some of the properties and behavior of the existing classes. An existing class that is “parent” of a new class is called a **base class**. New class that inherits properties of the base class is called a **derived class**.

Inheritance is a technique of code reuse. It also provides possibility to extend existing classes by creating derived classes.

Inheritance Syntax

The basic syntax of inheritance is:

```
class DerivedClass : accessSpecifier BaseClass
```

Access specifier can be public, protected and private. The default access specifier is **private.** Access specifiers affect accessibility of data members of base class from the derived class. In addition, it determines the accessibility of data members of base class outside the derived class.

Inheritance Access Specifiers

Public Inheritance

This inheritance mode is used mostly. In this the protected member of Base class becomes protected members of Derived class and public becomes public.

```
class DerivedClass : public BaseClass
```

This is how the members of Base class are defined.Either Public, Protected or Private.T

Accessing Base class members	**public**	**protected**	**private**
From Base class	Yes	Yes	Yes
From object of a Base class	Yes	No	No
From Derived classes	Yes (As Public)	Yes (As Protected)	No
From object of a Derived class	Yes	No	No
From Derived class of Derived Classes	Yes (As Public)	Yes (As Protected)	No

Derived class of Derived Classes: If we are inheriting a derived class using a public inheritance as shown below

class B : public A

class C : public B

then public and protected members of class A will be accessible in class C as public and protected respectively.

Protected Inheritance

In protected mode, the public and protected members of Base class becomes protected members of Derived class.

```
class DerivedClass : protected BaseClass
```

Accessing Base class members	**public**	**protected**	**private**
From Base class	Yes	Yes	Yes
From object of a Base class	Yes	No	No
From Derived classes	Yes (As Protected)	Yes (As Protected)	No
From object of a Derived class	No	No	No
From Derived class of Derived Classes	Yes (As Protected)	Yes (As Protected)	No

Derived class of Derived Classes: If we are inheriting a derived class using a protected inheritance as shown below

class B : protected A

class C : protected B

then public and protected members of class A will be accessible in class C as protected

Private Inheritance

In private mode the public and protected members of Base class become private members of Derived class.

```
class DerivedClass : private BaseClass
```

```
class DerivedClass : BaseClass   // By default inheritance is private
```

Accessing Base class members	public	protected	private
From Base class	Yes	Yes	Yes
From object of a Base class	Yes	No	No
From Derived classes	Yes (As Private)	Yes (As Private)	No
From object of a Derived class	No	No	No
From Derived class of Derived Classes	No	No	No

Derived class of Derived Classes: If we are inheriting a derived class using a private inheritance as shown below

class B : private A

class C : private B

then public and protected members of class A will not be accessible in class C

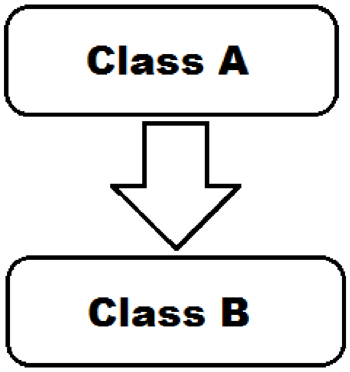
Types of Inheritance

There are different types of inheritance:

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Hybrid (Virtual) Inheritance

Single Inheritance

Single inheritance represents a form of inheritance when there is only one base class and one derived class. For example, a class describes a **Person:**

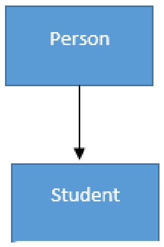


Example of Single Inheritance

```
//base class
class Person
{
public:
        Person(string szName, int iYear)
        {
                m_szLastName = szName;
                m_iYearOfBirth = iYear;
        }
        string m_szLastName;
        int m_iYearOfBirth;
        void print()
        {
                cout << "Last name: " << szLastName << endl;
                cout << "Year of birth: " << iYearOfBirth << endl;
        }
protected:
        string m_szPhoneNumber;
};
```

We want to create new class **Student** which should have the same information as **Person** class plus one new information about university. In this case, we can create a derived class **Student:**

```
//derived class
class Student:public Person
{
public:
        string m_szUniversity;
};
```



Class Student is having access to all the data members of the base class (Person).

Since class Student does not have a constructor so you can create a constructor as below

```
//will call default constructor of base class automatically
Student(string szName, int iYear, string szUniversity)
{
        m_szUniversity = szUniversity;
}
```

If you want to call the parameterized(user defined) constructor of a base class from a derived class then you need to write a parameterized constructor of a derived class as below

```
Student(string szName, int iYear, string szUniversity) :Person(szName, iYear)
{
        m_szUniversity = szUniversity;
}
```

Person(szName, iYear) represents call of a constructor of the base class **Person**. The passing of values to the constructor of a base class is done via member initialization list.

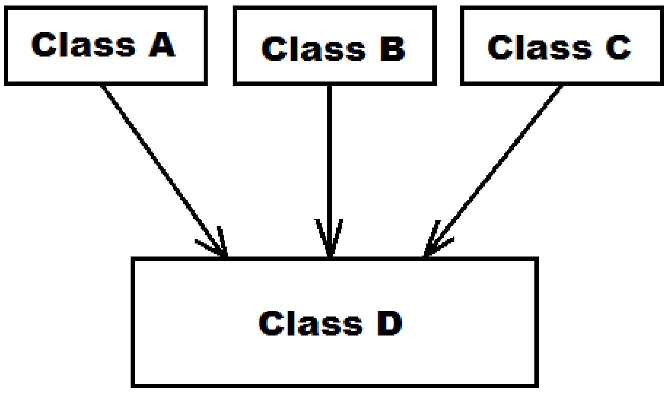
We can access member functions of a base class from a derived class. For example, we can create a new **print()** function in a derived class, that uses **print()** member function of a base class:

```
void print()
{
        //call function print from base class
        Person::print();
        cout << "University " << m_szUniversity << endl;
}
```

If you want to call the member function of the base class then you have to use the name of a base class

Multiple Inheritance

Multiple inheritance represents a kind of inheritance when a derived class inherits properties of **multiple** classes. For example, there are three classes A, B and C and derived class is D as shown below:



If you want to create a class with multiple base classes, you have to use following syntax:

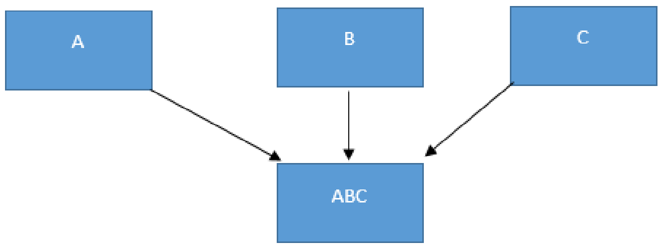
Class DerivedClass: accessSpecifier BaseClass1, BaseClass2, …, BaseClassN

Example of Multiple Inheritance

```
class A
{
        int m_iA;
        A(int iA) :m_iA(iA)
        {
        }
};

class B
{
        int m_iB;
        B(int iB) :m_iB(iB)
        {
        }
};

class C
{
        int m_iC;
        C(int iC) :m_iC(iC)
        {
        }
};
```

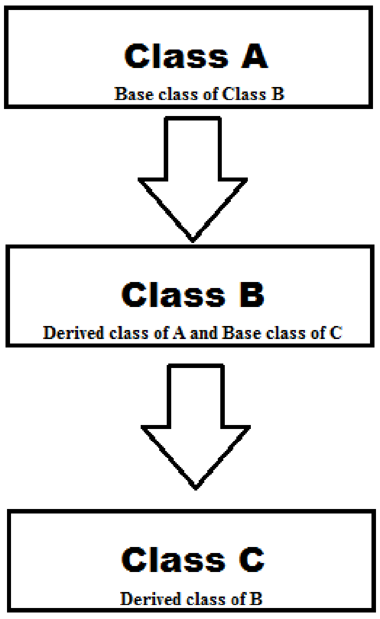
You can create a new class that will inherit all the properties of all these classes:

```
class ABC :public A, public B, public C
{
        int m_iABC;
        //here you can access m_iA, m_iB, m_iC
};
```



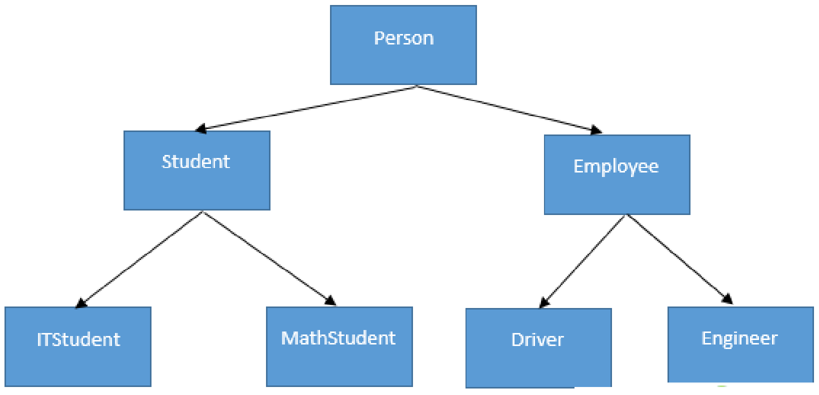
Multilevel Inheritance

Multilevel inheritance represents a type of inheritance when a Derived class is a base class for another class. In other words, deriving a class from a derived class is known as multi-level inheritance. Simple multi-level inheritance is shown in below image where Class A is a parent of Class B and Class B is a parent of Class C



Example of Multi-Level Inheritance

Below Image shows the example of multilevel inheritance



As you can see, Class **Person** is the base class of both **Student** and **Employee** classes. At the same time, Class **Student** is the base class for **ITStudent** and **MathStudent** classes. **Employee** is the base class for **Driver** and **Engineer** classes.

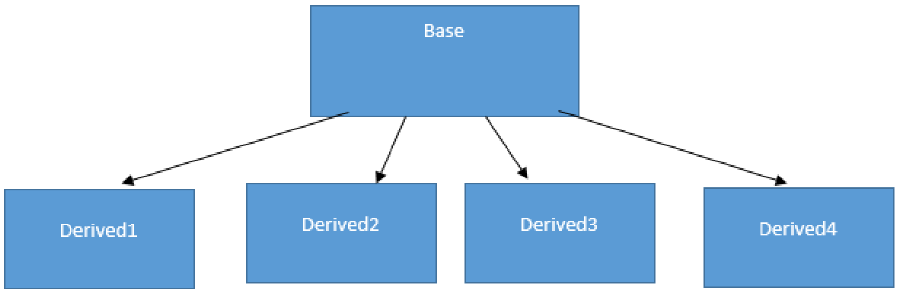
The code for above example of multilevel inheritance will be as shown below

```
class Person
{
        //content of class person
};

class Student :public Person
{
        //content of Student class
};
```

```
class Employee : public Person
{
        //content of Employee class
};

class ITStundet :public Student
{
        //content of ITStudent class
};

class MathStundet :public Student
{
        //content of MathStudent class
};

class Driver :public Employee
{
        //content of class Driver
};

class Engineer :public Employee
{
        //content of class Engineer
};
```

Hierarchical Inheritance

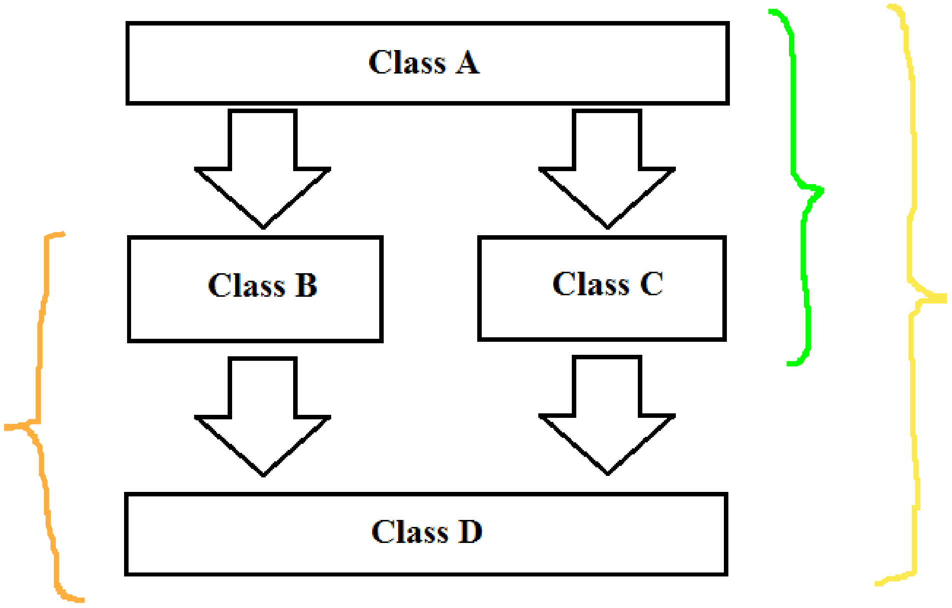
When there is a need to create multiple Derived classes that inherit properties of the same Base class is known as Hierarchical inheritance



```
class base
{
        //content of base class
};

class derived1 :public base
{
        //content of derived1
};

class derived2 :public base
{
        //content of derived
};

class derived3 :public base
{
        //content of derived3
};

class derived4 :public base
{
        //content of derived4
};
```

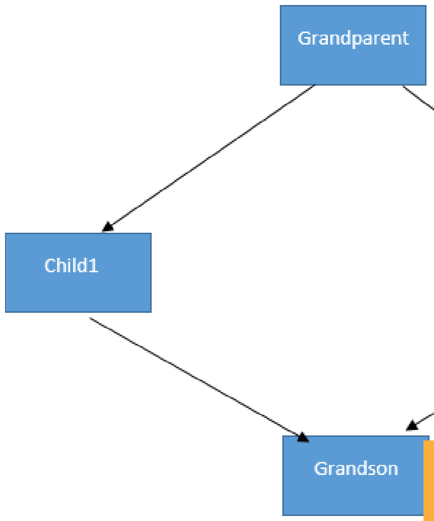
Hybrid Inheritance (also known as Virtual Inheritance)

Combination of Multi-level and Hierarchical inheritance will give you Hybrid inheritance.



Diamond Problem

When you have a hybrid inheritance then a Diamond problem may arise. In this problem a Derived class will have multiple paths to a Base class. This will result in duplicate inherited members of the Base class. This kind of problem is known as **Diamond problem**



- When we use Virtual Inheritance, we are guaranteed to get only a single ir

Virtual Inheritance

We can avoid Diamond problem easily with **Virtual Inheritance**. Child classes in this case should inherit Grandparent class by using virtual inheritance:

Memory Layout in Virtual Inheritance:-- Ir

```
class Grandparent
{
        //content of grandparent class
};

class Child1 :public virtual Grandparent
{
        //content of Child1 class
};

class Child2 :public virtual Grandparent
{
        //content of Child2 class
};

class grandson :public Child1, public Child2
{
        //content of grandson class
};
```

- Because there is only a single instance of a virtual base class(=Grandparent)

Now **grandson** class will have only one copy of data members of the Grandparent class.

Order of Constructor Call

When a default or parameterized constructor of a derived class is called, the default constructor of a base class is called automatically. As you create an object of a derived class, first the default constructor of a base class is called after that constructor of a derived class is called.

To call parameterized constructor of a base class you need to call it explicitly as shown below.

```
Student(string szName, int iYear, string szUniversity) :Person(szName, iYear)
{
}
```

Below program will show the order of execution that the default constructor of base class finishes first after that the constructor of a derived class starts. For example, there are two classes with single inheritance:

```
//base class
class Person
{
public:
        Person()
        {
                cout  << "Default constructor of base class called" << endl;
        }
        Person(string lName, int year)
        {
                cout  << "Parameterized constructor of base class called" << endl;
                lastName = lName;
                yearOfBirth = year;
        }
        string lastName;
        int yearOfBirth;
};

//derived class
class Student :public Person
{
public:
        Student()
        {
                cout  << "Default constructor of Derived class called" << endl;
        }
        Student(string lName, int year, string univer)
        {
                cout  << "Parameterized constructor of Derived class called" << endl;
                university  = univer;
        }
        string university;
};
```

There is no explicit call of constructor of a base class. But on creating two objects of Student class using default and parameterized constructors, both times default constructor of a base class get called.

```
Student student1; //Using default constructor of Student class
Student student2("John", 1990, "London School of  Economics"); //calling parameterize
```

In both the above cases, default constructor of a base class is called before the constructor of a derived class.

```
Default constructor of base class called
Default constructor of Derived class called
Default constructor of base class called
Parameterized constructor of Derived class called
```

When multiple inheritance is used, default constructors of base classes are called in the order as they are in inheritance list. For example, when a constructor of derived class is called:

```
class derived: public class1, public class 2
```

the order of constructors calls will be

```
class1 default constructor
class2 default constructor
derived constructor
```

If you want to call a parameterized constructor of the base class then this can be done using initializer list as shown below.

```
Student(string lName, int year, string univer) :Person(lName, year)
{
        cout << "Parameterized constructor of Derived class works" << endl;
        university  = univer;
}
```

Above code means that you are calling parametrized constructor of the base class and passing two parameters to it. Now the output will be

```
Default constructor of base class works
Default constructor of Derived class works
Parameterized constructor of base class works
Parameterized constructor of Derived class works
```

Now you can see that parameterized constructor of the base class is called from derived class constructor.

Rahul

If you have come this far, it means that you liked what you are reading. I am a software developer (graduated from BITS Pilani). I love writing technical articles on programming and data structures.