OOPC-03-Inheritance

- In Inheritance new classes are created from the existing classes.
- The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”.
- The derived class now is said to be inherited from the base class.
- The derived class inherits all the properties of the base class, without changing the properties of base class and may add new features to its own.
- These new features in the derived class will not affect the base class.
- Sub Class: The class that inherits properties from another class is called Subclass or Derived Class.
- Super Class: The class whose properties are inherited by a subclass is called Base Class or Superclass.

Implementation

```
class<derived_class_name>
<access-specifier>
<base-class-name>
{
    //body
}
```

Modes of Inheritance

- **Public Mode**:
  - If we derive a subclass 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 the derived class.
- **Protected Mode:**
  - If we derive a subclass from a Protected base class.
  - Then both public members and protected members of the base class will become protected in the derived class.
- **Private Mode:**
  - If we derive a subclass from a Private base class.
  - Then both public members and protected members of the base class will become Private in the derived class

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

```
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
};
```

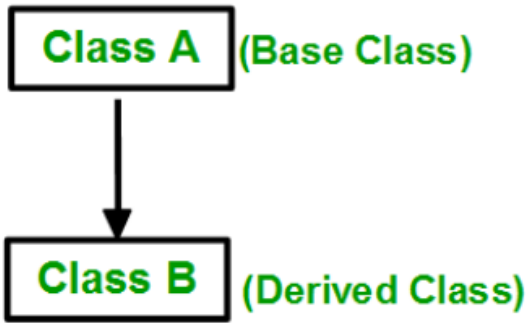
Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

Types of Inheritance

- 5 types:
  - **Single inheritance**
  - **Multilevel inheritance**
  - **Multiple inheritance**
  - **Hierarchical inheritance**
  - **Hybrid inheritance**

Single Inheritance

- In single inheritance, a class is allowed to inherit from only one class.
- I.e. one subclass is inherited by one base class only.

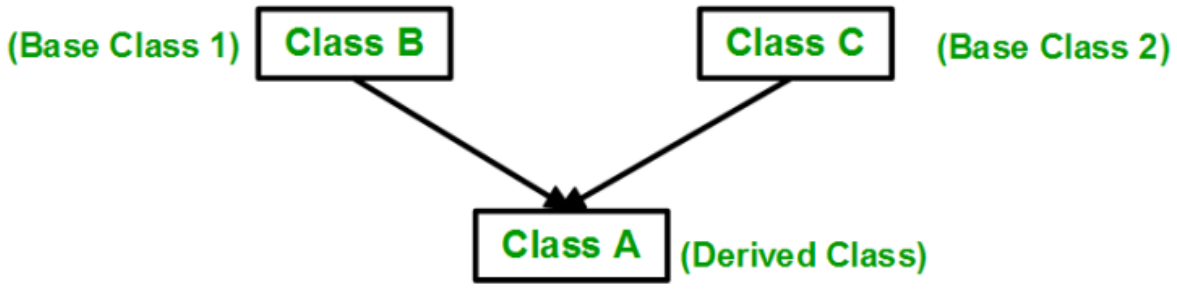



```
#include<iostream>
using namespace std;

//base class
class Vehicle{
    public:
        vehicle(){
        cout<<"This is a Vehicle\n"
    }
};

//sub class derived from a single base classes
class Car:public Vehicle{

};

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

```
This is Vehicle
```

Multiple Inheritance

- Multiple Inheritance is a feature of C++ where a class can inherit from more than one class.
- I.e. one subclass is inherited from more than one base class.



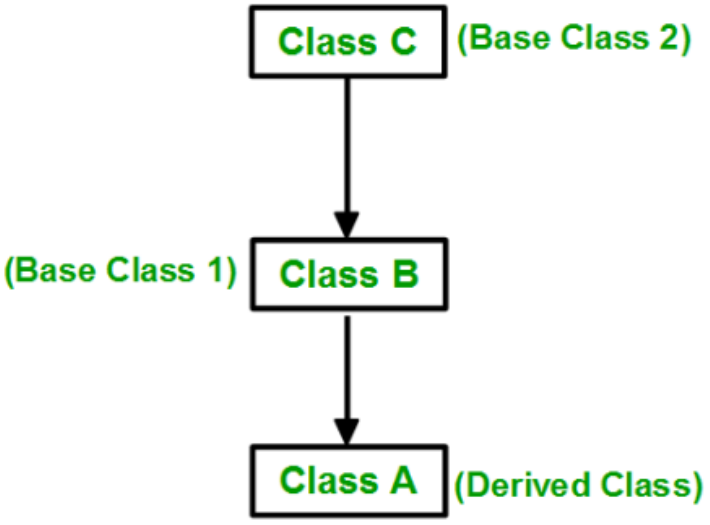

```
#include <iostream>
using namespace std;
```

```cpp
// first base class
class Vehicle {
    public:
        Vehicle() { cout << "This is a Vehicle\n"; }
};

// second base class
class FourWheeler {
    public:
        FourWheeler(){
            cout << "This is a 4 wheeler Vehicle\n";
    }
};

// sub class derived from two base classes
class Car : public Vehicle, public FourWheeler {
};

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

```
This is a Vehicle
This is a 4 Wheeler Vehicle
```

Multilevel Inheritance

- In this type of inheritance, a derived class is created from another derived class.



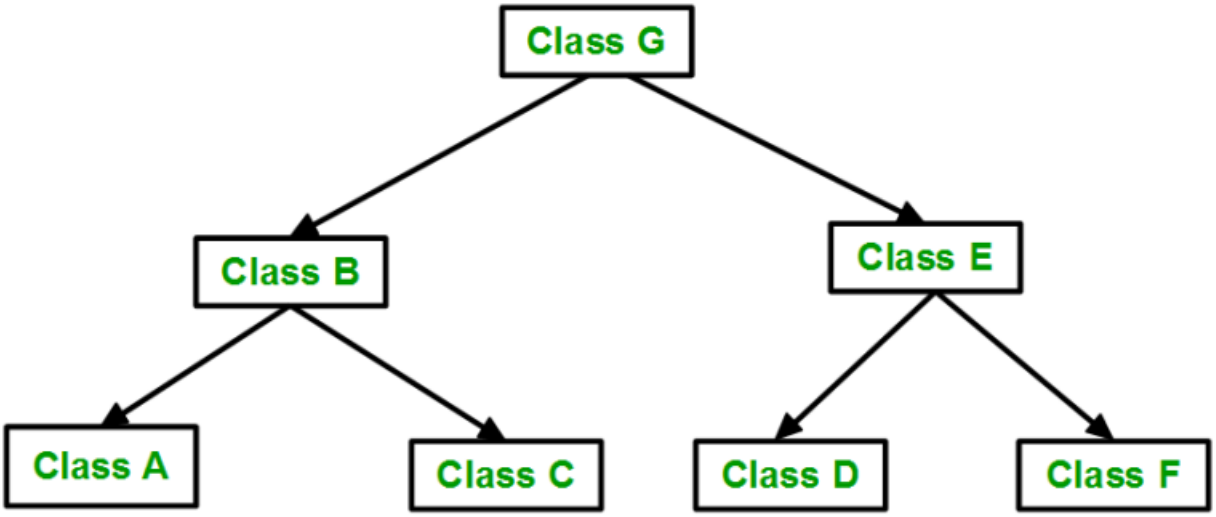

```cpp
#include <iostream>

using namespace std;
// base class
```

```
class Vehicle {
  public: Vehicle() {
    cout << "This is a Vehicle\n";
  }
};
// first sub_class derived from class vehicle
class fourWheeler: public Vehicle {
  public: fourWheeler() {
    cout << "Objects with 4 wheels are vehicles\n";
  }
};
// sub class derived from the derived base class fourWheeler
class Car: public fourWheeler {
  public: Car() {
    cout << "Car has 4 Wheels\n";
  }
};
// main function
int main() {
  // Creating object of sub class will
  // invoke the constructor of base classes.
  Car obj;
  return 0;
}
```

```
This is a Vehicle
Objects with 4 wheels are vehicles
Car has 4 Wheels
```

Hierarchical Inheritance

- In this type of inheritance, more than one subclass is inherited from a single base class.
- I.e. more than one derived class is created from a single base class.



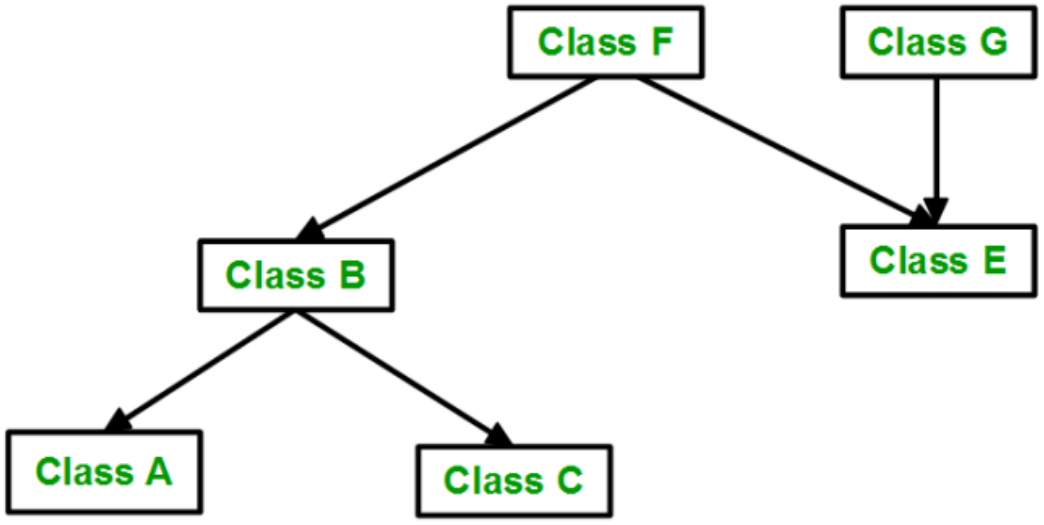

```
#include <iostream>

using namespace std;
// base class
class Vehicle {
```

```
  public: Vehicle() {
    cout << "This is a Vehicle\n";
  }
};
// first sub class
class Car: public Vehicle {};
// second sub class
class Bus: public Vehicle {};
// main function
int main() {
  // Creating object of sub class will
  // invoke the constructor of base class.
  Car obj1;
  Bus obj2;
  return 0;
}
```

```
This is a Vehicle
This is a Vehicle
```

Hybrid (Virtual) Inheritance

- Hybrid Inheritance is implemented by combining more than one type of inheritance.
- For example: Combining Hierarchical inheritance and Multiple Inheritance.




```
#include <iostream>

using namespace std;
// base class
class Vehicle {
  public: Vehicle() {
    cout << "This is a Vehicle\n";
  }
};
// base class
class Fare {
  public: Fare() {
    cout << "Fare of Vehicle\n";
```

```
  }
};
// first sub class
class Car: public Vehicle {};
// second sub class
class Bus: public Vehicle, public Fare {};
// main function
int main() {
  // Creating object of sub class will
  // invoke the constructor of base class.
  Bus obj2;
  return 0;
}
```

```
This is a Vehicle
Fare of Vehicle
```

Anomaly of Inheritance

- In multiple inheritances, when one class is derived from two or more base classes then there may be a possibility that the base classes have functions with the same name, and the derived class may not have functions with that name as those of its base classes.
- If the derived class object needs to access one of the similarly named member functions of the base classes then it results in ambiguity because the compiler gets confused about which base's class member function should be called.

```
#include<iostream>

using namespace std;
// Base class A
class A {
  public: void func() {
    cout << " I am in class A" << endl;
  }
};
// Base class B
class B {
  public: void func() {
    cout << " I am in class B" << endl;
  }
};
// Derived class C
class C: public A, public B {};
// Driver Code
int main() {
  // Created an object of class C
  C obj;
  // Calling function func()
  obj.func();
  return 0;
}
```

- In this example, derived class C inherited the two base classes A and B having the same function name func().
- When the object of class C is created and called the function func() then the compiler gets confused that which base class member function func() should be called.
- To solve this ambiguity scope resolution operator is used denoted by ' :: '

```
ObjectName.ClassName::FunctionName();
```

```
#include<iostream>

using namespace std;
// Base class A
class A {
  public: void func() {
    cout << " I am in class A" << endl;
  }
};
// Base class B
class B {
  public: void func() {
    cout << " I am in class B" << endl;
  }
};
// Derived class C
class C: public A, public B {};
// Driver Code
int main() {
  // Created an object of class C
  C obj;
  // Calling function func() in class A
  obj.A::func();
  // Calling function func() in class B
  obj.B::func();
  return 0;
}
```

```
I am in class A
I am in class B
```

Abstract Class

- Abstract classes act as expressions of general concepts from which more specific classes can be derived.
- You can't create an object of an abstract class type.
- However, you can use pointers and references to abstract class types.
- You create an abstract class by declaring at least one pure virtual member function.
- That's a virtual function declared by using the pure specifier (= 0) syntax.
- Classes derived from the abstract class must implement the pure virtual function or they, too, are abstract classes

```
// deriv_AbstractClasses.cpp
// compile with: /LD
class Account {
  public: Account(double d); // Constructor.
  virtual double GetBalance(); // Obtain balance.
  virtual void PrintBalance() = 0; // Pure virtual function.
  private: double _balance;
};
```

Restriction on Abstract Classes

- Abstract classes can't be used for:
  - Variables or member data
  - Argument types
  - Function return types
  - Types of explicit conversions

Questions

1. Define Inheritance with an example
2. Difference between multiple & multilevel inheritance
3. State various types of inheritance
4. Write a program to implement multilevel inheritance
5. What is hybrid inheritance in C++? Explain with example. (MID-06M)
6. What is an abstract class? Give an example (MID-04M)
7. What is single inheritance in C++? Explain in detail.
8. What is multiple inheritance in C++? Explain in detail. (MID-03M)
9. Explain Multilevel inheritance with example. (MID-06M)

Join Us on Telegram @SOU BCA