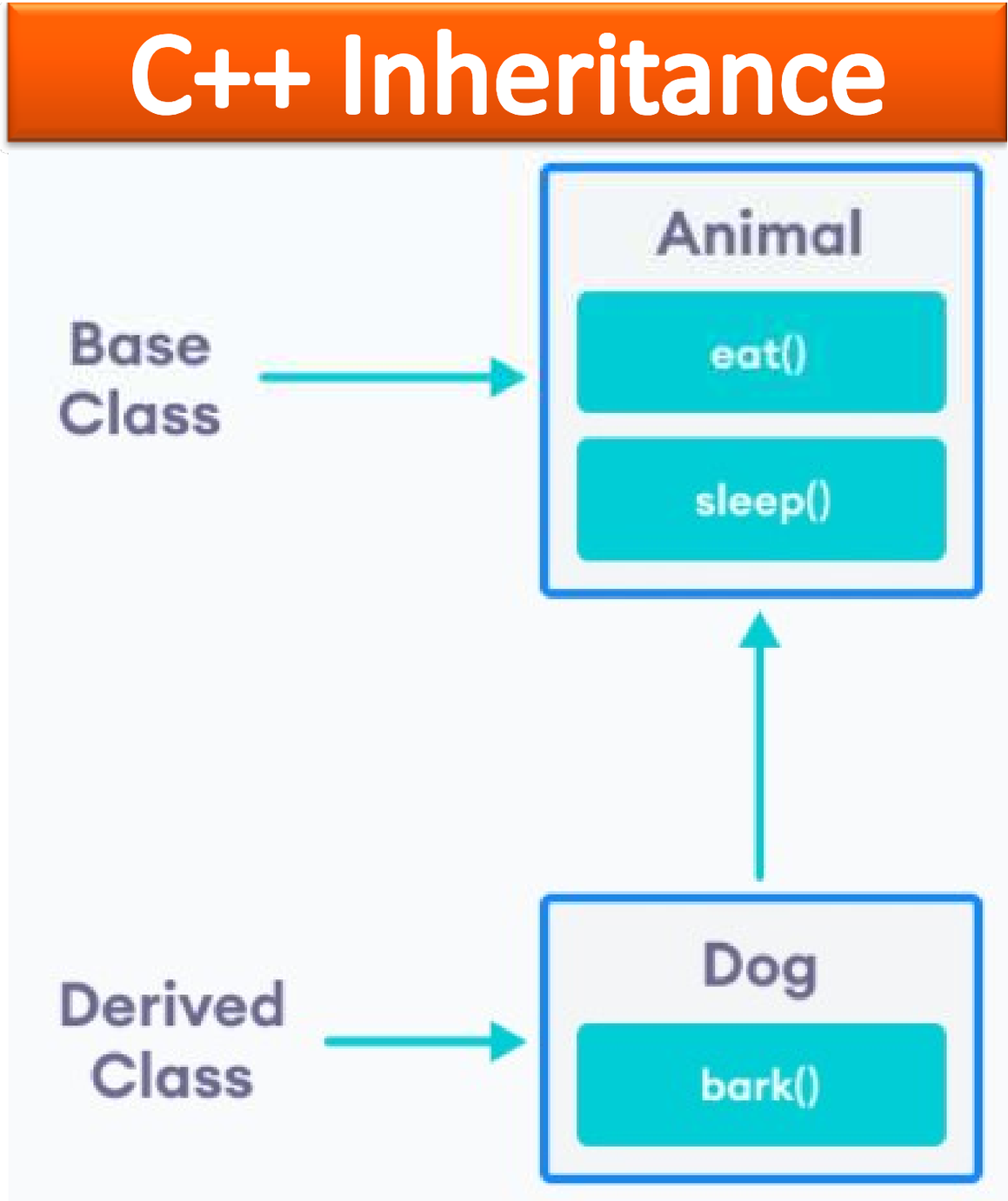
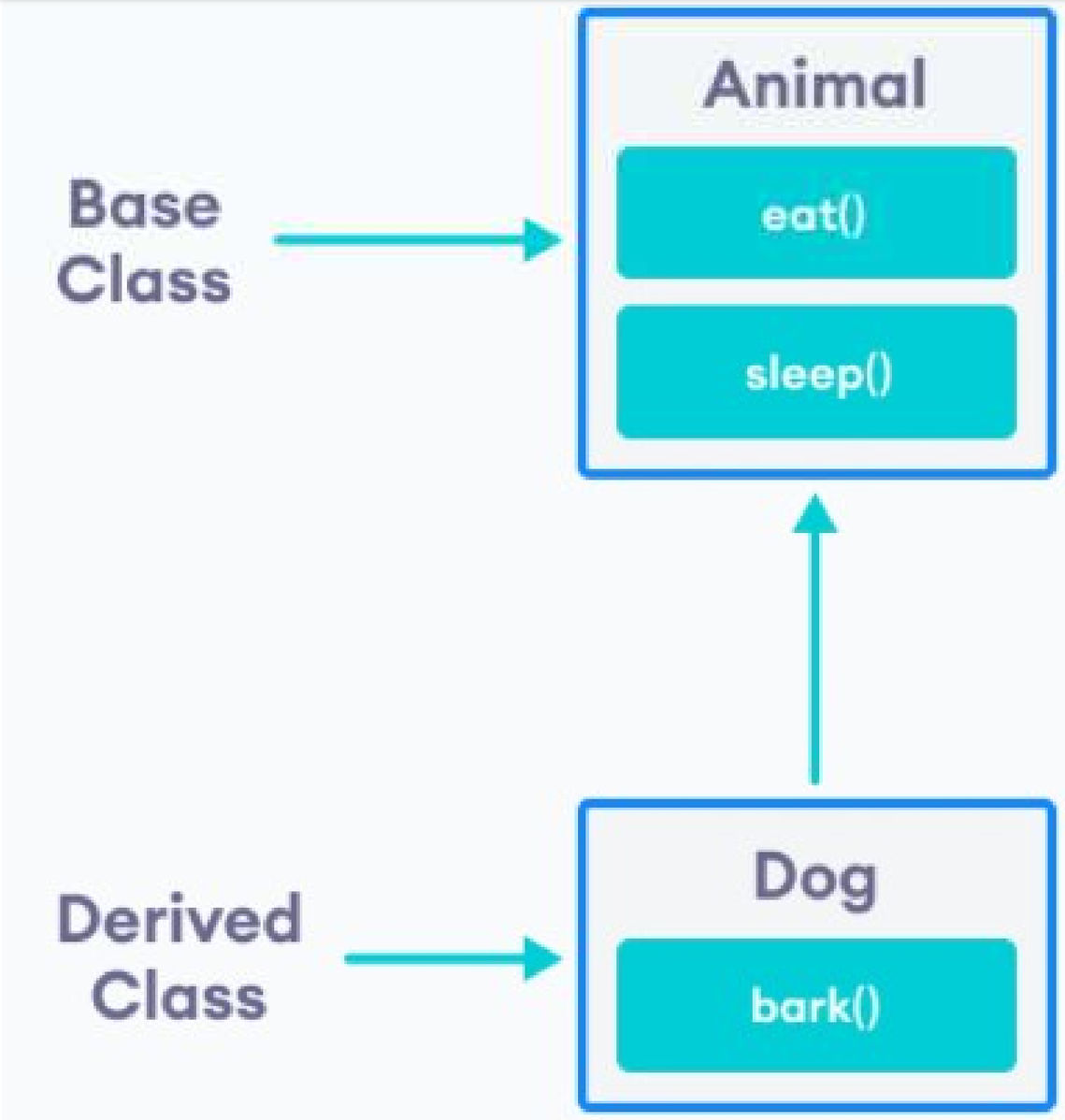
# C++ Inheritance

- The derived class inherits the features from the base class and can have additional features of its own.
- For example,
- Here, the Dog class is derived from the Animal class.
- Since Dog is derived from Animal, members of Animal are accessible to Dog.



# C++ Inheritance

```
class Animal {
    // eat() function
    // sleep() function
};

class Dog : public Animal {
    // bark() function
};
```



# Simple Example of C++ Inheritance

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

// base class
class Animal {

    public:
     void eat() {
         cout << "I can eat!" << endl;
     }

     void sleep() {
         cout << "I can sleep!" << endl;
     }
};

// derived class
class Dog : public Animal {

    public:
     void bark() {
         cout << "I can bark! Woof woof!!" << endl;
     }
};    int main() {
         // Create object of the Dog class
         Dog dog1;

         // Calling members of the base class
         dog1.eat();
         dog1.sleep();

         // Calling member of the derived class
         dog1.bark();

         return 0;
     }
```

```
class Animal  {  // base class
public:
   void eat()
   {
      cout << "I can eat!" << endl;
   }
   void sleep()
    {
      cout << "I can sleep!" << endl;
    }
};
class Dog : public Animal  // derived class
{
   public:
   void bark()
   {
      cout << "I can bark! Woof woof!!" << endl;
   }
};
```

```
int main()
{
   Dog dog1; // Create object of the Dog class
   // Calling members of the base class
   dog1.eat();
   dog1.sleep();
   // Calling member of the derived class
   dog1.bark();
   return 0;
}
```

Output

```
I can eat!
I can sleep!
I can bark! Woof woof!!
```

```
class Animal
{
public:
void eat()
{
cout<<" I can eat"<<endl;
}
void sleep()
{
cout<<" I can sleep"<<endl;
}
};
```

```
class dog : public Animal
{
public:
void bark()
{
cout<<" I can bark! woof woof!"<<endl;
}
};

class cat : public Animal
{
public:
void sound()
{
cout<<" I sound like meow meow!"<<endl;
} };
```

```
void  main()
{
dog d;  cat c;
cout<<“I am a Dog”<<endl;
d.eat();
d.sleep();
d.bark();
cout<<“I am a Cat”<<endl;
c.eat();
c.sleep();
c.sound();
}
```

# C++ Multiple, Multilevel and Hierarchical Inheritance

# C++ Inheritance

- **It allows software developers to derive a new class from the existing class.**
- **The derived class inherits the features of the base class (existing class).**
- **There are various models of inheritance in C++ programming.**

# C++ Multilevel Inheritance

- In C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as multilevel inheritance.

```
class A {
... .. ...
};
class B: public A {
... .. ...
};
class C: public B {
... ... ...
};
```

- Here, class B is derived from the base class A and the class C is derived from the derived class B.

# C++ Multilevel Inheritance

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

class A {
    public:
      void display() {
          cout<<"Base class content.";
      }
};

class B : public A {};

class C : public B {};

int main() {
    C obj;
    obj.display();
    return 0;
}
```

## Output

```
Base class content.
```

- In this program, class C is derived from class B (which is derived from base class A).
- The obj object of class C is defined in the main() function.
- When the display() function is called, display() in class A is executed. It's because there is no display() function in class C and class B.
- The compiler first looks for the display() function in class C. Since the function doesn't exist there, it looks for the function in class B (as C is derived from B).
- The function also doesn't exist in class B, so the compiler looks for it in class A (as B is derived from A).
- If display() function exists in C, the compiler overrides display() of class A (because of member function overriding).
-

# C++ Multiple Inheritance

- In C++ programming, a class can be derived from more than one parent.
- For example, A class Bat is derived from base classes Mammal and Winged Animal.
- It makes sense because bat is a mammal as well as a winged animal.

# Inheritance in C++ Programming

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

class Mammal {
  public:
    Mammal() {
      cout << "Mammals can give direct birth." << endl;
    }
};

class WingedAnimal {
  public:
    WingedAnimal() {
      cout << "Winged animal can flap." << endl;
    }
};

class Bat: public Mammal, public WingedAnimal {};

int main() {
    Bat b1;
    return 0;
}
```

Output

```
Mammals can give direct birth.
Winged animal can flap.
```

- **Ambiguity in Multiple Inheritance**
- The most obvious problem with multiple inheritance occurs during function overriding.
- Suppose, two base classes have a same function which is not overridden in derived class.
- If you try to call the function using the object of the derived class, compiler shows error. It's because compiler doesn't know which function to call. For example,

```
class base1 {
  public:
      void someFunction( ) {....}
};
class base2 {
    void someFunction( ) {....}
};
class derived : public base1, public base2 {};

int main() {
    derived obj;
    obj.someFunction() // Error!
}
```

# C++ Hierarchical Inheritance

- If more than one class is inherited from the base class, it's known as hierarchical inheritance.
- In hierarchical inheritance, all features that are common in child classes are included in the base class.
- For example, Physics, Chemistry, Biology are derived from Science class.
- Similarly, Dog, Cat, Horse are derived from Animal class.

# Syntax of Hierarchical Inheritance

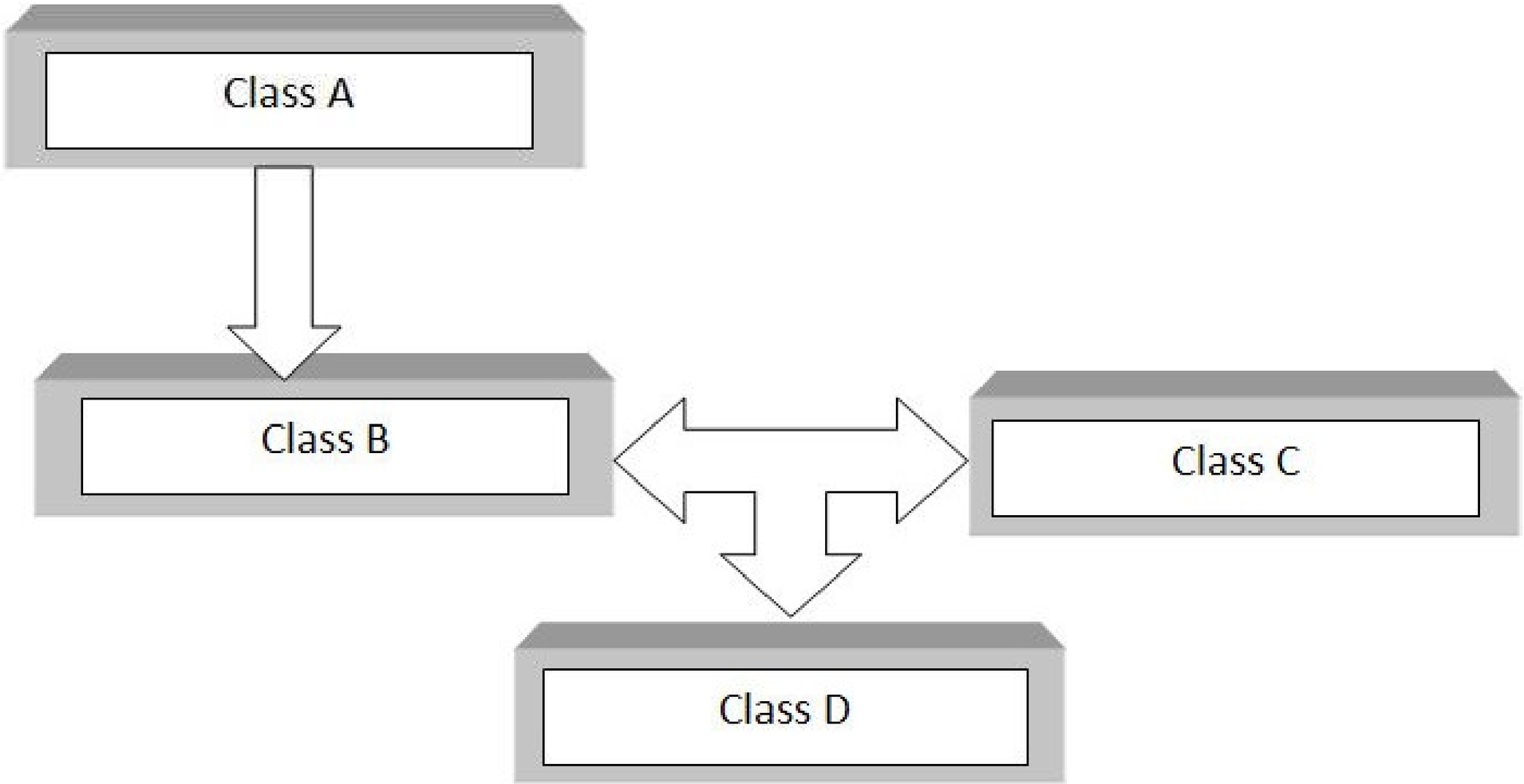
```
class base_class {
      ... .. ...
}

class first_derived_class: public base_class {
      ... .. ...
}

class second_derived_class: public base_class {
      ... .. ...
}

class third_derived_class: public base_class {
      ... .. ...
}
```

```cpp
// C++ program to demonstrate hierarchical inheritance

#include <iostream>
using namespace std;

// base class
class Animal {
   public:
    void info() {
        cout << "I am an animal." << endl;
    }
};

// derived class 1
class Dog : public Animal {
   public:
    void bark() {
        cout << "I am a Dog. Woof woof." << endl;
    }
};

// derived class 2
class Cat : public Animal {
   public:
    void meow() {
        cout << "I am a Cat. Meow." << endl;
    }
};

int main() {
    // Create object of Dog class
    Dog dog1;
    cout << "Dog Class:" << endl;
    dog1.info();  // Parent Class function
    dog1.bark();

    // Create object of Cat class
    Cat cat1;
    cout << "\nCat Class:" << endl;
    cat1.info();  // Parent Class function
    cat1.meow();

    return 0;
}
```

Output

```
Dog Class:
I am an animal.
I am a Dog. Woof woof.

Cat Class:
I am an animal.
I am a Cat. Meow.
```

# HYBRID INHERITANCE



# C++ Hybrid Inheritance Syntax

```
class A
{
      .........
};
class B : public A
{
      ..........
} ;
class C
{
      ..........
};
 class D : public B, public C
{
      ..........
};
```

```
class vehicle
{
public:
vehicle()
        {
        cout<< "This is a vehicle\n";
        }
};
class Car: public vehicle
{
public:
Car()
   {
cout<< "This is a car\n";
   }
};
```

```
class Racing
{
public:
Racing()
   {
cout<< "This is for Racing\n";
   }
};
class Ferrari: public Car, public Racing
{
public:
Ferrari()
   {
cout<< "Ferrari is a Racing Car\n";
   }

};
void main()
        {
        Ferrari f;
        }
```

# C++ Multiple, Multilevel and Hierarchical Inheritance

# C++ Inheritance

- **It allows software developers to derive a new class from the existing class.**
- **The derived class inherits the features of the base class (existing class).**
- **There are various models of inheritance in C++ programming.**

# C++ Multilevel Inheritance

- In C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as multilevel inheritance.

```
class A {
... .. ...
};
class B: public A {
... .. ...
};
class C: public B {
... ... ...
};
```

- Here, class B is derived from the base class A and the class C is derived from the derived class B.

# C++ Multilevel Inheritance

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

class A {
    public:
      void display() {
          cout<<"Base class content.";
      }
};

class B : public A {};

class C : public B {};

int main() {
    C obj;
    obj.display();
    return 0;
}
```

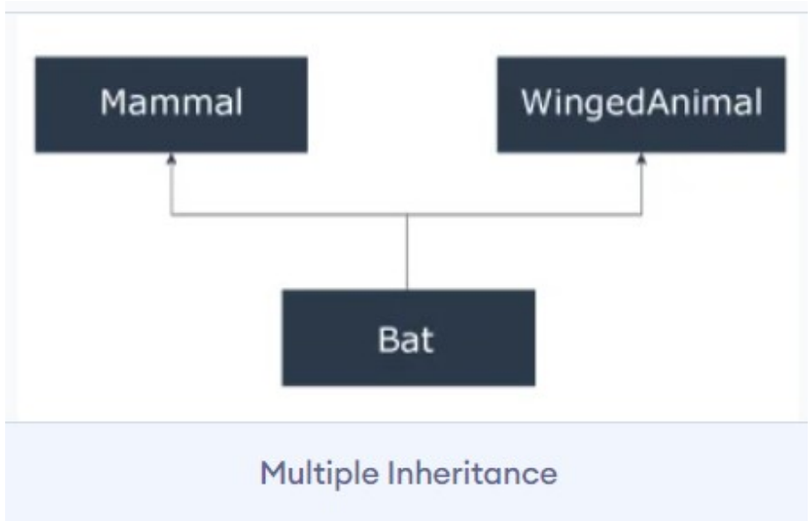
**Output**

```
Base class content.
```

- In this program, class C is derived from class B (which is derived from base class A).
- The obj object of class C is defined in the main() function.
- When the display() function is called, display() in class A is executed. It's because there is no display() function in class C and class B.
- The compiler first looks for the display() function in class C. Since the function doesn't exist there, it looks for the function in class B (as C is derived from B).
- The function also doesn't exist in class B, so the compiler looks for it in class A (as B is derived from A).
- If display() function exists in C, the compiler overrides display() of class A (because of member function overriding).
-

# C++ Multiple Inheritance

- In C++ programming, a class can be derived from more than one parent. For example, A class Bat is derived from base
classes Mammal and WingedAnimal. It makes sense because bat is a mammal as well as a winged animal.
-



# Example 2: Multiple Inheritance in C++ Programming

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

class Mammal {
  public:
    Mammal() {
      cout << "Mammals can give direct birth." << endl;
    }
};

class WingedAnimal {
  public:
    WingedAnimal() {
      cout << "Winged animal can flap." << endl;
    }
};

class Bat: public Mammal, public WingedAnimal {};

int main() {
    Bat b1;
    return 0;
}
```

**Output**

```
Mammals can give direct birth.
Winged animal can flap.
```

- **Ambiguity in Multiple Inheritance**
- The most obvious problem with multiple inheritance occurs during function overriding.
- Suppose, two base classes have a same function which is not overridden in derived class.
- If you try to call the function using the object of the derived class, compiler shows error. It's because compiler doesn't know which function to call. For example,

```
class base1 {
  public:
      void someFunction( ) {....}
};
class base2 {
    void someFunction( ) {....}
};
class derived : public base1, public base2 {};

int main() {
    derived obj;
    obj.someFunction() // Error!
}
```

- This problem can be solved using the scope resolution function to specify which function

```
int main() {
    obj.base1::someFunction( );  // Function of base1 class is called
    obj.base2::someFunction();   // Function of base2 class is called.
}
```

# C++ Hierarchical Inheritance

- If more than one class is inherited from the base class, it's known as hierarchical inheritance. In hierarchical inheritance, all features that are common in child classes are included in the base class.
- For example, Physics, Chemistry, Biology are derived from Science class. Similarly, Dog, Cat, Horse are derived from Animal class.

# Syntax of Hierarchical Inheritance

```
class base_class {
    ... .. ...
}

class first_derived_class: public base_class {
    ... .. ...
}

class second_derived_class: public base_class {
    ... .. ...
}

class third_derived_class: public base_class {
    ... .. ...
}
```

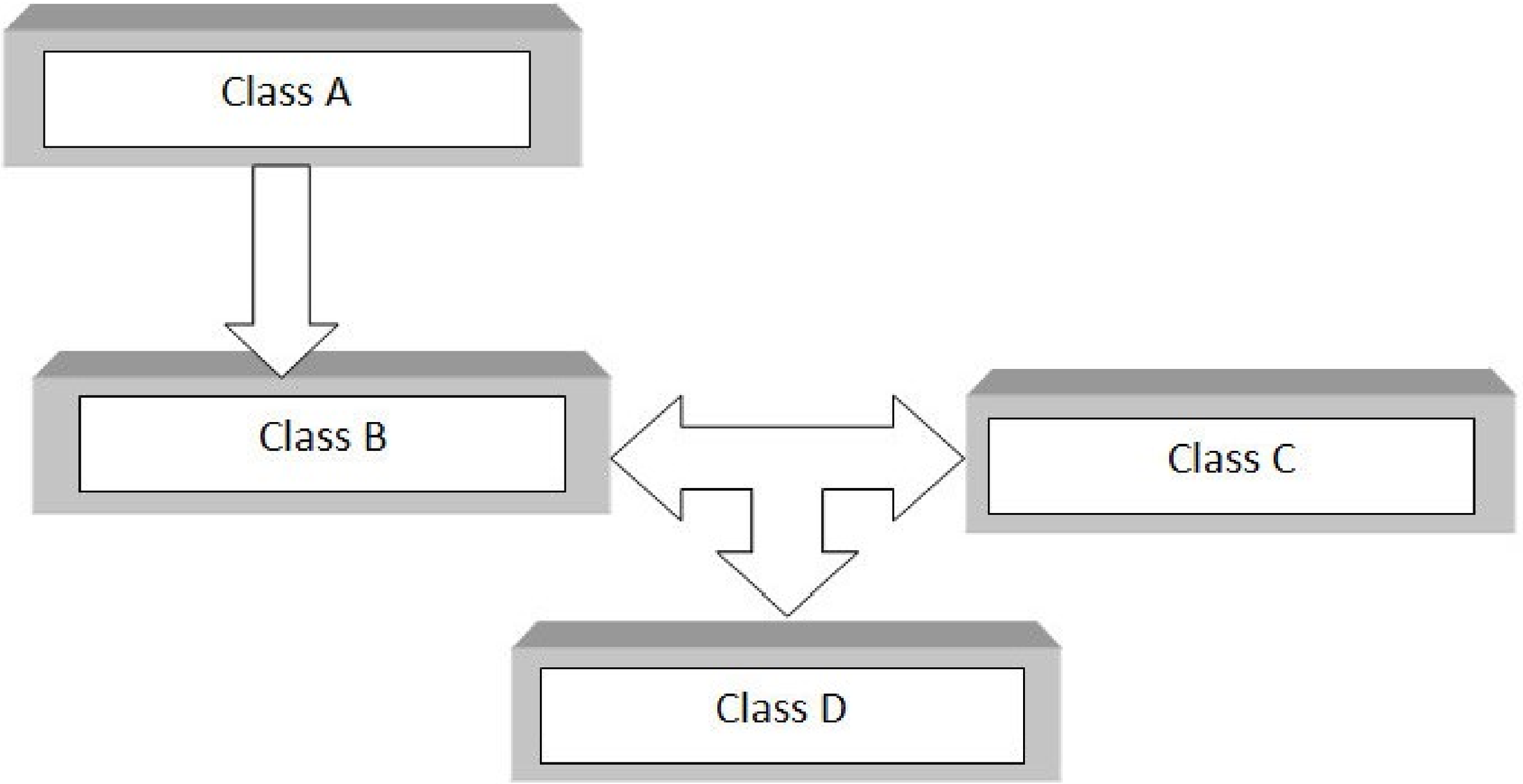
```cpp
// C++ program to demonstrate hierarchical inheritance

#include <iostream>
using namespace std;

// base class
class Animal {
   public:
    void info() {
        cout << "I am an animal." << endl;
    }
};

// derived class 1
class Dog : public Animal {
   public:
    void bark() {
        cout << "I am a Dog. Woof woof." << endl;
    }
};

// derived class 2
class Cat : public Animal {
   public:
    void meow() {
        cout << "I am a Cat. Meow." << endl;
    }
};

int main() {
    // Create object of Dog class
    Dog dog1;
    cout << "Dog Class:" << endl;
    dog1.info();  // Parent Class function
    dog1.bark();

    // Create object of Cat class
    Cat cat1;
    cout << "\nCat Class:" << endl;
    cat1.info();  // Parent Class function
    cat1.meow();

    return 0;
}
```

**Output**

```
Dog Class:
I am an animal.
I am a Dog. Woof woof.

Cat Class:
I am an animal.
I am a Cat. Meow.
```

# HYBRID INHERITANCE

- Here, both the Dog and Cat classes are derived from the Animal class. As such, both the derived classes can access the info() function belonging to the Animal class.


Class A
Class B
Class C
Class D

# C++ Hybrid Inheritance Syntax

```
class A
{
      .........
};
class B : public A
{
      ..........
} ;
class C
{
      ..........
};
 class D : public B, public C
{
      ..........
};
```

```
#include<iostream.h>
#include<conio.h>
class A
{
public:
int x;
};

class B: public A
{
public:
B()
{
x =344;
}
};

class C
{
public:
int y;
C()
{
y = 356;
}  };

class D: public B, public C
{
public:
void sum()
{
cout<<"SUM = sparrow =" << x +y;
} };

int main()
{
clrscr();
D sparrow1;
sparrow1.sum();
return 0;
}
```

# C++ Public, Protected and Private Inheritance

# program to demonstrate public access modifier

```
#include <iostream>


class Circle
{
public:
    double radius;
    double area()
    {
    return 3.14 * radius * radius;
    }
};
```

```
// accessing public data member outside class
int main()
{
Circle obj;
obj.radius = 5.5;
cout << "Radius is: " << obj.radius << "\n";
cout << "Area is: " << obj. area();
return 0;
}
```

# program to demonstrate public access modifier

```
#include<iostream.h>
#include<conio.h>
class parent
{
protected:
int a;
};
class child : public parent
{
public:
void setv(int b)
{
        a=b;
}

void displayv()
{
cout<<"Protected Value of a is:"<<a<<endl;
}
};
void main()
{
clrscr();
child c1;
c1.setv(50);
c1.displayv();
getch();
}
```

# Public, protected and private inheritance

- Public, protected, and private inheritance have the following features:
- Public inheritance makes public members of the base class public in the derived class, and the protected members of the base class remain protected in the derived class.
- Protected inheritance makes the public and protected members of the base class protected in the derived class.
- Private inheritance makes the public and protected members of the base class private in the derived class.
- Note: private members of the base class are inaccessible to the derived class.

```
class Base {
    public:
        int x;
    protected:
        int y;
    private:
        int z;
};

class PublicDerived: public Base {
    // x is public
    // y is protected
    // z is not accessible from PublicDerived
};
```

```
class ProtectedDerived: protected Base {
    // x is protected
    // y is protected
    // z is not accessible from ProtectedDerived
};

class PrivateDerived: private Base {
    // x is private
    // y is private
    // z is not accessible from PrivateDerived
}
```

```cpp
// C++ program to demonstrate the working of public inheritance

#include <iostream>
using namespace std;

class Base {
   private:
    int pvt = 1;

   protected:
    int prot = 2;

   public:
    int pub = 3;

    // function to access private member
    int getPVT() {
        return pvt;
    }
};

class PublicDerived : public Base {
   public:
    // function to access protected member from Base
    int getProt() {
        return prot;
    }
};

int main() {
    PublicDerived object1;
    cout << "Private = " << object1.getPVT(
    cout << "Protected = " << object1.getPr
    cout << "Public = " << object1.pub << e
    return 0;
}
```

Output

```
Private = 1
Protected = 2
Public = 3
```

- Here, we have
  derived PublicDerived from Base in public mode.
- As a result, in PublicDerived:
- prot is inherited as protected.
- pub and getPVT() are inherited as public.
- pvt is inaccessible since it is private in Base.
- Since private and protected members are not accessible from main(), we need to create public functions getPVT() and getProt() to access them:
-

## Accessibility in public Inheritance

| Accessibility | private members | protected members | public members |
|---|---|---|---|
| Base Class | Yes | Yes | Yes |
| Derived Class | No | Yes | Yes |

```
// Error: member "Base::pvt" is inaccessible
cout << "Private = " << object1.pvt;

// Error: member "Base::prot" is inaccessible
cout << "Protected = " << object1.prot;
```

- Notice that the getPVT() function has been defined inside Base. But the getProt() function has been defined inside PublicDerived.
- This is because pvt, which is private in Base, is inaccessible to PublicDerived.
- However, prot is accessible to PublicDerived due to public inheritance. So, getProt() can access the protected variable from within PublicDerived.
-

## Accessibility in public Inheritance

| Accessibility | private members | protected members | public members |
|---|---|---|---|
| Base Class | Yes | Yes | Yes |
| Derived Class | No | Yes | Yes |

```cpp
// C++ program to demonstrate the working of protected inheritance

#include <iostream>
using namespace std;

class Base {
   private:
    int pvt = 1;

   protected:
    int prot = 2;

   public:
    int pub = 3;

    // function to access private member
    int getPVT() {
        return pvt;
    }
};

class ProtectedDerived : protected Base {
   public:
    // function to access protected member from Base
    int getProt() {
        return prot;
    }

    // function to access public member from Base
    int getPub() {
        return pub;
    }
};

int main() {
    ProtectedDerived object1;
    cout << "Private cannot be accessed." << endl;
    cout << "Protected = " << ob
    cout << "Public = " << objec
    return 0;
}
```

Output

```
Private cannot be accessed.
Protected = 2
Public = 3
```

- Here, we have derived ProtectedDerived from Base in protected mode.
- As a result, in ProtectedDerived:
- prot, pub and getPVT() are inherited as protected.
- pvt is inaccessible since it is private in Base.
- As we know, protected members cannot be directly accessed from outside the class. As a result, we cannot use getPVT() from ProtectedDerived.
- That is also why we need to create the getPub() function in ProtectedDerived in order to access the pub variable.
-

```
// Error: member "Base::getPVT()" is inaccessible
cout << "Private = " << object1.getPVT();

// Error: member "Base::pub" is inaccessible
cout << "Public = " << object1.pub;
```

## Accessibility in protected Inheritance

| Accessibility | private members | protected members | public members |
|---|---|---|---|
| Base Class | Yes | Yes | Yes |
| Derived Class | No | Yes | Yes (inherited as protected variables) |

```cpp
// C++ program to demonstrate the working of private inheritance

#include <iostream>
using namespace std;

class Base {
   private:
    int pvt = 1;

   protected:
    int prot = 2;

   public:
    int pub = 3;

    // function to access private member
    int getPVT() {
        return pvt;
    }
};
```

```
class PrivateDerived : private Base {
   public:
    // function to access protected member from Base
    int getProt() {
        return prot;
    }

    // function to access private member
    int getPub() {
        return pub;
    }
};

int main() {
    PrivateDerived object1;
    cout << "Private cannot be accessed." << endl;
    cout << "Protected = " << object1.getProt() << endl;
    cout << "Public = " << object1.getPub() << endl;
    return 0;
}
```

**Output**

```
Private cannot be accessed.
Protected = 2
Public = 3
```

- Here, we have derived PrivateDerived from Base in private mode.
- As a result, in PrivateDerived:
- prot, pub and getPVT() are inherited as private.
- pvt is inaccessible since it is private in Base.
- As we know, private members cannot be directly accessed from outside the class. As a result, we cannot use getPVT() from PrivateDerived.
- That is also why we need to create the getPub() function in PrivateDerived in order to access the pub variable.
-

```
// Error: member "Base::getPVT()" is inaccessible
cout << "Private = " << object1.getPVT();

// Error: member "Base::pub" is inaccessible
cout << "Public = " << object1.pub;
```

## Accessibility in private Inheritance

| Accessibility | private members | protected members | public members |
|---|---|---|---|
| Base Class | Yes | Yes | Yes |
| Derived Class | No | Yes (inherited as private variables) | Yes (inherited as private variables) |