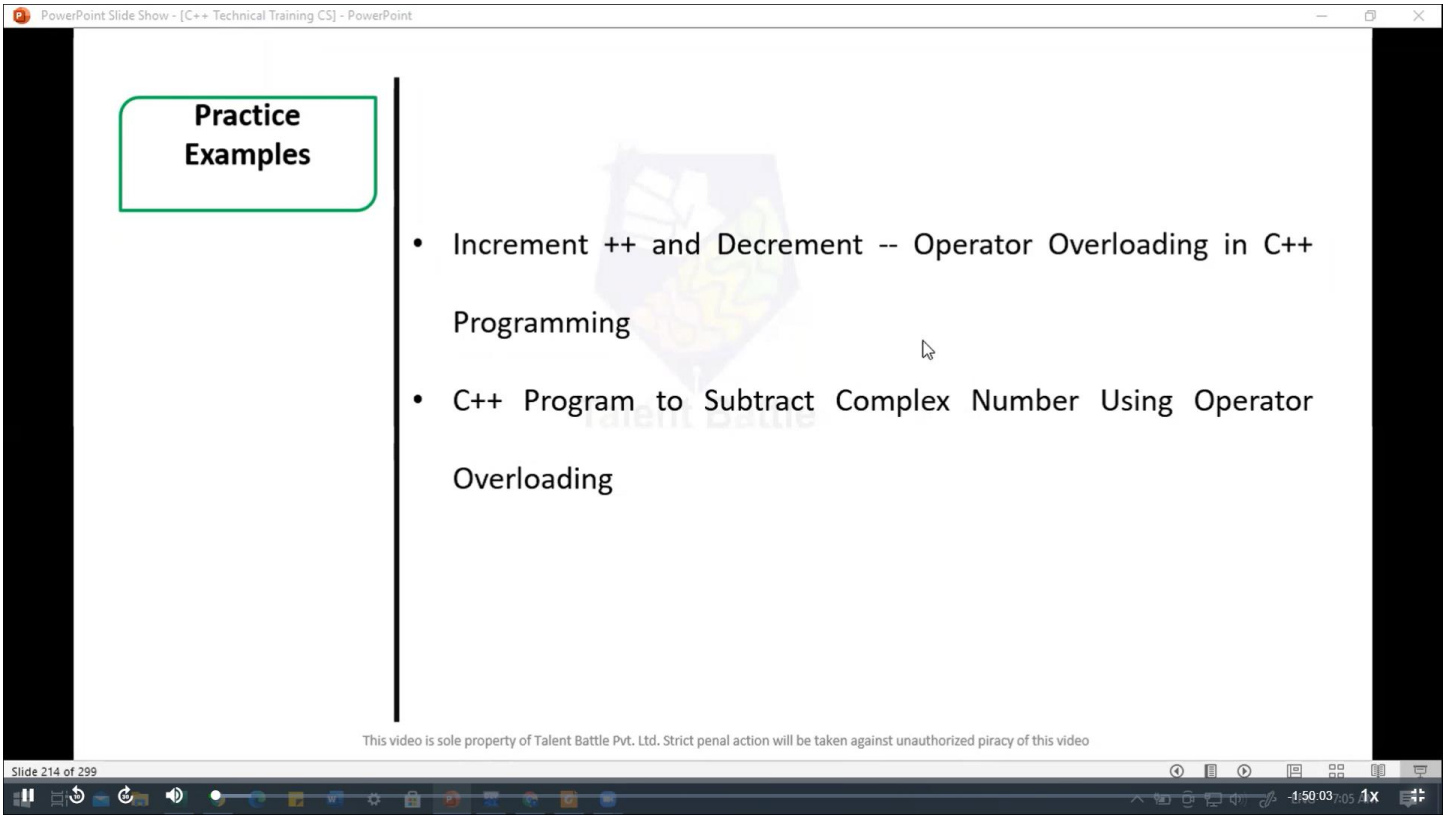
Day 7

C++ Inheritance



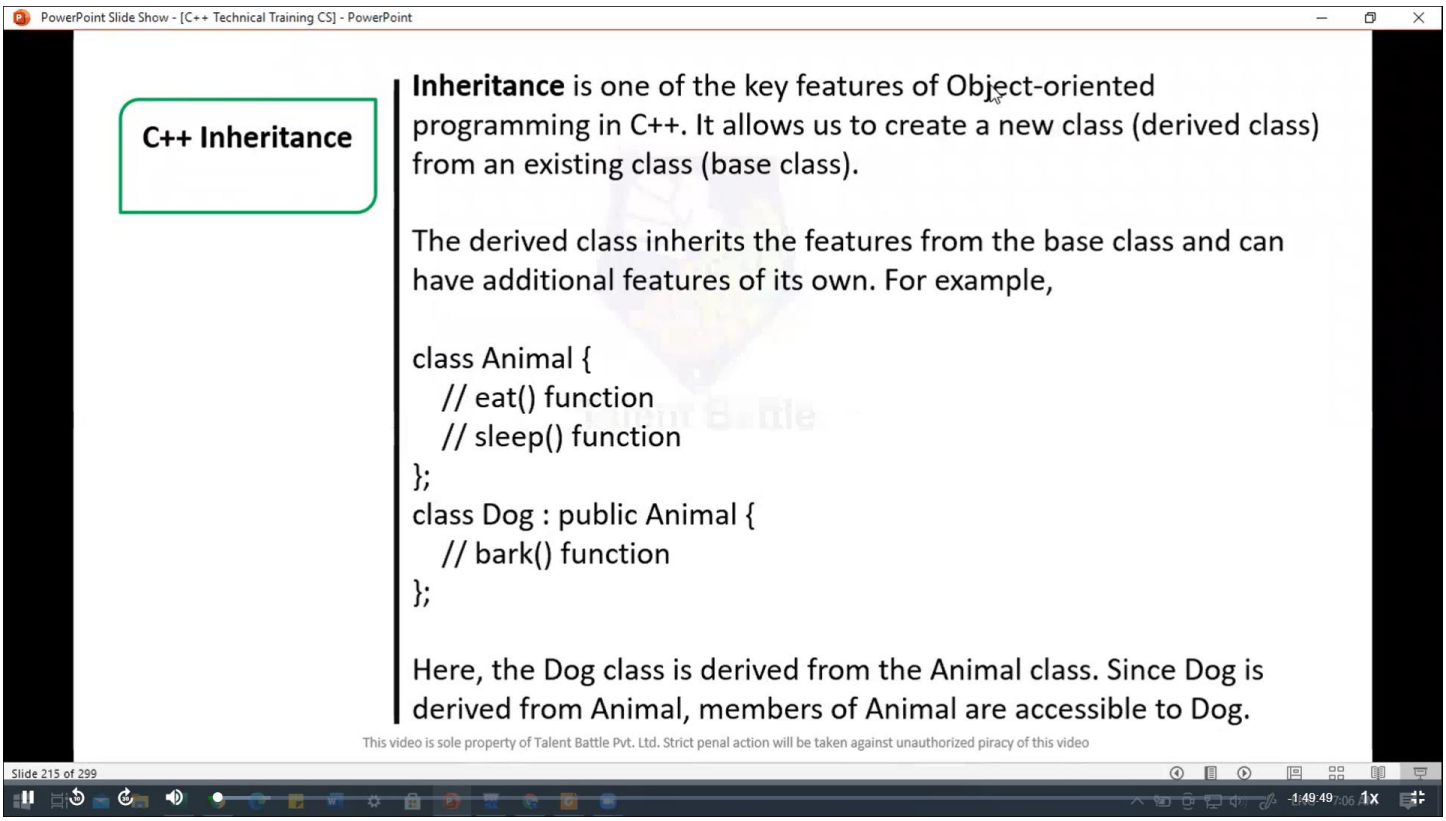

```
#include <iostream>
using namespace std;

class MyClass {
  private:
    int value;

  public:
    MyClass() : value(0){}

    // Prefic increment
    MyClass& operator++(){
      ++value;
      return *this;
    }

    // Postfix increment
    MyClass operator++ (int){
      MyClass temp = *this;
      ++value;
      return temp;
    }

    // Prefix decrement
```

```
    MyClass& operator-- (){
      --value;
      return *this;
    }

    // Postfix decrement
    MyClass operator--(int){
      MyClass temp = *this;
      --value;
      return temp;
    }

    void display() const {
       cout << "Value: " << value << endl;
    }
};

int main(){
  MyClass obj;

  cout << "Initial ";
  obj.display();

  cout << "After prefix increment ";
  ++obj;
  obj.display();

  cout << "After postfix increment ";
  obj++;
  obj.display();

  cout << "After prefix decrement ";
  --obj;
  obj.display();

  cout << "After postfix decrement ";
  obj--;
  obj.display();

  return 0;
}



/*
Initial Value: 0
After prefix increment Value: 1
After postfix increment Value: 2
After prefix decrement Value: 1
After postfix decrement Value: 0
```

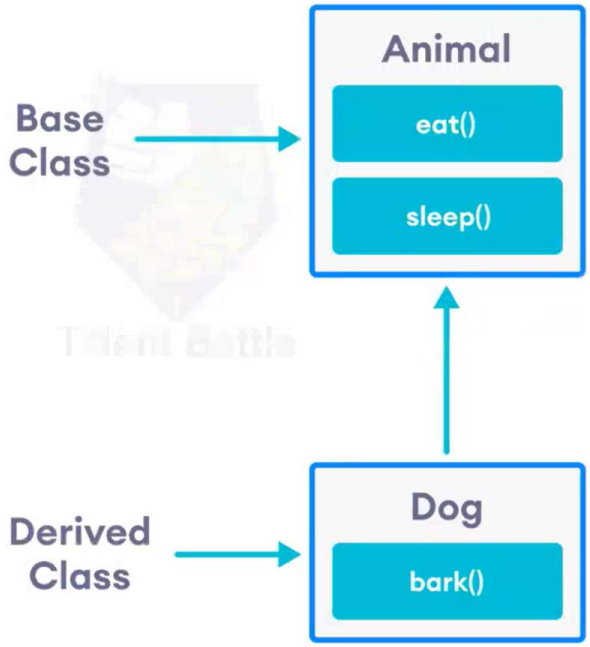
```
--------------------------------
Process exited after 0.1404 seconds with return value 0
*/
```

```
//=====================================================

#include <iostream>

class Complex {
private:
    double real;
    double imag;

public:
    // Constructor
    Complex(double r = 0, double i = 0) : real(r), imag(i) {}

    // Overloading the - operator
    Complex operator-(const Complex& other) const {
        return Complex(real - other.real, imag - other.imag);
    }

    // Display function
    void display() const {
        std::cout << real << " + " << imag << "i" << std::endl;
    }
};

int main() {
    double real1, imag1, real2, imag2;

    std::cout << "Enter real and imaginary part of the first complex number: ";
    std::cin >> real1 >> imag1;

    std::cout << "Enter real and imaginary part of the second complex number: ";
    std::cin >> real2 >> imag2;

    Complex c1(real1, imag1);
    Complex c2(real2, imag2);

    Complex c3 = c1 - c2;
```

```
    std::cout << "Result of subtraction: ";
    c3.display();

    return 0;
}



/*
Enter real and imaginary part of the first complex number: 4
3
Enter real and imaginary part of the second complex number: 6
5
Result of subtraction: -2 + -2i

--------------------------------
Process exited after 4.799 seconds with return value 0
Press any key to continue . . .
*/
```



C++ Inheritance



C++ Inheritance

is-a relationship

Inheritance is an is-a relationship. We use inheritance only if an is-a relationship is present between the two classes.

Here are some examples:

- A car is a vehicle.
- Orange is a fruit.
- A surgeon is a doctor.
- A dog is an animal.

C++ Inheritance

C++ protected Members

The access modifier protected is especially relevant when it comes to C++ inheritance.

Like private members, protected members are inaccessible outside of the class. However, they can be accessed by derived classes and friend classes/functions.

We need protected members if we want to hide the data of a class, but still want that data to be inherited by its derived classes.

C++ Inheritance

Access Modes in C++ Inheritance

So far, we have used the public keyword in order to inherit a class from a previously-existing base class. However, we can also use the private and protected keywords to inherit classes. For example,

```
class Animal {
   // code
};
class Dog : private Animal {
   // code
};
class Cat : protected Animal {
   // code
};
```

C++ Inheritance

The various ways we can derive classes are known as access modes. These access modes have the following effect:

public: If a derived class is declared in public mode, then the members of the base class are inherited by the derived class just as they are.

private: In this case, all the members of the base class become private members in the derived class.

protected: The public members of the base class become protected members in the derived class.

The private members of the base class are always private in the derived class.

C++ Inheritance

public, protected and private inheritance in C++

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.

C++ Inheritance

```
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
}
```


C++ Inheritance

Accessibility in public Inheritance

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

C++ Inheritance

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)

C++ Inheritance

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)

C++ Inheritance

Member Function Overriding in Inheritance

Suppose, base class and derived class have member functions with the same name and arguments.

If we create an object of the derived class and try to access that member function, the member function in the derived class is invoked instead of the one in the base class.

The member function of derived class overrides the member function of base class.

C++ Inheritance

C++ Function Overriding

As we know, inheritance is a feature of OOP that allows us to create derived classes from a base class. The derived classes inherit features of the base class.

Suppose, the same function is defined in both the derived class and the based class.

Now if we call this function using the object of the derived class, the function of the derived class is executed.

This is known as **function overriding** in C++. The function in derived class overrides the function in base class.

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Inheritance

```
class Base {
    public:
      void print() {
          // code
      }
};

class Derived : public Base {
    public:
      void print() {
          // code
      }
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```


PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Inheritance

Access Overridden Function in C++

To access the overridden function of the base class, we use the scope resolution operator :: .

```
class Base {
    public:
      void print() {
          // code
      }
};

class Derived : public Base {
    public:
      void print() {
          // code
      }
};

int main() {
    Derived derived1, derived2;

    derived1.print();

    derived2.Base::print();

    return 0;
}
```

C++ Inheritance

```
class Base {
    public:
      void print() {
          // code
      }
};

class Derived : public Base {
    public:
      void print() {
          // code
          Base::print();
      }
};

int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```

C++ Call Overridden Function From Derived Class

C++ Inheritance

C++ Multiple, Multilevel and Hierarchical Inheritance

Inheritance is one of the core feature of an object-oriented programming language. 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.

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Inheritance

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.

Slide 233 of 299

```
//=========================================================
// C++ Inheritance

// C++ program to demonstrate inheritance

/*
#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;
}
*/

/*
I can eat!
I can sleep!
I can bark! Woof woof!!

--------------------------------
Process exited after 0.06775 seconds with return value 0
*/



//===============================================
// C++ Inheritance
// Multi-level inheritance

/*
#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;
```

```
}
*/


/*
Base class content.
--------------------------------
Process exited after 0.1374 seconds with return value 0
*/



//===============================================


#include<iostream>
using namespace std;

class A;
class B{
  public:
    int  a;

    void getdata(int n){
      a = n;
    }

    friend int sum(A, B);
};

class A{
  public:
    int b;
    void getdata(int m){
      b = m;
    }

    friend int sum(A, B);
};

int sum(A m, B n){
  int result;
  result = m.b + n.a;
  return result;
}

int main(){
  B obj1;
  A obj2;
  obj2.getdata(10);
  obj1.getdata(20);
```

```
  cout << "Result of sum = " << sum(obj2, obj1);
  return 0;
}


/*
Result of sum = 30
--------------------------------
Process exited after 0.1154 seconds with return value 0
Press any key to continue . . .
*/
```

Project3 - [Project3.dev] - [Executing] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 64-bit Release

(globals)

Project Classes main.cpp

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

class A;// forward declaration
class B{
    public:
    int a;

    void getdata(int n)
    {
        a=n;
    }
    friend int sum(A,B);
};

class A
{
```

function
- func. prototype / Declarn
- func. Defn / Body
- func. call

Compiler | Resources | Compile Log | Debug | Find Results | Close

Abort Compilation

Shorten compiler paths

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Rohit\Desktop\Project3.exe
- Output Size: 1.83236885070801 MiB
- Compilation Time: 5.86s
```

Line: 4 Col: 31 Sel: 0 Lines: 50 Length: 504 Insert Done parsing in 0.046 seconds

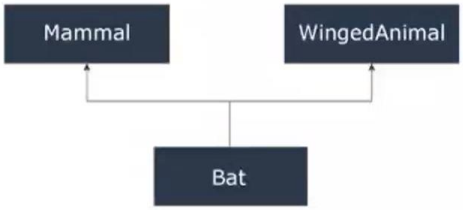
41:16 1x

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Multiple Inheritance

C++ 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.



Slide 235 of 299

ENG 8:23 AM

```cpp
//==========================================
// Multiple Inheritance

#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;
}



/*
Mammals can give direct birth.
Winged animal can flap.

--------------------------------
Process exited after 0.1364 seconds with return value 0
*/
```

C++ Inheritance

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.

C++ Inheritance

```
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 to class either base1or base2

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

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Inheritance

C++ Hierarchical Inheritance

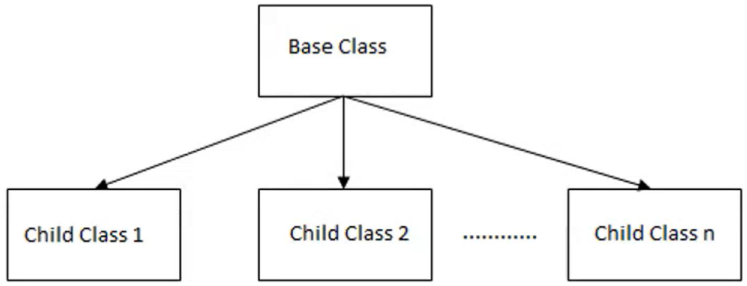


Fig: Hierarchical Inheritance

In hierarchical inheritance, all features that are common in child classes are included in the base class.

Slide 239 of 299 15:41 1.5x

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Inheritance

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 {
    ... .. ...
}
```


Slide 240 of 299 14:04 1.5x

```
//=====================================================
// C++ program to demonsrate 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;
}




/*
Dog Class:
I am an animal.
I am a Dog. Woof woof.

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

```
--------------------------------
Process exited after 0.1271 seconds with return value 0
Press any key to continue . . .

*/
```

Project3 - [Project3.dev] - Dev-C++ 5.11

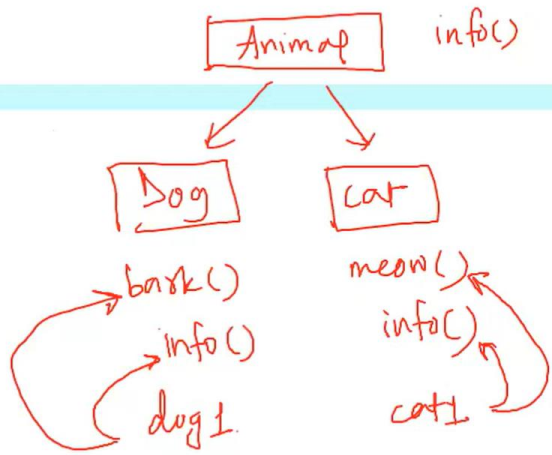
File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 64-bit Release

(globals)

Project Classe main.cpp

```
21      public:
22       void meow() {
23           cout << "I am a Cat. Meow." << endl;
24       }
25  };
26  int main() {
27      // Create object of Dog class
28      Dog dog1;
29      cout << "Dog Class:" << endl;
30      dog1.info();  // Parent Class function
31      dog1.bark();
32
33      // Create object of Cat class
34      Cat cat1;
35      cout << "\nCat Class:" << endl;
36      cat1.info();  // Parent Class function
37      cat1.meow();
38
39      return 0;
40  }
41
```



Compiler Resources Compile Log Debug Find Results Close

Abort Compilation

Compilation Time: 4.86s

Line: 26 Col: 1 Sel: 0 Lines: 41 Length: 831 Insert Done parsing in 0.11 seconds

11:15 1.5x