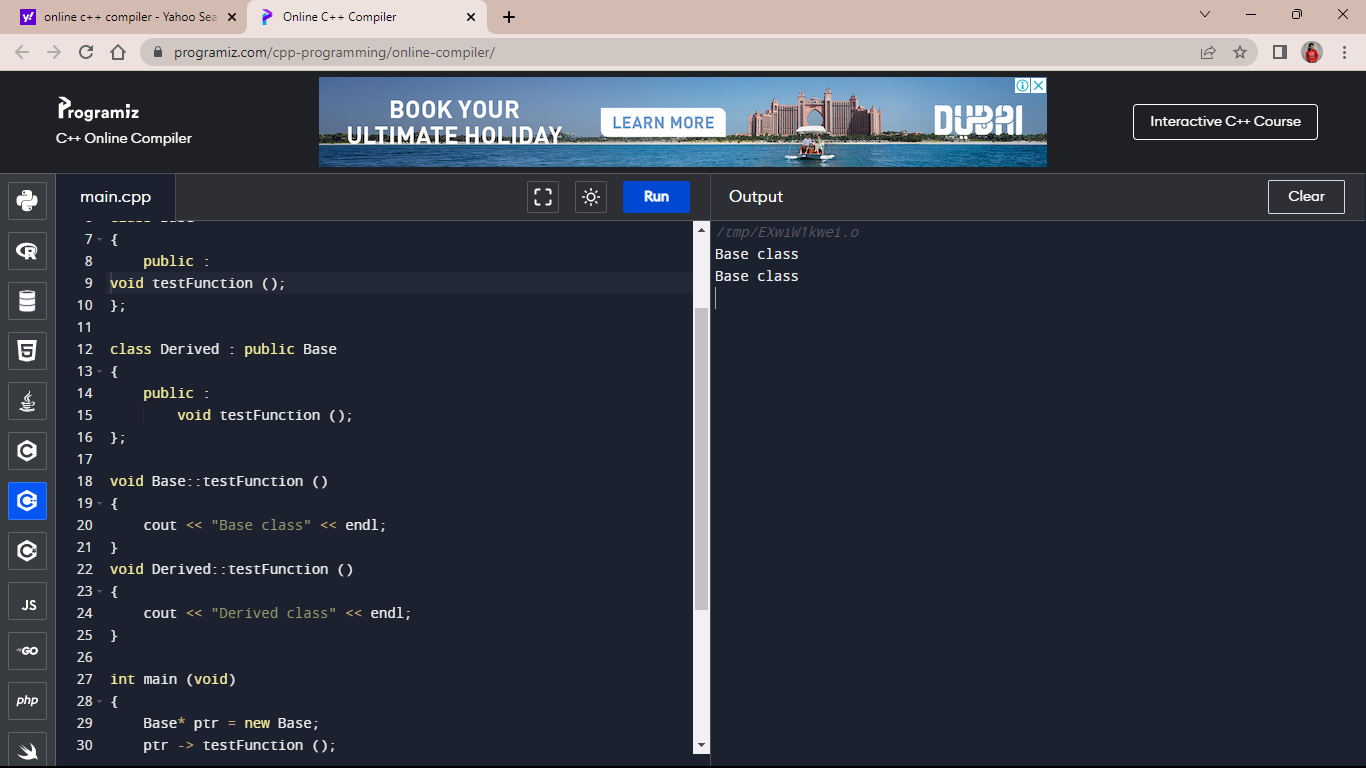
Zohaib Mehran

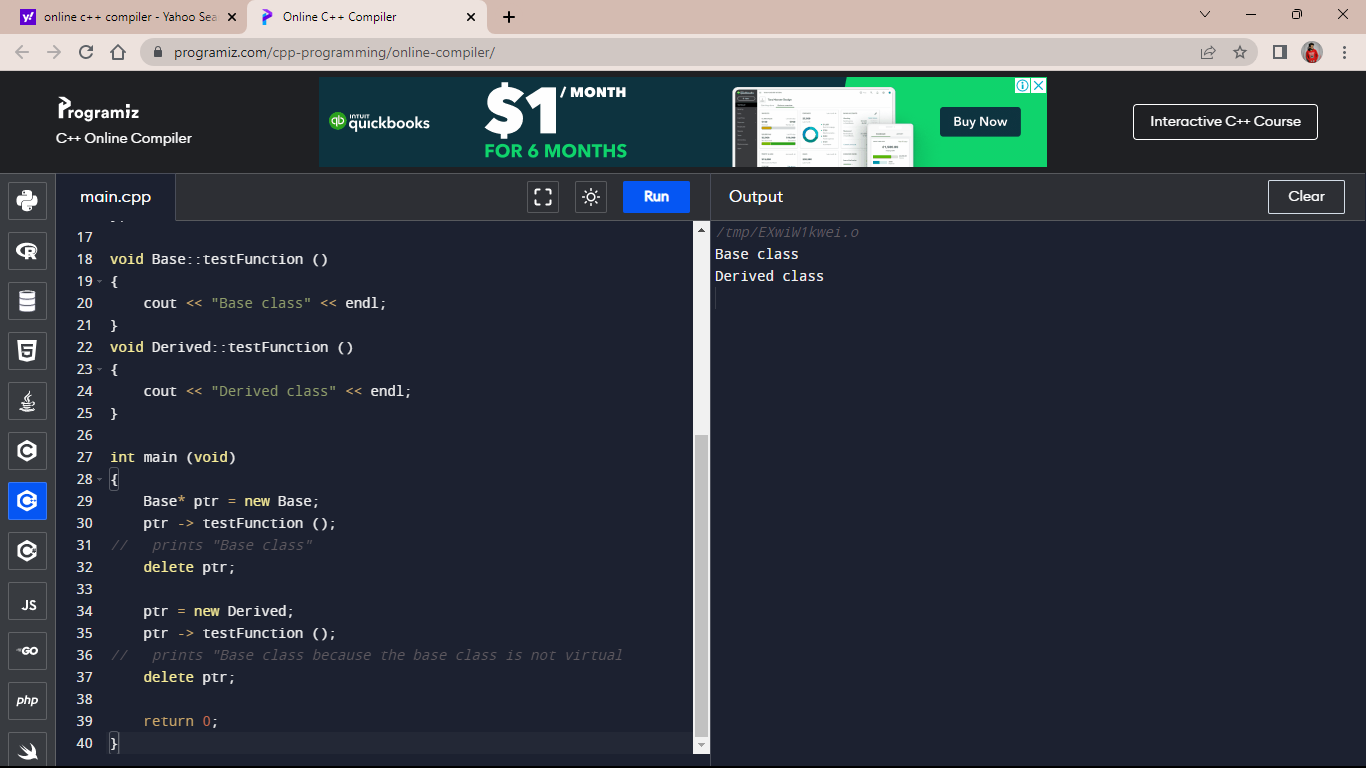
Sapid 46728

Task 1

* Before adding virtual function the code gives the output ;

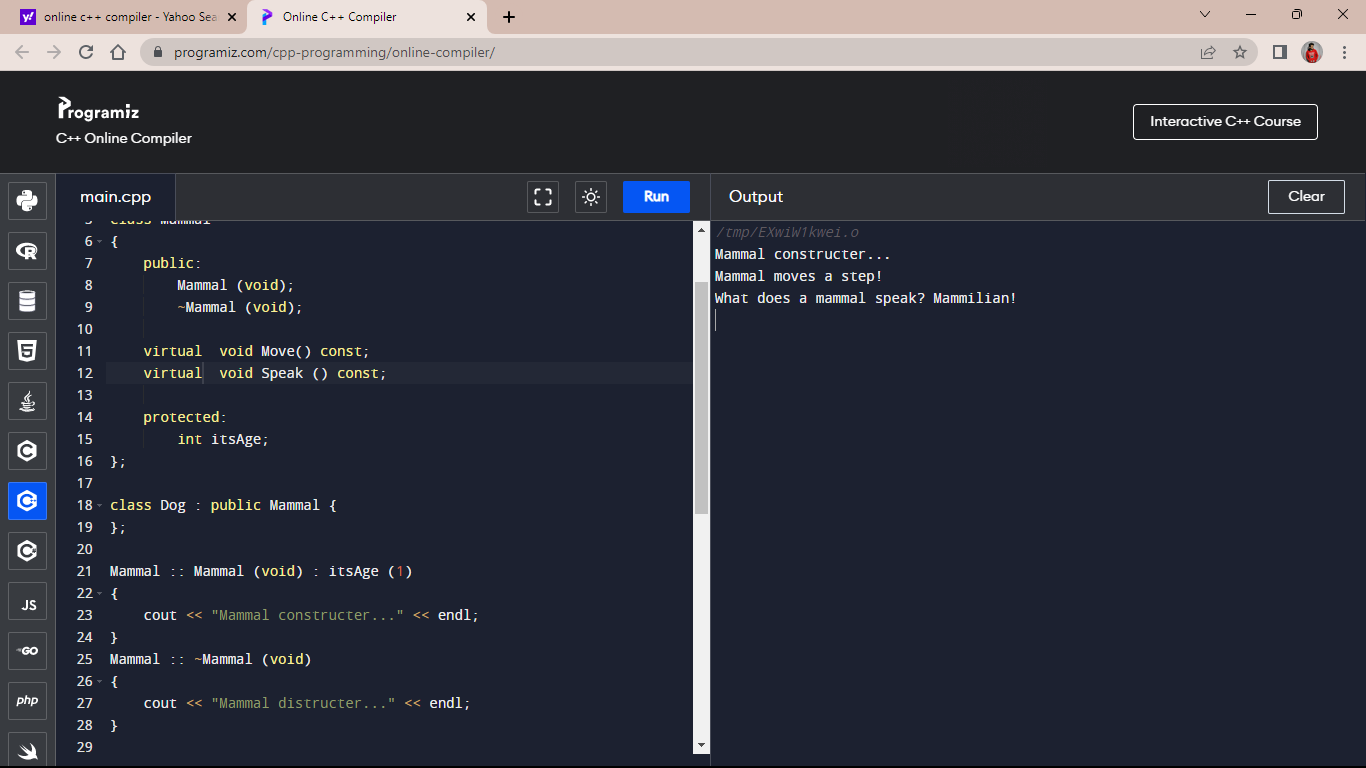


* And after doing the given changes the output was ;

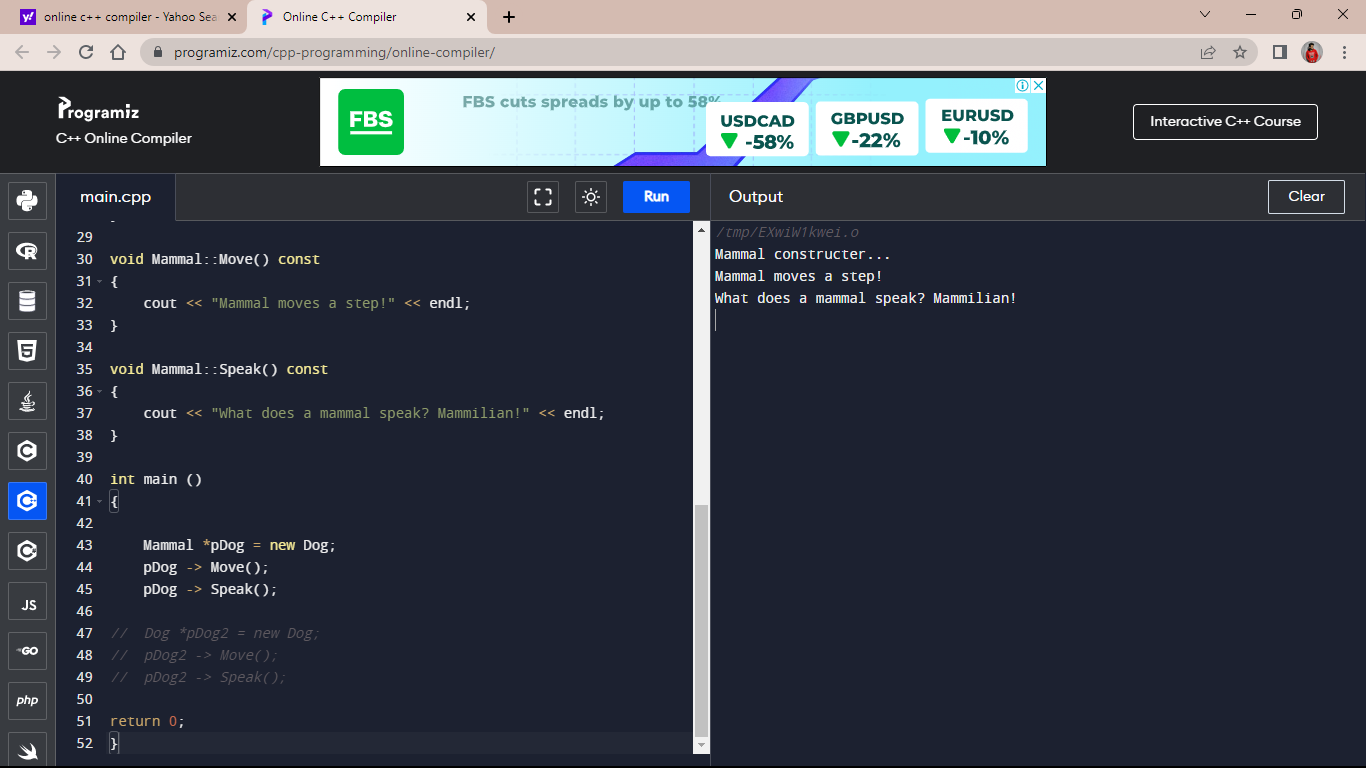


Task 2

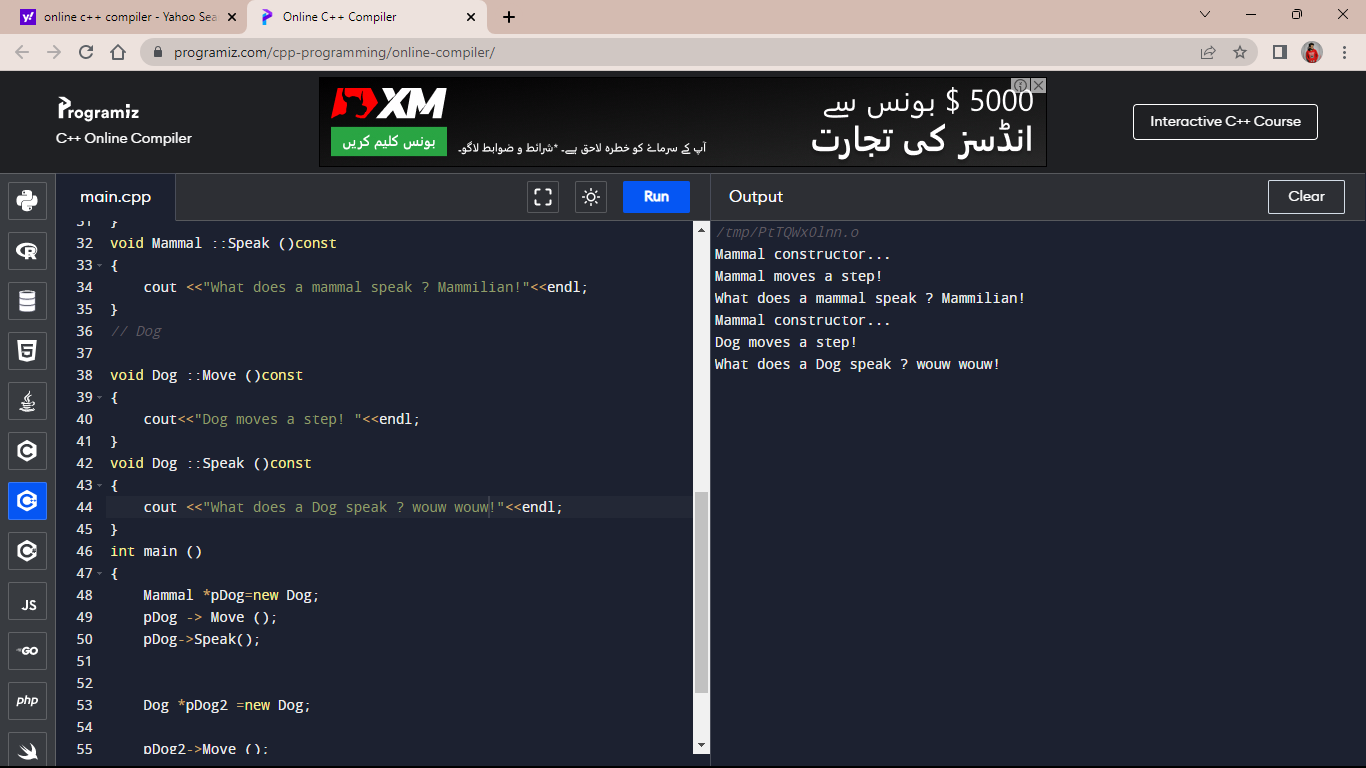
* With the given code ( virtual function included ) the output was;



* After removing the virtual function the output remains same;

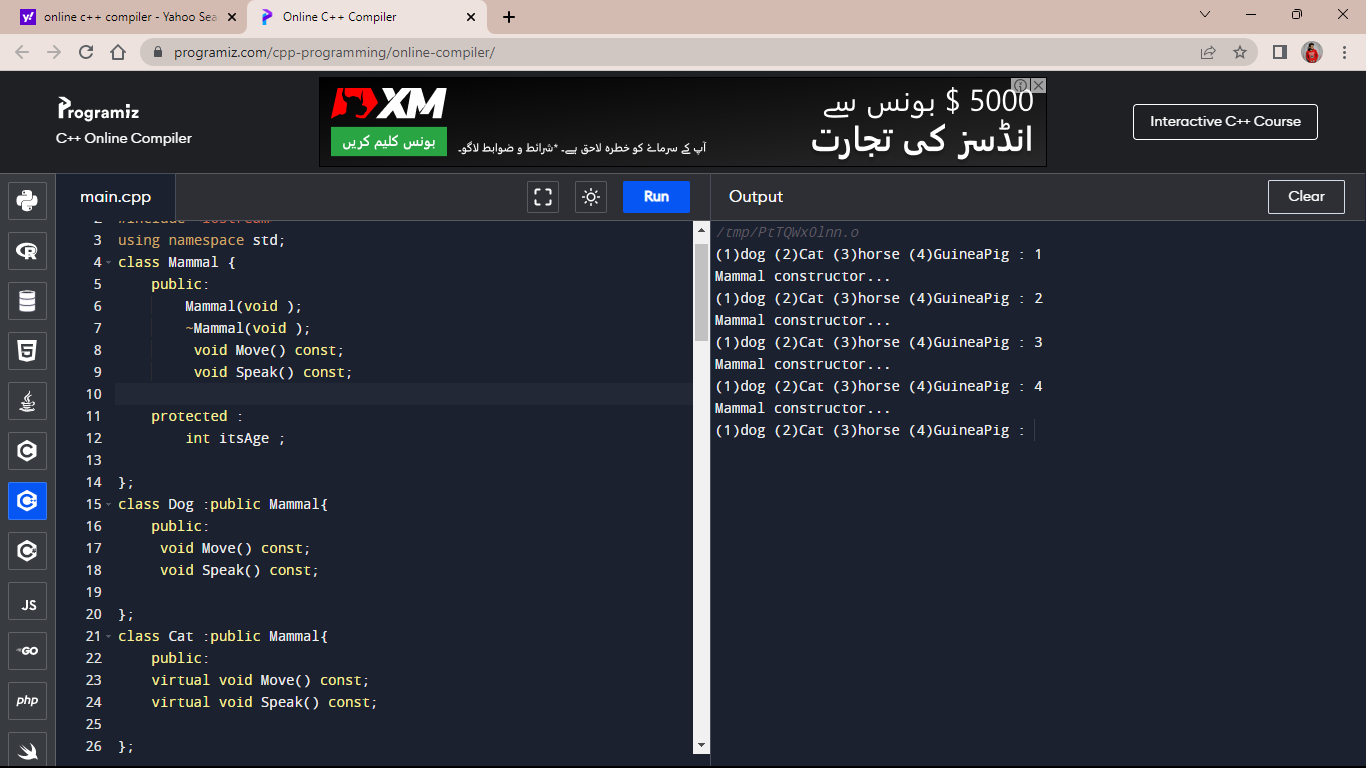


* When we add another dog pointer the output was;



Task 3

* After adding additional classes in the given code, the output was;



Answer the following questions:

Part 1

Question 1

Answer

Yes, when a class inherits from another class, it inherits all the members (data and functions) of the base class. This means that in your example, the Dog class would inherit all the members of both the Mammal and Animal classes, including any public and protected members.

Question 2

Answer

If the Mammal class overrides a function and the Dog class inherits from the Mammal class then the Dog class would get the overridden function from the Mammal class.

Question 3

Answer

Yes, a derived class make a public base function private.

Question 4

Answer

Making all class functions virtual can have performance implications because virtual functions are implemented using dynamic dispatch, which incurs some overhead compared to non-virtual functions. Additionally, making all functions virtual can make the class hierarchy more difficult to understand and maintain, as it can increase the complexity of the code.

Question 5

Answer

The base class's two-integer version of the function will be called. This is because the derived class has only overridden the one-integer version of the function .The call will be dynamically dispatched to the appropriate version of the function based on the actual type of the object being pointed to.

Part 2

Question 1

Answer

A v-table, short for virtual table, is a data structure used by compilers to implement polymorphic behavior in object-oriented programming languages that use virtual functions. A v-table is an array of function pointers that is associated with each class that has virtual functions. The v-table is created by the compiler and is used to dynamically dispatch calls to virtual functions at runtime.

Question 2

Answer

A virtual destructor is a destructor that is declared as virtual in a base class and is intended to be overridden in derived classes.

Question 3

Answer

In C++, constructors cannot be declared as virtual. This is because constructors are called during object construction, and the virtual table is not yet fully initialized at that point, so dynamic dispatch based on the actual object type is not yet possible.

Question 4

Answer

In C++, it's not possible to declare a constructor as virtual, including the copy constructor. However, it's possible to create a virtual copy constructor.

Question 5

Answer

To invoke a base member function from a derived class in which you've overridden that function, you can use the scope resolution operator:: to explicitly specify the base class name and function name.

Question 6

Answer

To invoke a base member function from a derived class in which you have not overridden that function, you can simply call the function by its name**.**

**Question 7**

**Answer**

Yes, if a base class declares a function to be virtual, and a derived class does not use the virtual keyword when overriding that function, it is still virtual when inherited by a third-generation class. The virtual keyword is not required when overriding a virtual function, but it is good practice to use it for clarity.

Question 8

Answer

The protected keyword in C++ is used to specify access control for class members. Members declared as protected are accessible within the class and its derived classes, but not from outside the class hierarchy.

Part 3

Question 1

Answer

class myclass {

public:

virtual void myFunction(int myParam) = 0;

};

Question 2

Answer

class Shape {

public:

virtual double area() const = 0;

};

class Rectangle : public Shape {

public:

Rectangle(double width, double height);

double area() const override;

protected:

double width\_;

double height\_;

};

class Square : public Rectangle {

public:

Square(double side);

};

Question 3

Answer

Square::Square(double side)

: Rectangle(side, side)

{

// Any additional initialization specific to Square can go here.

}

Question 4

Answer

class Shape {

public:

virtual ~Shape() {}

virtual Shape\* clone() const = 0;

virtual double area() const = 0;

};

class Rectangle : public Shape {

public:

Rectangle(double width, double height);

double area() const override;

Rectangle\* clone() const override;

protected:

double width\_;

double height\_;

};

class Square : public Rectangle {

public:

Square(double side);

Square\* clone() const override;

};

Rectangle\* Rectangle::clone() const {

return new Rectangle(\*this);

}

Square\* Square::clone() const {

return new Square(\*this);

}

Question 5

Answer

Perhaps nothing. SomeFunction expects a Shape object. You've passed it a Rectangle"sliced" down to a Shape. As long as you don't need any of the Rectangle parts, this will be fine. If you do need the rectangle parts, you'll need to change someFunction to take a pointer or a reference to a Shape.

Question 6

Answer

You can not declare a copy constructor to be virtual.