Name: Zainab Amjad

Sap id: 47596

Subject: OOP

Topic: Polymorphism

Task 1

First of all I type this code as it is its print Base class for 2 times.

CODE:

#include<iostream>

using std::cout;

using std::endl;

class Base

{

public:

void testfunction();

};

class Derived:public Base

{

public:

void testfunction();

};

void Base::testfunction()

{

cout<<"Base class"<<endl;

}

void Derived::testfunction()

{

cout<<"Derived class "<<endl;

}int main(void)

{

Base\* ptr=new Base;

ptr -> testfunction();

delete ptr;

ptr=new Derived;

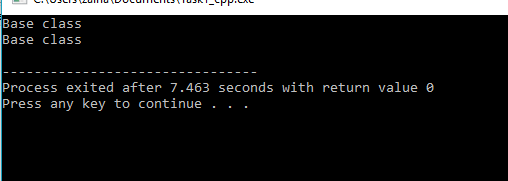
ptr -> testfunction();

delete ptr;

return 0;

}

OUTPUT:



Then I modify this code I add keyword virtual in base class its print 1 time Base class and 1 time Derived class on the screen.

CODE:

#include<iostream>

using std::cout;

using std::endl;

class Base

{

public:

virtual void testfunction();

};

class Derived:public Base

{

public:

void testfunction();

};

void Base::testfunction()

{

cout<<"Base class"<<endl;

}

void Derived::testfunction()

{

cout<<"Derived class "<<endl;

}int main(void)

{

Base\* ptr=new Base;

ptr -> testfunction();

delete ptr;

ptr=new Derived;

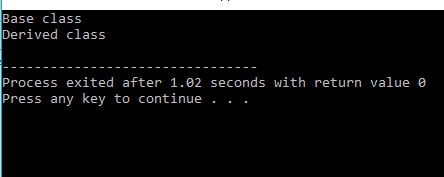
ptr -> testfunction();

delete ptr;

return 0;

}

OUTPUT:



Task2:

First of all I type this code as it is

CODE:

#pragma once

#include<iostream>

using std::cout;

using std::endl;

class Mammal

{

public:

Mammal(void);

~Mammal(void);

virtual void Move() const;

virtual void Speak() const;

protected:

int itsAge;

};

class Dog:public Mammal{

public:

virtual void Move()const;

virtual void Speak()const;

};

Mammal::Mammal(void):itsAge(1)

{

cout<<"Mammal constructor..."<<endl;

}

Mammal::~Mammal(void)

{

cout<<"Mammal destructor..."<<endl;

}

void Mammal::Move() const

{

cout<<"Mammal moves a step!"<<endl;

}

void Mammal::Speak() const

{

cout<<"What does a mammal speak?Mammilian!"<<endl;

}

void Dog::Move()const

{

cout<<"Dog moves a step!"<<endl;

}

void Dog::Speak()const

{

cout<<"What does a Dog speak ?bhao!"<<endl;

}

int main()

{

Mammal \*pDog = new Dog;

pDog->Move();

pDog->Speak();

Dog \*pDog2 = new Dog;

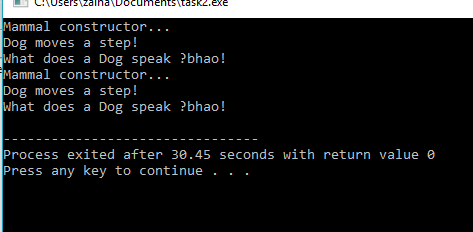
pDog2->Move();

pDog2->Speak();

return 0;

}

OUTPUT:



After removing the keyword virtual

CODE:

#pragma once

#include<iostream>

using std::cout;

using std::endl;

class Mammal

{

public:

Mammal(void);

~Mammal(void);

void Move() const;

void Speak() const;

protected:

int itsAge;

};

class Dog:public Mammal{

public:

virtual void Move()const;

virtual void Speak()const;

};

Mammal::Mammal(void):itsAge(1)

{

cout<<"Mammal constructor..."<<endl;

}

Mammal::~Mammal(void)

{

cout<<"Mammal destructor..."<<endl;

}

void Mammal::Move() const

{

cout<<"Mammal moves a step!"<<endl;

}

void Mammal::Speak() const

{

cout<<"What does a mammal speak?Mammilian!"<<endl;

}

void Dog::Move()const

{

cout<<"Dog moves a step!"<<endl;

}

void Dog::Speak()const

{

cout<<"What does a Dog speak ?bhao!"<<endl;

}

int main()

{

Mammal \*pDog = new Dog;

pDog->Move();

pDog->Speak();

Dog \*pDog2 = new Dog;

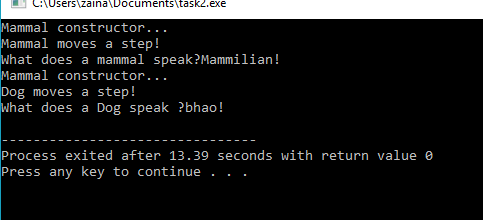
pDog2->Move();

pDog2->Speak();

return 0;

}

OUTPUT:



Task3

CODE:

#pragma once

#include<iostream>

using std::cout;

using std::endl;

class Mammal

{

public:

Mammal(void);

~Mammal(void);

void Move() const;

void Speak() const;

protected:

int itsAge;

};

class Dog:public Mammal{

public:

virtual void Move()const;

virtual void Speak()const;

};

class Cat:public Mammal{

public:

virtual void Move()const;

virtual void Speak()const;

};

class Horse:public Mammal{

public:

virtual void Move()const;

virtual void Speak()const;

};

class GuineaPig:public Mammal{

public:

virtual void Move()const;

virtual void Speak()const;

};

Mammal::Mammal(void):itsAge(1)

{

cout<<"Mammal constructor..."<<endl;

}

Mammal::~Mammal(void)

{

cout<<"Mammal destructor..."<<endl;

}

void Mammal::Move() const

{

cout<<"Mammal moves a step!"<<endl;

}

void Mammal::Speak() const

{

cout<<"What does a mammal speak?Mammilian!"<<endl;

}

void Dog::Move()const

{

cout<<"Dog moves a step!"<<endl;

}

void Dog::Speak()const

{

cout<<"What does a Dog speak ?bhao!"<<endl;

}

void Cat::Move()const

{

cout<<"Cat moves a step!"<<endl;

}

void Cat::Speak()const

{

cout<<"What does a Cat speak ?meow!"<<endl;

}

void Horse::Move()const

{

cout<<"Horse moves a step!"<<endl;

}

void Horse::Speak()const

{

cout<<"What does a Horse speak ?eenghhh!"<<endl;

}

void GuineaPig::Move()const

{

cout<<"GuineaPig moves a step!"<<endl;

}

void GuineaPig::Speak()const

{

cout<<"What does a GuineaPig speak ?weep weep!"<<endl;

}

int main()

{

Mammal \*pDog = new Dog;

pDog->Move();

pDog->Speak();

Dog \*pDog2 = new Dog;

pDog2->Move();

pDog2->Speak();

Mammal \*pCat = new Cat;

pCat->Move();

pCat->Speak();

Cat \*pCat2 = new Cat;

pCat2->Move();

pCat2->Speak();

Mammal \*pHorse = new Horse;

pHorse->Move();

pHorse->Speak();

Horse \*pHorse2 = new Horse;

pHorse2->Move();

pHorse2->Speak();

Mammal \*pGuineaPig = new GuineaPig;

pGuineaPig->Move();

pGuineaPig->Speak();

GuineaPig \*pGuineaPig2 = new GuineaPig;

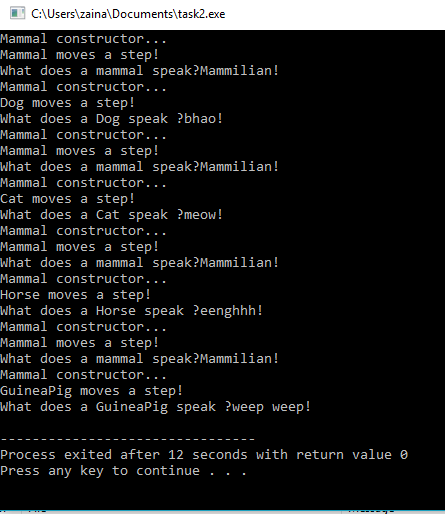
pGuineaPig2->Move();

pGuineaPig2->Speak();

return 0;

}

OUTPUT:



QUESTIONS/ANSWERS

Q1.If in the example................function?

Ans. If Dog inherits from mammal then mammal function is also a overrriden function.

Q2.Can..........private?

Ans.Yes its make a public class function private.

Q3.Why...........virtual?

Ans.Some function are virtual but it is not compulsory to make all function virtual.

Q4.If......form?

Ans.It takes one int but error occur at the compilation time.

Q5.What .....?

Ans.The v –table keeps a list of the addresses of all the virtual functions .

Q6.What ......constructor?

Ans.A virtual constructor is a mechanism of having base class pointer to a derived class object.

Q7.How...........................................?

Ans.The constructor cannot be virtual because when a constructor of a class is executed there is no virtual table in memory so,the constructor should be non virtual but the destructor is possible.

Q8.How...........................?

Ans.The duplicate object being created with the help of Clone()virtual function which is also considered as virtual copy constructor.

Q9.How.............?

Ans.By using the pointer of the base class to point to an object of the derived class and calling the function through the pointer.

Q10.What is protected...?

Ans.The keyword protected is an acess modifier used for attributes ,methods, constructors, making them accessiblein the same package and subclasses.