**Lab Tak OOP**

**Task 1:**

To gain a better understanding of polymorphic and virtual functions start with the following simple example. Notice we have not defined a virtual function yet.

#include<iostream>

using namespace std;

using std::endl;

class Base{

public:

virtual void testFunction ();

};

class Derived : public Base {

public:

void testFunction();

};

//#include "Task1.h"

void Base :: testFunction (){

cout<<" Base class "<<endl;

}

void Derived :: testFunction (){

cout<<" Derived class "<<endl;

}

//#include "Task1.h"

int main (void){

Base\*ptr=new Base ;

ptr -> testFunction();

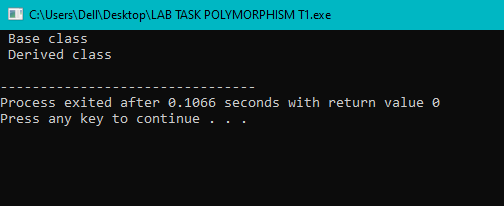
delete ptr;

ptr= new Derived;

ptr -> testFunction();

delete ptr;

return 0;

}

**Task 2 :**

You will first build two classes, Mammal and Dog. Dog will inherit from Mammal. Below is the Mammal class code. Once you have the Mammal class built, build a second class Dog that will inherit publicly from Mammal.

**Without Modification:**

#include<iostream>

using namespace std;

using std::endl;

// Mammal class : Base class

class Mammal{

public:

Mammal(void);

~Mammal(void);

virtual void Move () const ;

virtual void Speak () const ;

protected :

int itsAge;

};

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? "<<endl;

}

// Dog class : Derived class

class Dog : public Mammal{

public:

Dog(void);

~Dog(void);

virtual void Bark () const;

void Move () const;

protected:

int itsAge;

};

Dog :: Dog(void) : itsAge(2){

cout<<" Dog Constructor "<<endl;

}

Dog :: ~Dog (void){

cout<<" Dog Destructor "<<endl;

}

void Dog :: Move() const{

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

}

void Dog :: Bark () const {

cout<<" Dog is barking "<<endl;

}

int main(){

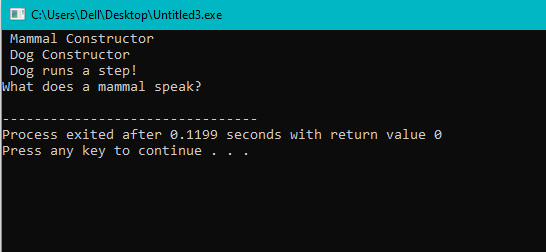
Mammal \*pDog = new Dog;

pDog -> Move();

pDog -> Speak ();

return 0;

}



**With Modification ( removing virtual from class mammal) :**

#include<iostream>

using namespace std;

using std::endl;

// Mammal class : Base class

class Mammal{

public:

Mammal(void);

~Mammal(void);

void Move () const ;

void Speak () const ;

protected :

int itsAge;

};

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? "<<endl;

}

// Dog class : Derived class

class Dog : public Mammal{

public:

Dog(void);

~Dog(void);

virtual void Bark () const;

void Move () const;

protected:

int itsAge;

};

Dog :: Dog(void) : itsAge(2){

cout<<" Dog Constructor "<<endl;

}

Dog :: ~Dog (void){

cout<<" Dog Destructor "<<endl;

}

void Dog :: Move() const{

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

}

void Dog :: Bark () const {

cout<<" Dog is barking "<<endl;

}

int main(){

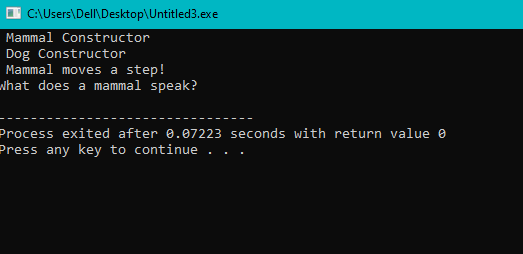
Mammal \*pDog = new Dog;

pDog -> Move();

pDog -> Speak ();

return 0;

}



**Change in pointers :**

#include<iostream>

using namespace std;

using std::endl;

// Mammal class : Base class

class Mammal{

public:

Mammal(void);

~Mammal(void);

void Move () const ;

void Speak () const ;

protected :

int itsAge;

};

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? "<<endl;

}

// Dog class : Derived class

class Dog : public Mammal{

public:

Dog(void);

~Dog(void);

virtual void Bark () const;

void Move () const;

protected:

int itsAge;

};

Dog :: Dog(void) : itsAge(2){

cout<<" Dog Constructor "<<endl;

}

Dog :: ~Dog (void){

cout<<" Dog Destructor "<<endl;

}

void Dog :: Move() const{

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

}

void Dog :: Bark () const {

cout<<" Dog is barking "<<endl;

}

int main(){

Mammal \*pDog = new Dog;

Dog \*pDog2 = new Dog ;

pDog -> Move();

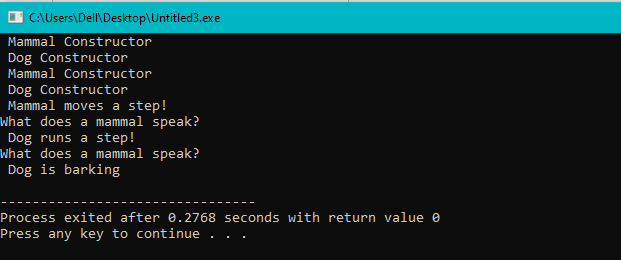
pDog -> Speak ();

pDog2 -> Move();

pDog2 -> Speak();

pDog2 -> Bark();

return 0;

}

**Task 3**:

Develop additional classes for Cat, Horse, and GuineaPig overriding the move and speak methods. (If you do not know guinea pigs go “weep weep”)

#include<iostream>

using namespace std;

using std::endl;

// Mammal class : Base class

class Mammal{

public:

Mammal(void);

~Mammal(void);

void Move () const ;

void Speak () const ;

protected :

int itsAge;

};

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? "<<endl;

}

// Dog class : Derived class

class Dog : public Mammal{

public:

Dog(void);

~Dog(void);

virtual void Bark () const;

void Move () const;

protected:

int itsAge;

};

Dog :: Dog(void) : itsAge(2){

cout<<" Dog Constructor "<<endl;

}

Dog :: ~Dog (void){

cout<<" Dog Destructor "<<endl;

}

void Dog :: Move() const{

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

}

void Dog :: Bark () const {

cout<<" Dog is barking "<<endl;

}

// Cat class : Derived class

class Cat : public Mammal{

public:

Cat(void);

~Cat(void);

virtual void Meow () const;

virtual void Move () const;

protected:

int itsAge;

};

Cat :: Cat(void) : itsAge(3){

cout<<" Cat Constructor "<<endl;

}

Cat :: ~Cat (void){

cout<<" Cat Destructor "<<endl;

}

void Cat :: Move() const{

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

}

void Cat :: Meow () const {

cout<<" Cat is meowing "<<endl;

}

// Horse class : Derived class

class Horse : public Mammal{

public:

Horse(void);

~Horse(void);

virtual void Neigh () const;

virtual void Move () const;

protected:

int itsAge;

};

Horse :: Horse(void) : itsAge(4){

cout<<" Horse Constructor "<<endl;

}

Horse :: ~Horse (void){

cout<<" Horse Destructor "<<endl;

}

void Horse :: Move() const{

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

}

void Horse :: Neigh () const {

cout<<" Horse is neighing "<<endl;

}

// GuineaPig class : Derived class

class GuineaPig : public Mammal{

public:

GuineaPig(void);

~GuineaPig(void);

virtual void Weep () const;

virtual void Move () const;

protected:

int itsAge;

};

GuineaPig :: GuineaPig(void) : itsAge(5){

cout<<" GuineaPig Constructor "<<endl;

}

GuineaPig :: ~GuineaPig (void){

cout<<" GuineaPig Destructor "<<endl;

}

void GuineaPig :: Move() const{

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

}

void GuineaPig :: Weep () const {

cout<<" GuineaPig is weeping "<<endl;

}

int main(){

// int theArray[5];

Mammal \*theArray[5];

Mammal \*ptr;

int choice,i;

for(i=0; i<5 ; i++){

cout<<"(1)dog (2)cat (3)horse (4)guinea pig : ";

cin>> choice ;

switch(choice){

case 1 : ptr = new Dog ;

break;

case 2 : ptr = new Cat ;

break;

case 3 : ptr = new Horse ;

break;

case 4 : ptr = new GuineaPig ;

break;

default : ptr = new Mammal ;

break ;

}

theArray[i]=ptr;

}

for(i=0;i<5;i++)

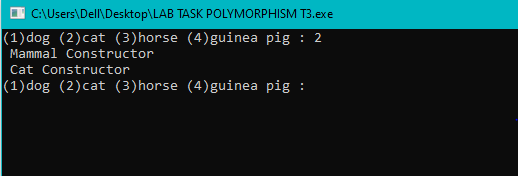
theArray[i] -> Speak();

for(i=0;i<5;i++)

delete theArray[i];

return 0;

}



**Q/Ans**

**1)** What is a v-table?

A v-table (short for virtual function table) is a mechanism used in object-oriented programming to implement dynamic dispatch or late binding of functions or methods in a polymorphic class hierarchy.

**2)** What is a virtual destructor?

A virtual destructor is a special kind of destructor that is declared virtual in the base class of a hierarchy of classes. When an object of a derived class is deleted, the virtual destructor is called in the base class, which ensures that all of the destructors in the hierarchy are called in the correct order.

**3)** How do you show the declaration of a virtual constructor?

There are no virtual constructors in c++.

**4)** How can you create a virtual copy constructor?

You need to create a base class that defines the virtual constructor.

**5)** How do you invoke a base member function from a derived class in which you've overridden that function?

Base::FunctionName(){

…

}

**6)** How do you invoke a base member function from a derived class in which you have not overridden that function?

FunctionName(){

…

}

**7)** If a base class declares a function to be virtual, and a derived class does not use the term virtual when overriding that class, is it still virtual when inherited by a third-generation class?

Yes

**8)** What is the protected keyword used for?

Used to modify the visibility of class members.

**9)** Show the declaration of a virtual function that takes an integer parameter and returns void.

virtual void SomeFunction(int);

**10)** Show the declaration of a class Square, which derives from Rectangle, which in turn derives from Shape.

class Square : public Rectangle {};

**11)** If, in Exercise 2, Shape takes no parameters, Rectangle takes two (length and width), but Square takes only one (length), show the constructor initialization for Square.

Square::Square(int length): Rectangle(length, length){}

**12)** Write a virtual copy constructor for the class Square (in Exercise 3).

classSquare { public: // ... virtual Square \* clone() const { return new Square(\*this); } // ... };

**13)** BUG BUSTERS: What is wrong with this code snippet? void SomeFunction (Shape); Shape \* pRect = new Rectangle; SomeFunction(\*pRect);

Nothing

**14)** BUG BUSTERS: What is wrong with this code snippet? class Shape() { public: Shape(); virtual ~Shape(); virtual Shape(const Shape &); };

Can't declare a copy constructor to be virtual