OOP LAB TASKS

NAME: ZEESHAN SHOUKAT

SAP ID: 47170

LAB TASK NO: 1

CODE:

```
#include<iostream>
using namespace std;
// Base Class
class Base {
        public:
   virtual void testfunction();
};
// Derived Class
        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;
}
```

OUTPUT:

1st:

```
C:\Users\Paradise Computers\Desktop\Lab Task.exe
Base class
Derived class

--------------------------------
Process exited after 13.66 seconds with return value 0
Press any key to continue . . .
```

2nd:

```
C:\Users\Paradise Computers\Desktop\Lab Task.exe
Base class
Base class

--------------------------------
Process exited after 13.87 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION:

When we use Virtual, keyword in the 1st screenshot in the Base Class the output will come Base and Derived class.

Otherwise in the 2nd screenshot if we don't use Virtual in the Base class Two Base class Output will Display on the console screen.

TASK NO: 2

CODE :

```
#include<iostream>
using namespace std;
 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;
}
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<<" the dog is run "<<endl;
}
void Dog :: Bark () const {
   cout<<"  The Dog is barking "<<endl;
}
int main(){
        Mammal *pDog = new Dog;
        pDog -> Move();
        pDog -> Speak ();
        Mammal *pDog2 = new Dog;
        pDog -> Move();
        pDog -> Speak ();

        return 0;
}
```

OUTPUT 1st:

```
Select C:\Users\Paradise Computers\Desktop\Lab Task 2.exe
 Mammal Constructor
 Dog Constructor
 the dog is run
What does a mammal speak?

--------------------------------
Process exited after 19.93 seconds with return value 0
Press any key to continue . . .
```

2nd:

```
C:\Users\Paradise Computers\Desktop\Lab Task 2.exe
 Mammal Constructor
 Dog Constructor
 the dog is run
What does a mammal speak?
 Mammal Constructor
 Dog Constructor
 the dog is run
What does a mammal speak?

--------------------------------
Process exited after 13.57 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION:

if we don't use virtual keyword then only Mammal class functions, constructors and destructors calls.

Mammal *pDog2 = new Dog;

If we add point pDog2 the 2nd output will display all the data.

Lab Task: 3

Code:

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

OUTPUT:

```
C:\Users\Paradise Computers\Desktop\Lab Task 3.exe
(1)dog (2)cat (3)horse (4)guinea pig : 1
 Mammal Constructor
 Dog Constructor
(1)dog (2)cat (3)horse (4)guinea pig : 2
 Mammal Constructor
 Cat Constructor
(1)dog (2)cat (3)horse (4)guinea pig : 3
 Mammal Constructor
 Horse Constructor
(1)dog (2)cat (3)horse (4)guinea pig : 4
 Mammal Constructor
 GuineaPig Constructor
(1)dog (2)cat (3)horse (4)guinea pig :
```

Conclusion:

Ever constructor will call after pressing the 1/2/3/4 dog cat horse guinea pig.

Question Answers:

Q1. If, in the example above, Mammal overrides a function in Animal, which does Dog get, the original or the overridden function?

Ans: In the example, Mammal overrides an Animal function, giving the overridden function to the Dog.

Q2. Can a derived class make a public base function private?

Ans. Yes, and it is kept private for all derived uses. The public members of the base class become the private members of the derived class when a base class is privately inherited by a derived class; as a result, the public members of the base class can only be accessible by the member functions of the derived class. For the objects of the derived class, they are unreachable.

Q3. Why not make all class functions virtual?

Ans: For the simple reason that a function only has to be virtual if a derived class will implement it differently.

Q4. If a function (SomeFunc()) is virtual in a base class and is also overloaded, so as to take either an integer or two integers, and the derived class overrides the form taking one integer, what is called when a pointer to

a derived object calls the two-integer form?

Ans: You will receive a build error claiming that the function only needs one int since the one-int form overriding hides the whole base class method.

More Questions:

Q1. What is a v-table?

Ans: A straightforward wrapper component for the table> element, the v-table component is. All of the standard table elements, such as thead>, tbody>, tr>, etc., may be used inside the component.

Q2. What is a virtual destructor?

Ans: Virtual destructors are useful when you might potentially delete an instance of a derived class through a pointer to base class.

Q3. How do you show the declaration of a virtual constructor?

Ans. Forward declare is insufficient since the compiler has to know the concrete type and constructor in order to return a pointer. You might write the following code in the C++ file: in header file: forward declare SubVirt and CreateClass function Add MyVirt.h and the CreateClass function definitions.

Q4. How can you create a virtual copy constructor?

Ans. A virtual copy constructor is a means to duplicate an object while keeping its dynamic type by using a pointer or reference to its base class.

Q5. How do you invoke a base member function from a derived class in which you have overridden that function?

Ans. Using the scope resolution operator: we can get to the base class' override function. By utilizing a pointer from the base class to point to an object of the derived class and then calling the method from that pointer, we can also access the overridden function.

Q6. How do you invoke a base member function from a derived class in which you have not overridden that function?

Ans: By invoking the method through the pointer and utilizing the base class's pointer to point to an object of the derived class.

Q7.If a base class Declare a Function to be Virtual and derived class does not use the keyword virtual when Inherited by third Generation class?

Ans. When defining overriding functions in a derived class, the virtual keyword can be used, but it is not necessary because overrides of virtual functions are always virtual. A base class's virtual functions must be defined unless they are explicitly stated using the pure- specifier.

Q8. What is the protected keyword used for?

Ans. The protected keyword is an access modifier that makes constructors, methods, and attributes accessible to other objects in the same package as well as subclasses.