OOP LABTASK

REPORT

NAME: Tauheed Ahmad

SAP ID: 46859

TASKNO1:

To gain better understanding of polymorphic and virtual functions we start with simple example: #include<iostream> using namespace std; 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:

Here we note use the keyword virtual so the output is only Base class. Virual function is base class member function that you can redefine in a derived class to achieve polymorphism.

THE NEXT EXAMPLE WHERE WE USE VIRTUAL FUNCTION:

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

```
class   Derived   :   public   Base{
        public:
                void testfunction();
};
```

OUTPUT:

Now, we use virtual function, so the output is as follows

TASKNO2:

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.

```
#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 ();

return 0;
}
```

OUTPUT:

Now, weremove thevirtual keyword fromclassmammal:

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

class          Mammal{
public:
Mammal(void);
~Mammal(void);
 void  Move  ()  const  ;
 void Speak () const ;
```

protected :

int itsAge;

};

Now, the output becomes:

OUTPUT:

NOW, weadd a pointer pdog2 this will happen

OUTPUT:

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;

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;

}
```

```
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;

}

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;
}
```

```
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;
}

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(){
  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;
}
```

ANSWER THE FOLLOWING QUESTIONS:

1. Are inherited members and functions passed along to subsequent generations? If Dog derives from Mammal, and Mammal derives from Animal, does Dog inherit Animal's functions and data?

ANS: Yes, Dog derives from Mammal, and Mammal derives from Animal, Dog will inherit all the members of Mammal and Animal.

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

ANS: No, a derived class cannot make a public base function private.

3. Why not make all class functions virtual?

ANS: Because a function only needs to be virtual iff a derived class will implement that function in a different way.

4. 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: If the virtual function SomeFunc() in the base class is overloaded to take either an integer or two integers, and the derived class overrides the form taking one integer, when a pointer to a derived object calls the two-integer form, the version of the function defined in the base class will be called.

SOME OTHER EXERCISES:

- What is a v-table?

VTables (or virtual tables) are arrays of virtual functions. Virtual functions are member functions of a C++ class that can be redefined in a child class. These are used to implement runtime polymorphism in C++ through dynamic dispatching.

- What is a virtual destructor?

A virtual destructor is a destructor function in C++ that is declared as virtual in a base class, and can be overridden by a derived class. When an

object is deleted through a pointer to a base class, the virtual destructor ensures that the destructor of the most derived class is called first When an object is deleted through a pointer to a base class, the virtual destructor ensures that the destructor of the most derived class is called first.

- How do you show the declaration of a virtual constructor?

- How can you create a virtual copy constructor?

A virtual copy constructor is a way to create a copy of an object using a pointer or reference to its base class, while preserving the object's dynamic type.

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

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 specify the base class.

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

To invoke a base member function from a derived class in which you have not overridden that function, you can use the scope resolution operator '::' to access the base class's version of the function.

- 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, if a base class declares a function to be virtual and a derived class overrides that function without using the virtual keyword, the function is still considered to be virtual.

- What is the protected keyword used for?

The protected keyword is an access modifier used for attributes, methods and constructors, making them accessible in the same subclasses.