

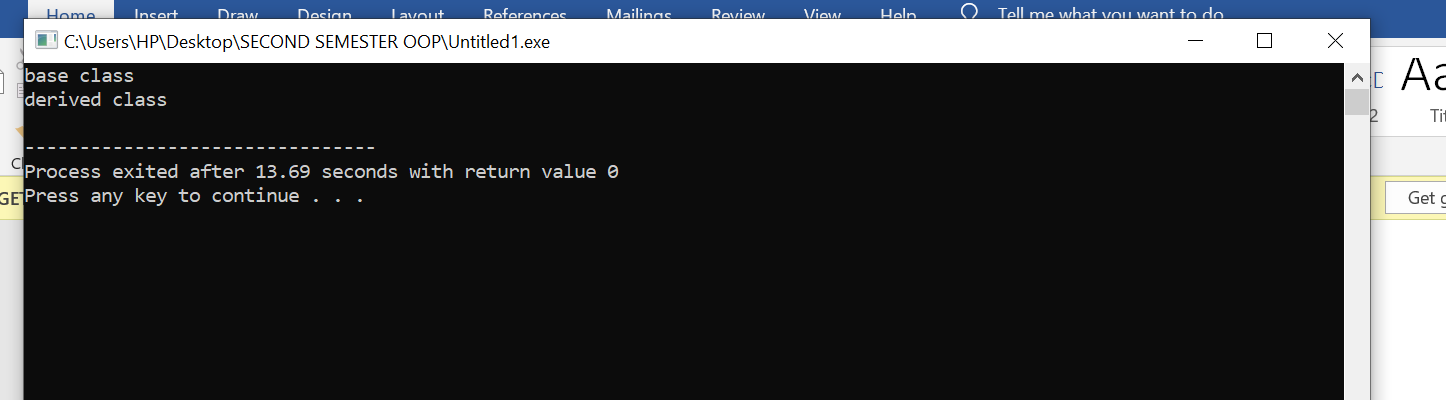
**NAME:MUHAMMAD IBRAHIM**

**SAPID:(46935)**

**SUBMITTED TO:SIR SHEHZAD SB.**

**REPORT LAB AFTER MID:**

**TASK NO 1:**

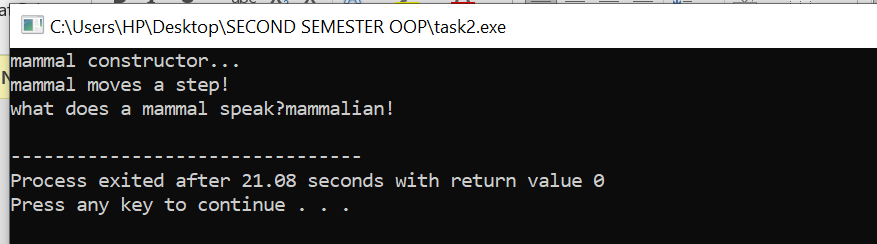
**Output :**

IN THIS code, It used virtual function with base class then it prints one time Base class and one time Derived class

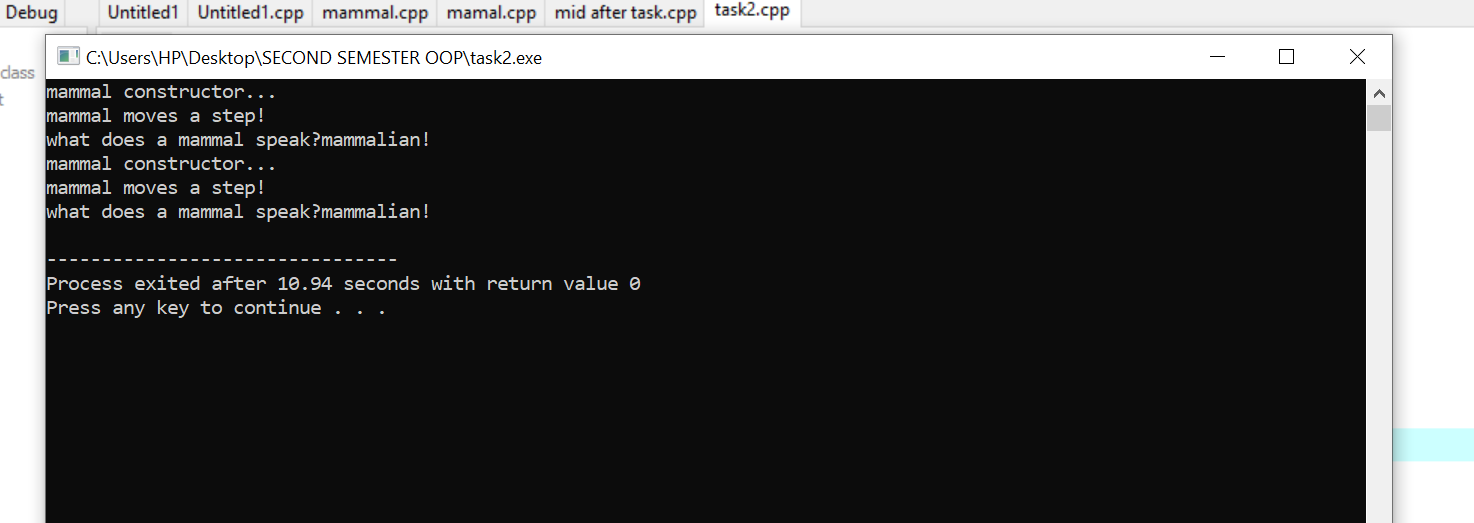
**Task 2.**

**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.**

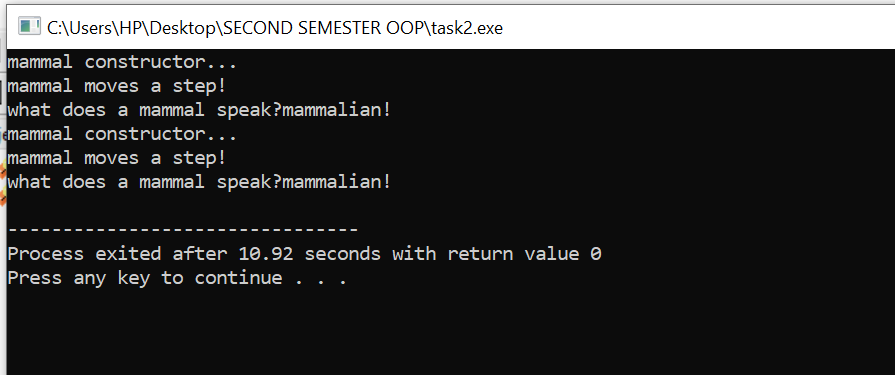
I inherit a Dog class with Mammal used public functions of mammal in Dog class then we create a pointer based object then it prints



After that I used the other pointer object with the name pdog2 then compiler give me this output.



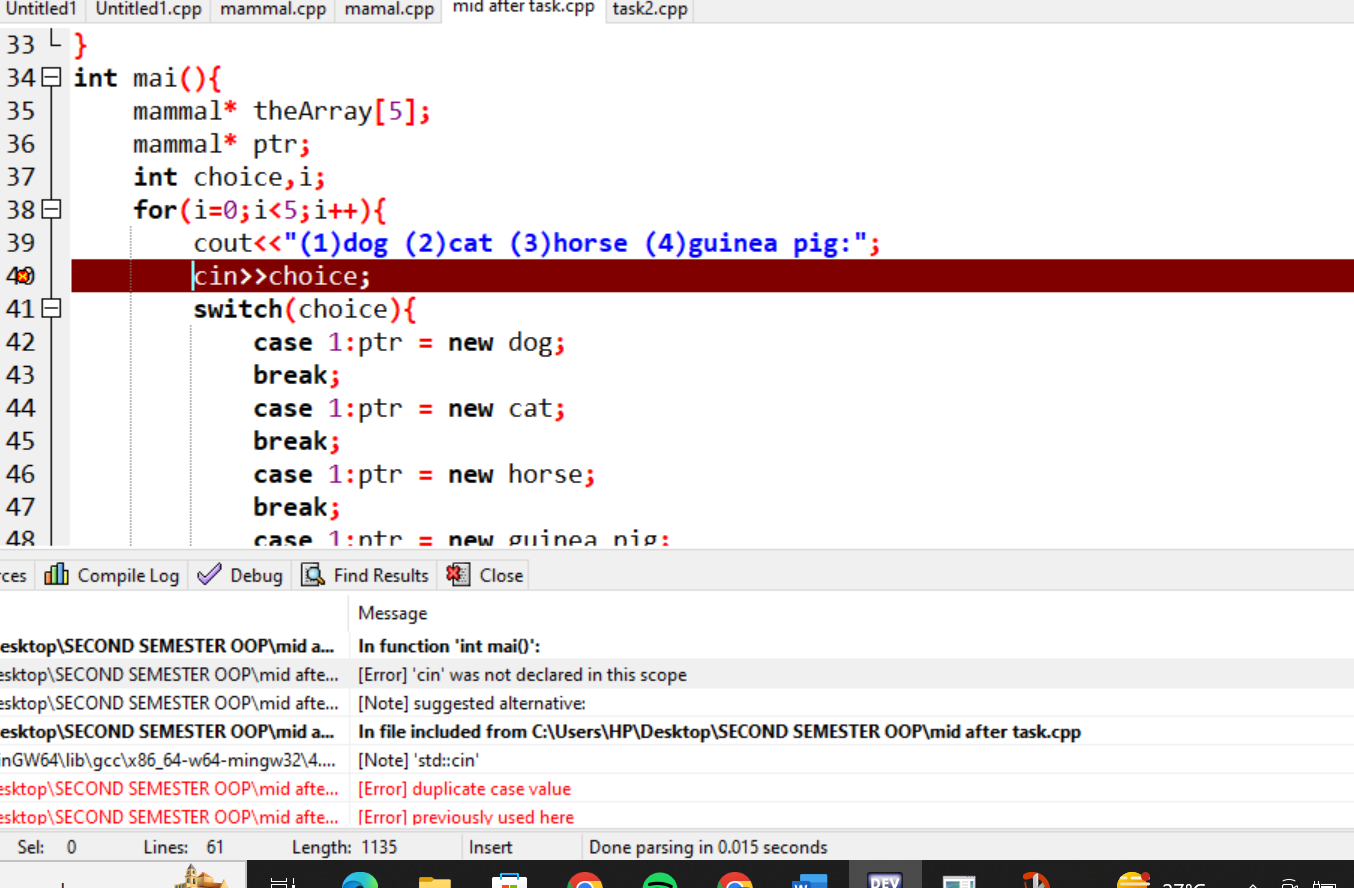
I removed the virtual keyword from the void function in mammal class. And then it give me this output.



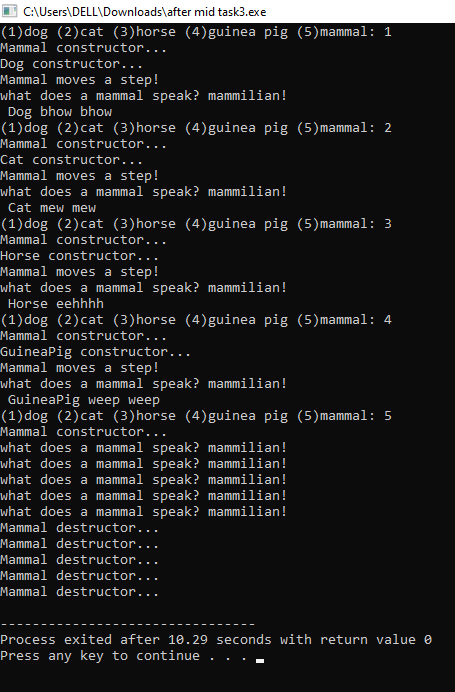
**Task no 3:**

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

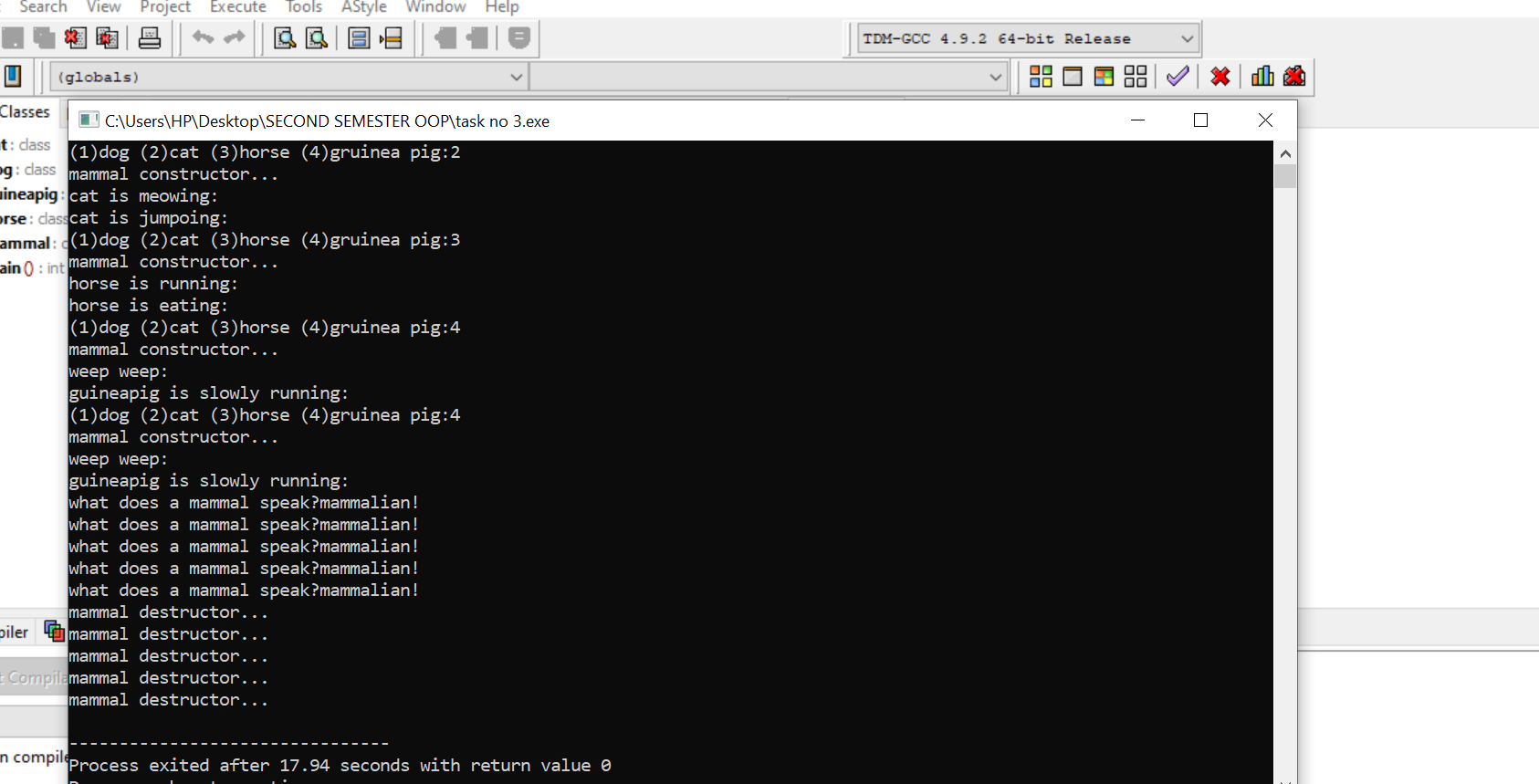
* **If i run the program it will give me an error (cin>>choice;)in this line .if we use std::cin then it will not give an error**

****

**FIRSTLY it will give me output like this..**



I used additional classes for Cat, Horse, and Guinea Pig and I overriding the moves and speaking methods and then compiler give me the input like this.



**Some questions that you should start to understand**

**Can a derived class make a public base function private?**

**With private inheritance, public and protected member of the base class become private members of the derived class.** That means the methods of the base class do not become the public interface of the derived object. However, they can be used inside the member functions of the derived class.

**Why not make all class functions virtual?**

Making all class functions virtual can have performance and design implications, and therefore may not be the best approach in all cases.

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?

When a pointer to a derived object calls the two-integer form of the overloaded virtual function that is defined in the base class, the version of the function that is called depends on how the derived class overrides the function. If the derived class overrides only the one-integer form of the function, and does not provide an implementation for the two-integer form, then the two-integer form in the base class will be called when invoked through a pointer to the derived class object.

**Here are some more questions**

**What is a v-table?**

A v-table, short for virtual function table, is a mechanism used by some object-oriented programming languages, such as C++, to implement virtual functions and achieve dynamic polymorphism.

**What is a virtual destructor?**

A virtual destructor is a special kind of destructor in C++ that is declared as virtual in the base class and overrides the destructor in any derived classes.

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

In C++, virtual constructors are not allowed. This is because constructors are responsible for initializing the object, including setting up the v-table pointer and other internal state, and making a constructor virtual would violate this responsibility.

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

In C++, there is no such thing as a virtual copy constructor because the copy constructor is a special kind of constructor that takes a reference to an existing object of the same type and creates a new object that is a copy of it.

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

In C++, you can invoke a base member function from a derived class using the scope resolution operator ‘**::’**.

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

In C++, you can invoke a base member function from a derived class even if you have not overridden that function by simply calling it directly. When you call a member function on a derived class object, the compiler will first look for an implementation of that function in the derived class. If it does not find one, it will look for an implementation in the base class.

**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 function is declared as virtual in a base class, and a derived class overrides that function without using the virtual keyword, the function is still virtual in the derived class and any further derived classes.

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

In C++, the protected keyword is used as an access specifier for class members. Class members that are declared as protected can be accessed by the class itself, as well as by any derived classes.

**Some more exercises**

**BUG BUSTERS: What is wrong with this code snippet?**

void SomeFunction (Shape);

Shape \* pRect = new Rectangle;

SomeFunction(\*pRect);

The issue with this code snippet is that the SomeFunction function takes its argument Shape by value instead of by reference or pointer. This means that when SomeFunction(\*pRect) is called, the Rectangle object pointed to by pRect is sliced, and only the Shape part of the object is passed to SomeFunction. As a result, any data or functionality specific to the Rectangle class will be lost, and SomeFunction will only be able to operate on the Shape part of the object.

**BUG BUSTERS: What is wrong with this code snippet?**

class Shape() { public: Shape(); virtual ~Shape();

virtual Shape(const Shape &);

};

There are a few issues with this code snippet:

1. The constructor for Shape is declared with empty parentheses, but it doesn't need to be declared at all since the default constructor will be automatically generated by the compiler. If you want to define your own constructor, you should provide an implementation with a body and any necessary parameters.
2. The virtual destructor for Shape is declared correctly with the virtual keyword, but it shouldn't also be declared as a constructor with the same name. Constructors in C++ don't have a return type, so the line virtual Shape(const Shape &); is actually declaring a virtual function that takes a const Shape& parameter and returns a Shape object. This is not what you want for a constructor.
3. Constructors can't be virtual in C++, so the virtual keyword shouldn't be used on the constructor.Top of Form

Top of Form

**-------------------------THE END­----------------------------------------------**