** RIPHAH INTERNATIONAL UNIVERSITY LAHORE**

NAME: **Anousha Ijaz**

SAP: **47544**

PROGRAM & SEM: **BSCS 2A**

OOP LAB TASK 5 MAY REPORT

**TASK 1**

**CREATION OF HEADER FILE:**

**TASK1.H**

#include<iostream>

using std::cout;

using std::endl;

class Base{

public:

void testFunction ();

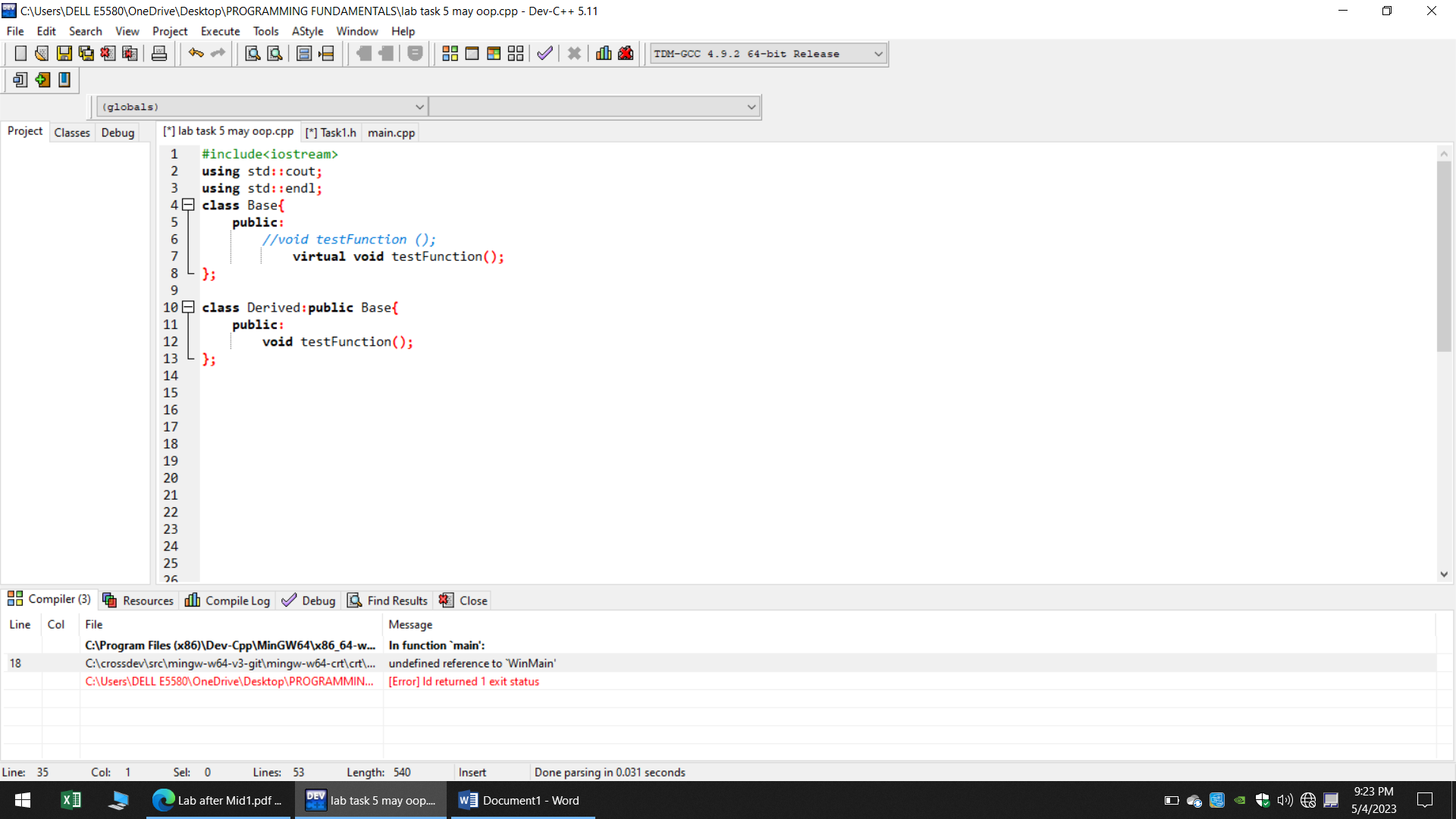
};

class Derived:public Base{

public:

void testFunction();

};



#include"Task1.h"

void Base::test Function()

{

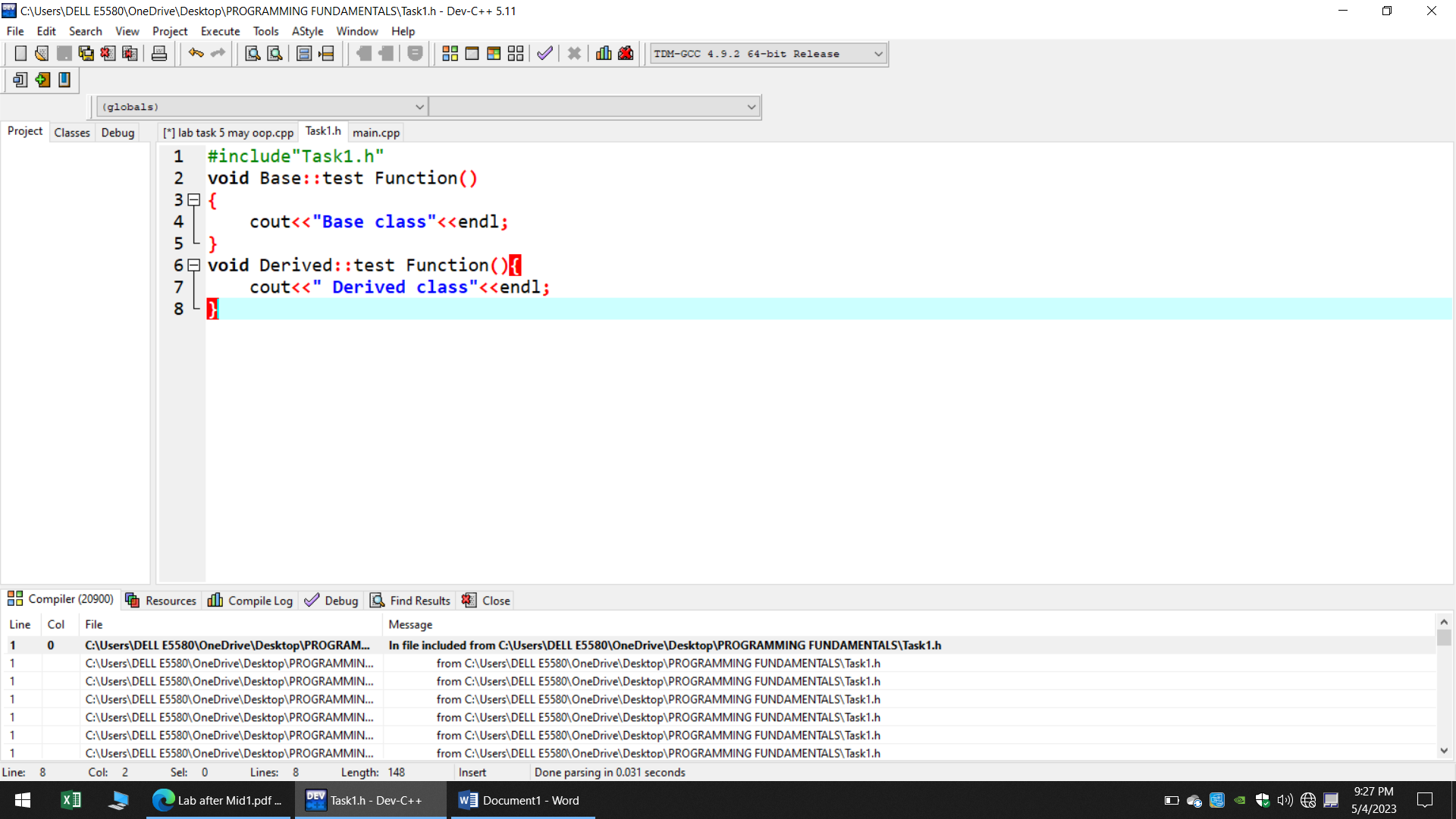
cout<<"Base class"<<endl;

}

void Derived::test Function(){

cout<<" Derived class"<<endl;

}



**TASK 1.CPP:**

#include"Task1.h"

int main (void)

{

Base\*ptr= new Base;

ptr-> testFunction();

delete ptr;

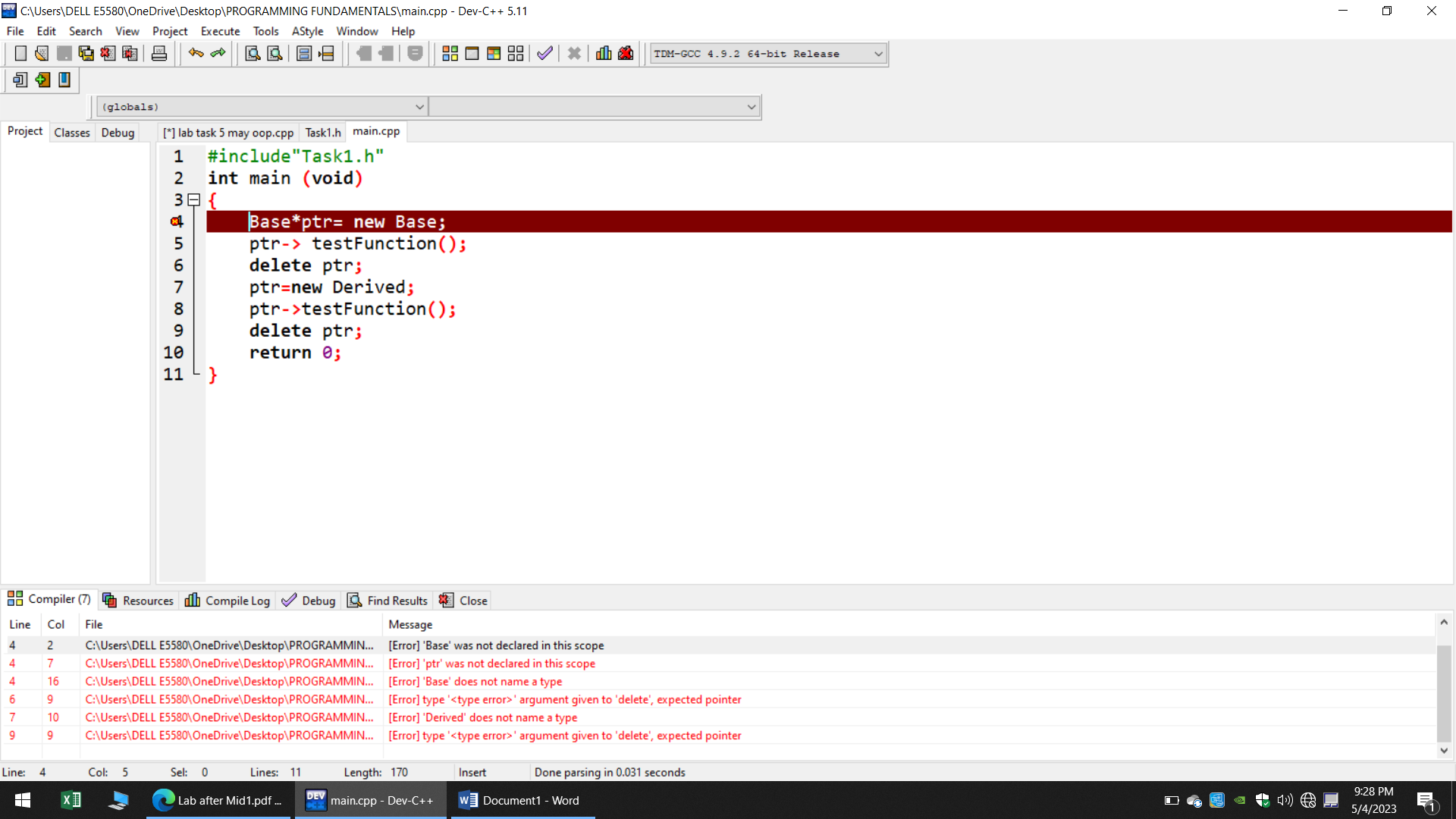
ptr=new Derived;

ptr->testFunction();

delete ptr;

return 0;

}



**CREATION OF MAIN BODY:**

#include<iostream>

using std::cout;

using std::endl;

class Base{

public:

**Addition of virtual function**

virtual void testFunction();

};

class Derived:public Base{

public:

void testFunction();

};

int main (void)

{

Base\*ptr= new Base;

ptr-> testFunction();

delete ptr;

ptr=new Derived;

ptr->testFunction();

delete ptr;

return 0;

}

void Base::testFunction()

{

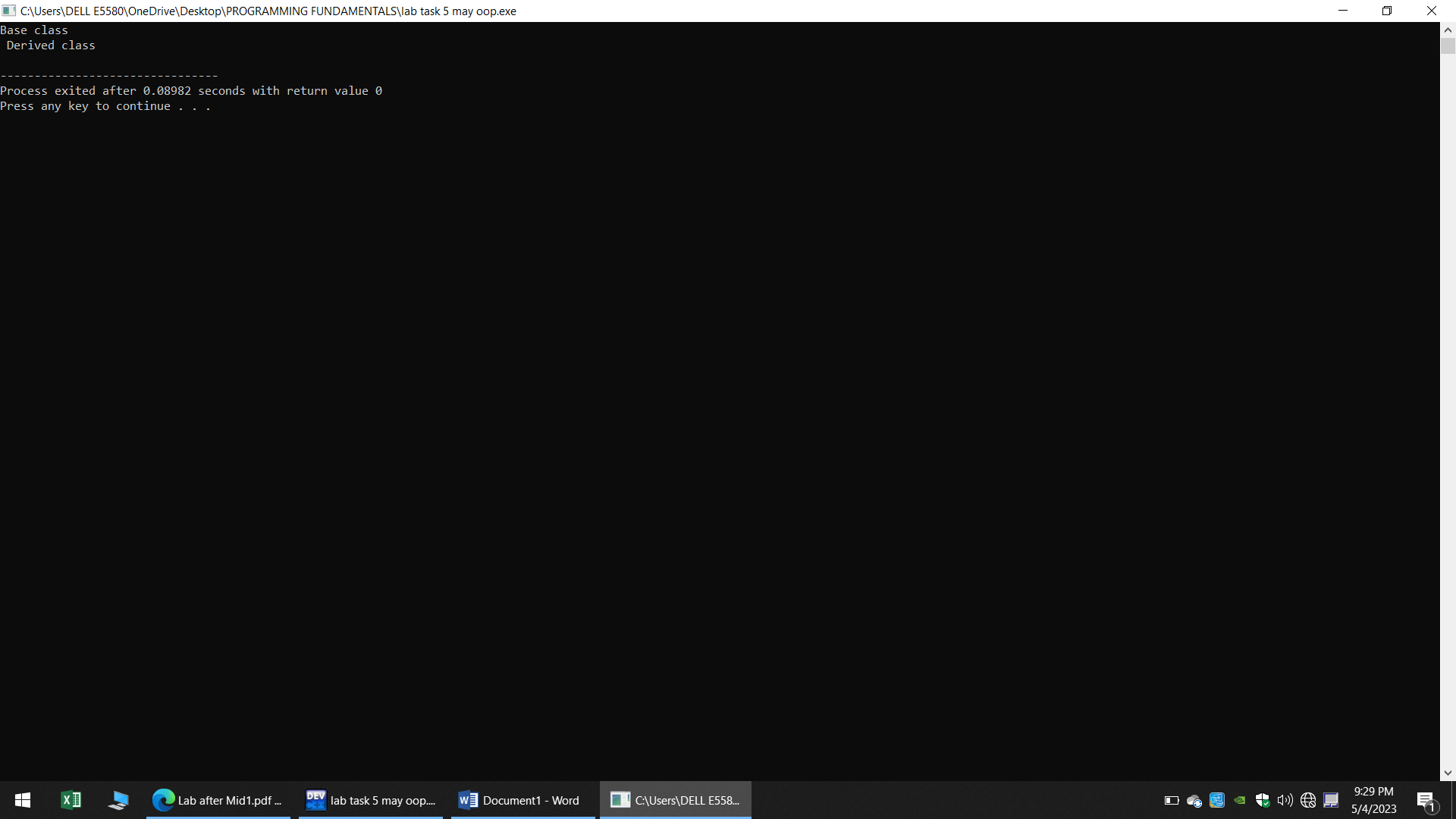
cout<<"Base class"<<endl;

}

void Derived::testFunction(){

cout<<" Derived class"<<endl;

}



**When I compiled all code together it shows output such that given in the picture**

Base class

Derived class

**TASK 2**

#include <iostream >

using namespace std;

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;

}

// Dog

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;

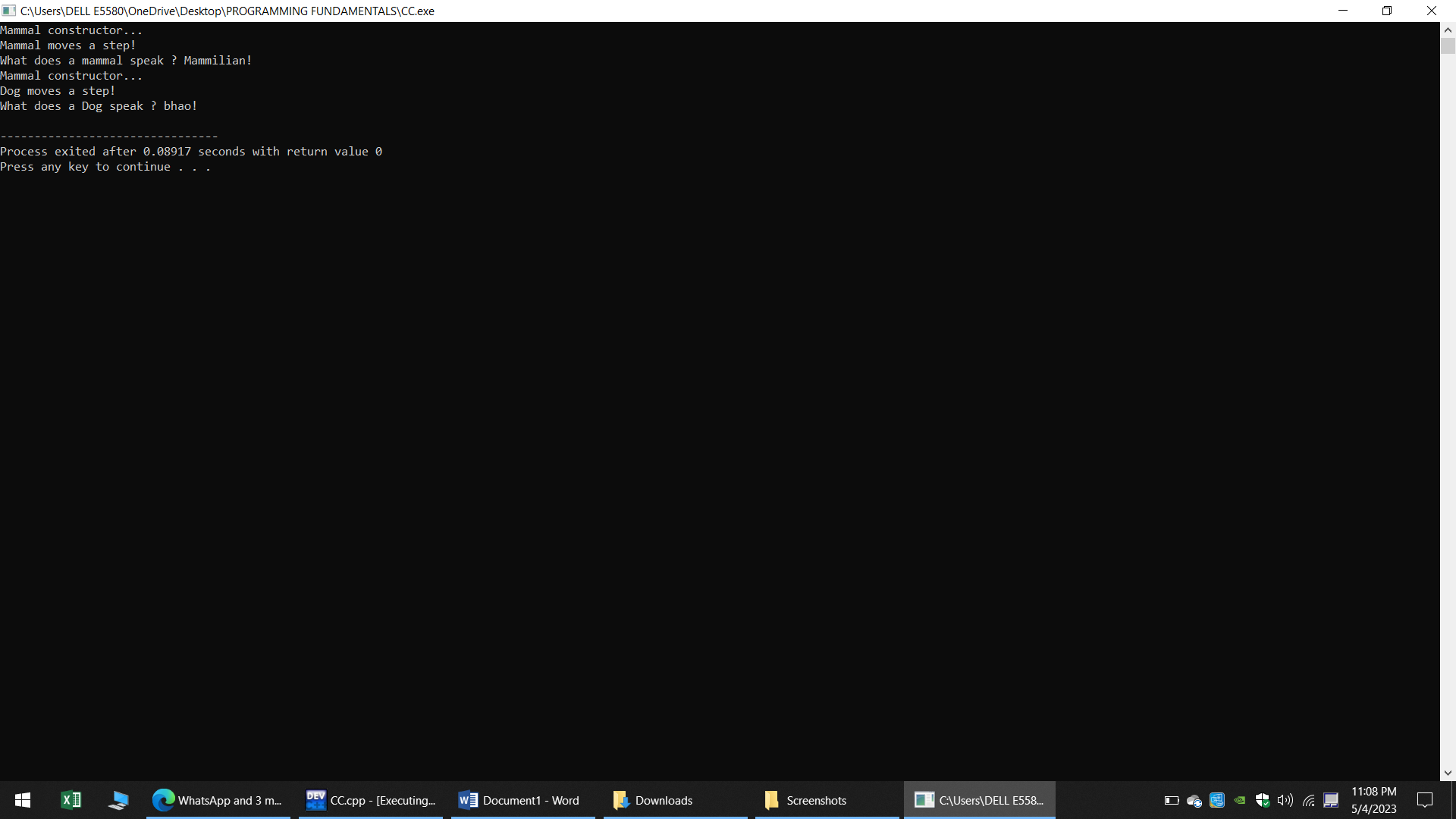
pDog2->Move ();

pDog2->Speak();

return 0 ;

}

**OUTPUT:**



**Task 3**

#include <iostream>

using namespace std;

class Mammal {

public:

Mammal(void );

~Mammal(void );

void Move() const;

void Speak() const;

protected :

int itsAge ;

};

class Dog :public Mammal{

public:

void Move() const;

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::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;

}

// Dog

void Dog ::Move ()const

{

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

}

void Dog ::Speak ()const

{

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

}

//CAT

void Cat ::Move ()const

{

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

}

void Cat ::Speak ()const

{

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

}

// HORSE

void Horse ::Move ()const

{

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

}

void Horse ::Speak ()const

{

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

}

// GuineaPig

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\* theArray[5];

Mammal\* ptr;

int choice ,i;

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

{

cout<<"(1)dog (2)Cat (3)horse (4)GuineaPig : ";

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();

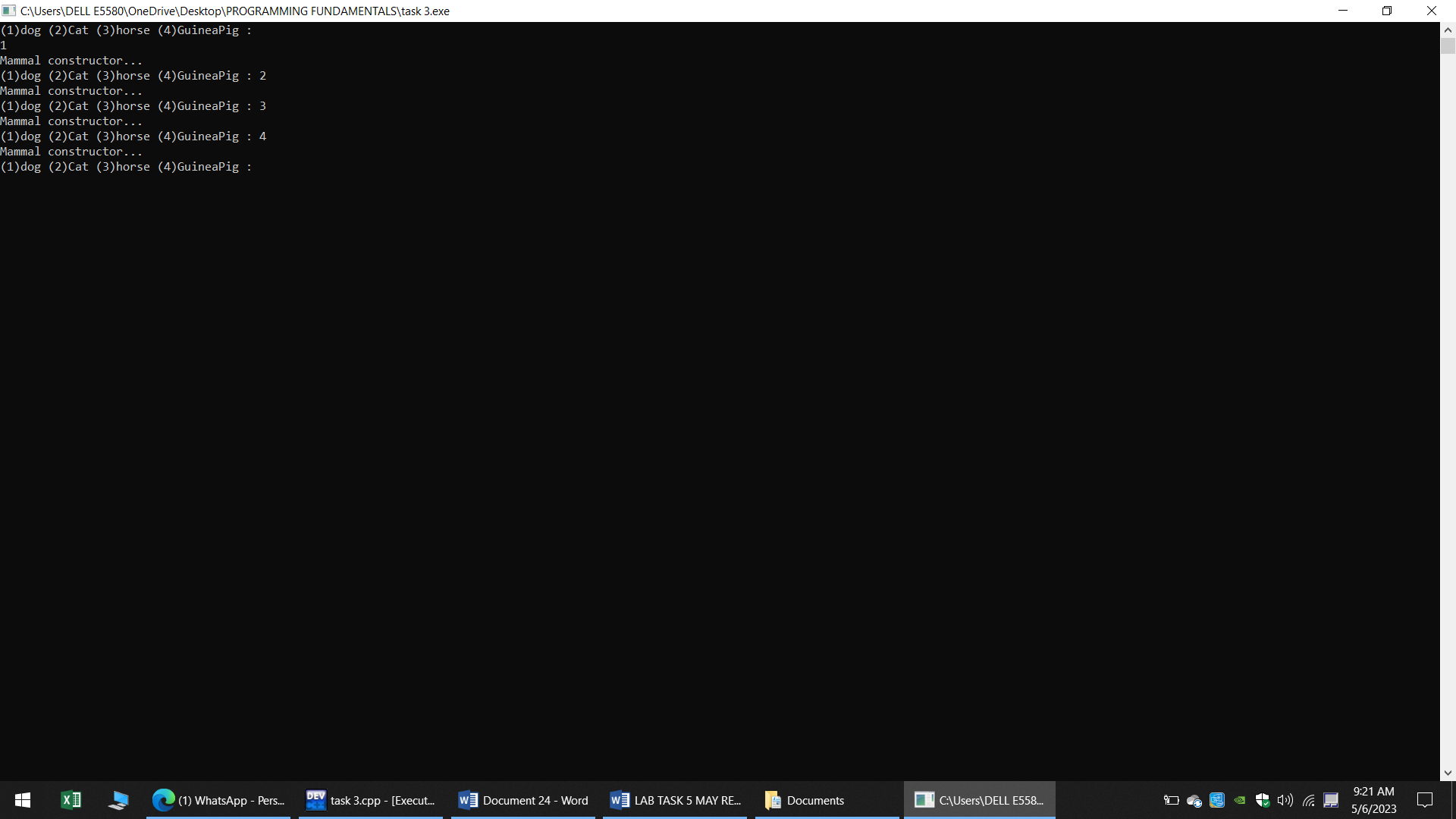
for (i=0;i<5;i++) //Always free dynamicallu Allocated objects

delete theArray[i];

return 0;

}

**Output:**



**QUESTIONS AND ANSWERS:**

**Q1:** Can a derived class make a public base function private?

Ans: No, a derived class cannot make a public base function private. In object-oriented programming, a derived class inherits the members of its base class. If a member function is declared as public in the base class, it will be inherited as public in the derived class

**Q2:** Why not make all class functions virtual?

**Ans**: Making all class functions virtual can have performance implications because virtual functions are implemented using dynamic dispatch, which incurs some overhead compared to non-virtual functions. Additionally, making all functions virtual can make the class hierarchy more difficult to understand and maintain, as it can increase the complexity of the code

**Q3**: If a function (Some Func( )) 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**: The base class's two-integer version of the function will be called. This is because the derived class has only overridden the one-integer version of the function. The call will be dynamically dispatched to the appropriate version of the function based on the actual type of the object being pointed to.

**Q4:** What is a v-table?

**Ans**: A v-table, short for virtual table, is a data structure used by compilers to implement polymorphic behavior in object-oriented programming languages that use virtual functions. A v-table is an array of function pointers that is associated with each class that has virtual functions. The v-table is created by the compiler and is used to dynamically dispatch calls to virtual functions at runtime.

**Q5:** What is a virtual destructor?

**Ans**: A virtual destructor is a destructor that is declared as virtual in a base class and is intended to be overridden in derived classes.

**Q6**: How do you show the declaration of a virtual constructor?

**Ans**: if you want to create a virtual constructor ,you can use a factory method a factory member is a static function of a class that creates and returns object of that class but it can be overridden by derived classes to return objects of the appropriate type.

**Q7:** How can you create a virtual copy constructor?

**Ans:** Sometimes we may need to construct an object from another existing object. Precisely the copy constructor does the same. The initial state of new object will be based on another existing object state. The compiler places call to copy constructor when an object being instantiated from another object. However, the compiler needs concrete type information to invoke appropriate copy constructor.

**Q8:** How do you invoke a base member function from a derived class in which you have not overridden that function?

**Ans:** You must use the scope resolution operator, “::” to access the overridden function. Another way to access the overridden function is by using the pointer of the base class to point to an object of the derived class and calling the function through the pointer

**Q9:** 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?

**Ans**: Yes, the virtuality is inherited and cannot be turned off.

**Q10**: What is the protected keyword used for?

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