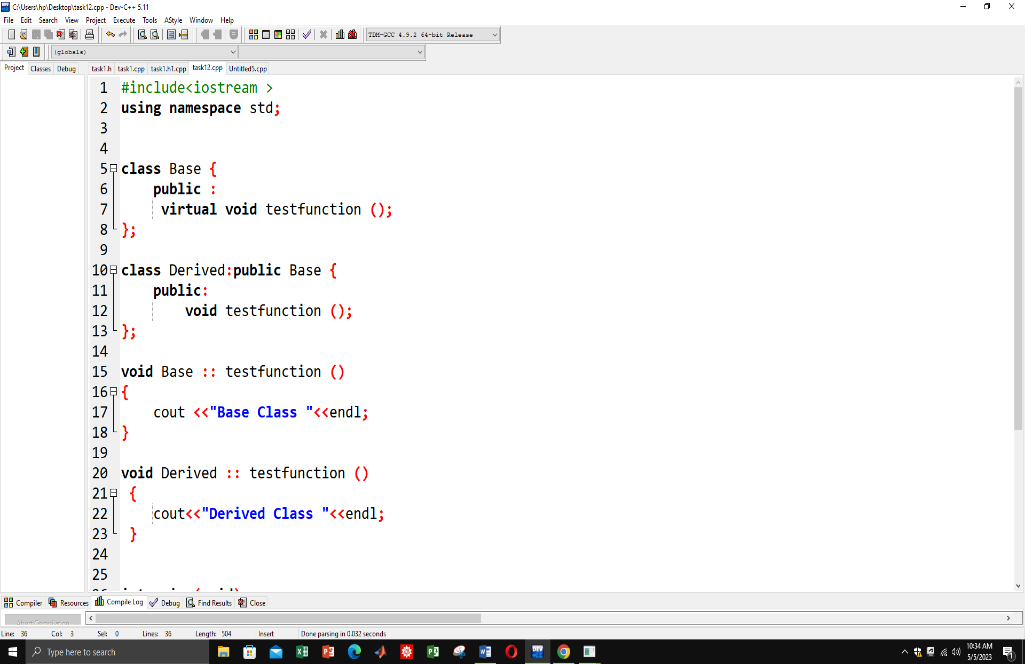
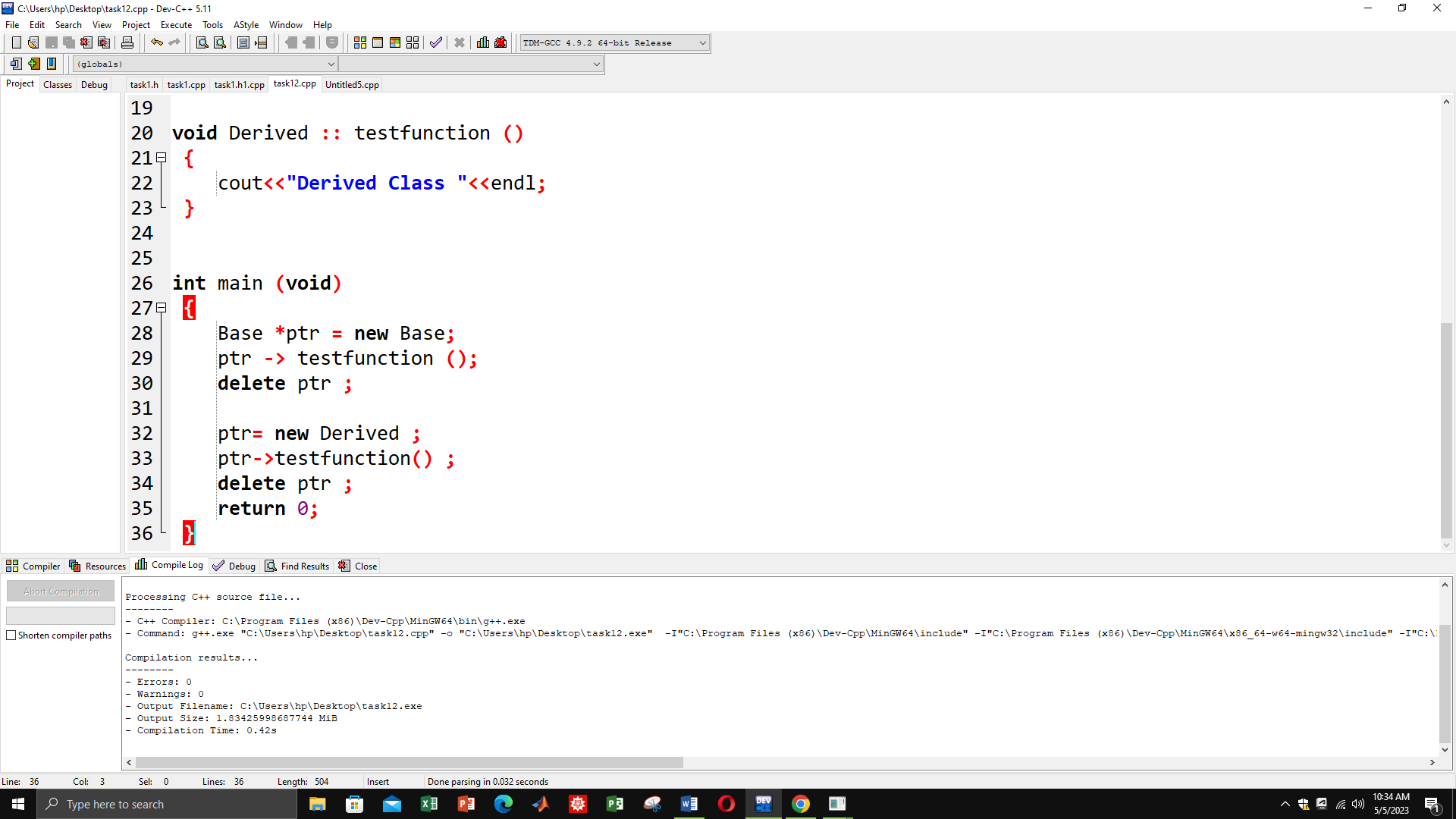
**Name : Usman khan**

**Sap ID : 46729**

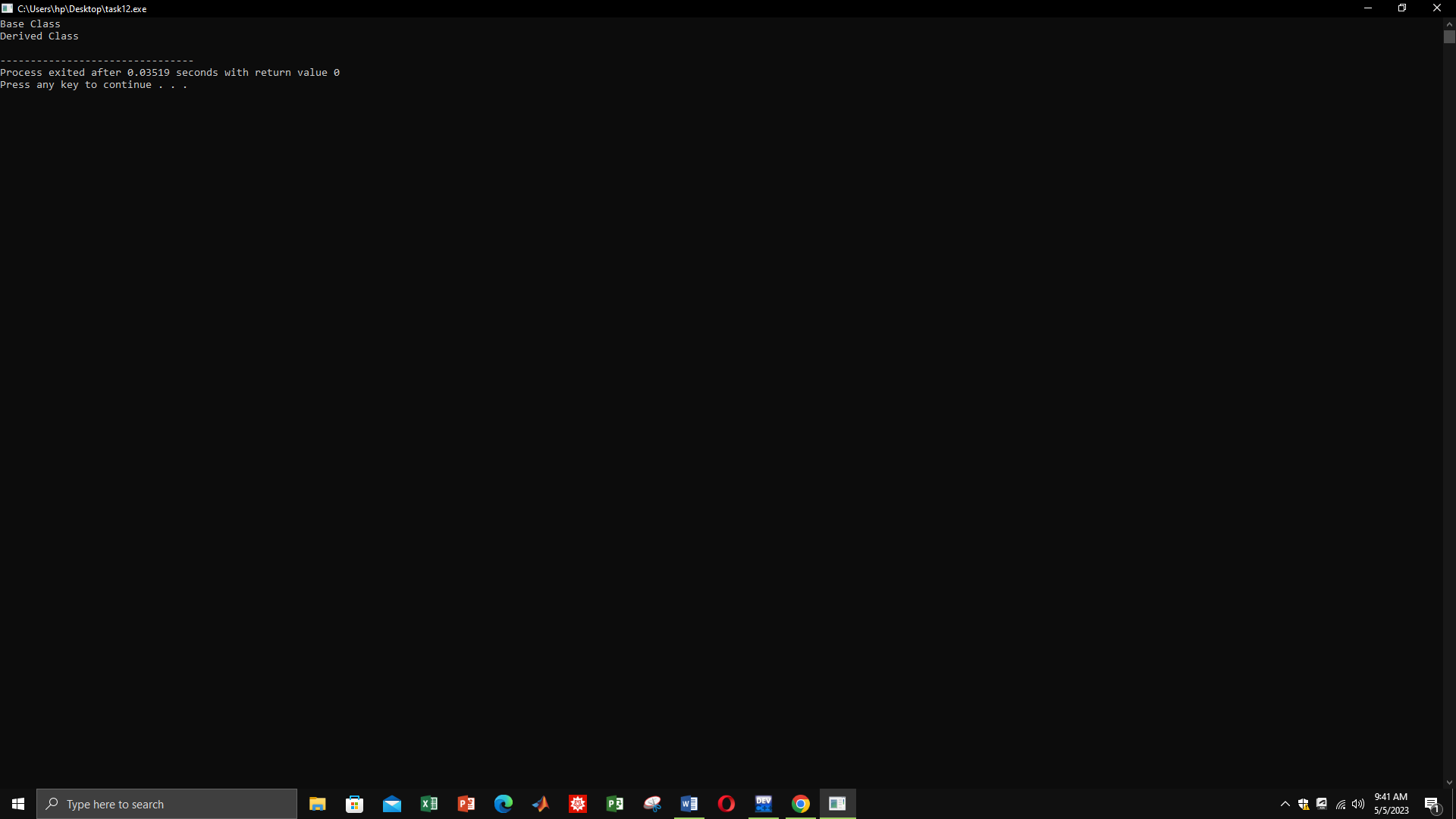
TASK 1:

Code1:



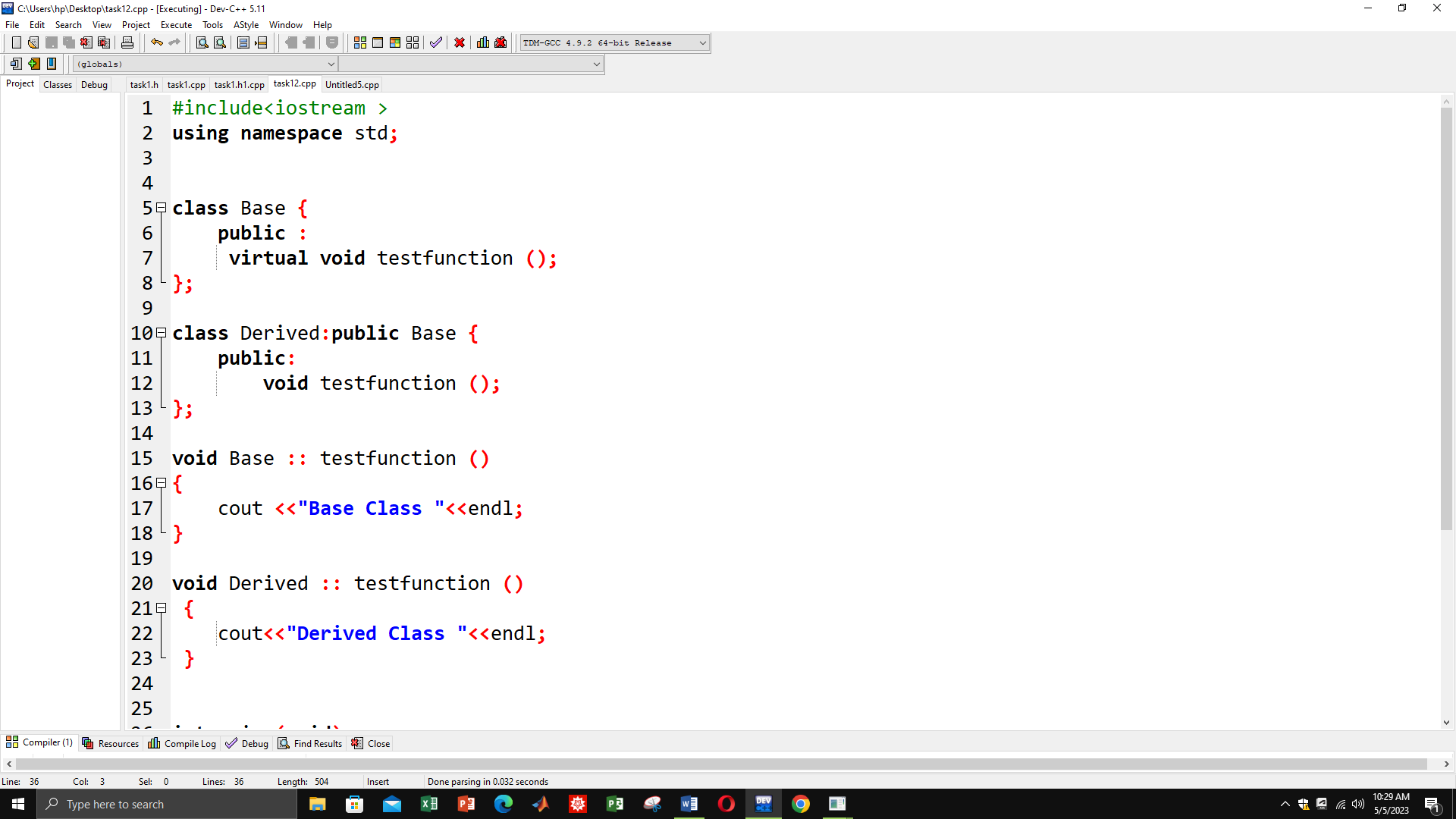
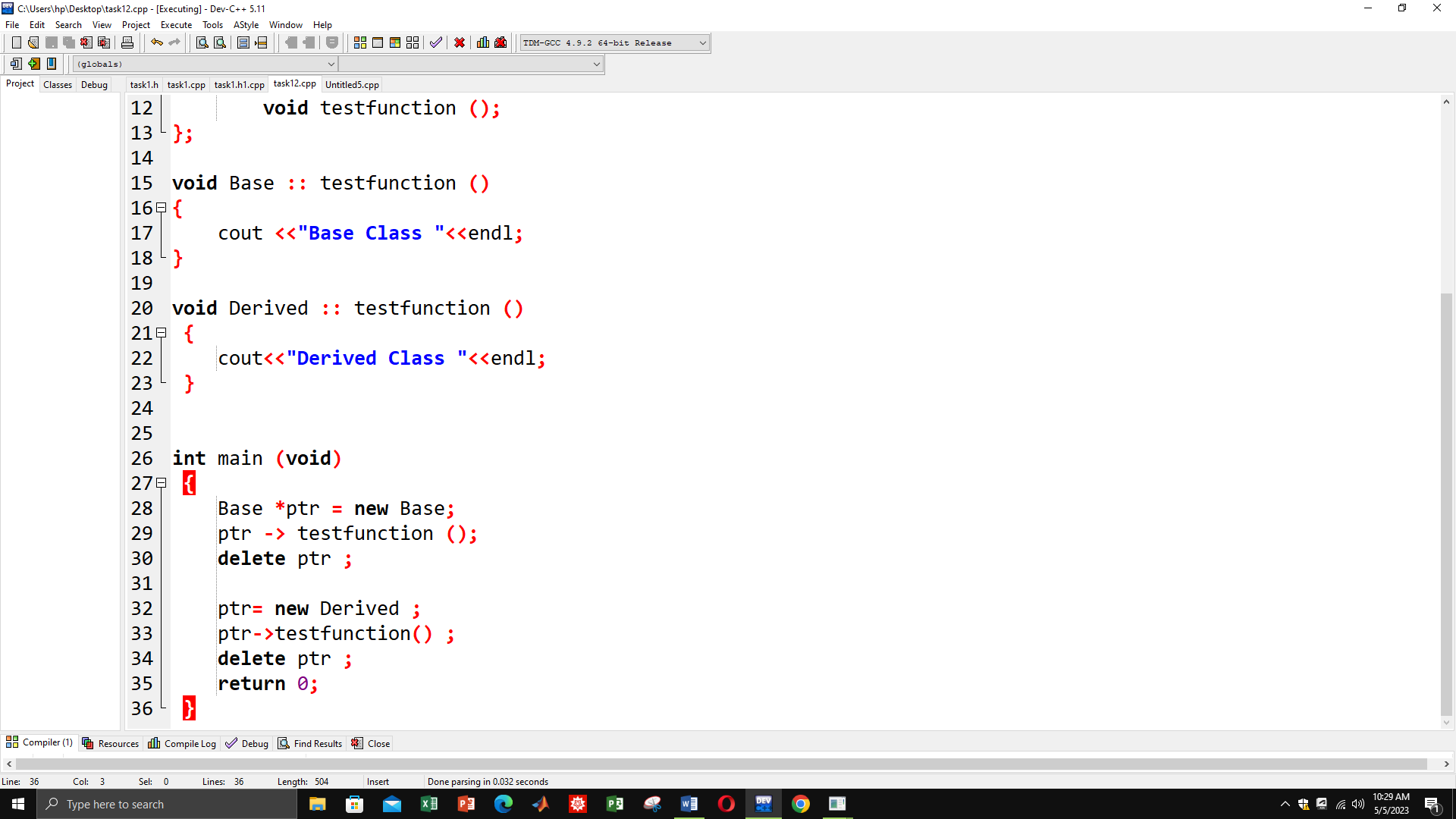


Output:

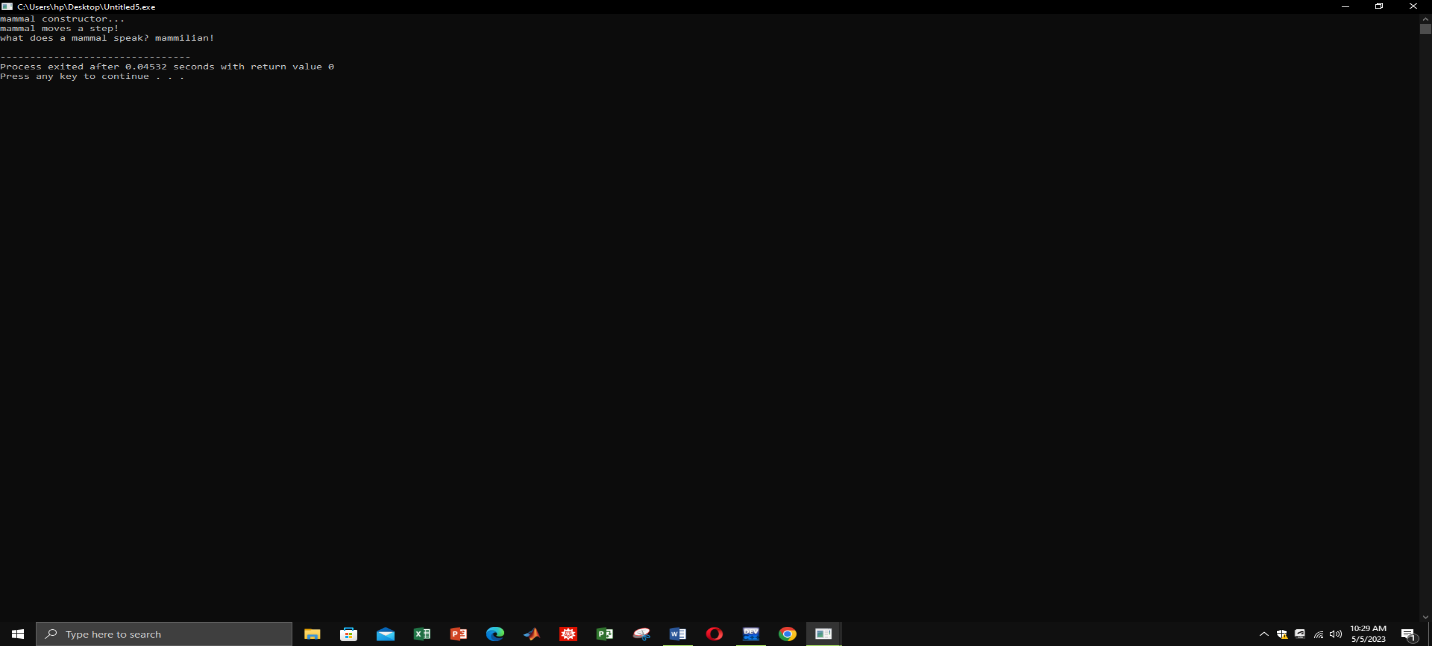


TASK 2:

CODE 2:



Output:



**TASK 3**

**CODE:**

#include<iostream>

using namespace std;

#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;

};

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();

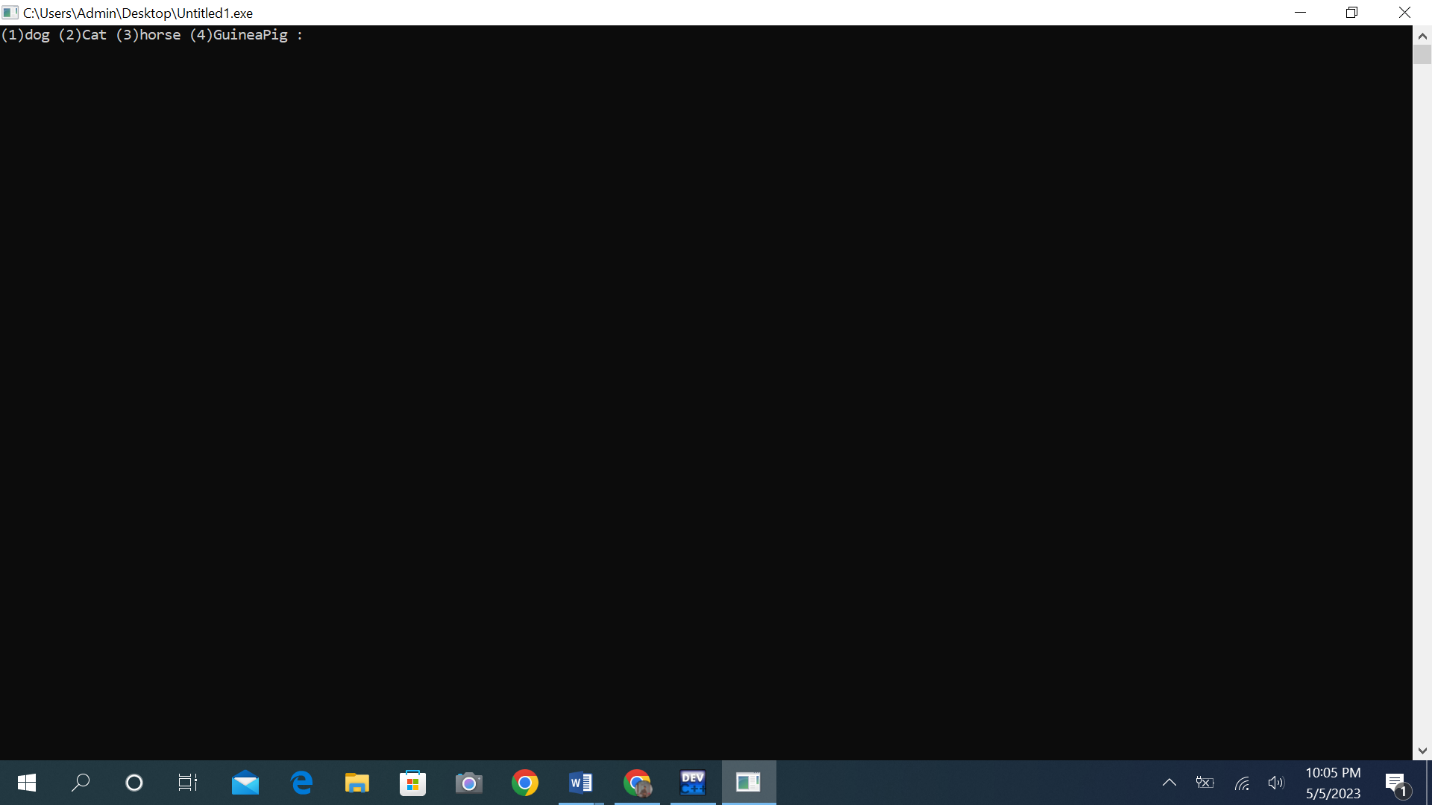
for (i=0;i<5;i++) //Always free dynamicallu Allocated objects

delete theArray[i];

return 0;

}

**OUTPUT**

****

**QUESTIONS**

**Q1: Can a derived 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: In C++, constructors cannot be declared as virtual. This is because constructors are called during object construction, and the virtual table is not yet fully initialized at that point, so dynamic dispatch based on the actual object type is not yet possible.

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

ANS: In C++, it's not possible to declare a constructor as virtual, including the copy constructor. However, it's possible to create a virtual copy constructor.

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

ANS: 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 explicitly specify the base class name and function name.

**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, if a base class declares a function to be virtual, and a derived class does not use the virtual keyword when overriding that function, it is still virtual when inherited by a third-generation class. The virtual keyword is not required when overriding a virtual function, but it is good practice to use it for clarity.

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

ANS: The protected keyword in C++ is used to specify access control for class members. Members declared as protected are accessible within the class and its derived classes, but not from outside the class hierarchy.

**Q11:** **How do you invoke a base member function from a derived class in which you have not overridden that function?**

**ANS:** To invoke a base member function from a derived class in which you have not overridden that function, you can simply call the function by its name**.**

**Q12:** **How do you invoke a base member function from a derived class in which you have not overridden that function?**

**ANS:**

To invoke a base member function from a derived class in which you have not overridden that function, you can simply call the function by its name.

**Q13:** **Show the declaration of a virtual function that takes an integer parameter and returns void?**

**Ans:**

class MyClass {

public:

virtual void myFunction(int myParam) = 0;

};

**Q14:**

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

class Shape {

public:

virtual double area() const = 0;

};

class Rectangle : public Shape {

public:

Rectangle(double width, double height);

double area() const override;

protected:

double width\_;

double height\_;

};

class Square : public Rectangle {

public:

Square(double side);

};

**Q15: If, in Exercise 14, Shape takes no parameters, Rectangle takes two(length and width), but Square takes only one (length), show the constructor initialization for Square?**

**Ans:**

Square::Square(double side)

: Rectangle(side, side)

{

// Any additional initialization specific to Square can go here.

}

**Q16: Write a virtual copy constructor for the class Square?**

**Ans:**

class Shape {

public:

virtual ~Shape() {}

virtual Shape\* clone() const = 0;

virtual double area() const = 0;

};

class Rectangle : public Shape {

public:

Rectangle(double width, double height);

double area() const override;

Rectangle\* clone() const override;

protected:

double width\_;

double height\_;

};

class Square : public Rectangle {

public:

Square(double side);

Square\* clone() const override;

};

Rectangle\* Rectangle::clone() const {

return new Rectangle(\*this);

}

Square\* Square::clone() const {

return new Square(\*this);

}

**Q17: BUG BUSTERS: What is wrong with this code snippet?**

**void SomeFunction (Shape); Shape \* pRect = new Rectangle; SomeFunction(\*pRect);**

**Ans:**

Perhaps nothing. SomeFunction expects a Shape object. You've passed it a Rectangle"sliced" down to a Shape. As long as you don't need any of the Rectangle parts, this will be fine. If you do need the rectangle parts, you'll need to change someFunction to take a pointer or a reference to a Shape.

**Q18: BUG BUSTERS: What is wrong with this code snippet?**

**class Shape() { public:**

**Shape();**

**virtual ~Shape();**

**virtual Shape(const Shape &);**

**};**

**Ans :**

You can not declare a copy constructor to be virtual.