** RIPHAH INTERNATIONAL UNIVERSITY LAHORE**

NAME: **ANOUSHA IJAZ**

SAP: **47544**

PROGRAM & SEM: **BSCS 2A**

SUBJECT: **OBJECT ORIENTED PROGRAMMING**

**ASSIGNMENT NO. 4**

**PART 1:**

**1. Explain what polymorphism is and how it relates to object-oriented programming.**

Polymorphism is a feature of object-oriented programming languages that allows a specific routine to use variables of different types at different times. Polymorphism in programming gives a program the ability to redefine methods for derived classes

**2. What is the difference between static and dynamic polymorphism?**

Dynamic polymorphism happens at run time, and static polymorphism at compile time. Dynamic polymorphism typically requires a pointer indirection at run time (read the post "Demystifying virtual functions, V table, and VPTR in C++"), but static polymorphism has no performance costs at run time.

**3. Describe the two types of polymorphism in C++.**

|  |  |
| --- | --- |
| **COMPILE-TIME** | **RUN-TIME** |
| Compile-time polymorphism is also known as static or early binding polymorphism. | Run-time polymorphism is also known as dynamic or late binding polymorphism. |
| The function calls are resolved by the compiler. | The function calls are not resolved by the compiler. |

**4.What is a virtual function? Explain why it is used.**

A virtual function in C++ is a base class member function that you can redefine in a derived class to achieve polymorphism. You can declare the function in the base class using the virtual keyword. A virtual function in C++ helps ensure you call the correct function via a reference or pointer. The C++ programming language allows you only to use a single pointer to refer to all the derived class objects

**5.Can a class have both virtual and non-virtual functions? Explain your answer**

A virtual function is a member function that can be overridden in a derived class. a non-virtual function is a member function that is not designed to be overridden in derived classes. A class can have both virtual and non-virtual functions. The choice of whether to make a function virtual or non-virtual depends on the specific requirements of the class and its intended use in a program.

**PART 2**

**1. Write a C++ program that demonstrates the concept of function overloading.**

#include<iostream>

using namespace std;

int add(int a,int b){

cout<<"function 1"<<endl;

return(a+b);

}

double add(double a,double b){

cout<<"function 2"<<endl;

return(a+b);

}

int add(int a,int b,int c){

cout<<"function 3"<<endl;

return(a+b+c);

}

double add(int a,double b){

cout<<"function 4"<<endl;

return(a+b);

}

double add(double a,int b){

cout<<"function 5"<<endl;

return(a+b);

}

double add(double a,int b ,int c)

{

cout<<" function 6"<<endl;

return(a+b+c);

}

int main(){

cout<<add(1,2)<<endl;

cout<<add(1.1,2.3)<<endl;

cout<<add(1,2,4)<<endl;

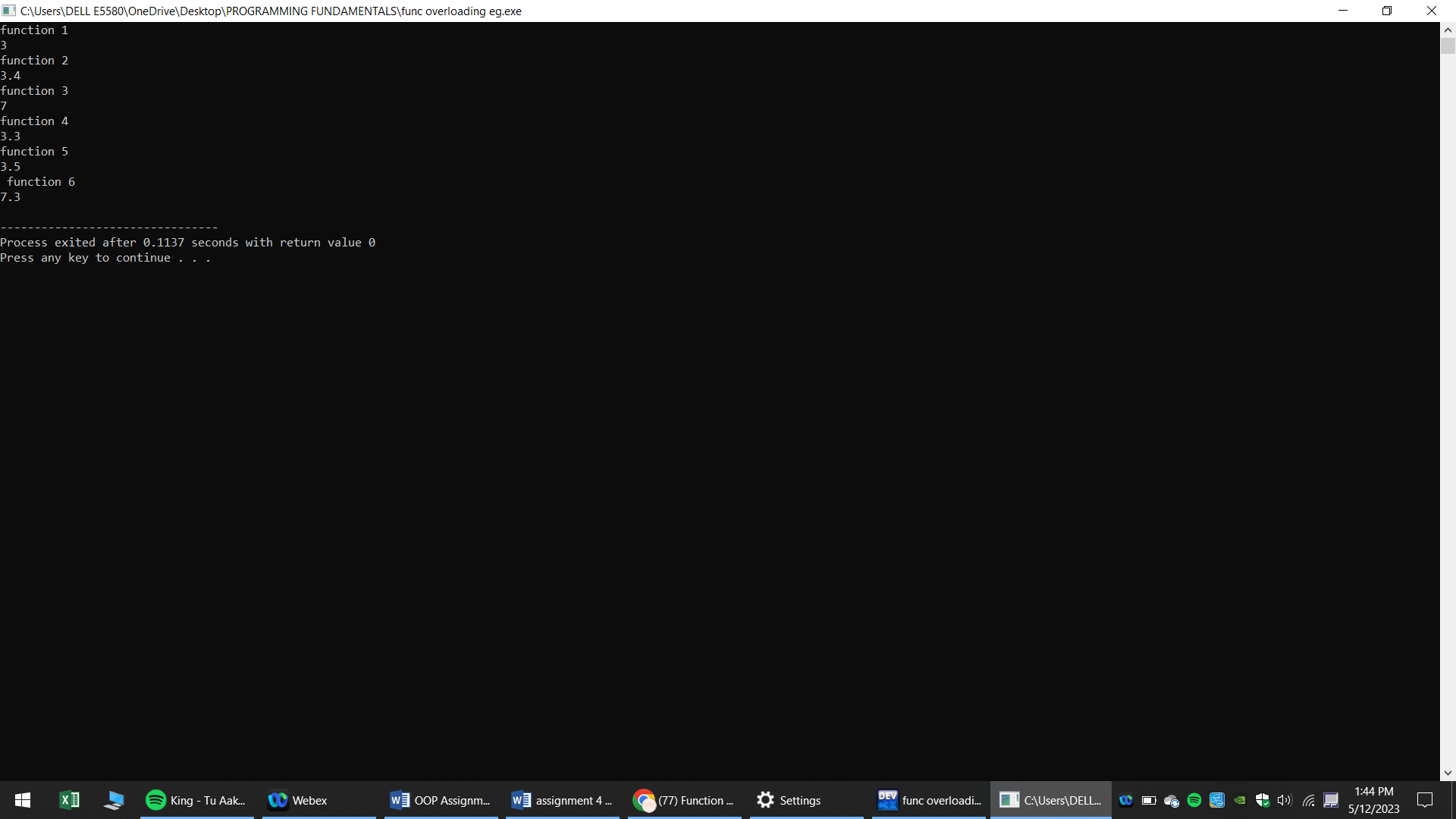
cout<<add(1,2.3)<<endl;

cout<<add(1.5,2)<<endl;

cout<<add(1.3,2,4)<<endl;

}

**OUTPUT:**



2. Write a C++ program that demonstrates the concept of operator overloading.

#include<iostream>

using namespace std;

class Complex{

private:

int real,imaginary;

public:

Complex(){

real=0;

imaginary=0;}

Complex(int r,int i){

real=r;

imaginary=i;

}

void print()

{

cout<<" the complex number is "<<endl <<" "<<real<< "+"<<imaginary<<"i"<<endl;

}

Complex operator +(Complex c)

{

Complex temp;

temp.real=real+c.real;

temp.imaginary=imaginary+c.imaginary;

return temp;

}

};

int main(){

Complex c1(5,4);

Complex c2(4,2);

Complex c3(1,2);

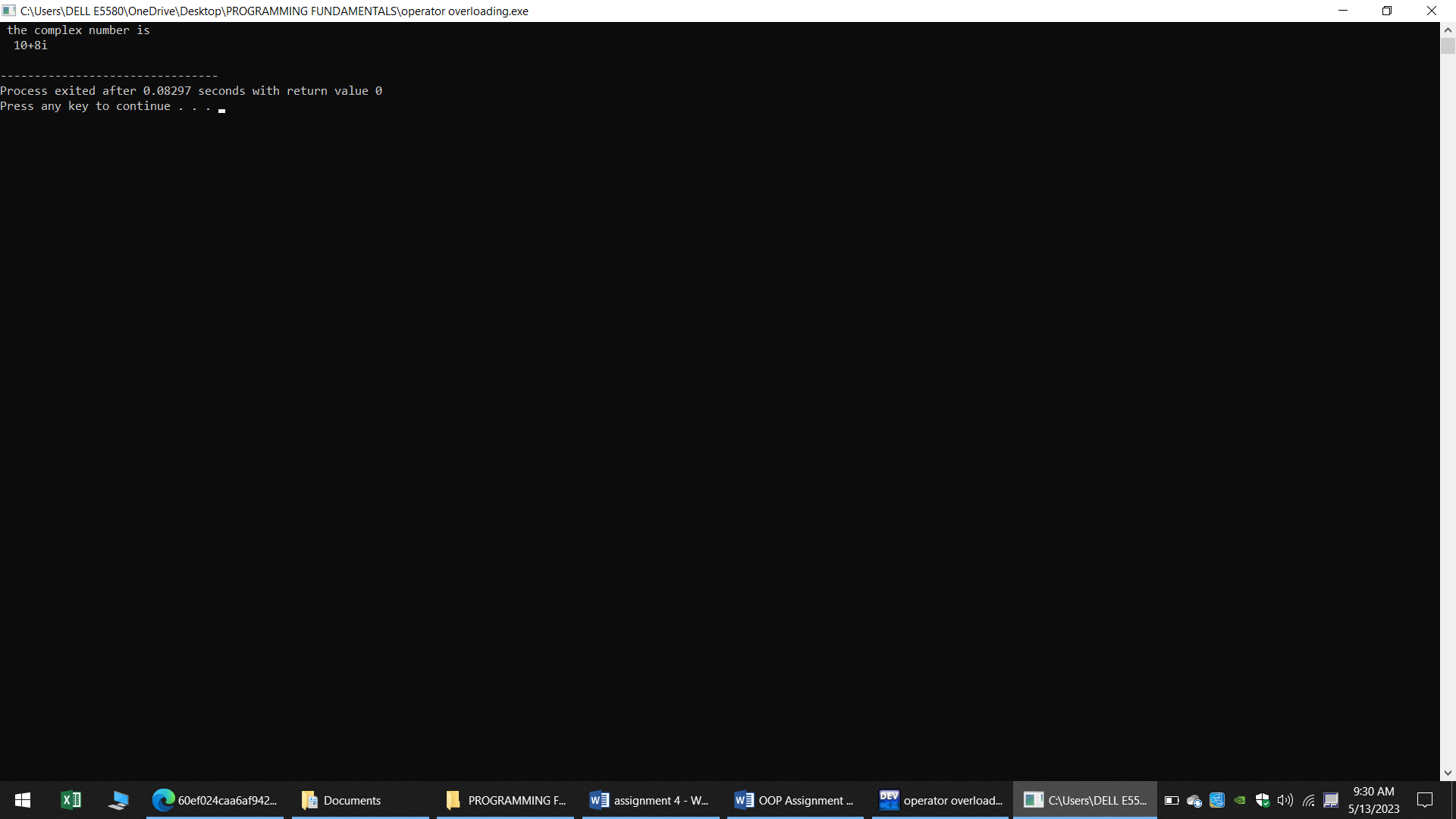
Complex c4;

c4=c1+c2+c3;

c4.print();

}

**OUTPUT:**



3. Write a C++ program that demonstrates the concept of runtime polymorphism using virtual functions.

#include <iostream>

using namespace std;

class B{

public :

virtual void show()

{

cout<<"base class show function"<<endl;

}

};

class D:public B

{

public:

void show()

{

cout<<"derived class show function"<<endl;

}

};

int main()

{

B \*b;

D \*d;

B b1;

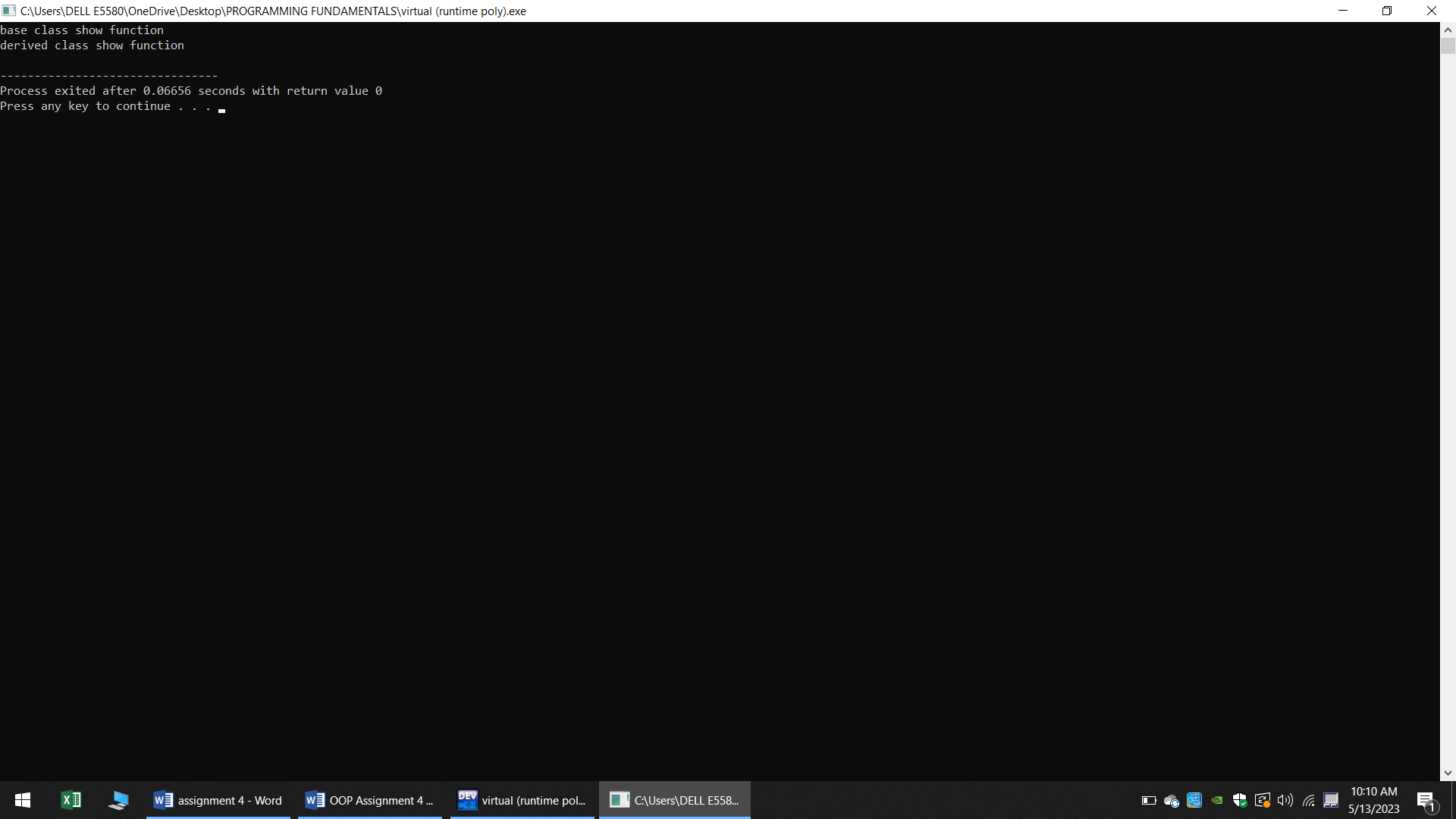
D d1;

b=&b1;b->show();

b=&d1;b->show();

}

**Output:**



4. Write a C++ program that demonstrates the concept of compile-time polymorphism using templates.

#include <iostream>

template <class T>

class StudentFee : T

{

public:

void annualFees()

{

this->fees();

}

};

class EngStudent

{

public:

void fees()

{

std::cout << "EngStudent Fees = 70,000" << std::endl;

}

};

class MathStudent

{

public:

void fees()

{

std::cout << "MathStudent Fees = 95,000" << std::endl;

}

};

int main()

{

StudentFee<EngStudent> engStudent;

StudentFee<MathStudent> mathStudent;

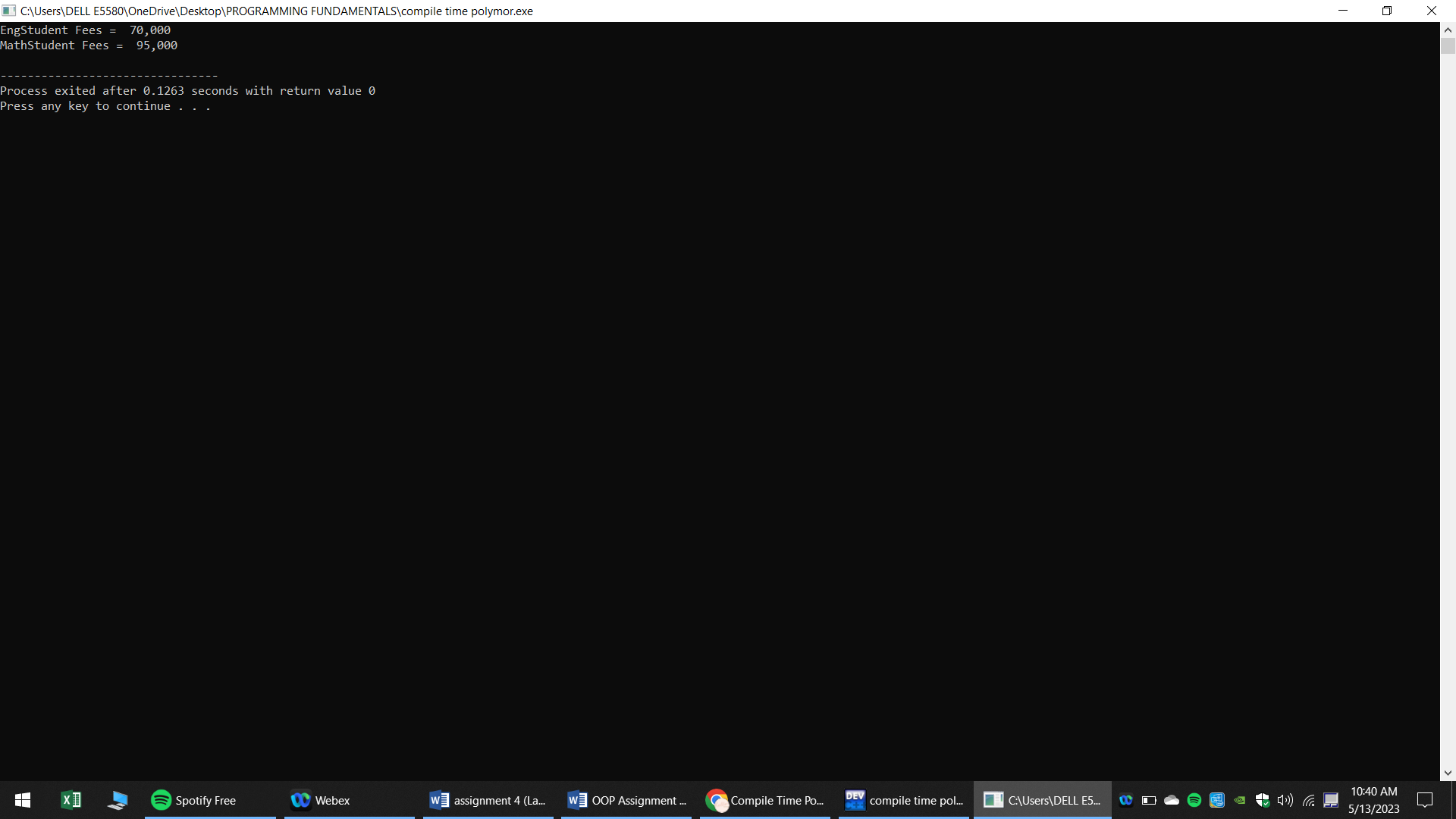
engStudent.annualFees();

mathStudent.annualFees();

return 0;

}

**OUTPUT:**



**PROGRAMS**

1. **Write a C++ program that uses polymorphism to create a hierarchy of shapes. The program should have a base class called `Shape` and derived classes for different types of shapes (e.g. `Circle`, `Rectangle`, `Triangle`). Each derived class should implement a function called `area()` that calculates the area of the shape. The program should allow the user to create objects of different shapes and calculate their areas using polymorphism.**

#include <iostream>

using namespace std;

class Shape {

public:

virtual double area() const = 0;

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double area() const override {

return 3.14159 \* radius \* radius;

}

};

class Rectangle : public Shape {

private:

double width;

double height;

public:

Rectangle(double w, double h) : width(w), height(h) {}

double area() const override {

return width \* height;

}

};

class Triangle : public Shape {

private:

double base;

double height;

public:

Triangle(double b, double h) : base(b), height(h) {}

double area() const override {

return 0.5 \* base \* height;

}

};

int main() {

Shape\* shapes[3];

shapes[0] = new Circle(5.0);

shapes[1] = new Rectangle(4.0, 6.0);

shapes[2] = new Triangle(3.0, 8.0);

for (int i = 0; i < 3; i++) {

cout << "Shape " << i + 1 << " area: " << shapes[i]->area() <<endl;

delete shapes[i];

}

return 0;

}

1. **Extend the previous program to include a function that sorts an array of shapes based on their area. The function should use polymorphism to determine the area of each shape and compare them. The program should allow the user to create an array of shapes of different types and sizes and sort them by area.**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

class Shape {

public:

virtual double area() const = 0;

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double area() const override {

return 3.14159 \* radius \* radius;

}

};

class Rectangle : public Shape {

private:

double width;

double height;

public:

Rectangle(double w, double h) : width(w), height(h) {}

double area() const override {

return width \* height;

}

};

class Triangle : public Shape {

private:

double base;

double height;

public:

Triangle(double b, double h) : base(b), height(h) {}

double area() const override {

return 0.5 \* base \* height;

}

};

bool compareArea(const Shape\* shape1, const Shape\* shape2) {

return shape1->area() < shape2->area();

}

int main() {

vector<Shape\*> shapes;

Circle circle(5.0);

Rectangle rectangle(4.0, 6.0);

Triangle triangle(3.0, 8.0);

shapes.push\_back(&circle);

shapes.push\_back(&rectangle);

shapes.push\_back(&triangle);

cout << "Unsorted Shapes:" << endl;

for (const auto& shape : shapes) {

cout << "Area: " << shape->area() << endl;

}

sort(shapes.begin(), shapes.end(), compareArea);

cout << "\nSorted Shapes:" << endl;

for (const auto& shape : shapes) {

cout << "Area: " << shape->area() << endl;

}

return 0;

}

**Reflection**

1. **Reflect on what you learned in this assignment. What was challenging, and what did you find interesting?**

It was difficult to learn and understand the concept of polymorphism, but in the end I figured it out, and I have now learned and understood it quite well.

1. **How can you apply what you learned in this assignment to future projects or your future career?**

The knowledge I gained from this assignment will be useful to me in our upcoming OOP Project. In our upcoming project, we will also include classes, polymorphism, and inheritance.