**LAB 4: USING FRIEND FUNCTION IN C++.**

**OBJECTIVE:**

1. To create the programs in C++ using friend functions.
2. To access the private and protected members from outside the class.

**THEORY:**If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.By using the keyword friend compiler knows the given function is a friend function.For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.The friend function allows operations between two different classes. Such type of functions have objects as arguments but cannot be called using the object of class. Generally the use of friend functions is out of an object oriented programming methodology because it violates the concept of data hiding.So whenever possible it is better to use members of the same class.To make an outside function friendly to a class,we simply declare the function as a friend of class.

**PROGRAM 1: Define a class Rectangle with private data members length and breadth.Create a friend function getArea that calculates and returns the area of rectangle.**

**//SOURCE CODE:**

#include <iostream>

using namespace std;

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

friend double getArea(const Rectangle& rect);

};

double getArea(const Rectangle& rect) {

return rect.length \* rect.width;

}

int main() {

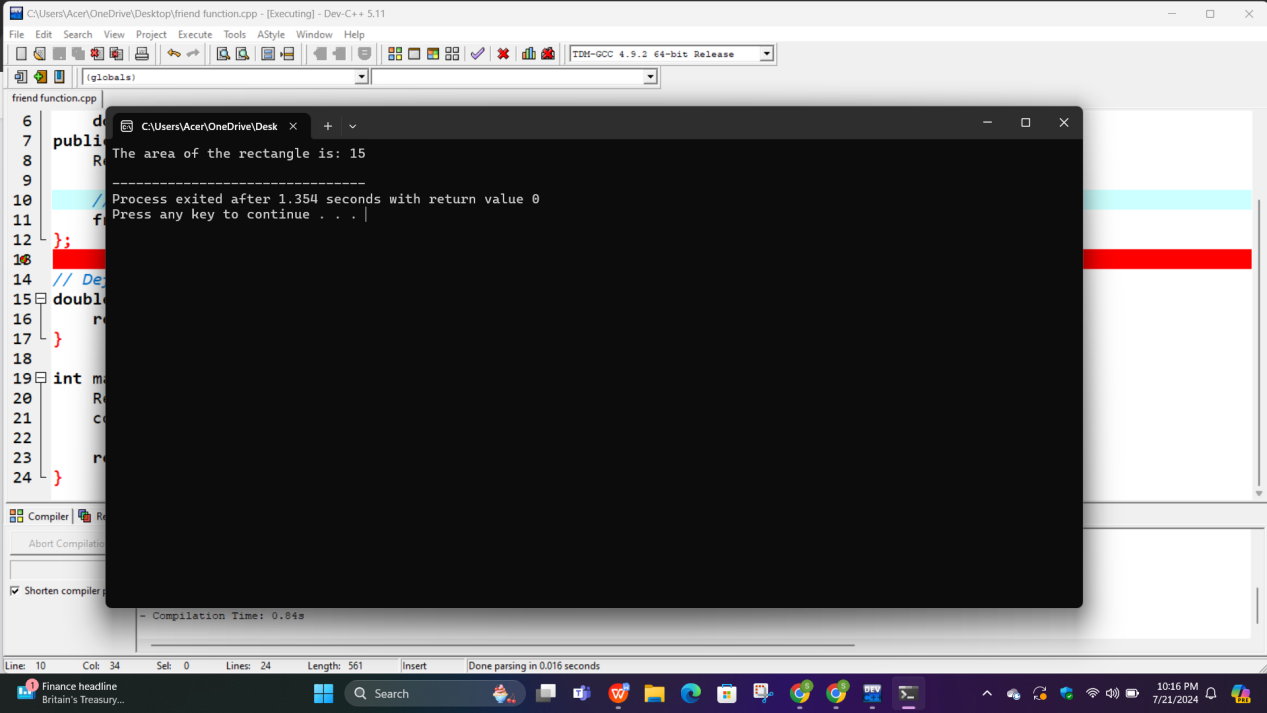
Rectangle rect(5.0, 3.0);

cout << "The area of the rectangle is: " << getArea(rect) << endl;

return 0;

}

**OUTPUT:**

****

**PROGRAM 2.Define a person class to include a friend function compareAge that compares the ages of two person objects and prints which one has a greater age.**

#include <iostream>

using namespace std;

class Person {

private:

string name;

int age;

public:

Person(string n, int a) : name(n), age(a) {}

friend void compareAge(Person &p1, Person &p2);

};

void compareAge(Person &p1,Person &p2) {

if (p1.age > p2.age) {

cout <<p1.name << " is older than " << p2.name<< endl;

} else if (p1.age < p2.age) {

cout << p2.name << " is older than " << p1.name<< endl;

} else {

cout << p1.name << " and " << p2.name << " are of the same age." << endl;

}

}

int main() {

Person p1("Samikshya", 19);

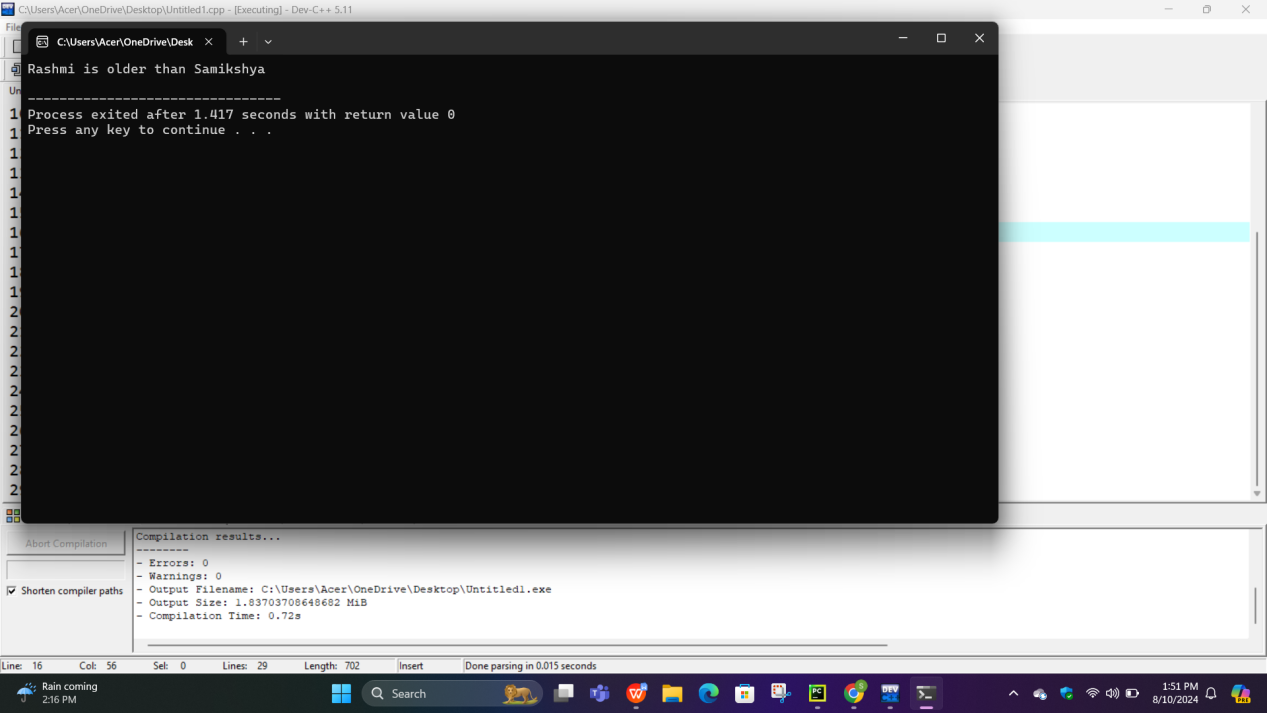
Person p2("Rashmi", 20);

compareAge(p1,p2);

return 0;

}

OUTPUT:



**PROGRAM 3. Create classes Circle and Rectangle.Define a friend function compareArea outside both classes that compares the area of Circle and Rectangle object and prints the larger one.**

**//SOURCE CODE:**

#include <iostream>

using namespace std;

class Circle {

private:

double radius;

public:

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

double getArea(){

return 3.14 \* radius \* radius;

}

};

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double getArea(){

return length \* width;

}

friend void compareArea(Circle& c,Rectangle& r);

};

void compareArea( Circle& c,Rectangle& r) {

double area\_circle = c.getArea();

double area\_rectangle = r.getArea();

if (area\_circle > area\_rectangle) {

cout << "Circle has a larger area than Rectangle." << endl;

} else if (area\_rectangle > area\_circle) {

cout << "Rectangle has a larger area than Circle." << endl;

} else {

cout << "Circle and Rectangle have the same area." << endl;

}

}

int main() {

Circle circle(7.0);

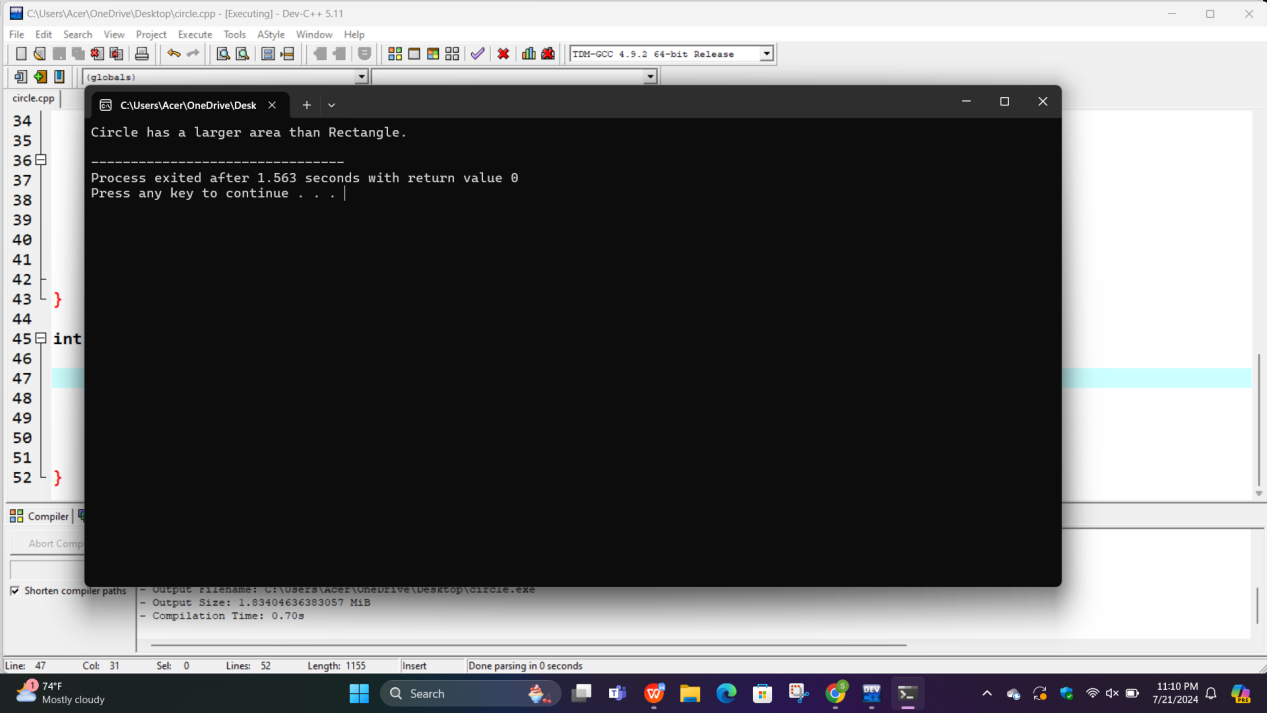
Rectangle rectangle(9.0, 3.0);

compareArea(circle, rectangle);

return 0;

}

**OUTPUT:**

****

**PROGRAM 4:Design a class BankAccount with private data members balance and ownername.Implement a friend function transferFunds that allows transfering funds between two BankAccount objects.**

**//SOURCE CODE:**

#include <iostream>

using namespace std;

class BankAccount{

private:

string accHolderName;

float balance;

public:

BankAccount(string n, float b):accHolderName(n),balance(b){}

friend void transferFund(BankAccount& b1, BankAccount& b2, float amount);

void checkBalance(){

cout<<" Dear "<<accHolderName<< "Youer balnce is "<<balance<<endl;

}

**};**

void transferFund(BankAccount& b1, BankAccount& b2, float amount){

b1.balance -= amount;

b2.balance += amount;

cout<<" Rs. "<<amount <<" is transferred from "<< b1.accHolderName<<" to "<<b2.accHolderName<<"."<<endl;

}

int main(){

BankAccount b1("Deepika ",12000);

BankAccount b2("Yudika ",15000);

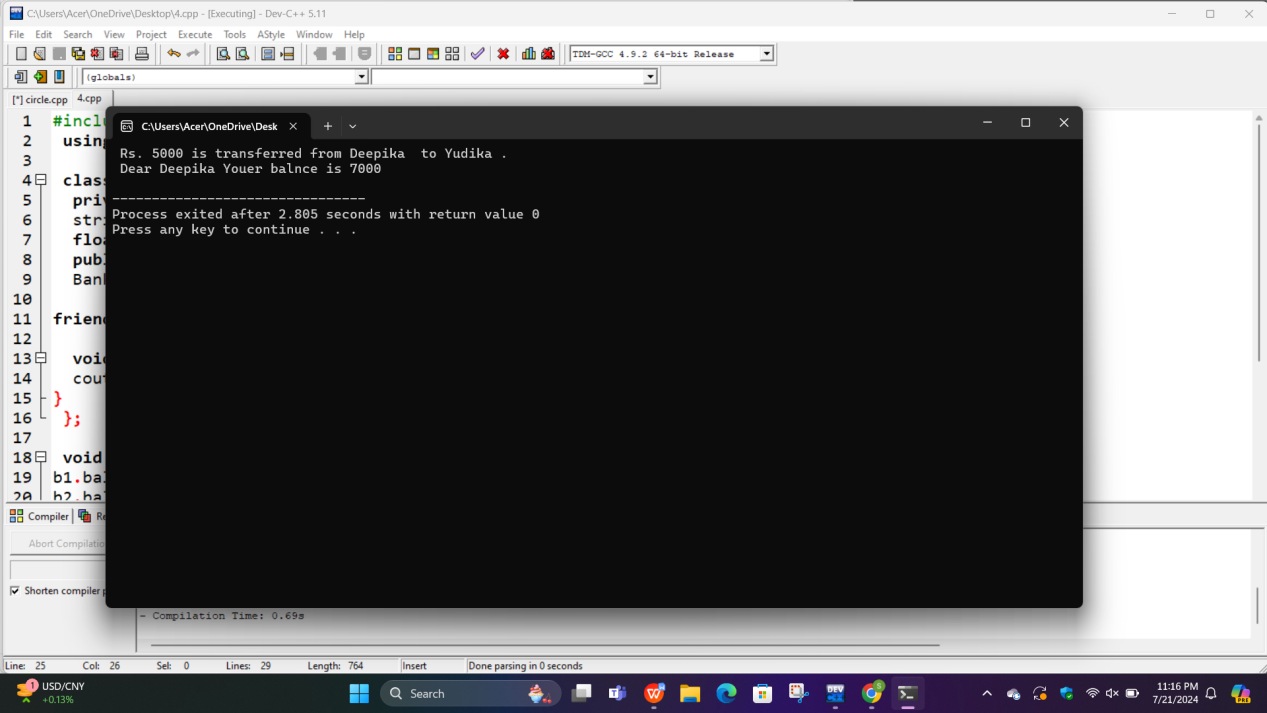
transferFund(b1,b2,5000);

b1.checkBalance();

return 0;

}

**OUTPUT:**

****

**CONCLUSION:**

In this lab we become familiar with the friend functions and run different types of programs based on it.Also we got to know that friend function can be declared either in the public or the private part of class without affecting its meaning.And friend functions can be invoked(called) like a normal function without the help of any object.

**LAB 5:PASSING OBJECT AS A FUNCTION ARGUMENT AND RETURNING OBJECT FROM A FUNCTION IN C++.**

**OBJECTIVE:**

1. Understand the concept of passing objects as a function arguments.
2. Learn how to modify object properties within a function.
3. Explore the process of returning objects from functions.
4. Practice using objects effectively in real world programming scenarios.

**THEORY:**

Passing an object as an argument and returning the objects from function is a fundamental concept in OOP.This is useful, when we want to initialize all data members of an object with another object, we can pass objects and assign the values of supplied object to the current object. For complex or large projects, we need to use objects as an argument or parameter.

**1.Passing Objects as arguments.**

When you pass an object as an argument to a function ,you essentially provide that function with a reference to that object.This reference allows the function to access and manipulate the objects properties and call its methods. This is particularly useful when you want to perform operations on an object without modifying it directly in the calling code.It also promotes code reusability and modularity.

1. **Returning Objects from functions.**

Returning objects from a function allow you to create a new object or modify an existing one within the function and then provide the result back to the calling code.This is particularly useful when you need to encapsulate complex operations or data transformation within a function and present the result as a new object.

**PROGRAM 1:WAP to create a class Rectangle and data members length, breadth and color and initializing length and breadth from constructor also creating a member function paint that takes rectangle object and color as arguments and returns the colored rectangle.**

**//SOURCE CODE:**

#include<iostream>

using namespace std;

class Rectangle{

private:

int length, breadth;

string color;

public:

Rectangle(int l, int b): length(l),breadth(b){}

void info(){

cout << "Length of the rectangle: " << length << endl;

cout << "Breadth of the rectangle: " << breadth << endl;

}

void colour(Rectangle &r,string color){

r.color = color;

cout<<color<<" colored rectangle with length "<<length<<" and breadth "<<breadth<<"."<<endl;

}

};

int main(){

Rectangle r1(40, 60);

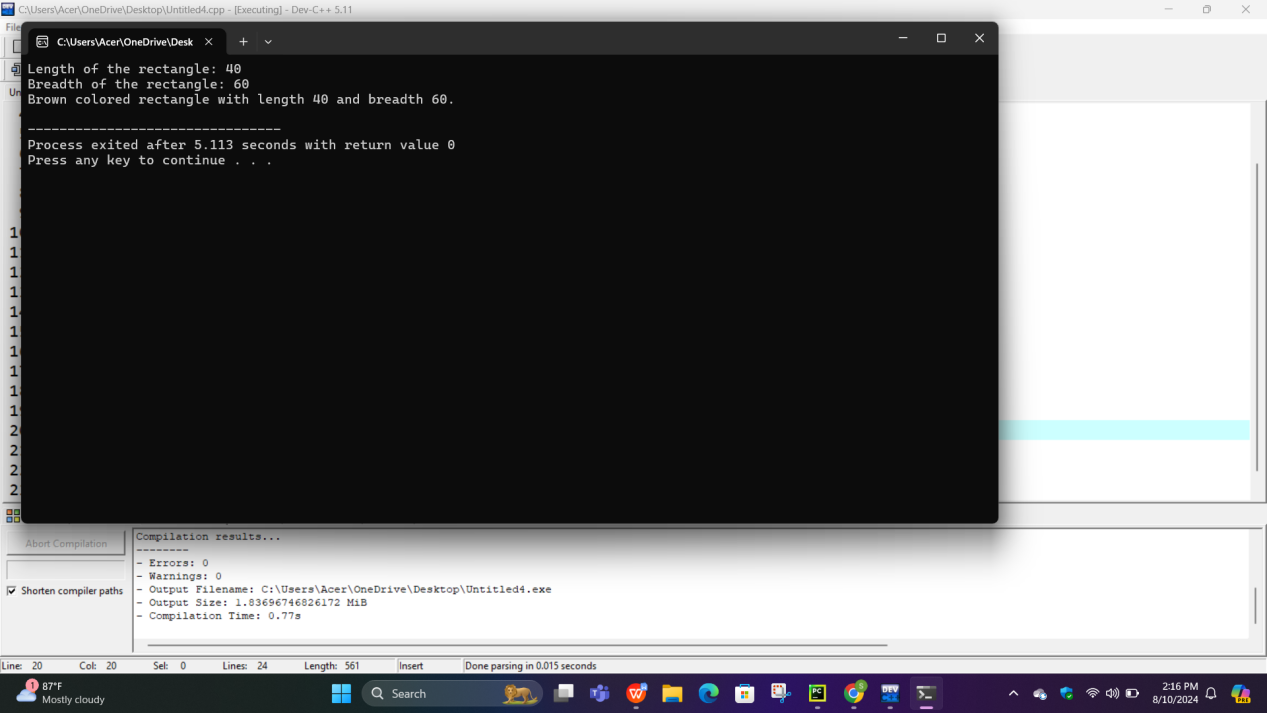
r1.info();

r1.colour(r1,"Brown");

return 0;

}

**OUTPUT:**



**PROGRAM 2:Create Car and Driver classes and necessary data members. Write a drive function in car class so that when a driver object is passed as argument, it displays the information about the car condition according to driver skill.**

**//SOURCE CODE:**

#include<iostream>

using namespace std;

class Driver{

private:

string name;

bool license;

public:

Driver(string n, bool l):name(n), license(l){}

string getname(){

return name;

}

bool getlicense(){

return license;

}

};

class Car{

public:

void drive(Driver &dri){

if(dri.getlicense()==true){

cout<<dri.getname()<<" have got license.So the condition of car might be better."<<endl;

}

else{

cout<<dri.getname()<<" have got no license which have high risk of car accident."<<endl;

}

}

};

int main(){

Car c;

Driver d1("Samikshya Ranabhat",false);

c.drive(d1);

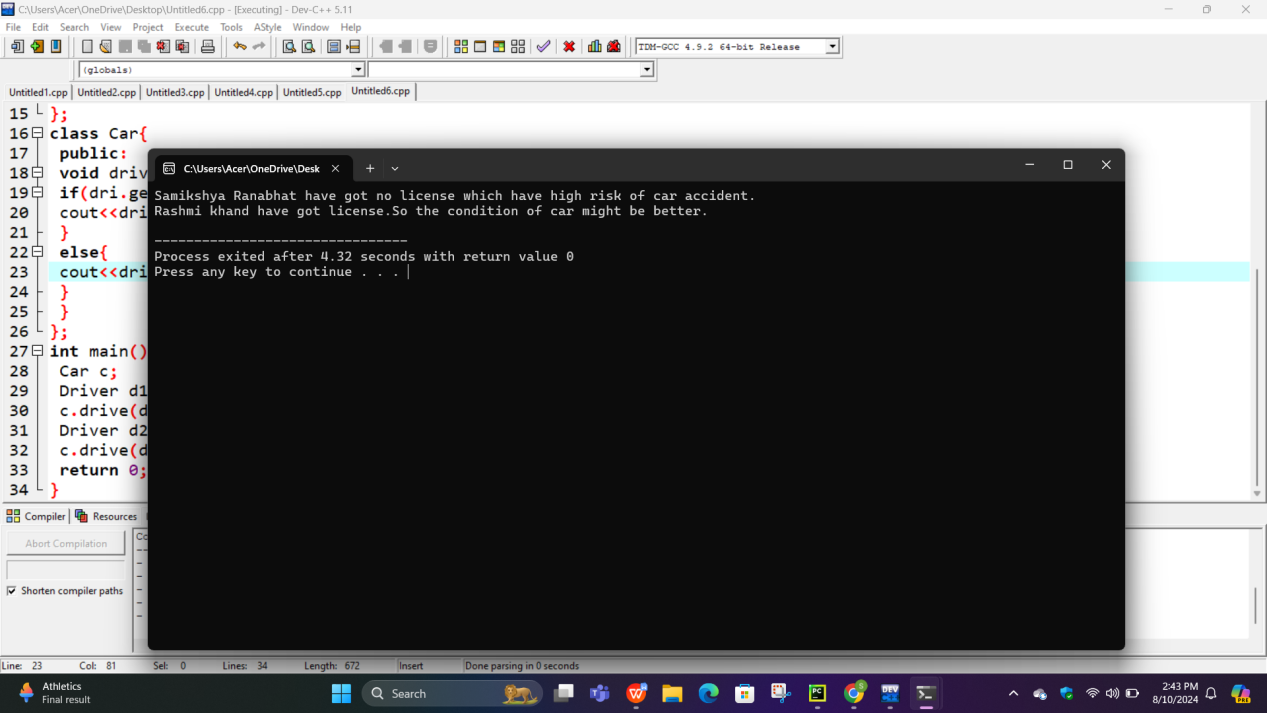
Driver d2("Rashmi khand", true);

c.drive(d2);

return 0;

}

OUTPUT:



**PROGRAM 3:Create a ComplexNumber class and necessary members. Also create a member function add, that adds two complex numbers and returns a new complex number as a sum of two.**

**//SOURCE CODE:**

#include<iostream>

using namespace std;

class ComplexNumber{

private:

int real, img;

public:

ComplexNumber(int r=0, int i=0): real(r),img(i){}

void add(ComplexNumber &c1, ComplexNumber &c2){

c1.real = c1.real + c2.real;

c1.img = c1.img + c2.img;

cout<<"Sum of two complex numbers: "<<c1.real<<" + "<<c1.img<<"i"<<endl;

}

};

int main(){

ComplexNumber c1(4, 3);

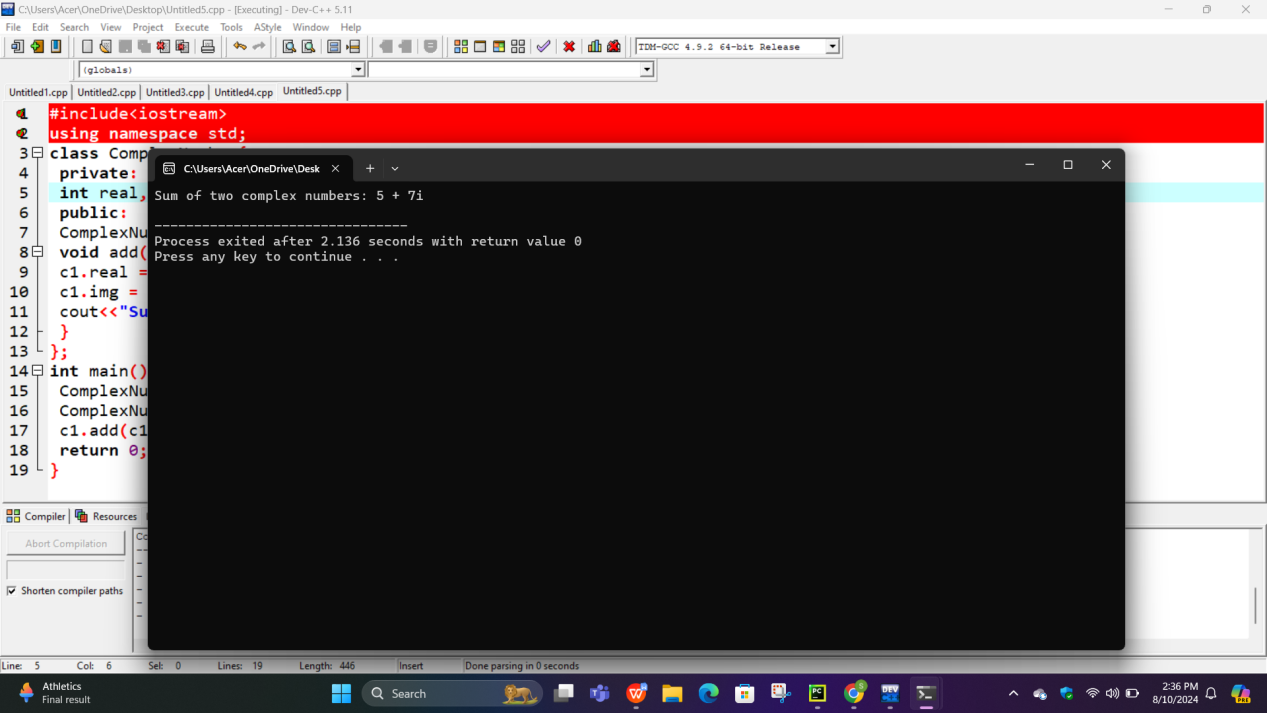
ComplexNumber c2(1, 4);

c1.add(c1, c2);

return 0;

}

**OUTPUT:**



**PROGRAM 4:Create a class Point with necessary data members. Write a function that takes two points as arguments and returns the mid point**

**//SOURCE CODE:**

#include<iostream>

using namespace std;

class Point{

private:

int x, y;

public:

Point(int a, int b):x(a),y(b){}

void display(){

cout<<"("<<x<<","<<y<<")"<<endl;

}

Point Mp(Point &p){

float midX=(x+p.x)/2;

float midY=(y+p.y)/2;

return Point(midX,midY);

}

};

int main(){

Point p1(1,2);

Point p2(3,4);

p1.display();

p2.display();

cout<<"Midpoint of the given points: "<<endl;

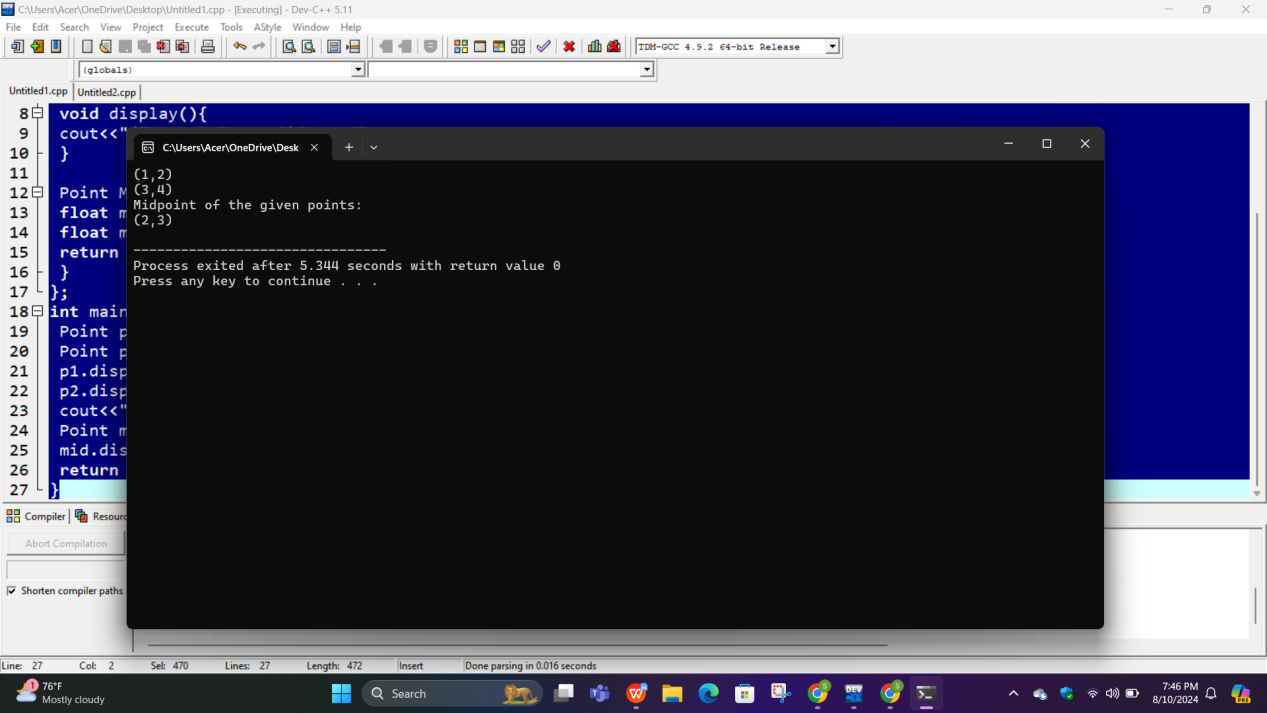
Point mid = p1.Mp(p2);

mid.display();

return 0;

}

**OUTPUT:**

****

**CONCLUSION:**

In this C++ program, we've successfully demonstrated the concept of passing objects as arguments and returning objects from functions. This approach allows for the modification and creation of objects within functions, promoting code modularity and reusability.