Assignment # 4

Name: Minahil Dar

SAP ID: 47134

Subject: Object-Oriented Programming

Submitted To: Sir Shahzad

**Part-01**

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

**Answer:** Polymorphism is one of the concepts of OOP which include inheritance and overriding. Polymorphism describes a situation that can occur several times and in different forms. This means we can use the same object in the same interface as many times as we want with the help of inheritance and overriding. The objects inherited from the base class in the derived class.

It relates to object-oriented programming like we have some objects whose functionality is the same as we want to use some other place but in a different way than we use polymorphism. A function of the base class in the derived class by using polymorphism.

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

**Answer:**

|  |  |
| --- | --- |
| **Static Polymorphism** | **Dynamic Polymorphism** |
| * Static polymorphism happens at compile time. * It is also known as early binding. * Static polymorphism is achieved by overloading functions in which different functions have the same name but different parameters. When the function is called the compiler determines itself which function should be called depending on the argument type. | * Dynamic polymorphism happens at run time. * It is also known as late binding. * Dynamic polymorphism is achieved by inheritance and virtual function. Inheritance is the process in which we can inherit a data member of the base class in the child class. Virtual functions are the function that is declared in the base class and can be used in the child class by overriding. When the virtual function is to be called is determined at run time based on the object. |

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

**Answer:** There are two types of polymorphism in C++ Static polymorphism (compile time) and dynamic polymorphism (run time).

**Static polymorphism** is also called compile-time polymorphism. It occurs at compile time by overloading functions. The functions have the same name but different parameters or arguments. When the overloading functions are called they are determined by the compiler based on the type of parameters or arguments.

**Dynamic polymorphism** is also called run-time polymorphism. Dynamic polymorphism is achieved by inheritance and virtual function. Inheritance is the process in which we can inherit a data member of the base class in the child class. Virtual functions are the function that is declared in the base class and can be used in the child class by overriding. When the virtual function is to be called is determined at run time based on the object.

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

**Answer:** Virtual function is used in dynamic polymorphism. These are the functions in the base class and can be used or overridden by the derived class. The keyword virtual is written before the function name. It is used if we have a base class and so many derived class which uses the same function of the base class but in different ways than we use the virtual function, those functions are overridden by the derived class. If we want to use a virtual function then in the base class use virtual with the function but not in the derived class. When we use the virtual function in the base class then that function will necessarily use in the derived class.

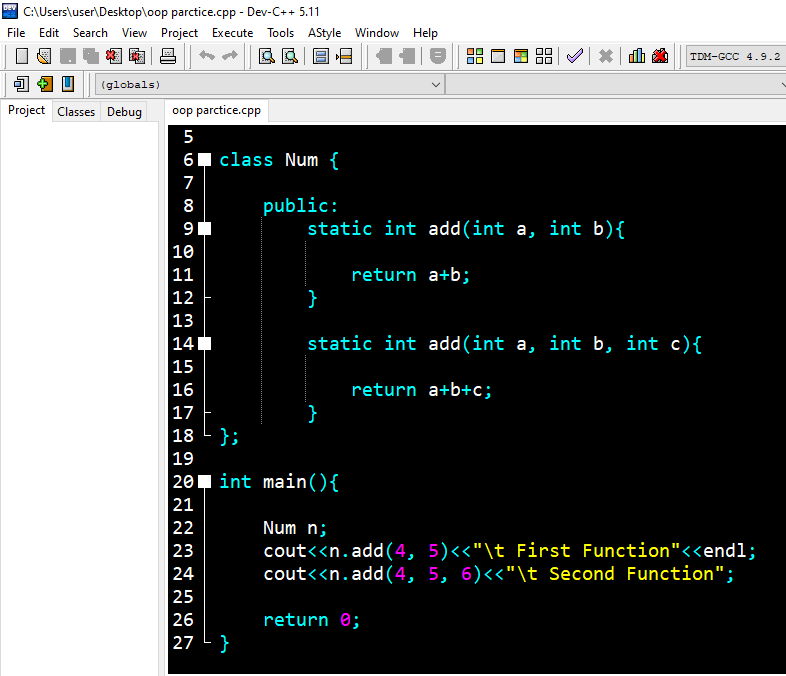
1. **Can a class have both virtual and non-virtual functions? Explain your answer.**

**Answer:** Yes, a class can have both virtual and non-virtual functions. A virtual function is used to override a function in a derived class, which provides its own implementation but a non-virtual function does not implement its own comparatively non-virtual functions are independent and are not supposed to be overridden in a class. In conclusion, both non-virtual and virtual functions can be used in a class.

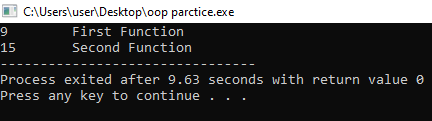
**Part-02**

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

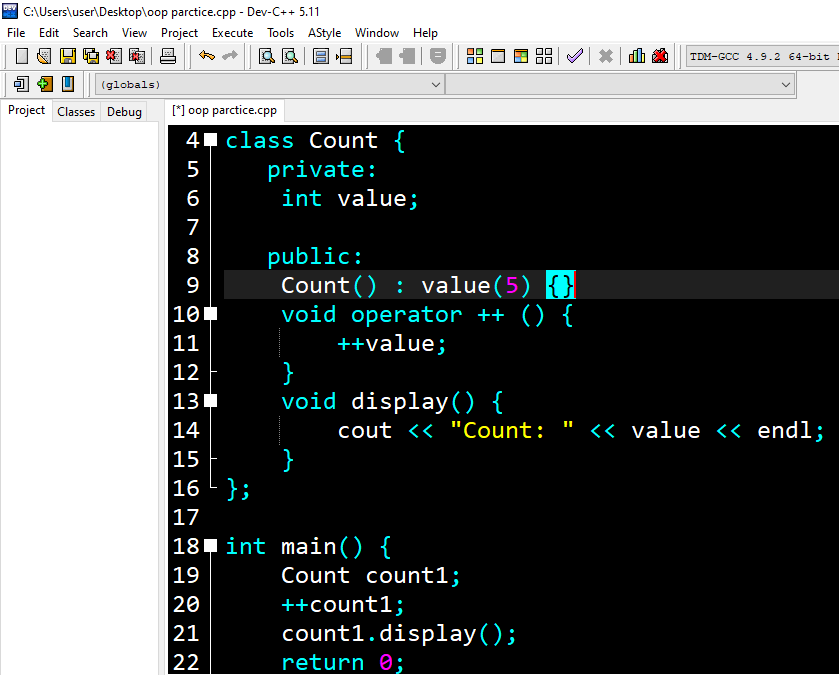
**Answer:**



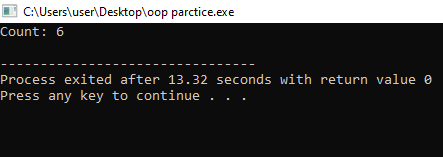
**Output:**



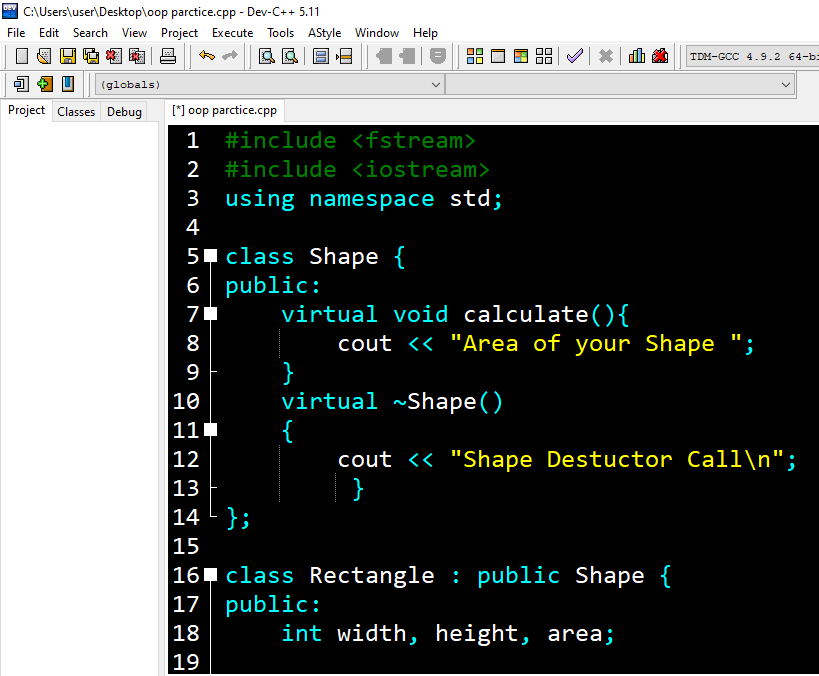
1. **Write a C++ program that demonstrates the concept of operator overloading.**

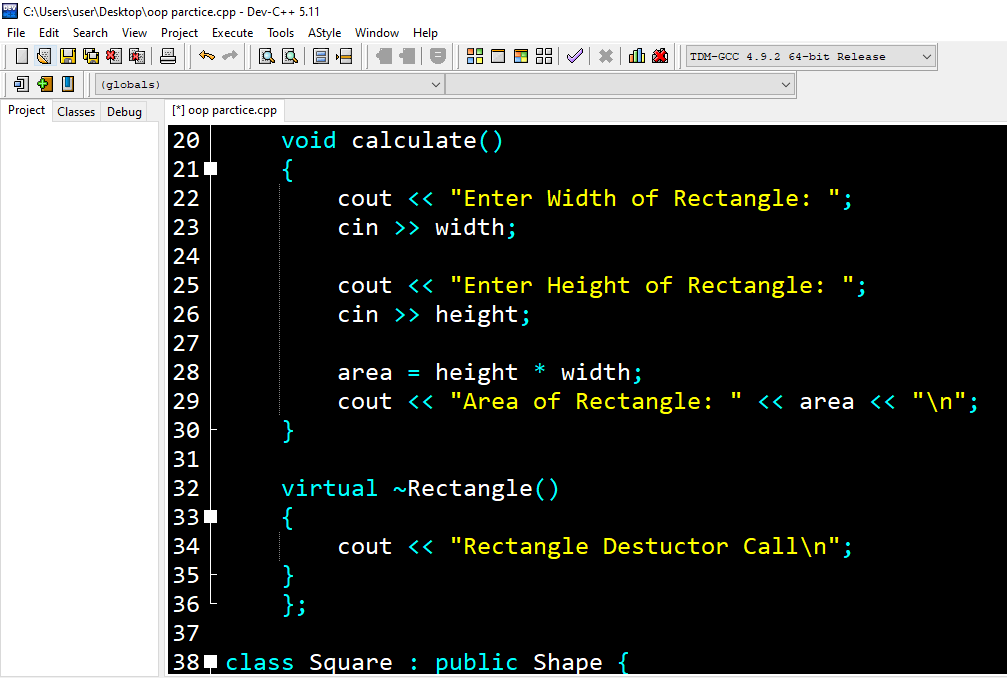


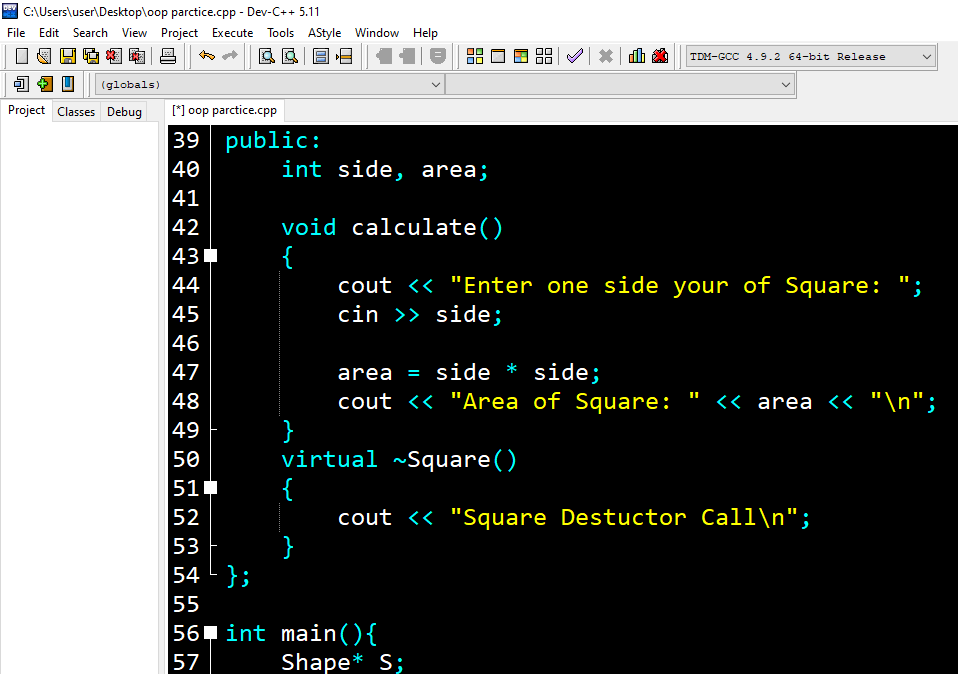
**Output:**

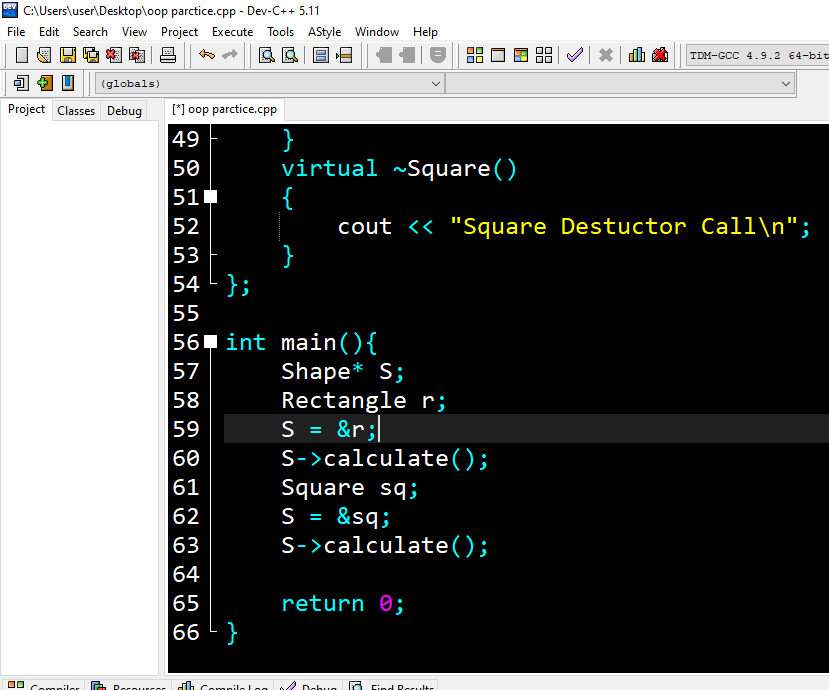


3. Write a C++ program that demonstrates the concept of runtime polymorphism using virtual functions.









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

Answer:

#include <iostream>

template <class T>

class StudentFee : T

{

public:

void annualFees()

{

this->fees();

}

};

class EngStudent

{

public:

void fees()

{

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

}

};

class MbaStudent

{

public:

void fees()

{

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

}

};

int main()

{

StudentFee<EngStudent> engStudent;

StudentFee<MbaStudent> mbaStudent;

engStudent.annualFees();

mbaStudent.annualFees();

return 0;

}

Part-03

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.

**Answer:**

#include <fstream>

#include <iostream>

using namespace std;

class Shape {

public:

virtual void calculate(){

cout << "Area of your Shape ";

}

virtual ~Shape()

{

cout << "Shape Destuctor Call\n";

}

};

class Rectangle : public Shape {

public:

int width, height, area;

void calculate()

{

cout << "Enter Width of Rectangle: ";

cin >> width;

cout << "Enter Height of Rectangle: ";

cin >> height;

area = height \* width;

cout << "Area of Rectangle: " << area << "\n";

}

virtual ~Rectangle()

{

cout << "Rectangle Destuctor Call\n";

}

};

class Square : public Shape {

public:

int side, area;

void calculate()

{

cout << "Enter one side your of Square: ";

cin >> side;

area = side \* side;

cout << "Area of Square: " << area << "\n";

}

virtual ~Square()

{

cout << "Square Destuctor Call\n";

}

};

int main(){

Shape\* S;

Rectangle r;

S = &r;

S->calculate();

Square sq;

S = &sq;

S->calculate();

return 0;

}