**OBJECT ORIENTED CONCEPT & PROGRAMMING**

**(SE-201)**

**ASSIGNMENT**

**TAQI HAIDER\_CSIT\_SECTION:B\_ROLL#CT-22092**

**Q1. Compare compile time and runtime polymorphism.**

|  |  |
| --- | --- |
| **Compile Time Polymorphism** | **Run Time Polymorphism** |
| **Achieved through function overloading and operator overloading.** | **Achieved through function overriding** |
| **Multiple functions with the same name but different parameters exist. The choice of which function to call is determined at compile time based on the number and types of arguments.** | **Occurs when a derived class provides a definition for a member function that is already defined in the base class. The choice of which function to call is determined at runtime based on the actual object type.** |
| **Involves method overloading and operator overloading.** | **Involves method overriding.** |
| **Resolved by the compiler during compile time.** | **Resolved at runtime, allowing the program to select the appropriate method to invoke based on the actual object type.** |
| **Also known as early binding.** | **Also known as late binding or dynamic polymorphism.** |

**Q2. Apply OOP concepts to write a program to find maximum out of two numbers using friend function and also note one number is a member of one class and other number is member of some other class. By using set function set values of data members.**

#include <iostream>

using namespace std;

class CarParts

{

protected:

    string Brand;

public:

    CarParts(string brand) : Brand(brand) {}

    string get\_brand()

    {

        return Brand;

    }

    virtual void info()

    {

        cout << "The name of " << Brand << endl;

    }

};

class Doors : public CarParts

{

    string location;

public:

    Doors(string loc, string brand) : CarParts(brand), location(loc) {}

    void open()

    {

        cout << location << " door is opened." << endl;

    }

    void close()

    {

        cout << location << " door is closed." << endl;

    }

    void info() override

    {

        cout << "This is a " << get\_brand() << " at " << location << endl;

    }

};

class Windows : public CarParts

{

    string location;

public:

    Windows(string loc, string brand) : CarParts(brand), location(loc) {}

    void open()

    {

        cout << location << " window is opened." << endl;

    }

    void close()

    {

        cout << location << " window is closed." << endl;

    }

    void info() override

    {

        cout << "This is a " << get\_brand() << " at " << location << endl;

    }

};

class Engine : public CarParts

{

    string horsepower;

public:

    Engine(string hp, string brand) : CarParts(brand), horsepower(hp) {}

    void start()

    {

        cout << "The Engine of Horsepower " << horsepower << " is started." << endl;

    }

    void stop()

    {

        cout << "The Engine of Horsepower " << horsepower << " is stopped." << endl;

    }

    void info() override

    {

        cout << "This is a " << get\_brand() << " & the hp of " << horsepower << endl;

    }

};

class Wheels : public CarParts

{

    string mirror;

public:

    Wheels(string m, string brand) : CarParts(brand), mirror(m) {}

    void rotate()

    {

        cout << "Wheels are rotated" << endl;

    }

    void setMirror(string m)

    {

        if (m == "yes")

        {

            cout << "Mirror is on" << endl;

        }

        else

        {

            cout << "Mirror is off" << endl;

        }

    }

    void info() override

    {

        cout << "This is a " << get\_brand() << " & Mirror status : " << mirror << endl;

    }

};

class Car

{

    Doors frontDoor;

    Doors rearDoor;

    Windows frontWindow;

    Windows rearWindow;

    Engine engine;

    Wheels wheels;

public:

    Car(string doorLocation, string windowLocation, string engineHP, string wheelMirror)

        : frontDoor(doorLocation, "BMW"),

          rearDoor(doorLocation, "BMW"),

          frontWindow(windowLocation, "Mercedes"),

          rearWindow(windowLocation, "Mercedes"),

          engine(engineHP, "Audi"),

          wheels(wheelMirror, "Civic") {}

    void start()

    {

        engine.start();

    }

    void stop()

    {

        engine.stop();

    }

    void openFrontDoor()

    {

        frontDoor.open();

    }

    void closeFrontDoor()

    {

        frontDoor.close();

    }

    void openRearDoor()

    {

        rearDoor.open();

    }

    void closeRearDoor()

    {

        rearDoor.close();

    }

    void openFrontWindow()

    {

        frontWindow.open();

    }

    void closeFrontWindow()

    {

        frontWindow.close();

    }

    void openRearWindow()

    {

        rearWindow.open();

    }

    void closeRearWindow()

    {

        rearWindow.close();

    }

    void rotateWheels()

    {

        wheels.rotate();

    }

    void setWheelMirror(string m)

    {

        wheels.setMirror(m);

    }

    void carInfo()

    {

        cout << "Car Brand: " << wheels.get\_brand() << endl;

        cout << "Car Parts Information:" << endl;

        frontDoor.info();

        rearDoor.info();

        frontWindow.info();

        rearWindow.info();

        engine.info();

        wheels.info();

    }

};

int main()

{

    Car myCar("Front", "Front", "300hp", "yes");

    myCar.carInfo();

    myCar.start();

    myCar.openFrontDoor();

    myCar.openFrontWindow();

    myCar.rotateWheels();

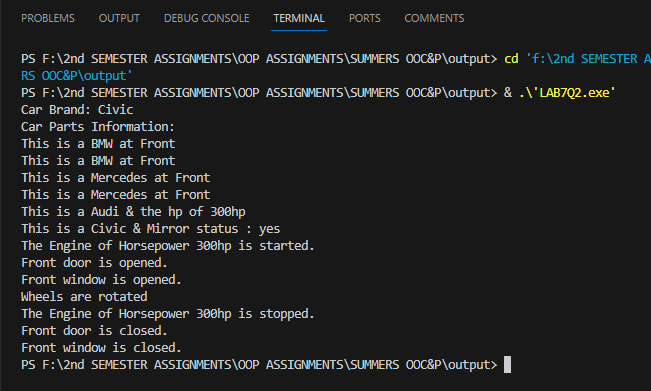
    myCar.stop();

    myCar.closeFrontDoor();

    myCar.closeFrontWindow();

    return 0;

}



**Q3 :-**

#include <iostream>

using namespace std;

class BOX

{

    double width, length, height;

public:

    BOX() : width(0.0), length(0.0), height(0.0) {}

    BOX(double w, double l, double h) : width(w), length(l), height(h) {}

    void showdata()

    {

        cout << "The width of the BOX: " << width << endl;

        cout << "The length of the BOX: " << length << endl;

        cout << "The height of the BOX: " << height << endl;

    }

    void calcVolume()

    {

        cout << "The volume is :" << width \* length \* height << endl;

    }

    BOX operator+(BOX &b)

    {

        return BOX(width + b.width, length + b.length, height + b.height);

    }

    BOX operator-(BOX &b)

    {

        return BOX(width - b.width, length - b.length, height - b.height);

    }

    BOX operator\*(double scalar)

    {

        return BOX(width \* scalar, length \* scalar, height \* scalar);

    }

    BOX operator/(double scalar) const

    {

        if (scalar == 0.0)

        {

            std::cerr << "Error: Division by zero." << endl;

            return \*this;

        }

        return BOX(width / scalar, length / scalar, height / scalar );

    }

};

int main()

{

    BOX box1(2.0, 3.0, 4.0);

    BOX box2(1.0, 1.5, 2.0);

    // Display data of box1 and box2

    cout << "Box1 data: ";

    box1.showdata();

    cout << "Box2 data: ";

    box2.showdata();

    // Perform operations on boxes

    BOX additionResult = box1 + box2;

    BOX subtractionResult = box1 - box2;

    BOX multiplicationResult = box1 \* 2.0;

    BOX divisionResult = box1 / 2.0;

    // Display results

    cout << "Addition Result: " << endl;

    additionResult.showdata();

    cout << "Subtraction Result: " << endl;

    subtractionResult.showdata();

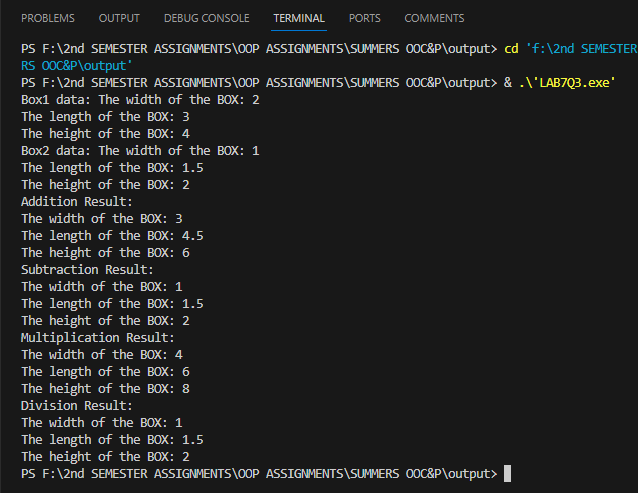
    cout << "Multiplication Result: " << endl;

    multiplicationResult.showdata();

    cout << "Division Result: " << endl;

    divisionResult.showdata();

}

****