**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.**

// 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 num2;

class num1{

    int number1;

    public:

    num1():number1(0){}

    void set\_value(int n){

        number1=n;

    }

    void get\_value(){

        cout<<"Number 1 Value:"<<number1<<endl;

    }

    friend int max(num1,num2);

};

class num2{

    int number2;

    public:

    num2():number2(0){}

    void set\_value(int n){

        number2=n;

    }

    void get\_value(){

        cout<<"Number 2 Value:"<<number2<<endl;

    }

    friend int max(num1,num2);

};

int max(num1 n1,num2 n2){

    if(n1.number1>n2.number2){

        return n1.number1;

    }

    else {

        return  n2.number2;

    }

}

int main(){

    num1 a;

    num2 b;

    a.set\_value(10);

    a.get\_value();

    b.set\_value(20);

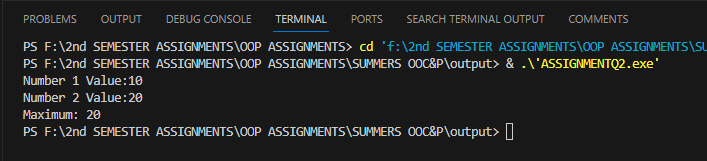
    b.get\_value();

    int maximum = max(a, b);

    cout << "Maximum: " << maximum << endl;

    return 0;

}

****

**Q3. How are virtual functions different from pure virtual functions. Elaborate using an example**.

Virtual Functions:

A virtual function is a member function that is declared in a base class using the virtual keyword and is meant to be overridden by derived classes. It provides a way for a base class to define a method that can be customized by derived classes. In other words, the base class provides a default implementation, but derived classes can provide their own implementations.

Here's an example:

#include<iostream>

using namespace std;

class Animal {

public:

    virtual void makeSound() {

        cout << "Animal makes a sound" << endl;

    }

};

class Dog : public Animal {

public:

    void makeSound() override {

        cout << "Dog barks" << endl;

    }

};

class Cat : public Animal {

public:

    void makeSound() override {

        cout << "Cat meows" << endl;

    }

};

int main() {

    Animal\* animal1 = new Dog();

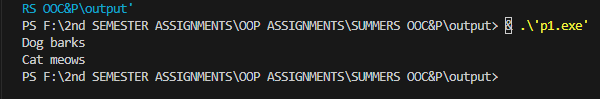
    Animal\* animal2 = new Cat();

    animal1->makeSound();

    animal2->makeSound();

    return 0;

}



Pure Virtual Functions:

A pure virtual function is a virtual function that is declared in a base class but has no implementation in the base class. It is meant to be overridden by derived classes, and any class containing at least one pure virtual function becomes an abstract class. Abstract classes cannot be instantiated; they are meant to serve as base classes for other classes.

Here's an example:

#include<iostream>

using namespace std;

class Shape {

public:

    virtual void area() = 0; // Pure virtual function

};

class Circle : public Shape {

private:

    double radius;

public:

    Circle(double r) : radius(r) {}

    void area() override {

        double result = 3.1415 \* radius \* radius;

        cout << "Area of the circle: " << result << endl;

    }

};

int main() {

    // Shape\* shape = new Shape();

    // Will throw Error: Cannot create object of an abstract class

    Circle circle(5.0);

    circle.area(); // Calls Circle's area function

    return 0;

}

