CS-112 Assignment # 03

Name : Muhammad khuzaim

Reg no # 2024421

Section : J (Software Engineering)

Question 1 :

Write a program that demonstrates simple single inheritance.

Create a base class Person with members name, age, and a function display().

Create a derived class Student that inherits Person and adds studentID and displayStudent() function.

In main(), create an object of Student and call both functions.

Code :

```
#include <iostream>
using namespace std;

// Base class
class Person {
public:
    string name;
    int age;

   void display() {
      cout << "Name: " << name << endl;
      cout << "Age: " << age << endl;
```

```
    }
};

// Derived class
class Student : public Person {
public:
    string studentID;
    void displayStudent() {
        cout << "Student ID: " << studentID << endl;
    }
};

int main() {
    Student s;
    s.name = "Ali";
    s.age = 20;
    s.studentID = "CS112-1234";

    // Call both functions
    s.display();
    s.displayStudent();
    return 0;
}
```

Screenshot :

Question 2 :

Design a program using multilevel inheritance.

Base class: Vehicle → contains brand, and a function start().

Derived class: Car → adds model, and a function drive().

Final derived class: ElectricCar → adds batteryPercentage, and a function charge().

Demonstrate how access specifiers (public, protected, private) affect accessibility.

Add one protected member in base and use it in the derived class.

Code :

```
#include <iostream>

using namespace std;


class Vehicle {

protected:

   string type = "Electric"; // Protected member

public:

   string brand;
```

```
    void start() {
        cout << "Vehicle " << brand << " is starting..." << endl;
    }
};

class Car : public Vehicle {
public:
    string model;

    void drive() {
        cout << "Driving " << brand << " " << model << " [" << type << "]" << endl;
    }
};

class ElectricCar : public Car {
public:
    int batteryPercentage;

    void charge() {
        cout << "Charging " << model << ". Battery at " << batteryPercentage << "%" << endl;
    }
};

int main() {
```

```
    ElectricCar ec;
    ec.brand = "Tesla";
    ec.model = "Model 3";
    ec.batteryPercentage = 85;

    ec.start();
    ec.drive();
    ec.charge();

    return 0;
}
```

Screenshot :

Question 3 :

Write a program that shows constructor chaining and function overriding.

Base class Shape with a constructor and function display().

Derived class Rectangle and Circle, both override display().

Demonstrate calling base class constructor using initializer list.

Call overridden function from derived class using scope resolution Shape::display().

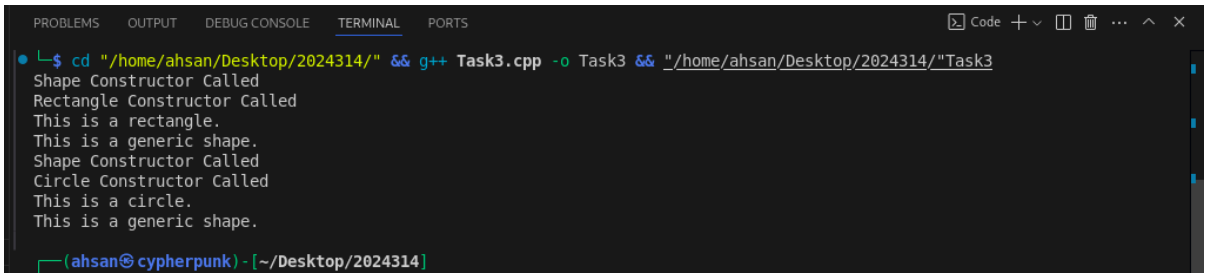
Code :

```
#include <iostream>
using namespace std;

class Shape {
public:
   Shape() {
      cout << "Shape Constructor Called" << endl;
   }

   virtual void display() {
      cout << "This is a generic shape." << endl;
   }
};

class Rectangle : public Shape {
public:
   Rectangle() : Shape() {
      cout << "Rectangle Constructor Called" << endl;
   }

   void display() override {
```

```
        cout << "This is a rectangle." << endl;
        Shape::display(); // Call base version
    }
};

class Circle : public Shape {
public:
    Circle() : Shape() {
        cout << "Circle Constructor Called" << endl;
    }

    void display() override {
        cout << "This is a circle." << endl;
        Shape::display();
    }
};

int main() {
    Rectangle r;
    r.display();

    Circle c;
    c.display();

    return 0;
```

}

Screenshot :



Question 4 :

Create a program to demonstrate the Diamond Problem and solve it using virtual base classes.

Base class: Employee

Derived: Manager and Engineer, both inherit from Employee.

Class TechLead inherits from both Manager and Engineer.

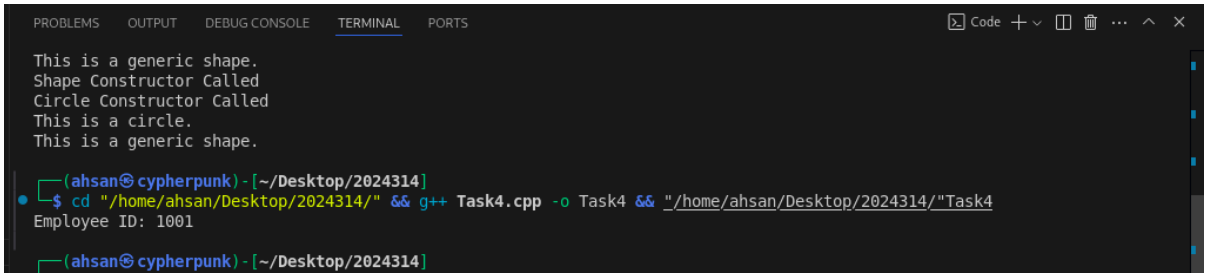
Add a function in Employee to show the employee's ID and demonstrate that it's not ambiguous using virtual inheritance.

Code :

```
#include <iostream>

using namespace std;


class Employee {

public:

   int empID;
```

```
    void showID() {
        cout << "Employee ID: " << empID << endl;
    }
};

// Virtual inheritance to avoid diamond problem
class Manager : virtual public Employee {
};

class Engineer : virtual public Employee {
};

class TechLead : public Manager, public Engineer {
};

int main() {
    TechLead t;
    t.empID = 1001;
    t.showID();  // No ambiguity

    return 0;
}
```

Screenshot :



Question 5 :

You are tasked to build a basic University Management System simulation using inheritance.

Requirements:

Base class: Person with members like name, age, gender, and a virtual function displayInfo().

Derived class: Faculty (inherits from Person) → adds designation, subject.

Derived class: Student (inherits from Person) → adds studentID, department, semester.

Further derive TeachingAssistant from both Student and Faculty.

Task:

Demonstrate multiple inheritance and use of constructor initialization lists.

Resolve ambiguity using scope resolution and/or virtual inheritance.

Override displayInfo() in all derived classes to show their full info.

In main(), create one object each of Faculty, Student, and TeachingAssistant, and call displayInfo() on

each.

Code :

```
#include <iostream>
using namespace std;

class Person {
public:
   string name;
   int age;
   string gender;

   virtual void displayInfo() {
      cout << "Name: " << name << ", Age: " << age << ", Gender: " << gender
<< endl;
   }
};

class Faculty : virtual public Person {
public:
   string designation;
   string subject;

   void displayInfo() override {
      cout << "Faculty Info:\n";
```

```
    Person::displayInfo();
    cout << "Designation: " << designation << ", Subject: " << subject << endl;
  }
};

class Student : virtual public Person {
public:
  string studentID;
  string department;
  int semester;

  void displayInfo() override {
    cout << "Student Info:\n";
    Person::displayInfo();
    cout << "Student ID: " << studentID << ", Department: " << department << ", Semester: " << semester << endl;
  }
};

class TeachingAssistant : public Student, public Faculty {
public:
  void displayInfo() override {
    cout << "Teaching Assistant Info:\n";
    Person::displayInfo();
```

```
    cout << "Student ID: " << studentID << ", Dept: " << department << ", Semester: " << semester << endl;
    cout << "Designation: " << designation << ", Subject: " << subject << endl;
  }
};

int main() {
  Faculty f;
  f.name = "Dr. Ahsan";
  f.age = 40;
  f.gender = "Male";
  f.designation = "Professor";
  f.subject = "OOP";
  f.displayInfo();

  cout << endl;

  Student s;
  s.name = "Sara";
  s.age = 21;
  s.gender = "Female";
  s.studentID = "CS112-5678";
  s.department = "CS";
  s.semester = 5;
  s.displayInfo();
```

```
    cout << endl;

    TeachingAssistant ta;
    ta.name = "Ali";
    ta.age = 23;
    ta.gender = "Male";
    ta.studentID = "CS112-3456";
    ta.department = "CS";
    ta.semester = 7;
    ta.designation = "TA";
    ta.subject = "Data Structures";
    ta.displayInfo();

    return 0;
}
```

Screenshot :

Question 6 :

Objective: Understand and verify the order of constructor and destructor calls in a multilevel inheritance

scenario.

Task:

Implement the following class hierarchy:

Point class (base)

Circle class derived from Point

Cylinder class derived from Circle.

Each class should:

Have a constructor and a destructor.

Each constructor should print when it's being called.

Each destructor should also print when it's being destroyed.

Then create an object of type Cylinder in main() and observe the output.

Code :

```
#include <iostream>
using namespace std;

class Point {
public:
  Point() {
    cout << "Point constructor called" << endl;
  }
  ~Point() {
    cout << "Point destructor called" << endl;
```

```
    }
};

class Circle : public Point {
public:
    Circle() {
        cout << "Circle constructor called" << endl;
    }
    ~Circle() {
        cout << "Circle destructor called" << endl;
    }
};

class Cylinder : public Circle {
public:
    Cylinder() {
        cout << "Cylinder constructor called" << endl;
    }
    ~Cylinder() {
        cout << "Cylinder destructor called" << endl;
    }
};

int main() {
    Cylinder c;
```

```
    return 0;
}
```

Screenshot :