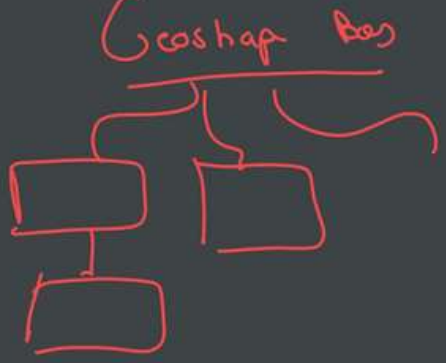
Assignments

- Geoshape Example.

  Design rest of classes Circle, Triangle (Applying features of changing types of inheritance).

- implement Print function → Public inheritance — what should I do?
  for all shapes That → Protected inheritance — what happened or need to be changed
  show each shape area & Perimeter in Design.
  - Add Rhombus, Cube → shapes (who's Parent class).
    - which type of inheritance I should use.
  - implement Area, Volume.
- Create stand alone function That Compare any 2 shapes 2gether → (Area)
- in Previous Example ⇒ Consider creating CTrs in each Class That is Suitable To Class design → show when. To use default of Base & when create my own.
- overLoad = operator in each Class of shapes.

- Turn Geoshap into Abstract class
  & change what must be changed in your design.
- Create mevin function → use objects Dynamically created.
  → Try creating Pointer of Base.
  - after creating & calculate areas, volume, Perimeter of shapes → create sort to Print all shapes ASC, Desc.



```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;

class GeoShape {
protected:
    double dim1, dim2;
public:
    virtual double Area() const = 0;
    virtual double Perimeter() const = 0;
    virtual double Volume() const = 0;

    virtual void Print() const = 0;

    virtual ~GeoShape() {}
};

class Circle : public GeoShape {
public:
    Circle(double r = 0) {
        dim1 = r;
        dim2 = 0;
```

```
    double Area() const override { return 3.14159 * dim1 * dim1; }
    double Perimeter() const override { return 2 * 3.14159 * dim1; }
    double Volume() const override { return 0; }

    Circle& operator=(const Circle& other) {
        if (this != &other) {
            dim1 = other.dim1;
            dim2 = other.dim2;
        }
        return *this;
    }

    void Print() const override {
        cout << "\n[Circle]\n";
        cout << "Radius      : " << dim1 << endl;
        cout << "Area        : " << Area() << endl;
        cout << "Perimeter   : " << Perimeter() << endl;
        cout << "Volume      : " << Volume() << endl;
    }
};

class Triangle : protected GeoShape {
```

```
cout << "Circle A Before assignment: Area = " << A.Area() << endl;
Circle B(2);
cout << "Circle B Before assignment: Area = " << B.Area() << endl;
B = A;
cout << "Circle B after assignment: Area = " << B.Area() << endl;

cout << "\n===== Testing CompareArea() Alone =====\n";
Circle X(5);
Circle Y(3);

cout << "X Area = " << X.Area() << endl;
cout << "Y Area = " << Y.Area() << endl;
cout << "CompareArea(X, Y) : "
     << (CompareArea(&X, &Y) ? "X < Y" : "X >= Y") << endl;


cout << "\n===== Testing Dynamic Objects =====\n";
GeoShape* s1 = new Circle(5);
GeoShape* s2 = new Circle(3);
GeoShape* s4 = new Rhombus(6, 8);
GeoShape* s5 = new Cube(4);
GeoShape* s6 = new Sphere(3);

s1->Print();
```

```
    }

    double Volume() const override { return 0; }

    Triangle& operator=(const Triangle& other) {
        if (this != &other) {
            a = other.a;
            b = other.b;
            c = other.c;
        }
        return *this;
    }

    void Print() const override {
        cout << "\n[Triangle]\n";
        cout << "Sides       : " << a << ", " << b << ", " << c << endl;
        cout << "Area        : " << Area() << endl;
        cout << "Perimeter   : " << Perimeter() << endl;
        cout << "Volume      : " << Volume() << endl;
    }
};

class Rhombus : public GeoShape {
private:
```

```
Rhombus(double D1 = 0, double D2 = 0) {
    d1 = D1;
    d2 = D2;
    dim1 = D1;
    dim2 = D2;
}

double Area() const override { return (d1 * d2) / 2; }

double Perimeter() const override {
    double side = sqrt(pow(d1 / 2, 2) + pow(d2 / 2, 2)
    return 4 * side;
}

double Volume() const override { return 0; }

Rhombus& operator=(const Rhombus& other) {
    if (this != &other) {
        d1 = other.d1;
        d2 = other.d2;
        dim1 = other.dim1;
        dim2 = other.dim2;
    }
    return *this;
```

```
148         return *this;
149     }
150
151     void Print() const override {
152         cout << "\n[Cube]\n";
153         cout << "Side         : " << side << endl;
154         cout << "Area         : " << Area() << endl;
155         cout << "Perimeter    : " << Perimeter() << endl;
156         cout << "Volume       : " << Volume() << endl;
157     }
158 };
159
160 class Sphere : public GeoShape {
161 private:
162     double radius;
163
164 public:
165     Sphere(double r = 0) {
166         radius = r;
167         dim1 = r;
168         dim2 = 0;
169     }
170
171     double Area() const override { return 4 * 3.14159 * radius * radius; }
```

```
        return *this;
    }

    void Print() const override {
        cout << "\n[Sphere]\n";
        cout << "Radius       : " << radius << endl;
        cout << "Surface Area: " << Area() << endl;
        cout << "Perimeter    : " << Perimeter() << endl;
        cout << "Volume       : " << Volume() << endl;
    }
};

bool CompareArea(const GeoShape* s1, const GeoShape* s2) {
    return s1->Area() < s2->Area();
    }

void PrintSortedAreas(const vector<GeoShape*>& shapes) {
    cout << "\n=== Sorted Areas (Ascending) ===\n";
    for (auto* p : shapes) {
        p->Print();
        cout << "Area = " << p->Area() << "\n--------------------\n";
```

```
int main() {

    cout << "\n===== Testing operator= Alone =====\n";
    Circle A(10);
    cout << "Circle A Before assignment: Area = " << A.Area() << endl;
    Circle B(2);
    cout << "Circle B Before assignment: Area = " << B.Area() << endl;
    B = A;
    cout << "Circle B after assignment: Area = " << B.Area() << endl;

    cout << "\n===== Testing CompareArea() Alone =====\n";
    Circle X(5);
    Circle Y(3);

    cout << "X Area = " << X.Area() << endl;
    cout << "Y Area = " << Y.Area() << endl;
    cout << "CompareArea(X, Y) : "
         << (CompareArea(&X, &Y) ? "X < Y" : "X >= Y") << endl;


    cout << "\n===== Testing Dynamic Objects =====\n";
    GeoShape* s1 = new Circle(5);
    GeoShape* s2 = new Circle(3);
    GeoShape* s4 = new Rhombus(6, 8);
```

```
    GeoShape* s2 = new Circle(3);
    GeoShape* s4 = new Rhombus(6, 8);
    GeoShape* s5 = new Cube(4);
    GeoShape* s6 = new Sphere(3);

    s1->Print();
    s2->Print();
    s4->Print();
    s5->Print();


    cout << "\n====== Sorting All Shapes =====\n";
    vector<GeoShape*> shapes = {s1, s2, s4, s5,s6};

    sort(shapes.begin(), shapes.end(), CompareArea);

    PrintSortedAreas(shapes);

    for (auto* p : shapes)
        delete p;

    return 0;
}
```