**Task # 01**

#include <iostream>

#include <math.h>

using namespace std;

class Complex {

private:

    float real;

    float imag;

public:

    // Constructor

    Complex(float r = 0, float i = 0) : real(r), imag(i) {}

    // Overload +

    Complex operator+(const Complex& other) const {

        return Complex(real + other.real, imag + other.imag);

    }

    // Overload \*

    Complex operator\*(const Complex& other) const {

        float r = (real \* other.real) - (imag \* other.imag);

        float i = (real \* other.imag) + (imag \* other.real);

        return Complex(r, i);

    }

    // Display

    void display() const {

        cout << real << (imag >= 0 ? " + " : " - ") << abs(imag) << "i" << endl;

    }

};

int main() {

    Complex c1(3, 2);     // 3 + 2i

    Complex c2(1, 7);     // 1 + 7i

    Complex sum = c1 + c2;

    Complex product = c1 \* c2;

    cout << "c1 = "; c1.display();

    cout << "c2 = "; c2.display();

    cout << "\nSum (c1 + c2) = "; sum.display();

    cout << "Product (c1 \* c2) = "; product.display();

    return 0;

}

**Task # 02**

#include <iostream>

using namespace std;

// Base class

class Transport {

public:

    virtual float calculateFare(int distance) = 0;

    virtual ~Transport() {}

};

// Derived class: Bus

class Bus : public Transport {

public:

    float calculateFare(int distance) override {

        return distance \* 1.5;

    }

};

class Taxi : public Transport {

public:

    float calculateFare(int distance) override {

        return 100 + (distance \* 5);

    }

};

int main() {

    int distance;

    cout << "Enter travel distance in km: ";

    cin >> distance;

    Transport\* vehicle;

    // Bus Fare

    Bus bus;

    vehicle = &bus;

    cout << "Bus Fare: Rs. " << vehicle->calculateFare(distance) << endl;

    // Taxi Fare

    Taxi taxi;

    vehicle = &taxi;

    cout << "Taxi Fare: Rs. " << vehicle->calculateFare(distance) << endl;

    return 0;

}

**Task # 03**

#include <iostream>

#include <iomanip>

using namespace std;

void generateInvoice(string itemName, int quantity, float pricePerUnit) {

    float total = quantity \* pricePerUnit;

    cout << fixed << setprecision(2);

    cout << "\nInvoice (Unit-based):" << endl;

    cout << "Item: " << itemName << endl;

    cout << "Quantity: " << quantity << " x $" << pricePerUnit << endl;

    cout << "Total: $" << total << endl;

}

void generateInvoice(string itemName, float weightKg, float pricePerKg) {

    float total = weightKg \* pricePerKg;

    cout << fixed << setprecision(2);

    cout << "\nInvoice (Weight-based):" << endl;

    cout << "Item: " << itemName << endl;

    cout << "Weight: " << weightKg << " kg x $" << pricePerKg << endl;

    cout << "Total: $" << total << endl;

}

int main() {

    generateInvoice("Pen", 3, 2.0);

    generateInvoice("Apples", 1.5f, 4.0f);

    return 0;

}

**Task # 04**

#include <iostream>

using namespace std;

class Date {

private:

    int day, month, year;

public:

    Date(int d, int m, int y) : day(d), month(m), year(y) {}

    // Overload > operator

    bool operator>(const Date& other) const {

        if (year > other.year)

            return true;

        else if (year == other.year && month > other.month)

            return true;

        else if (year == other.year && month == other.month && day > other.day)

            return true;

        return false;

    }

    // Overload == operator

    bool operator==(const Date& other) const {

        return (day == other.day && month == other.month && year == other.year);

    }

    void display() const {

        cout << day << "/" << month << "/" << year;

    }

};

int main() {

    Date date1(15, 5, 2025);

    Date date2(10, 6, 2025);

    cout << "Date 1: ";

    date1.display();

    cout << "\nDate 2: ";

    date2.display();

    cout << "\n\n";

    if (date1 > date2)

        cout << "Date 1 is later than Date 2\n";

    else if (date1 == date2)

        cout << "Both dates are the same\n";

    else

        cout << "Date 1 is earlier than Date 2\n";

    return 0;

}

**Task # 05**

#include <iostream>

using namespace std;

class Employee {

protected:

    string name;

    float salary;

public:

    Employee(string n, float s) : name(n), salary(s) {}

    virtual void calculateBonus() {

        cout << name << ": Bonus not defined.\n";

    }

    virtual ~Employee() {}

};

// Derived class: Manager

class Manager : public Employee {

private:

    float performanceRating;

public:

    Manager(string n, float s, float rating)

        : Employee(n, s), performanceRating(rating) {}

    void calculateBonus() override {

        float bonus = salary \* (performanceRating / 10);

        cout << name << " (Manager) Bonus: $" << bonus << endl;

    }

};

// Derived class: Engineer

class Engineer : public Employee {

private:

    int skillLevel;

public:

    Engineer(string n, float s, int level)

        : Employee(n, s), skillLevel(level) {}

    void calculateBonus() override {

        float bonus = skillLevel \* 500;

        cout << name << " (Engineer) Bonus: $" << bonus << endl;

    }

};

class Intern : public Employee {

public:

    Intern(string n, float s) : Employee(n, s) {}

    void calculateBonus() override {

        cout << name << " (Intern) Bonus: $0 (Not eligible)\n";

    }

};

int main() {

    Employee\* emp1 = new Manager("Alice", 80000, 9.2);

    Employee\* emp2 = new Engineer("Bob", 60000, 4);

    Employee\* emp3 = new Intern("Charlie", 20000);

    emp1->calculateBonus();

    emp2->calculateBonus();

    emp3->calculateBonus();

    // Clean

    delete emp1;

    delete emp2;

    delete emp3;

    return 0;

}