TASK 01

#include <iostream>

#include <cmath>

using namespace std;

class Complex {

private:

float real;

float imag;

public:

Complex(float r = 0, float i = 0) {

real = r;

imag = i;

}

Complex operator + (const Complex& obj) {

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

}

Complex operator - (const Complex& obj) {

return Complex(real - obj.real, imag - obj.imag);

}

Complex operator \* (const Complex& obj) {

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

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

return Complex(r, i);

}

Complex operator / (const Complex& obj) {

float denominator = obj.real \* obj.real + obj.imag \* obj.imag;

float r = (real \* obj.real + imag \* obj.imag) / denominator;

float i = (imag \* obj.real - real \* obj.imag) / denominator;

return Complex(r, i);

}

friend float magnitude(const Complex& obj);

friend ostream& operator << (ostream& out, const Complex& obj);

};

float magnitude(const Complex& obj) {

return sqrt(obj.real \* obj.real + obj.imag \* obj.imag);

}

ostream& operator << (ostream& out, const Complex& obj) {

out << "(" << obj.real << (obj.imag >= 0 ? " + " : " - ") << fabs(obj.imag) << "i)";

return out;

}

int main() {

Complex c1(2, 5);

Complex c2(3, 4);

Complex sum = c1 + c2;

Complex diff = c1 - c2;

Complex prod = c1 \* c2;

Complex quot = c1 / c2;

cout << "First Complex Number: " << c1 << endl;

cout << "Second Complex Number: " << c2 << endl;

cout << "Sum: " << sum << endl;

cout << "Difference: " << diff << endl;

cout << "Product: " << prod << endl;

cout << "Quotient: " << quot << endl;

cout << "Magnitude of " << c1 << " is " << magnitude(c1) << endl;

cout << "Magnitude of " << c2 << " is " << magnitude(c2) << endl;

return 0;

}

TASK 02

#include <iostream>

using namespace std;

class Account;

class Manager {

public:

void displayAccount(const Account& acc);

void deposit(Account& acc, float amount);

void withdraw(Account& acc, float amount);

};

class Account {

private:

int accountNumber;

float balance;

public:

Account(int accNo = 0, float bal = 0.0) {

accountNumber = accNo;

balance = bal;

}

friend class Manager;

friend void transferFunds(Account& from, Account& to, float amount);

};

void Manager::displayAccount(const Account& acc) {

cout << "Account Number: " << acc.accountNumber << endl;

cout << "Balance: $" << acc.balance << endl;

}

void Manager::deposit(Account& acc, float amount) {

if (amount > 0) {

acc.balance += amount;

cout << "Deposited $" << amount << " to Account " << acc.accountNumber << endl;

} else {

cout << "Invalid deposit amount." << endl;

}

}

void Manager::withdraw(Account& acc, float amount) {

if (amount > 0 && acc.balance >= amount) {

acc.balance -= amount;

cout << "Withdrew $" << amount << " from Account " << acc.accountNumber << endl;

} else {

cout << "Insufficient balance or invalid amount." << endl;

}

}

void transferFunds(Account& from, Account& to, float amount) {

if (amount > 0 && from.balance >= amount) {

from.balance -= amount;

to.balance += amount;

cout << "Transferred $" << amount << " from Account " << from.accountNumber

<< " to Account " << to.accountNumber << endl;

} else {

cout << "Transfer failed due to insufficient balance or invalid amount." << endl;

}

}

int main() {

Account acc1(115, 500.0);

Account acc2(123, 300.0);

Manager manager;

cout << "Initial Account States:\n";

manager.displayAccount(acc1);

manager.displayAccount(acc2);

cout << "\nPerforming Transactions:\n";

manager.deposit(acc1, 250.0);

manager.withdraw(acc2, 120.0);

transferFunds(acc1, acc2, 230.0);

cout << "\nFinal Account States:\n";

manager.displayAccount(acc1);

manager.displayAccount(acc2);

return 0;

}

TASK 03

#include <iostream>

using namespace std;

class Vector2D {

private:

float x, y;

public:

Vector2D(float xVal = 0, float yVal = 0) {

x = xVal;

y = yVal;

}

Vector2D operator+(const Vector2D& other) {

return Vector2D(x + other.x, y + other.y);

}

Vector2D operator-(const Vector2D& other) {

return Vector2D(x - other.x, y - other.y);

}

Vector2D operator\*(float scalar) {

return Vector2D(x \* scalar, y \* scalar);

}

friend float dotProduct(const Vector2D& a, const Vector2D& b);

friend ostream& operator<<(ostream& out, const Vector2D& v) {

out << "(" << v.x << ", " << v.y << ")";

return out;

}

};

float dotProduct(const Vector2D& a, const Vector2D& b) {

return a.x \* b.x + a.y \* b.y;

}

int main() {

Vector2D v1(3, 5), v2(2, 4);

Vector2D sum = v1 + v2;

Vector2D diff = v1 - v2;

Vector2D scaled = v1 \* 2.5;

cout << "v1 = " << v1 << endl;

cout << "v2 = " << v2 << endl;

cout << "v1 + v2 = " << sum << endl;

cout << "v1 - v2 = " << diff << endl;

cout << "v1 \* 2.5 = " << scaled << endl;

cout << "Dot product = " << dotProduct(v1, v2) << endl;

return 0;

}

TASK 04

#include <iostream>

#include <string>

using namespace std;

class Car;

void comparePrice(const Car& car1, const Car& car2);

class InventoryManager;

class Car {

private:

string model;

double price;

public:

Car(string m = "", double p = 0.0) {

model = m;

price = p;

}

friend class InventoryManager;

friend void comparePrice(const Car& car1, const Car& car2);

};

class InventoryManager {

public:

void updatePrice(Car& car, double newPrice) {

car.price = newPrice;

}

void displayDetails(const Car& car) {

cout << "Model: " << car.model << ", Price: $" << car.price << endl;

}

};

void comparePrice(const Car& car1, const Car& car2) {

if (car1.price > car2.price) {

cout << car1.model << " is more expensive.\n";

} else if (car1.price < car2.price) {

cout << car2.model << " is more expensive.\n";

} else {

cout << "Both cars are priced equally.\n";

}

}

int main() {

Car car1("Toyota Lexus", 50000.0);

Car car2("kia sportage", 32000.0);

InventoryManager manager;

manager.displayDetails(car1);

manager.displayDetails(car2);

manager.updatePrice(car1, 28000.0);

cout << "\nAfter price update:\n";

manager.displayDetails(car1);

cout << "\nComparing cars:\n";

comparePrice(car1, car2);

return 0;

}

TASK 05

#include <iostream>

using namespace std;

class Matrix2x2;

class MatrixHelper {

public:

void updateElement(Matrix2x2& mat, int row, int col, int value);

};

class Matrix2x2 {

private:

int a, b, c, d;

public:

Matrix2x2(int a1 = 0, int b1 = 0, int c1 = 0, int d1 = 0) {

a = a1;

b = b1;

c = c1;

d = d1;

}

Matrix2x2 operator+(const Matrix2x2& other) {

return Matrix2x2(a + other.a, b + other.b, c + other.c, d + other.d);

}

Matrix2x2 operator-(const Matrix2x2& other) {

return Matrix2x2(a - other.a, b - other.b, c - other.c, d - other.d);

}

Matrix2x2 operator\*(const Matrix2x2& other) {

return Matrix2x2(

a \* other.a + b \* other.c,

a \* other.b + b \* other.d,

c \* other.a + d \* other.c,

c \* other.b + d \* other.d

);

}

friend int determinant(const Matrix2x2& mat);

friend class MatrixHelper;

void display() {

cout << "[ " << a << " " << b << " ]\n";

cout << "[ " << c << " " << d << " ]\n";

}

};

int determinant(const Matrix2x2& mat) {

return mat.a \* mat.d - mat.b \* mat.c;

}

void MatrixHelper::updateElement(Matrix2x2& mat, int row, int col, int value) {

if (row == 0 && col == 0) mat.a = value;

else if (row == 0 && col == 1) mat.b = value;

else if (row == 1 && col == 0) mat.c = value;

else if (row == 1 && col == 1) mat.d = value;

else cout << "Invalid index.\n";

}

int main() {

Matrix2x2 m1(1, 2, 3, 4);

Matrix2x2 m2(5, 6, 7, 8);

cout << "Matrix 1:\n";

m1.display();

cout << "\nMatrix 2:\n";

m2.display();

Matrix2x2 sum = m1 + m2;

Matrix2x2 diff = m1 - m2;

Matrix2x2 prod = m1 \* m2;

cout << "\nSum:\n";

sum.display();

cout << "\nDifference:\n";

diff.display();

cout << "\nProduct:\n";

prod.display();

cout << "\nDeterminant of Matrix 1: " << determinant(m1) << endl;

MatrixHelper helper;

helper.updateElement(m1, 0, 0, 9);

cout << "\nMatrix 1 after update:\n";

m1.display();

return 0;

}