1. Класс – комплексное число. Методы – сумма, разность, произведение комплексных чисел. Поля – действительная и мнимая часть комплексного числа.

#include <iostream>

using namespace std;

class Complex {

public:

Complex() {

real = 0;

imaginary = 0;

}

Complex(double r, double i) {

real = r;

imaginary = i;

}

Complex(const Complex& other) {

real = other.real;

imaginary = other.imaginary;

}

Complex& operator=(const Complex& other) {

real = other.real;

imaginary = other.imaginary;

return \*this;

}

Complex operator+(const Complex& other) const {

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

}

Complex operator-(const Complex& other) const {

return Complex(real - other.real, imaginary - other.imaginary);

}

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

double r = real \* other.real - imaginary \* other.imaginary;

double i = real \* other.imaginary + imaginary \* other.real;

return Complex(r, i);

}

Complex operator+(const double& number) const {

return Complex(real + number, imaginary);

}

Complex operator-(const double& number) const {

return Complex(real - number, imaginary);

}

Complex operator\*(const double& number) const {

return Complex(real \* number, imaginary \* number);

}

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

return (real == other.real && imaginary == other.imaginary);

}

bool operator!=(const Complex& other) const {

return !(\*this == other);

}

bool operator>(const Complex& other) const {

return (real > other.real);

}

bool operator<(const Complex& other) const {

return (real < other.real);

}

bool operator>=(const Complex& other) const {

return !(\*this < other);

}

bool operator<=(const Complex& other) const {

return !(\*this > other);

}

void print() const {

if (imaginary < 0) {

cout << real << " - " << -imaginary << "i" << endl;

} else {

cout << real << " + " << imaginary << "i" << endl;

}

}

private:

double real;

double imaginary;

};

int main() {

Complex a(2, 3);

Complex b(4, 5);

Complex c = a + b;

Complex d = a - b;

Complex e = a \* b;

Complex f = a + 2.5;

Complex g = b - 1.2;

Complex h = a \* 1.5;

bool equal = (a == b);

bool not\_equal = (a != b);

bool greater = (a > b);

bool less = (a < b);

bool greater\_or\_equal = (a >= b);

bool less\_or\_equal = (a <= b);

cout << "a = ";

a.print();

cout << "b = ";

b.print();

cout << "a + b = ";

c.print();

cout << "a - b = ";

d.print();

cout << "a \* b = ";

e.print();

cout << "a + 2.5 = ";

f.print();

cout << "b - 1.2 = ";

g.print();

/////////////////////////////////////////////////////////

2. Класс – десятичная дробь. Методы класса – сумма, разность, произведение, частное десятичных дробей. Поля класса – целая и дробная часть числа.

#include <iostream>

class Decimal {

public:

Decimal(double value = 0.0) {

integerPart = static\_cast<int>(value);

decimalPart = value - integerPart;

}

Decimal(const Decimal& other) {

integerPart = other.integerPart;

decimalPart = other.decimalPart;

}

Decimal& operator=(const Decimal& other) {

if (this != &other) {

integerPart = other.integerPart;

decimalPart = other.decimalPart;

}

return \*this;

}

Decimal operator+(const Decimal& other) const {

return Decimal(static\_cast<double>(\*this) + static\_cast<double>(other));

}

Decimal operator-(const Decimal& other) const {

return Decimal(static\_cast<double>(\*this) - static\_cast<double>(other));

}

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

return Decimal(static\_cast<double>(\*this) \* static\_cast<double>(other));

}

Decimal operator/(const Decimal& other) const {

return Decimal(static\_cast<double>(\*this) / static\_cast<double>(other));

}

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

return integerPart == other.integerPart && decimalPart == other.decimalPart;

}

bool operator!=(const Decimal& other) const {

return !(\*this == other);

}

bool operator>(const Decimal& other) const {

return static\_cast<double>(\*this) > static\_cast<double>(other);

}

bool operator<(const Decimal& other) const {

return static\_cast<double>(\*this) < static\_cast<double>(other);

}

bool operator>=(const Decimal& other) const {

return static\_cast<double>(\*this) >= static\_cast<double>(other);

}

bool operator<=(const Decimal& other) const {

return static\_cast<double>(\*this) <= static\_cast<double>(other);

}

operator double() const {

return static\_cast<double>(integerPart) + decimalPart;

}

private:

int integerPart;

double decimalPart;

};

int main() {

Decimal d1(3.14);

Decimal d2(2.71);

Decimal d3 = d1 + d2;

std::cout << d3 << std::endl; // Output: 5.85

Decimal d4 = d1 \* 2;

std::cout << d4 << std::endl; // Output: 6.28

bool b1 = d1 == d2;

std::cout << b1 << std::endl; // Output: 0

bool b2 = d1 != d2;

std::cout << b2 << std::endl; // Output: 1

bool b3 = d1 > d2;

std::cout << b3 << std::endl; // Output: 1

bool b4 = d1 <= d2;

std::cout << b4 << std::endl; // Output: 0

return 0;

}

/////////////////////////////////////////////////////////

3. Класс – натуральная дробь. Методы класса – сумма, разность, произведение, частное натуральных дробей. Поля класса – числитель и знаменатель дроби.

class NaturalFraction {

public:

// конструктор по умолчанию

NaturalFraction() : numerator(0), denominator(1) {}

// конструктор с параметрами

NaturalFraction(int num, int den = 1) : numerator(num), denominator(den) {}

// перегрузка оператора присваивания

NaturalFraction& operator=(const NaturalFraction& other) {

numerator = other.numerator;

denominator = other.denominator;

return \*this;

}

// перегрузка арифметических операторов для операций над экземпляром класса и числами

NaturalFraction operator+(const NaturalFraction& other) const {

int lcm = get\_lcm(denominator, other.denominator);

int num = numerator \* (lcm / denominator) + other.numerator \* (lcm / other.denominator);

return NaturalFraction(num, lcm);

}

NaturalFraction operator-(const NaturalFraction& other) const {

int lcm = get\_lcm(denominator, other.denominator);

int num = numerator \* (lcm / denominator) - other.numerator \* (lcm / other.denominator);

return NaturalFraction(num, lcm);

}

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

return NaturalFraction(numerator \* other.numerator, denominator \* other.denominator);

}

NaturalFraction operator/(const NaturalFraction& other) const {

return NaturalFraction(numerator \* other.denominator, denominator \* other.numerator);

}

NaturalFraction operator+(int num) const {

return NaturalFraction(numerator + num \* denominator, denominator);

}

NaturalFraction operator-(int num) const {

return NaturalFraction(numerator - num \* denominator, denominator);

}

NaturalFraction operator\*(int num) const {

return NaturalFraction(numerator \* num, denominator);

}

NaturalFraction operator/(int num) const {

return NaturalFraction(numerator, denominator \* num);

}

// перегрузка арифметических операторов для операций над двумя экземплярами

NaturalFraction operator+(const NaturalFraction& other) {

\*this = \*this + other;

return \*this;

}

NaturalFraction operator-(const NaturalFraction& other) {

\*this = \*this - other;

return \*this;

}

NaturalFraction operator\*(const NaturalFraction& other) {

\*this = \*this \* other;

return \*this;

}

NaturalFraction operator/(const NaturalFraction& other) {

\*this = \*this / other;

return \*this;

}

// перегрузка операторов сравнения

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

return numerator == other.numerator && denominator == other.denominator;

}

bool operator!=(const NaturalFraction& other) const {

return !(\*this == other);

}

bool operator<(const NaturalFraction& other) const {

int lcm = get\_lcm(denominator, other.denominator);

return numerator \* (lcm / denominator) < other.numerator \* (lcm / other.denominator);

}

bool operator<=(const NaturalFraction& other) const {

return \*this == other || \*this < other;

}

bool operator>(const NaturalFraction& other) const {

return !(\*this

/////////////////////////////////////////////////////////

4. Класс – отрезок. Методы – расчет середины и длины. Поля класса – координаты начала, конца, середины и длина.

#include <iostream>

#include <cmath>

using namespace std;

class Segment {

private:

double start\_x, start\_y, end\_x, end\_y;

public:

Segment(double x1, double y1, double x2, double y2): start\_x(x1), start\_y(y1), end\_x(x2), end\_y(y2) {}

// Расчет середины отрезка

double middle\_x() const { return (start\_x + end\_x) / 2; }

double middle\_y() const { return (start\_y + end\_y) / 2; }

// Расчет длины отрезка

double length() const {

double dx = end\_x - start\_x;

double dy = end\_y - start\_y;

return sqrt(dx\*dx + dy\*dy);

}

// Перегрузка оператора присваивания

Segment& operator=(const Segment& other) {

start\_x = other.start\_x;

start\_y = other.start\_y;

end\_x = other.end\_x;

end\_y = other.end\_y;

return \*this;

}

// Перегрузка операторов сложения, вычитания и умножения для операций над экземпляром класса и целым и вещественным числами

Segment operator+(double n) const { return Segment(start\_x + n, start\_y + n, end\_x + n, end\_y + n); }

Segment operator-(double n) const { return Segment(start\_x - n, start\_y - n, end\_x - n, end\_y - n); }

Segment operator\*(double n) const { return Segment(start\_x \* n, start\_y \* n, end\_x \* n, end\_y \* n); }

// Перегрузка операторов сложения, вычитания и умножения для операций над двумя экземплярами класса

Segment operator+(const Segment& other) const { return Segment(start\_x + other.start\_x, start\_y + other.start\_y, end\_x + other.end\_x, end\_y + other.end\_y); }

Segment operator-(const Segment& other) const { return Segment(start\_x - other.start\_x, start\_y - other.start\_y, end\_x - other.end\_x, end\_y - other.end\_y); }

Segment operator\*(const Segment& other) const { return Segment(start\_x \* other.start\_x, start\_y \* other.start\_y, end\_x \* other.end\_x, end\_y \* other.end\_y); }

// Перегрузка операторов сравнения

bool operator==(const Segment& other) const { return start\_x == other.start\_x && start\_y == other.start\_y && end\_x == other.end\_x && end\_y == other.end\_y; }

bool operator!=(const Segment& other) const { return !(\*this == other); }

bool operator>(const Segment& other) const { return length() > other.length(); }

bool operator<(const Segment& other) const { return length() < other.length(); }

bool operator>=(const Segment& other) const { return length() >= other.length(); }

bool operator<=(const Segment& other) const { return length() <= other.length(); }

// Перегрузка логических операторов

bool operator&&(const Segment&

/////////////////////////////////////////////////////////

5. Класс – окружность. Методы – расчет длины окружности и ее площади. Поля класса – координаты центра, радиус, длина, площадь окружности.

#include <cmath>

class Circle {

public:

double x, y; // координаты центра окружности

double r; // радиус окружности

double l; // длина окружности

double s; // площадь окружности

// конструктор по умолчанию

Circle() : x(0), y(0), r(0), l(0), s(0) {}

// конструктор с параметрами

Circle(double x, double y, double r) : x(x), y(y), r(r) {

l = 2 \* M\_PI \* r;

s = M\_PI \* r \* r;

}

// оператор присваивания

Circle& operator=(const Circle& other) {

if (this != &other) {

x = other.x;

y = other.y;

r = other.r;

l = other.l;

s = other.s;

}

return \*this;

}

// оператор сложения

Circle operator+(const Circle& other) const {

return Circle(x + other.x, y + other.y, r + other.r);

}

// оператор вычитания

Circle operator-(const Circle& other) const {

return Circle(x - other.x, y - other.y, r - other.r);

}

// оператор умножения на число

Circle operator\*(double num) const {

return Circle(x \* num, y \* num, r \* num);

}

// оператор деления на число

Circle operator/(double num) const {

return Circle(x / num, y / num, r / num);

}

// оператор сложения с числом (double)

Circle operator+(double num) const {

return Circle(x + num, y + num, r + num);

}

// оператор вычитания числа (double)

Circle operator-(double num) const {

return Circle(x - num, y - num, r - num);

}

// оператор умножения на число (double)

Circle operator\*(double num) const {

return Circle(x \* num, y \* num, r \* num);

}

// оператор деления на число (double)

Circle operator/(double num) const {

return Circle(x / num, y / num, r / num);

}

// оператор сравнения на равенство

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

return x == other.x && y == other.y && r == other.r;

}

// оператор сравнения на неравенство

bool operator!=(const Circle& other) const {

return !(\*this == other);

}

// оператор больше

bool operator>(const Circle& other) const {

return r > other.r;

}

// оператор меньше

bool operator<(const Circle& other) const {

return r < other.r;

}

// оператор больше или равно

bool operator>=(const Circle& other) const {

return r >= other.r;

}

// оператор меньше или равно

bool operator<=(const Circle& other) const {

return r <= other.r;

/////////////////////////////////////////////////////////

6. Класс – шар. Методы – расчет площади и объема шара. Поля – координаты центра (x, y, z), радиус шара

#include <iostream>

#include <cmath>

class Complex {

private:

double real, imag;

public:

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

// операторы присваивания

Complex& operator=(const Complex& other) {

real = other.real;

imag = other.imag;

return \*this;

}

// арифметические операторы для операций над экземпляром класса

Complex operator+(const Complex& other) const {

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

}

Complex operator-(const Complex& other) const {

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

}

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

return Complex(real \* other.real - imag \* other.imag,

imag \* other.real + real \* other.imag);

}

// арифметические операторы для операций над целыми и вещественными числами

Complex operator+(const double& scalar) const {

return Complex(real + scalar, imag);

}

Complex operator-(const double& scalar) const {

return Complex(real - scalar, imag);

}

Complex operator\*(const double& scalar) const {

return Complex(real \* scalar, imag \* scalar);

}

// операторы сравнения

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

return (real == other.real) && (imag == other.imag);

}

bool operator!=(const Complex& other) const {

return !(\*this == other);

}

bool operator>(const Complex& other) const {

return sqrt(real \* real + imag \* imag) > sqrt(other.real \* other.real + other.imag \* other.imag);

}

bool operator<(const Complex& other) const {

return sqrt(real \* real + imag \* imag) < sqrt(other.real \* other.real + other.imag \* other.imag);

}

bool operator>=(const Complex& other) const {

return sqrt(real \* real + imag \* imag) >= sqrt(other.real \* other.real + other.imag \* other.imag);

}

bool operator<=(const Complex& other) const {

return sqrt(real \* real + imag \* imag) <= sqrt(other.real \* other.real + other.imag \* other.imag);

}

// логические операторы

bool operator&&(const Complex& other) const {

return (real && other.real) && (imag && other.imag);

}

bool operator||(const Complex& other) const {

return (real || other.real) || (imag || other.imag);

}

// дополнительные методы

double magnitude() const {

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

}

void print() const {

std::cout << "(" << real << ", " << imag << ")";

}

};

// перегруженный оператор вывода в поток

std::ostream& operator<<(std::ostream& os, const Complex& c) {

os << "(" << c.real << ", " << c.imag << ")";

return os;

}

int main() {

Complex

/////////////////////////////////////////////////////////

7. Класс – конус. Методы – расчет площади и объема конуса. Поля – радиус основания, высота конуса.

#include <cmath>

class Cone {

private:

double radius;

double height;

public:

Cone(double r, double h) : radius(r), height(h) {}

// методы для расчета площади и объема конуса

double surfaceArea() const {

return M\_PI \* radius \* (radius + sqrt(height\*height + radius\*radius));

}

double volume() const {

return M\_PI \* radius \* radius \* height / 3.0;

}

// перегрузка операторов арифметических операций

Cone operator+(double val) const {

return Cone(radius + val, height + val);

}

Cone operator-(double val) const {

return Cone(radius - val, height - val);

}

Cone operator\*(double val) const {

return Cone(radius \* val, height \* val);

}

Cone operator+(const Cone& other) const {

return Cone(radius + other.radius, height + other.height);

}

Cone operator-(const Cone& other) const {

return Cone(radius - other.radius, height - other.height);

}

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

return Cone(radius \* other.radius, height \* other.height);

}

// перегрузка операторов сравнения

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

return (radius == other.radius) && (height == other.height);

}

bool operator!=(const Cone& other) const {

return !(\*this == other);

}

bool operator>(const Cone& other) const {

return volume() > other.volume();

}

bool operator<(const Cone& other) const {

return volume() < other.volume();

}

bool operator>=(const Cone& other) const {

return volume() >= other.volume();

}

bool operator<=(const Cone& other) const {

return volume() <= other.volume();

}

// перегрузка оператора присваивания

Cone& operator=(const Cone& other) {

radius = other.radius;

height = other.height;

return \*this;

}

};

/////////////////////////////////////////////////////////

8. Класс – параллелепипед. Методы – расчет площади поверхности и объема параллелепипеда. Поля – ребра a, b, c, площадь и объем.

#include <iostream>

class Parallelepiped {

public:

double a, b, c, area, volume;

Parallelepiped(double a, double b, double c) : a(a), b(b), c(c) {

area = 2 \* (a \* b + b \* c + a \* c);

volume = a \* b \* c;

}

Parallelepiped& operator=(const Parallelepiped& other) {

a = other.a;

b = other.b;

c = other.c;

area = other.area;

volume = other.volume;

return \*this;

}

Parallelepiped operator+(const Parallelepiped& other) const {

return Parallelepiped(a + other.a, b + other.b, c + other.c);

}

Parallelepiped operator-(const Parallelepiped& other) const {

return Parallelepiped(a - other.a, b - other.b, c - other.c);

}

Parallelepiped operator\*(double scalar) const {

return Parallelepiped(a \* scalar, b \* scalar, c \* scalar);

}

friend Parallelepiped operator\*(double scalar, const Parallelepiped& p) {

return p \* scalar;

}

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

return a == other.a && b == other.b && c == other.c;

}

bool operator!=(const Parallelepiped& other) const {

return !(\*this == other);

}

bool operator>(const Parallelepiped& other) const {

return volume > other.volume;

}

bool operator<(const Parallelepiped& other) const {

return volume < other.volume;

}

bool operator>=(const Parallelepiped& other) const {

return volume >= other.volume;

}

bool operator<=(const Parallelepiped& other) const {

return volume <= other.volume;

}

};

int main() {

Parallelepiped p1(2, 3, 4);

Parallelepiped p2(1, 2, 3);

// операторы присваивания

Parallelepiped p3 = p1;

p3 = p2;

// арифметические операторы

Parallelepiped p4 = p1 + p2;

Parallelepiped p5 = p1 - p2;

Parallelepiped p6 = p1 \* 2.0;

Parallelepiped p7 = 3.0 \* p1;

// операторы сравнения

std::cout << (p1 == p2) << std::endl;

std::cout << (p1 != p2) << std::endl;

std::cout << (p1 > p2) << std::endl;

std::cout << (p1 < p2) << std::endl;

std::cout << (p1 >= p2) << std::endl;

std::cout << (p1 <= p2) << std::endl;

return 0;

}

/////////////////////////////////////////////////////////

9. Класс – целочисленный вектор фиксированной размерности. Методы – сумма, разность, скалярное произведение векторов, сравнение двух векторов на равенство, вычисление длины вектора. Поля – размерность вектора, координаты вектора.

#include <iostream>

#include <cmath>

class Vector {

public:

Vector(int n) : size(n), data(new int[n]) {}

Vector(const Vector& other) : size(other.size), data(new int[other.size]) {

for (int i = 0; i < size; i++) {

data[i] = other.data[i];

}

}

~Vector() { delete[] data; }

Vector& operator=(const Vector& other) {

if (this != &other) {

delete[] data;

size = other.size;

data = new int[size];

for (int i = 0; i < size; i++) {

data[i] = other.data[i];

}

}

return \*this;

}

Vector operator+(const Vector& other) const {

Vector result(size);

for (int i = 0; i < size; i++) {

result.data[i] = data[i] + other.data[i];

}

return result;

}

Vector operator-(const Vector& other) const {

Vector result(size);

for (int i = 0; i < size; i++) {

result.data[i] = data[i] - other.data[i];

}

return result;

}

Vector operator\*(int scalar) const {

Vector result(size);

for (int i = 0; i < size; i++) {

result.data[i] = data[i] \* scalar;

}

return result;

}

int operator\*(const Vector& other) const {

int result = 0;

for (int i = 0; i < size; i++) {

result += data[i] \* other.data[i];

}

return result;

}

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

if (size != other.size) {

return false;

}

for (int i = 0; i < size; i++) {

if (data[i] != other.data[i]) {

return false;

}

}

return true;

}

bool operator!=(const Vector& other) const {

return !(\*this == other);

}

bool operator<(const Vector& other) const {

for (int i = 0; i < size; i++) {

if (data[i] < other.data[i]) {

return true;

}

else if (data[i] > other.data[i]) {

return false;

}

}

return false;

}

bool operator>(const Vector& other) const {

for (int i = 0; i < size; i++) {

if (data[i] > other.data[i]) {

return true;

}

else if (data[i] < other.data[i]) {

return false;

}

}

return false;

}

bool operator<=(const Vector& other) const {

return (\*this < other) || (\*this == other);

}

bool operator>=(const Vector& other) const {

return (\*this > other) || (\*this == other);

}

double length() const {

double result = 0;

for (int i = 0; i < size; i++) {

result += data[i] \* data[i];

}

return std::sqrt(result);

}

private:

int size;

int\* data;

};

/////////////////////////////////////////////////////////

10. Класс – треугольник. Методы – расчет площади и периметра треугольника. Поля – стороны треугольника.

#include <cmath>

class Vector {

public:

Vector(int dim = 3) : dim(dim) {

coords = new int[dim];

for (int i = 0; i < dim; i++) {

coords[i] = 0;

}

}

Vector(const Vector& other) : dim(other.dim) {

coords = new int[dim];

for (int i = 0; i < dim; i++) {

coords[i] = other.coords[i];

}

}

~Vector() {

delete[] coords;

}

Vector& operator=(const Vector& other) {

if (this != &other) {

delete[] coords;

dim = other.dim;

coords = new int[dim];

for (int i = 0; i < dim; i++) {

coords[i] = other.coords[i];

}

}

return \*this;

}

Vector operator+(const Vector& other) const {

Vector result(dim);

for (int i = 0; i < dim; i++) {

result.coords[i] = coords[i] + other.coords[i];

}

return result;

}

Vector operator-(const Vector& other) const {

Vector result(dim);

for (int i = 0; i < dim; i++) {

result.coords[i] = coords[i] - other.coords[i];

}

return result;

}

Vector operator\*(int scalar) const {

Vector result(dim);

for (int i = 0; i < dim; i++) {

result.coords[i] = coords[i] \* scalar;

}

return result;

}

Vector operator\*(double scalar) const {

Vector result(dim);

for (int i = 0; i < dim; i++) {

result.coords[i] = round(coords[i] \* scalar);

}

return result;

}

int operator\*(const Vector& other) const {

int result = 0;

for (int i = 0; i < dim; i++) {

result += coords[i] \* other.coords[i];

}

return result;

}

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

if (dim != other.dim) {

return false;

}

for (int i = 0; i < dim; i++) {

if (coords[i] != other.coords[i]) {

return false;

}

}

return true;

}

bool operator!=(const Vector& other) const {

return !(\*this == other);

}

bool operator>(const Vector& other) const {

return length() > other.length();

}

bool operator<(const Vector& other) const {

return length() < other.length();

}

bool operator>=(const Vector& other) const {

return length() >= other.length();

}

bool operator<=(const Vector& other) const {

return length() <= other.length();

}

double length() const {

double sum = 0.0;

for (int i = 0; i < dim; i++) {

sum += coords[i] \* coords[i];

}

return sqrt(sum);

}

private:

int dim;

int\* coords;

};

/////////////////////////////////////////////////////////

11. Класс – трапеция. Методы – расчет площади и периметра трапеции. Поля – стороны трапеции, площадь и периметр.

#include <iostream>

#include <cmath>

class Trapezoid {

public:

double side1, side2, side3, side4, area, perimeter;

// Конструктор класса

Trapezoid(double s1 = 0, double s2 = 0, double s3 = 0, double s4 = 0)

: side1(s1), side2(s2), side3(s3), side4(s4), area(0), perimeter(0) {}

// Расчет площади трапеции

void calcArea() {

area = (side1 + side2) / 2 \* sqrt(side3 \* side3 - pow((pow((side2 - side1), 2) + side3 \* side3 - side4 \* side4) / (2 \* (side2 - side1)), 2));

}

// Расчет периметра трапеции

void calcPerimeter() {

perimeter = side1 + side2 + side3 + side4;

}

// Перегрузка оператора присваивания

Trapezoid& operator=(const Trapezoid& t) {

side1 = t.side1;

side2 = t.side2;

side3 = t.side3;

side4 = t.side4;

area = t.area;

perimeter = t.perimeter;

return \*this;

}

// Перегрузка оператора сложения для операций над экземпляром класса и целым/вещественным числом

Trapezoid operator+(double val) const {

return Trapezoid(side1 + val, side2 + val, side3 + val, side4 + val);

}

// Перегрузка оператора вычитания для операций над экземпляром класса и целым/вещественным числом

Trapezoid operator-(double val) const {

return Trapezoid(side1 - val, side2 - val, side3 - val, side4 - val);

}

// Перегрузка оператора умножения для операций над экземпляром класса и целым/вещественным числом

Trapezoid operator\*(double val) const {

return Trapezoid(side1 \* val, side2 \* val, side3 \* val, side4 \* val);

}

// Перегрузка оператора сложения для операций над двумя экземплярами класса

Trapezoid operator+(const Trapezoid& t) const {

return Trapezoid(side1 + t.side1, side2 + t.side2, side3 + t.side3, side4 + t.side4);

}

// Перегрузка оператора вычитания для операций над двумя экземплярами класса

Trapezoid operator-(const Trapezoid& t) const {

return Trapezoid(side1 - t.side1, side2 - t.side2, side3 - t.side3, side4 - t.side4);

}

// Перегрузка оператора умн