**ОТЧЕТ**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**Вариант №11**

**Выполнил:**

**Студент гр. ДИНРБ-21/2**

**Ахмедов Шамиль**

Ахмедов Шамиль ДИНРБ-21/2

Вариант № 11

Создать класс vector3D, задаваемый тройкой координат. Обязательно должны быть реализованы: операции сложения и вычитания векторов,

векторное произведение векторов, скалярное произведение векторов, умножения на скаляр, сравнение векторов на совпадение,

вычисление длины вектора, сравнение длины векторов, вычисление угла между векторами.

**Заголовочный файл для класса Vector3D.h:**

#ifndef VECTOR3D\_H

#define VECTOR3D\_H

#pragma once

#include <iostream>

#include <cmath>

class Vector3D {

private:

double x, y, z;

public:

// Конструкторы

Vector3D();

Vector3D(double x, double y, double z);

// Методы-операции с присваиванием

Vector3D& operator+=(const Vector3D& other);

Vector3D& operator-=(const Vector3D& other);

Vector3D& operator\*=(double scalar);

// Методы класса

double length() const noexcept;

double dotProduct(const Vector3D& other) const noexcept;

Vector3D crossProduct(const Vector3D& other) const noexcept;

double angle(const Vector3D& other) const noexcept;

// Дружественные функции операции без присваивания

friend Vector3D operator+(const Vector3D& v1, const Vector3D& v2);

friend Vector3D operator-(const Vector3D& v1, const Vector3D& v2);

friend Vector3D operator\*(const Vector3D& v, double scalar);

friend Vector3D operator\*(double scalar, const Vector3D& v);

friend bool operator==(const Vector3D& v1, const Vector3D& v2);

friend bool operator!=(const Vector3D& v1, const Vector3D& v2);

friend bool operator<(const Vector3D& v1, const Vector3D& v2);

// Дружественные функции ввода/вывода

friend std::ostream& operator<<(std::ostream& os, const Vector3D& v);

friend std::istream& operator>>(std::istream& is, Vector3D& v);

// Метод преобразования в строку

std::string toString() const;

};

#endif // VECTOR3D\_H

**Код файла реализации Vector3D.cpp:**

#include "Vector3D.h"

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

// Конструктор по умолчанию

Vector3D::Vector3D() : x(0.0), y(0.0), z(0.0) {}

// Конструктор с параметрами

Vector3D::Vector3D(double x, double y, double z) : x(x), y(y), z(z) {}

// Методы-операции с присваиванием

Vector3D& Vector3D::operator+=(const Vector3D& other) {

x += other.x;

y += other.y;

z += other.z;

return \*this;

}

Vector3D& Vector3D::operator-=(const Vector3D& other) {

x -= other.x;

y -= other.y;

z -= other.z;

return \*this;

}

Vector3D& Vector3D::operator\*=(double scalar) {

x \*= scalar;

y \*= scalar;

z \*= scalar;

return \*this;

}

// Методы класса

double Vector3D::length() const noexcept {

return sqrt(x \* x + y \* y + z \* z);

}

double Vector3D::dotProduct(const Vector3D& other) const noexcept {

return x \* other.x + y \* other.y + z \* other.z;

}

Vector3D Vector3D::crossProduct(const Vector3D& other) const noexcept {

return Vector3D(y \* other.z - z \* other.y, z \* other.x - x \* other.z, x \* other.y - y \* other.x);

}

double Vector3D::angle(const Vector3D& other) const noexcept {

double dot = dotProduct(other);

double lenProduct = length() \* other.length();

return acos(dot / lenProduct);

}

// Дружественные функции операции без присваивания

Vector3D operator+(const Vector3D& v1, const Vector3D& v2) {

Vector3D result = v1;

result += v2;

return result;

}

Vector3D operator-(const Vector3D& v1, const Vector3D& v2) {

Vector3D result = v1;

result -= v2;

return result;

}

Vector3D operator\*(const Vector3D& v, double scalar) {

Vector3D result = v;

result \*= scalar;

return result;

}

Vector3D operator\*(double scalar, const Vector3D& v) {

return v \* scalar;

}

bool operator==(const Vector3D& v1, const Vector3D& v2) {

return (v1.x == v2.x) && (v1.y == v2.y) && (v1.z == v2.z);

}

bool operator!=(const Vector3D& v1, const Vector3D& v2) {

return !(v1 == v2);

}

bool operator<(const Vector3D& v1, const Vector3D& v2) {

return v1.length() < v2.length();

}

// Дружественные функции ввода/вывода

ostream& operator<<(ostream& os, const Vector3D& v) {

os << "(" << v.x << ", " << v.y << ", " << v.z << ")";

return os;

}

istream& operator>>(istream& is, Vector3D& v) {

is >> v.x >> v.y >> v.z;

return is;

}

// Метод преобразования в строку

string Vector3D::toString() const {

return "(" + to\_string(x) + ", " + to\_string(y) + ", " + to\_string(z) + ")";

}

**Код главной функции:**

#include "Vector3D.h"

#include <iostream>

#include <cassert>

using namespace std;

int main() {

setlocale(LC\_ALL,"ru");

try {

// Создание объектов Vector3D

Vector3D v1(1.0, 2.0, 3.0);

Vector3D v2(4.0, 5.0, 6.0);

Vector3D v3(3.0, 4.0, 5.0);

// Операции с векторами

Vector3D sum = v1 + v2;

Vector3D diff = v3 - v2;

double scalar = 2.0;

Vector3D scaled = v1 \* scalar;

// Вывод результатов

cout << "v1: " << v1 << endl;

cout << "v2: " << v2 << endl;

cout << "v3: " << v3 << endl;

cout << "Сумма: " << sum << endl;

cout << "Разность: " << diff << endl;

cout << "Умножение на скаляр: " << scaled << endl;

// Другие операции и вывод результатов

cout << "Скалярное произведение: " << v1.dotProduct(v2) << endl;

cout << "Векторное произведение: " << v1.crossProduct(v2) << endl;

cout << "Угол между векторами: " << v1.angle(v2) << " радиан" << endl;

cout << "Длина v1: " << v1.length() << endl;

cout << "Длина v2: " << v2.length() << endl;

if (v1 == v2) {

cout << "Векторы равны" << endl;

}

else {

cout << "Векторы не равны" << endl;

}

assert(sum == Vector3D(5.0, 7.0, 9.0));

assert(diff == Vector3D(-1.0, -1.0, -1.0));

assert(scaled == Vector3D(2.0, 4.0, 6.0));

// Проверка метода dotProduct

assert(v1.dotProduct(v2) == 32.0);

// Проверка метода crossProduct

assert(v1.crossProduct(v2) == (-3.0, 6.0, -3.0));

// Проверка метода angle

assert(v1.angle(v2) == 0.225726);

// Проверка метода length

assert(v1.length() == 3.74166);

// Проверка метода length

assert(v2.length() == 8.77496);

//Векторы не равны

// Демонстрация операций сравнения

cout << "v1 == v2: " << (v1 == v2) << endl;

cout << "v1 != v2: " << (v1 != v2) << endl;

cout << "v1 < v2: " << (v1 < v2) << endl;

}

catch (const exception& e) {

cerr << "Исключение: " << e.what() << endl;

}

return 0;

}