

Лабораторная работа №1

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: Капичников Ярослав Андреевич, группа М80-207Б-20

Преподаватель: Дорохов Евгений Павлович, каф. 806

Задание:(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.

Описание программы:

Исходный код разделён на несколько файлов:

- `vector.h` – описание класса вектора.
- `vector.cpp` – реализация функций класса вектор.

Дневник отладки

№	Дата	Событие	Действие по исправлению
1			

Вывод:

Проделав данную работу, я изучил основы принципов ооп, создал класс векторов, для которого реализовал множество арифметических операций. Создание класса в C++ очень схоже с созданием структур в си, с чем мы уже сталкивались в предыдущем семестре.

Исходный код:

Vector.h

```
#pragma
once

#include <iostream>
class Vector {
public:
    Vector();
    Vector(std::istream& is);
    Vector(double x, double y, double z);
```

```

        double dist(Vector& other);
        double getX();
        double getY();
        double getZ();
        void setX(double a);
        void setY(double a);
        void setZ(double a);
        friend std::istream& operator>>(std::istream& is, Vector& p);
        friend std::ostream& operator<<(std::ostream& os, Vector& p);
        friend bool operator== (Vector& p1, Vector& p2);
        friend bool operator!= (Vector& p1, Vector& p2);
        friend Vector operator+ (Vector& v1, Vector& v2);
        friend Vector operator- (Vector& v1, Vector& v2);
        friend Vector operator* (Vector& v1, Vector& v2);
        friend Vector operator* (Vector& v1, double a);

        double Len();
        double Angle( Vector& v2);
        double SkalarUm(Vector& v2);
private:
        double x_;
        double y_;
        double z_;
};

```

Vector.cpp

```

#include
"vector.h"

#include <cmath>
Vector::Vector() : x_(0.0), y_(0.0), z_(0.0) {}
Vector::Vector(double x, double y, double z) : x_(x), y_(y), z_(z) {}
Vector::Vector(std::istream& is) {
    is >> x_ >> y_ >> z_;
}
double Vector::dist(Vector& other) {
    double dx = (other.x_ - x_);
    double dy = (other.y_ - y_);
    return std::sqrt(dx * dx + dy * dy);
}
double Vector::getX()
{
    return x_;
}
double Vector::getY()
{
    return y_;
}
double Vector::getZ()
{
    return z_;
}
void Vector::setX(double a)
{
    x_ = a;
}
void Vector::setY(double a)
{
    y_ = a;
}

```

```

}
void Vector::setZ(double a)
{
    z_ = a;
}
double Vector::Len()
{
    double l = sqrt(x_ * x_ + y_ * y_ + z_ * z_);
    return l;
}
double Vector:: Angle( Vector& v2)
{
    double cos = (x_ * v2.x_ + y_ * v2.y_ + z_ * v2.z_) / (Len() * v2.Len());
    return acos(cos) * 180/3.1415;
}
std::istream& operator>>(std::istream& is, Vector& p) {
    is >> p.x_ >> p.y_ >> p.z_;
    return is;
}
std::ostream& operator<<(std::ostream& os, Vector& p) {
    os << "(" << p.x_ << ", " << p.y_ << ", " << p.z_ << ")";
    return os;
}
bool operator== (Vector& p1, Vector& p2)
{
    return (p1.getX() == p2.getX() &&
            p1.getY() == p2.getY() &&
            p1.getZ() == p2.getZ());
}
bool operator!= (Vector& p1, Vector& p2)
{
    return !(p1 == p2);
}
Vector operator+ (Vector& v1, Vector& v2)
{
    Vector v3;
    v3.x_ = v1.x_ + v2.x_;
    v3.y_ = v1.y_ + v2.y_;
    v3.z_ = v1.z_ + v2.z_;
    return v3;
}
Vector operator- (Vector& v1, Vector& v2)
{
    Vector v3;
    v3.x_ = v1.x_ - v2.x_;
    v3.y_ = v1.y_ - v2.y_;
    v3.z_ = v1.z_ - v2.z_;
    return v3;
}
Vector operator* (Vector& v1, Vector& v2)
{
    Vector v3;
    v3.x_ = v1.y_ * v2.z_ - v1.z_ * v2.y_;
    v3.y_ = v1.z_ * v2.x_ - v1.x_ * v2.z_;
    v3.z_ = v1.x_ * v2.y_ - v1.y_ * v2.x_;
    return v3;
}
Vector operator* (Vector& v1, double a)
{
    Vector v3;

```

```
    v3.x_ = v1.x_ * a;  
    v3.y_ = v1.y_ * a;  
    v3.z_ = v1.z_ * a;  
    return v3;  
}  
double Vector::SkalarUm(Vector& v2)  
{  
    double s = x_ * v2.x_ + y_ * v2.y_ + z_ * v2.z_;  
    return s;  
}
```