Московский авиационный институт

(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 2

Тема: Операторы, литералы

Студент: Кудинов Сергей

Преподаватель: Журавлев А.А.

Дата:

Оценка:

Москва, 2019

1. **Постановка задачи**

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.

Операции сложения, вычитания, сравнения (на равенство, больше и меньше) должны быть выполнены в виде перегрузки операторов.

Необходимо реализовать пользовательский литерал для работы с константами типа vector3D.

2. **Репозиторий github**

https://github.com/StormStudioAndroid2/oop_exercise_02

3. **Описание программы**

Реализован класс Vector3D, в котором хранятся три переменные, отображающие координаты. Написаны Get функции для их получения. Также перезагружены операторы, указанные в задании, для получения суммы и разности(+, -), и для сравнения различных объектов класса(==, <,>). В начале выводится сложение первого и второго вектора, затем их вычитание, затем – величина угла между ними, затем длины первого и второго вектора, затем – равны они или нет. Последним выводится значение длины вектора, полученного в результате векторного произведения. Каждая величина выводится с новой строки.

4. **Набор testcases**

Тестовые файлы: **test_01.test**, **test_02.test, test_03.test**

test_01.test:

1 1 1

-1 -1 -1

Проверка правильности для коллинеарных векторов с противоположным направлением

Результат работы программы

0 0 0

2 2 2

180

1.73205

1.73205

Not equal

d(v1)=d(v2)

0

test_02.test :

1 2 3

4 5 6

Проверка корректности работы для случайных векторов

Результат работы программы

5 7 9

-3 -3 -3

12.9332

3.74166

8.77496

Not equal

d(v1)<d(v2)

7.34847

test_03.test:

1 1 1

1 1 1

Проверка корректности работы операций сложения и вычитания для равных векторов

Результат работы программы

2 2 2

0 0 0

0

1.73205

1.73205

Equal

d(v1)=d(v2)

0

5. Результаты выполнения тестов

Все тесты успешно пройдены, программа выдаёт верные результаты, корректно обрабатывает время.

6. Листинг программы

main.cpp

```
#include<iostream>
#include "Vector3D.h"
#include <iomanip>

int  main() {
  int x1,x2,y1,y2,z1,z2;
  //literal using
  Vector3D literal = 3.0_vect3D;

  std::cout.precision(6);
  Vector3D vector1;
  Vector3D vector2;
  std::cin >> vector1;
  std::cin >> vector2;
  Vector3D result = vector1+vector2;
  std::cout << result << std::endl;
  result = vector1-vector2;
  std::cout << result <<  std::endl;
```

```
  std::cout << vector1.getAngle(vector2) <<  std::endl;
  std::cout << vector1.getLength() <<  std::endl;
  std::cout << vector2.getLength() <<  std::endl;
    if (vector1 == vector2) {
    std::cout << "Equal" <<  std::endl;
    } else {
      std::cout << "Not equal" <<  std::endl;
    }
    if (!(vector1>vector2) && !(vector1<vector2)) {
      std::cout << "d(v1)=d(v2)" <<  std::endl;
    } else {
      if (vector1>vector2) {
        std::cout << "d(v1)>d(v2)" <<  std::endl;
      } else {
        if (vector1<vector2) {
          std::cout << "d(v1)<d(v2)" <<  std::endl;
        }
      }
    }
     std::cout << vector1.crossProduct(vector2).getLength() <<  std::endl;

    return 0;
}
```

Vector3D.h

```
#include <iostream>
#include <cmath>


class  Vector3D
{
private :
  double x;
  double y;
  double z;

 public:
  Vector3D(double x,double y,double z);
  Vector3D();
  friend Vector3D operator+(const Vector3D& left, const Vector3D& right);
  friend Vector3D operator*(const Vector3D& left, const double right);
  friend std::ostream& operator<< (std::ostream &out, const Vector3D &vector);
  friend Vector3D operator-(const Vector3D& left, const Vector3D& right);
  friend bool operator>(const Vector3D& left, const Vector3D& right);
  friend std::istream& operator>> (std::istream &in, Vector3D &vector);
  friend bool operator<(const Vector3D& left, const Vector3D& right);
  friend bool operator==(const Vector3D& left, const Vector3D& right);
  Vector3D crossProduct(const Vector3D& vector);
  void lambdaProduct(double lambda);
  double scalarProduct(const Vector3D& vector);
  const double getLength() const ;
  double getAngle( Vector3D& vector);
  double getX();
  double getY();
  double getZ();

} ;
  Vector3D operator""_vect3D(long double n);\
```

Vector3D.cpp

```
#include "Vector3D.h"


#include <iostream>
#include <cmath>
```

```
Vector3D ::Vector3D(double x,double y,double z)
:  x(x), y(y),z(z) {}
   Vector3D::Vector3D()
:  x(0), y(0),z(0) {}
  Vector3D operator+(const Vector3D& left, const Vector3D& right) {
    Vector3D result;
    result.x = left.x+right.x;
    result.y = left.y+right.y;
    result.z = left.z+right.z;

    return result;
  }
  Vector3D operator-(const Vector3D& left, const Vector3D& right) {
    Vector3D result;
    result.x = left.x-right.x;
    result.y = left.y-right.y;
    result.z = left.z-right.z;

    return result;
  }
  bool operator>(const Vector3D& left, const Vector3D& right) {
    return left.getLength()>right.getLength();
  }
  bool operator<(const Vector3D& left, const Vector3D& right) {
    return left.getLength()<right.getLength();
  }
  bool operator==(const Vector3D& left, const Vector3D& right) {
    return (left.x==right.x && left.y==right.y && left.z==right.z);
  }
   Vector3D Vector3D::crossProduct(const Vector3D& vector) {
    Vector3D result;
    result.x = this->y*vector.z-this->z*vector.y;
    result.y = this->z*vector.x-this->x*vector.z;
    result.z = this->x*vector.y-this->y*vector.x;

    return result;
  }
 Vector3D operator*(const Vector3D& left, const double right) {
    Vector3D result;
     result.x = left.x*right;
    result.y = left.y*right;
    result.z = left.z*right;

    return result;
  }
std ::ostream& operator<< (std::ostream &out, const Vector3D &vector)
{
```

```
  // Поскольку operator<< является другом класса Point, то мы имеем прямой доступ к членам Point
  out << vector.x << " " << vector.y << " " << vector.z << "";

  return out;
}
  void Vector3D::lambdaProduct(double lambda) {
    this->x*=lambda;
    this->y*=lambda;
    this->z*=lambda;

  }
std ::istream& operator>> (std::istream &in, Vector3D &vector)
{
  // Поскольку operator>> является другом класса Point, то мы имеем прямой доступ к членам Point
  // Обратите внимание, параметр point (объект класса Point) должен быть не константным, чтобы мы имели возможность изменить члены класса
  in >> vector.x;
  in >> vector.y;
  in >> vector.z;

  return in;
}
  double Vector3D::scalarProduct(const Vector3D& vector) {
    return this->x*vector.x+this->y*vector.y+this->z*vector.z;
  }
  const double Vector3D::getLength() const {
    return sqrt(this->x*this->x+this->y*this->y+this->z*this->z);
  }

  double Vector3D::getAngle( Vector3D& vector)  {
    if ((vector.getLength()==0) || (this->getLength())==0) {
      return 0;
    }
      const double halfC = 180/M_PI;
     double cos1 = (this->scalarProduct(vector)/(this->getLength()*vector.getLength())) ;
     if (cos1 < -1) {
       return 180;
     }
     if (cos1>1) {
       return 0;
     }
     return halfC*acos(cos1);

  }

  double  Vector3D::getX() {
```

```
    return x;
}
double  Vector3D::getY() {
    return y;
}
double  Vector3D::getZ() {
    return z;
}
Vector3D operator""_vect3D(long double n) {
    return  Vector3D(n,n,n);
}
```

7. Вывод

Реализована программа, включающая в себя простой класс с перезагруженными операторами и реализованным литералом.

Список литературы

1. Шилдт, Герберт. C++: базовый курс, 3-е изд. : Пер. с англ. - М. : ООО “И.Д. Вильямс”, 2018. - 624 с. : ил. - Парал. тит. англ.