МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

#### Факультет экономики, менеджмента и информационных технологий

(факультет)

### Кафедра Систем управления и информационных технологий в строительств

# ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

по дисциплине Объектно-ориентированное программирование

Тема «Наследование»

Выполнил студент группы ИСТ-214 Д. В. Тюленев Подпись, дата Инициалы, фамилия

## Принял Е. Н. Королев Подпись, дата Инициалы, фамилия

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

2022

Лабораторная работа №6

«Наследование»

Вариант №4

1. Цель работы: получить навыки в создании иерархии классов при помощи наследования.
2. Задания на лабораторную работу:

Создать иерархию классов, используя наследование. В каждой программе необходимо соблюсти принцип разделения интерфейса и реализации класса. В каждом варианте необходимо написать программу, иллюстрирующую применение всех методов ваших классов.

Прежде чем приступить к написанию программ, продумать, какие необходимы функции в каждом из классов: как в базовом, так и в классах-наследниках. Также продумать, что следует поместить в закрытые переменные. Предусмотреть возможность переопределения методов базового класса в производном. Возможно, в некоторых вариантах лучше воспользоваться композицией, чем наследованием.

1. Ход выполнения:

Первым делом определим структуру для хранения величины направления, имеет 3 поля для фиксирования значения каждой координаты (x, y, z) и конструктор для инициализации этих полей

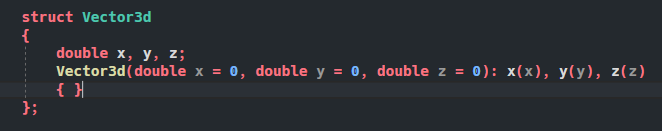
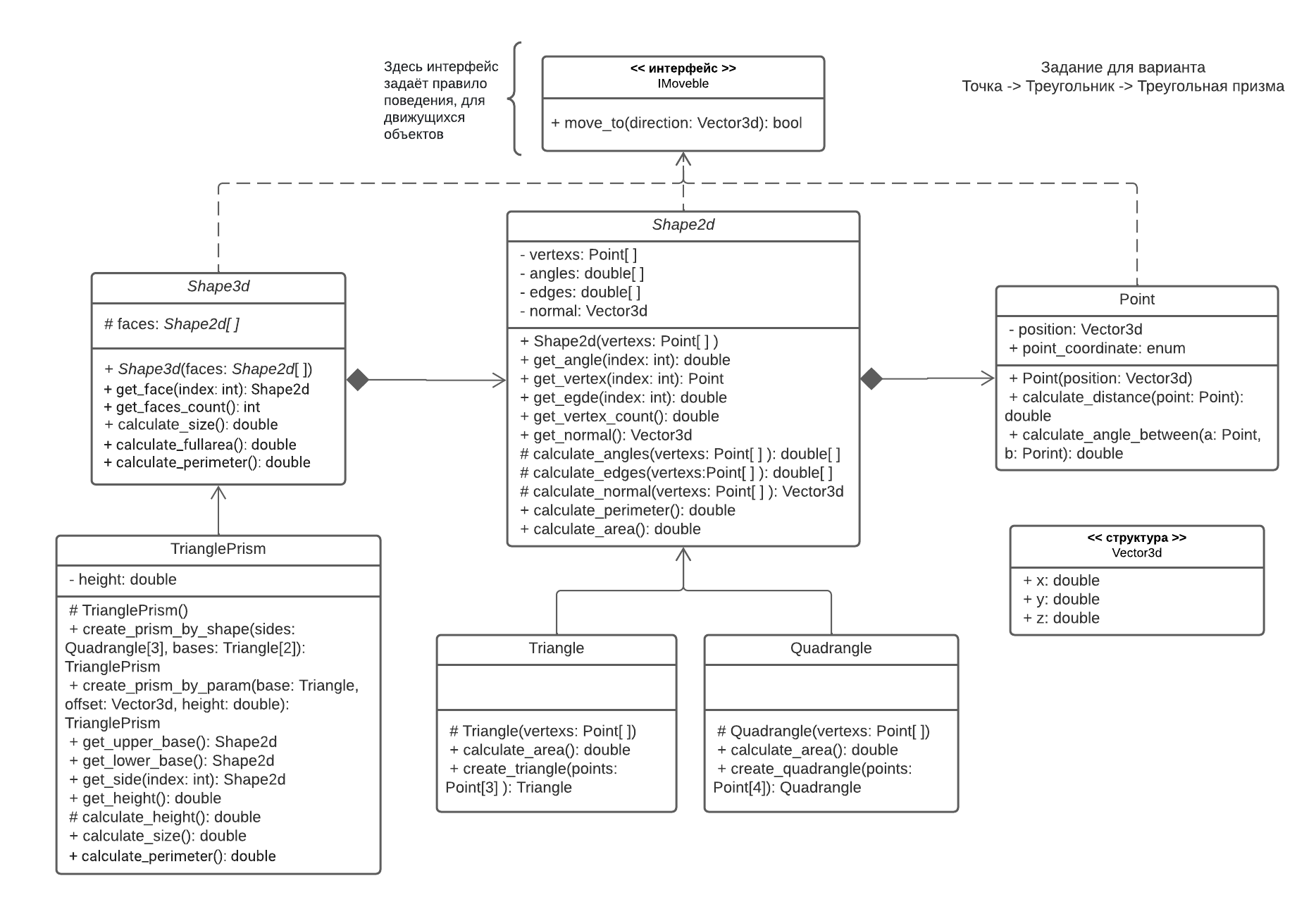


Рисунок 1 － Структура, характеризующая направленный отрезок (вектор).

Ключевой задачей стоит проектировка модели, которая позволит обозначит отношения между классами и их экземплярами. Для удобства разработки системы, была спроектирована UML диаграмма.

 Рисунок 2 － UML диаграмма работы

Для начала реализуем класс «Point», который имеет: поле «position» типа «Vector3d» для хранения текущей позиции, конструктор для инициализации позиции, метод, позволяющий рассчитать дистанцию между двумя точками, и метод, позволяющий рассчитать угол между точками (в радианах). У данного класса не будет дочерних, но он реализует интерфейс «IMoveble». Имеет перегруженный оператор индексации для доступа к сокрытым координатам.

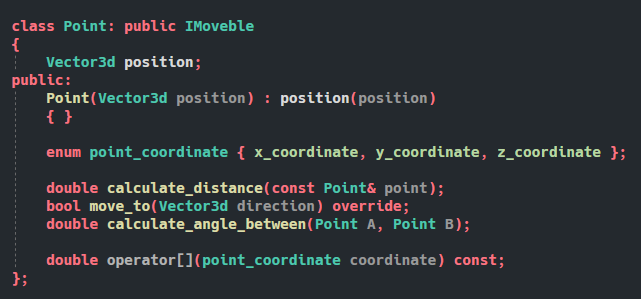


Рисунок 3 － Класс «Точка»

Метод «Point::calculate\_distance» рассчитывает дистанцию между точками, в качестве параметра принимается неизменяемая копия объекта класса «Point» и возвращает рассчитанную длину. Используется формула:

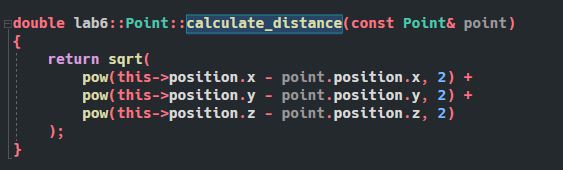


Рисунок 4 － Метод «Point::calculate\_distance»

Метод «Point::calculate\_angle\_between» рассчитывает угол (в радианах) между прямыми проведёнными через точки, в качестве параметров принимает объекты класса «Point» и возвращает рассчитанный угол.

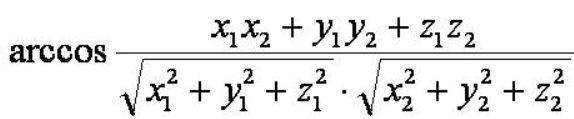


Рисунок 5 － Формула для вычисления угла

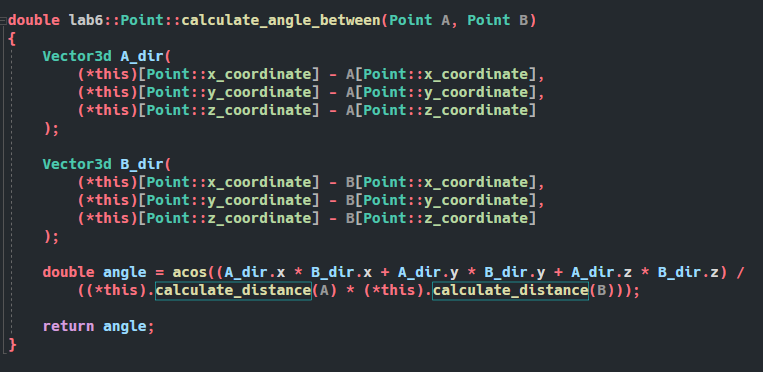


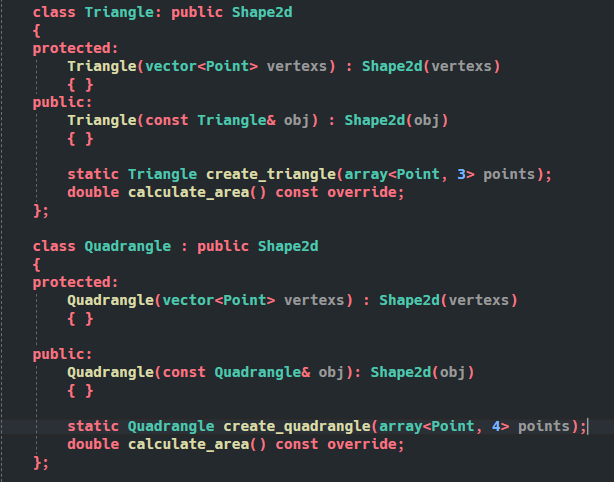
Рисунок 6 － Метод «Point::calculate\_angle\_between»

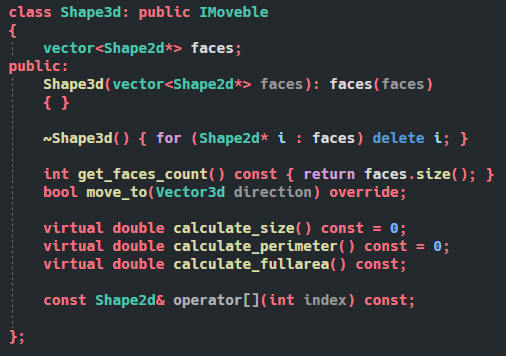
Далее реализовывается абстрактный класс «Shape2d», который описывает двумерные фигуры, в качестве полей: массивы для хранения значений углов фигуры и сторон фигуры, контейнер для хранения вершин (экземпляры класса «Point») и переменная для хранения векторного значения нормали данной плоскости.

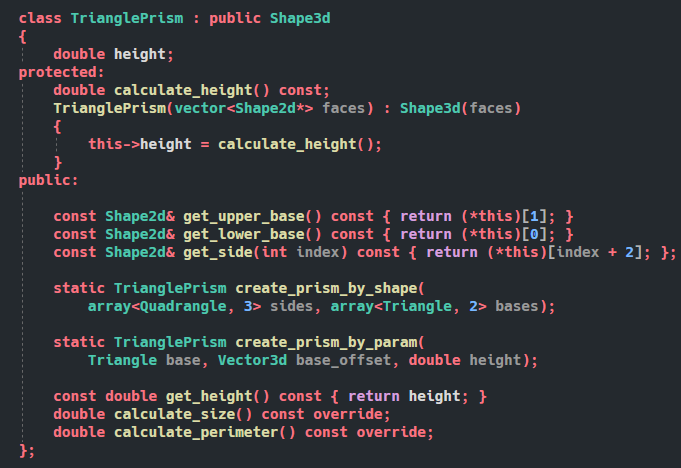


Рисунок 7 － Абстрактный класс «Shape2d»

Имеет конструктор для инициализации всех сокрытых полей, реализует интерфейс «IMoveble».







Вывод: получили навыки в создании иерархии классов при помощи наследования.

Приложение А

Листинг программного кода языка C++

Файл «lab6.h»

#pragma once

#include <iostream>

#include <vector>

#include <string>

#include <array>

#include <sstream>

# define M\_PI 3.14159265358979323846

#include <cmath>

// Точка -> Треугольник -> Треугольная призма

// (отношения будут реализованы через композицию)

namespace lab6

{

using namespace std;

struct Vector3d

{

double x, y, z;

Vector3d(double x = 0, double y = 0, double z = 0): x(x), y(y), z(z)

{ }

};

class IMoveble

{

public:

virtual bool move\_to(Vector3d direction) = 0;

};

class Point: public IMoveble

{

Vector3d position;

public:

Point(Vector3d position) : position(position)

{ }

enum point\_coordinate { x\_coordinate, y\_coordinate, z\_coordinate };

double calculate\_distance(const Point& point);

bool move\_to(Vector3d direction) override;

double calculate\_angle\_between(Point A, Point B);

double operator[](point\_coordinate coordinate) const;

};

class Shape2d: public IMoveble

{

double \*angles, \*edges;

Vector3d normal;

vector<Point> vertexs;

protected:

double\* calculate\_angles(vector<Point> vertexs);

double\* calculate\_edges(vector<Point> vertexs);

Vector3d calculate\_normal(vector<Point> vertexs);

public:

Shape2d(vector<Point> vertexs) : vertexs(vertexs)

{

this->angles = calculate\_angles(vertexs);

this->edges = calculate\_edges(vertexs);

this->normal = calculate\_normal(vertexs);

}

Shape2d(const Shape2d& obj) : Shape2d(obj.vertexs)

{ }

~Shape2d() { delete[] angles, edges; }

int get\_vertex\_count() const { return vertexs.size(); }

Vector3d get\_normal() const { return normal; }

double get\_egde(int index) const;

double get\_angle(int index) const;

Point operator[](int index) const;

virtual double calculate\_perimeter() const;

virtual double calculate\_area() const = 0;

bool move\_to(Vector3d direction) override;

};

class Triangle: public Shape2d

{

protected:

Triangle(vector<Point> vertexs) : Shape2d(vertexs)

{ }

public:

Triangle(const Triangle& obj) : Shape2d(obj)

{ }

static Triangle create\_triangle(array<Point, 3> points);

double calculate\_area() const override;

};

class Quadrangle : public Shape2d

{

protected:

Quadrangle(vector<Point> vertexs) : Shape2d(vertexs)

{ }

public:

Quadrangle(const Quadrangle& obj): Shape2d(obj)

{ }

static Quadrangle create\_quadrangle(array<Point, 4> points);

double calculate\_area() const override;

};

class Shape3d: public IMoveble

{

vector<Shape2d\*> faces;

public:

Shape3d(vector<Shape2d\*> faces): faces(faces)

{ }

~Shape3d() { for (Shape2d\* i : faces) delete i; }

int get\_faces\_count() const { return faces.size(); }

bool move\_to(Vector3d direction) override;

virtual double calculate\_size() const = 0;

virtual double calculate\_perimeter() const = 0;

virtual double calculate\_fullarea() const;

const Shape2d& operator[](int index) const;

};

class TrianglePrism : public Shape3d

{

double height;

protected:

double calculate\_height() const;

TrianglePrism(vector<Shape2d\*> faces) : Shape3d(faces)

{

this->height = calculate\_height();

}

public:

const Shape2d& get\_upper\_base() const { return (\*this)[1]; }

const Shape2d& get\_lower\_base() const { return (\*this)[0]; }

const Shape2d& get\_side(int index) const { return (\*this)[index + 2]; };

static TrianglePrism create\_prism\_by\_shape(

array<Quadrangle, 3> sides, array<Triangle, 2> bases);

static TrianglePrism create\_prism\_by\_param(

Triangle base, Vector3d base\_offset, double height);

const double get\_height() const { return height; }

double calculate\_size() const override;

double calculate\_perimeter() const override;

};

void lab(void);

}

Файл «lab6.cpp»

#include "lab6.h"

///////////////////////////////////--Point--///////////////////////////////////

double lab6::Point::operator[](point\_coordinate coordinate) const

{

switch (coordinate)

{

case Point::x\_coordinate: return this->position.x;

case Point::y\_coordinate: return this->position.y;

case Point::z\_coordinate: return this->position.z;

default:

throw exception("Невозможно выбрать координату вершины");

}

}

double lab6::Point::calculate\_distance(const Point& point)

{

return sqrt(

pow(this->position.x - point.position.x, 2) +

pow(this->position.y - point.position.y, 2) +

pow(this->position.z - point.position.z, 2)

);

}

bool lab6::Point::move\_to(Vector3d direction)

{

this->position.x += direction.x;

this->position.y += direction.y;

this->position.z += direction.z;

return true;

}

double lab6::Point::calculate\_angle\_between(Point A, Point B)

{

Vector3d A\_dir(

(\*this)[Point::x\_coordinate] - A[Point::x\_coordinate],

(\*this)[Point::y\_coordinate] - A[Point::y\_coordinate],

(\*this)[Point::z\_coordinate] - A[Point::z\_coordinate]

);

Vector3d B\_dir(

(\*this)[Point::x\_coordinate] - B[Point::x\_coordinate],

(\*this)[Point::y\_coordinate] - B[Point::y\_coordinate],

(\*this)[Point::z\_coordinate] - B[Point::z\_coordinate]

);

double angle = acos((A\_dir.x \* B\_dir.x + A\_dir.y \* B\_dir.y + A\_dir.z \* B\_dir.z) /

((\*this).calculate\_distance(A) \* (\*this).calculate\_distance(B)));

return angle;

}

///////////////////////////////////--Shape2d--/////////////////////////////////

double\* lab6::Shape2d::calculate\_angles(vector<Point> vertexs)

{

double\* angles = new double[vertexs.size()];

angles[0] = vertexs[0].calculate\_angle\_between(vertexs[vertexs.size() - 1], vertexs[1]);

for (int i = 1; i < vertexs.size() - 1; i++)

angles[i] = vertexs[i].calculate\_angle\_between(vertexs[i - 1], vertexs[i + 1]);

angles[vertexs.size() - 1] = vertexs[vertexs.size() - 1]

.calculate\_angle\_between(vertexs[vertexs.size() - 2], vertexs[0]);

return angles;

}

double\* lab6::Shape2d::calculate\_edges(vector<Point> vertexs)

{

double\* edges = new double[vertexs.size()];

edges[0] = this->vertexs[0].calculate\_distance(vertexs[vertexs.size() - 1]);

for (int i = 1; i < vertexs.size(); i++)

edges[i] = this->vertexs[i - 1].calculate\_distance(vertexs[i]);

return edges;

}

lab6::Vector3d lab6::Shape2d::calculate\_normal(vector<Point> vertexs)

{

Vector3d dirA(

vertexs[vertexs.size() - 1][Point::x\_coordinate] - vertexs[0][Point::x\_coordinate],

vertexs[vertexs.size() - 1][Point::y\_coordinate] - vertexs[0][Point::y\_coordinate],

vertexs[vertexs.size() - 1][Point::z\_coordinate] - vertexs[0][Point::z\_coordinate]

), dirB(

vertexs[1][Point::x\_coordinate] - vertexs[0][Point::x\_coordinate],

vertexs[1][Point::y\_coordinate] - vertexs[0][Point::y\_coordinate],

vertexs[1][Point::z\_coordinate] - vertexs[0][Point::z\_coordinate]

);

Vector3d normal = Vector3d(

dirB.y \* dirA.z - (dirB.z \* dirA.y),

dirB.x \* dirA.z - (dirB.z \* dirA.x),

dirB.x \* dirA.y - (dirB.y \* dirA.x)

);

const double length = Point(Vector3d()).calculate\_distance(normal);

return Vector3d(normal.x / length, normal.y / length, normal.z / length);

}

double lab6::Shape2d::calculate\_perimeter() const

{

double result(0);

for (int i = 0; i < this->get\_vertex\_count(); i++) result += this->edges[i];

return result;

}

double lab6::Shape2d::get\_angle(int index) const

{

if (index < 0 || index >= this->vertexs.size())

throw std::exception("Индекс вне доступного диапазона");

return this->angles[index];

}

double lab6::Shape2d::get\_egde(int index) const

{

if (index < 0 || index >= this->vertexs.size())

throw std::exception("Индекс вне доступного диапазона");

return this->edges[index];

}

lab6::Point lab6::Shape2d::operator[](int index) const

{

if (index < 0 || index >= this->vertexs.size())

throw std::exception("Индекс вне доступного диапазона");

return vertexs[index];

}

bool lab6::Shape2d::move\_to(Vector3d direction)

{

for (auto& i : this->vertexs) i.move\_to(direction);

return true;

}

////////////////////////////////////--Triangle--///////////////////////////////

double lab6::Triangle::calculate\_area() const

{

const double p = this->calculate\_perimeter() / 2.;

double result(p);

for (int i = 0; i < this->get\_vertex\_count(); i++) result \*= (p - this->get\_egde(i));

return sqrt(result);

}

lab6::Triangle lab6::Triangle::create\_triangle(array<Point, 3> points)

{

vector<Point> vertexs;

for (const auto& i : points) vertexs.push\_back(i);

return Triangle(vertexs);

}

/////////////////////////////////--Quadrangle--///////////////////////////////

double lab6::Quadrangle::calculate\_area() const

{

Vector3d A(

(\*this)[0][Point::x\_coordinate] - (\*this)[2][Point::x\_coordinate],

(\*this)[0][Point::y\_coordinate] - (\*this)[2][Point::y\_coordinate],

(\*this)[0][Point::z\_coordinate] - (\*this)[2][Point::z\_coordinate]

), B(

(\*this)[1][Point::x\_coordinate] - (\*this)[3][Point::x\_coordinate],

(\*this)[1][Point::y\_coordinate] - (\*this)[3][Point::y\_coordinate],

(\*this)[1][Point::z\_coordinate] - (\*this)[3][Point::z\_coordinate]

);

const double angle = Point(Vector3d(0, 0, 0)).calculate\_angle\_between(Point(A), Point(B));

const double S = (\*this)[0].calculate\_distance((\*this)[2]) \*

(\*this)[1].calculate\_distance((\*this)[3]) / 2. \* sin(angle);

return S;

}

lab6::Quadrangle lab6::Quadrangle::create\_quadrangle(array<Point, 4> points)

{

vector<Point> vertexs;

for (const auto& i : points) vertexs.push\_back(i);

return Quadrangle(vertexs);

}

////////////////////////////////////--Shape3d--///////////////////////////////

const lab6::Shape2d& lab6::Shape3d::operator[](int index) const

{

if (index < 0 || index >= this->faces.size())

throw std::exception("Индекс вне доступного диапазона");

return \*(this->faces[index]);

}

bool lab6::Shape3d::move\_to(Vector3d direction)

{

for (Shape2d\* face : this->faces)

{

for (int i = 0; i < face->get\_vertex\_count(); i++)

face[i].move\_to(direction);

}

return true;

}

double lab6::Shape3d::calculate\_fullarea() const

{

double result(0);

for (Shape2d\* i : this->faces) result += i->calculate\_area();

return result;

}

////////////////////////////////--TrianglePrism--//////////////////////////////

lab6::TrianglePrism lab6::TrianglePrism::create\_prism\_by\_shape

(array<Quadrangle, 3> sides, array<Triangle, 2> bases)

{

vector<Shape2d\*> faces;

for (auto i : bases) faces.push\_back(new Triangle(i));

for (auto i : sides) faces.push\_back(new Quadrangle(i));

return TrianglePrism(faces);

}

lab6::TrianglePrism lab6::TrianglePrism::create\_prism\_by\_param(

Triangle base, Vector3d base\_offset, double height)

{

vector<Shape2d\*> faces = { new Triangle(base) };

Triangle\* upper\_base = new Triangle(base);

Vector3d shift(

base.get\_normal().x \* height + base\_offset.x,

base.get\_normal().y \* height + base\_offset.y,

base.get\_normal().z \* height + base\_offset.z

);

if (upper\_base->move\_to(shift) != true)

throw std::exception("Невозможно создать объект призмы");

faces.push\_back(upper\_base);

auto connect\_vertexs = [&faces, &base, &upper\_base]

(int p1, int p2, int p3, int p4) -> void

{

Quadrangle side = Quadrangle::create\_quadrangle(array<Point, 4>{

base[p1], base[p2], (\*upper\_base)[p3], (\*upper\_base)[p4]

});

faces.push\_back(new Quadrangle(side));

};

for (int i = 0; i < base.get\_vertex\_count() - 1; i++) connect\_vertexs(i, i + 1, i + 1, i);

connect\_vertexs(base.get\_vertex\_count() - 1, 0, 0, base.get\_vertex\_count() - 1);

return TrianglePrism(faces);

}

double lab6::TrianglePrism::calculate\_size() const

{

return this->height \* this->get\_lower\_base().calculate\_area();

}

double lab6::TrianglePrism::calculate\_height() const

{

Point base\_point = this->get\_upper\_base()[0];

Vector3d normal = this->get\_lower\_base().get\_normal();

double upper = fabs(

normal.x \* base\_point[Point::x\_coordinate] +

normal.y \* base\_point[Point::y\_coordinate] +

normal.z \* base\_point[Point::z\_coordinate]

),lower = sqrt(

pow(normal.x, 2) + pow(normal.y, 2) + pow(normal.z, 2)

);

return upper / lower;

}

double lab6::TrianglePrism::calculate\_perimeter() const

{

double P = this->get\_lower\_base().calculate\_perimeter()

+ this->get\_upper\_base().calculate\_perimeter();

for (int i = 0; i < this->get\_lower\_base().get\_vertex\_count(); i++)

{

P += this->get\_lower\_base()[i].calculate\_distance(this->get\_upper\_base()[i]);

}

return P;

}

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

void lab6::lab(void)

{

int operation = -1;

string line;

while(true)

{

system("cls");

cout << "Выбор объекта для работы: " << endl;

cout << "1 - Объект класса \"Точка\"" << endl

<< "2 - Объект класса \"Треугольник\"" << endl

<< "3 - Объект класса \"Треугольная Призма\"" << endl

<< "4 - Выход из программы" << endl;

try

{

getline(cin, line);

operation = stoi(line);

}

catch (std::exception error)

{

cout << "Неверный ввод номера операции" << endl;

continue;

}

switch (operation)

{

case 1:

{

Vector3d dir;

cout << "Введите значение координаты (x, y, z) -> " << endl;

cin >> dir.x >> dir.y >> dir.z;

Point point(dir);

cout << "Введите номер операции" << endl

<< "1 - Вычислить расстояние до точки" << endl

<< "2 - Переместить точку" << endl

<< "3 - Угол между точками" << endl;

cin >> operation;

switch (operation)

{

case 1:

{

cout << "Координаты точки (x, y, z) -> ";

cin >> dir.x >> dir.y >> dir.z;

cout << "Расстояние: " << point.calculate\_distance(Point(dir)) << endl;

break;

}

case 2:

{

cout << "Предыдущее положение: x = " << point[Point::x\_coordinate]

<< "; y = " << point[Point::y\_coordinate]

<< "; z = " << point[Point::z\_coordinate] << endl;

cout << "Вектор перемещения (x, y, z) -> ";

cin >> dir.x >> dir.y >> dir.z;

point.move\_to(dir);

cout << "Текущее положение: x = " << point[Point::x\_coordinate]

<< "; y = " << point[Point::y\_coordinate]

<< "; z = " << point[Point::z\_coordinate] << endl;

}

break;

case 3:

{

Vector3d dir2;

cout << "Координаты точки 1 (x, y, z) -> ";

cin >> dir.x >> dir.y >> dir.z;

cout << "Координаты точки 2 (x, y, z) -> ";

cin >> dir2.x >> dir2.y >> dir2.z;

cout << " Угол: " << point.calculate\_angle\_between(dir, dir2) / M\_PI \* 180

<< " (deg)" << endl;

}

break;

default:

cout << "Неверный номер операции" << endl;

}

}

break;

case 2:

{

Vector3d p1, p2, p3;

cout << "Введите значение координаты 1 (x, y, z) -> ";

cin >> p1.x >> p1.y >> p1.z;

cout << "Введите значение координаты 2 (x, y, z) -> ";

cin >> p2.x >> p2.y >> p2.z;

cout << "Введите значение координаты 3 (x, y, z) -> ";

cin >> p3.x >> p3.y >> p3.z;

Triangle triangle = Triangle::create\_triangle(array<Point, 3>{

Point(p1), Point(p2), Point(p3)

});

if (triangle.calculate\_area() <= 0 || isnan(triangle.calculate\_area()))

{

cout << "Нельзя создать такой треугольник" << endl;

break;

}

cout << "Введите номер операции" << endl

<< "1 - Получить кол-во вершин" << endl

<< "2 - Получить вектор нормали" << endl

<< "3 - Получить длину стороны" << endl

<< "4 - Получить значение угола" << endl

<< "5 - Рассчитать периметр" << endl

<< "6 - Рассчитать площадь" << endl

<< "7 - Получить вершину" << endl

<< "8 - Переместить плоскость" << endl;

cin >> operation;

switch (operation)

{

case 1:

{

cout << "Количество вершин: " << triangle.get\_vertex\_count() << endl;

}

break;

case 2:

{

cout << "Вектор нормали: { " << triangle.get\_normal().x << "; "

<< triangle.get\_normal().y << "; " << triangle.get\_normal().z << " }" << endl;

}

break;

case 3:

{

int index;

cout << "Введите индекс стороны: ";

cin >> index;

cout << "Длина стороны: " << triangle.get\_egde(index) << endl;

}

break;

case 4:

{

int index;

cout << "Введите индекс угла: ";

cin >> index;

cout << "Значение угла: " << triangle.get\_angle(index) / M\_PI \* 180 << endl;

}

break;

case 5:

{

cout << "Периметр: " << triangle.calculate\_perimeter() << endl;

}

break;

case 6:

{

cout << "Площадь: " << triangle.calculate\_area() << endl;

}

break;

case 7:

{

int index;

cout << "Введите индекс вершины: ";

cin >> index;

cout << "x: " << triangle[index][Point::x\_coordinate]

<< " y: " << triangle[index][Point::y\_coordinate]

<< " z: " << triangle[index][Point::z\_coordinate] << endl;

}

break;

case 8:

{

cout << "Предыдущее положение: " << endl;

for (int i = 0; i < triangle.get\_vertex\_count(); i++)

{

cout << "x = " << triangle[i][Point::x\_coordinate]

<< "; y = " << triangle[i][Point::y\_coordinate]

<< "; z = " << triangle[i][Point::z\_coordinate] << endl;

}

cout << endl;

Vector3d dir;

cout << "Вектор перемещения (x, y, z) -> ";

cin >> dir.x >> dir.y >> dir.z;

triangle.move\_to(dir);

cout << "Текущее положение: " << endl;

for (int i = 0; i < triangle.get\_vertex\_count(); i++)

{

cout << "x = " << triangle[i][Point::x\_coordinate]

<< "; y = " << triangle[i][Point::y\_coordinate]

<< "; z = " << triangle[i][Point::z\_coordinate] << endl;

}

cout << endl;

}

break;

default:

cout << "Неверный номер операции" << endl;

}

}

break;

case 3:

{

Vector3d p1, p2, p3, dir;

double height;

cout << "Введите значение координаты 1 (x, y, z) -> ";

cin >> p1.x >> p1.y >> p1.z;

cout << "Введите значение координаты 2 (x, y, z) -> ";

cin >> p2.x >> p2.y >> p2.z;

cout << "Введите значение координаты 3 (x, y, z) -> ";

cin >> p3.x >> p3.y >> p3.z;

cout << "Введите значение высоты: ";

cin >> height;

cout << "Введите вектор смещение верхнего основания (x, y, z) -> ";

cin >> dir.x >> dir.y >> dir.z;

auto prism = TrianglePrism::create\_prism\_by\_param(

Triangle::create\_triangle(array<Point, 3>{Point(p1), Point(p2), Point(p3)}), dir, height

);

if (prism.calculate\_size() <= 0 || isnan(prism.calculate\_size()))

{

cout << "S(осн) = " << prism.get\_lower\_base().calculate\_area() << endl;

cout << "H = " << prism.get\_height() << endl;

cout << "Нельзя создать такую призму" << endl;

break;

}

cout << "Введите номер операции" << endl

<< "1 - Получить кол-во граней" << endl

<< "2 - Переместить призму" << endl

<< "3 - Вычислить объём" << endl

<< "4 - Вычислить площадь полной поверхности" << endl

<< "5 - Вычислить полный периметр" << endl

<< "6 - Получить высоту призмы" << endl;

cin >> operation;

switch (operation)

{

case 1:

{

cout << "Количество граней: " << prism.get\_faces\_count() << endl;

}

break;

case 2:

{

cout << "Предыдущее положение основания: " << endl;

for (int i = 0; i < prism.get\_lower\_base().get\_vertex\_count(); i++)

{

cout << "x = " << prism.get\_lower\_base()[i][Point::x\_coordinate]

<< "; y = " << prism.get\_lower\_base()[i][Point::y\_coordinate]

<< "; z = " << prism.get\_lower\_base()[i][Point::z\_coordinate] << endl;

}

cout << endl;

Vector3d dir;

cout << "Вектор перемещения (x, y, z) -> ";

cin >> dir.x >> dir.y >> dir.z;

prism.move\_to(dir);

cout << "Текущее положение: " << endl;

for (int i = 0; i < prism.get\_lower\_base().get\_vertex\_count(); i++)

{

cout << "x = " << prism.get\_lower\_base()[i][Point::x\_coordinate]

<< "; y = " << prism.get\_lower\_base()[i][Point::y\_coordinate]

<< "; z = " << prism.get\_lower\_base()[i][Point::z\_coordinate] << endl;

}

cout << endl;

}

break;

case 3:

{

cout << "Объем призмы: " << prism.calculate\_size() << endl;

}

break;

case 4:

{

cout << "Площадь полной поверхности: " << prism.calculate\_fullarea() << endl;

}

break;

case 5:

{

cout << "Полный периметр: " << prism.calculate\_perimeter() << endl;

}

break;

case 6:

{

cout << "Высота: " << prism.get\_height() << endl;

}

break;

default:

cout << "Неверный номер операции" << endl;

}

}

break;

case 4: return;

default:

cout << "Неверный номер" << endl;

}

system("pause");

}

}

Файл «main.cpp»

#include <iostream>

#include <locale>

#include "labs/lab6/lab6.h"

int main(void)

{

setlocale(LC\_ALL, "rus");

Lab6::lab();

return 0;

}