МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра: Математического обеспечения и суперкомпьютерных технологий**

Направление подготовки: Прикладная математика и информатика

Профиль подготовки: Общий профиль

**ОТЧЕТ**

по

лабораторной работе

на тему:

**«Класс треугольники и четырехугольники»**

**Выполнил(а):** студент(ка) группы 3821Б1ПМ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ширшов М.С.

Подпись

**Научный руководитель:**

Преподаватель, к.ф.-м.н.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Баркалов А.В.

Подпись

Нижний Новгород  
2022

**Содержание**

Введение…………………………………………………………………………...3

1. Постановка учебно-практической задачи……………………………………….4
2. Руководство пользователя………………………………………………………..6
3. Руководство программиста……………………………………………………….7

3.1. Интерфейс класса треугольник…………..………………………………….7

3.2. Интерфейс класса четырехугольник ………………………………………..7

3.3. Переменные, объявленные в «теле» программы …………………………...8

3.4. Описание структуры программного комплекса…………...……………......8

Заключение………………………………………………………………………...9

Список литературы………………………………………………………………10

Приложение………………………………………………………………………11

**Введение**

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Объект – это нечто, имеющее четко определенные границы. Однако, этого недостаточно, чтобы отделить один объект от другого или дать оценку качества абстракции. Объект обладает состоянием, поведением и идентичностью; структура и поведение схожих объектов определяет общий для них класс; термины «экземпляр класса» и «объект» взаимозаменяемы.

Класс – это множество объектов, обладающих общей структурой, поведением и семантикой. Отдельный объект – это экземпляр класса. Класс представляет лишь абстракцию существенных свойств объекта.

Индивидуальность объекта – это такое свойство объекта, которое отличает его от всех других объектов. В большинстве языков программирования при создании объект именуется, поэтому многие путают адресуемость и индивидуальность. Невозможность отличить имя объекта от самого объекта является источником множества ошибок в ООП.

Объектно-ориентированное программирование строится на трех основополагающих принципах: инкапсуляция, полиморфизм и наследование.

1. *Инкапсуляция* – это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение; инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации. Пусть члену класса требуется защита от «несанкционированного доступа». Как разумно ограничить множество функций, которым такой член будет доступен? Очевидный ответ для языков, поддерживающих объектно-ориентированное программирование, таков: доступ имеют все операции, которые определены для этого объекта, иными словами, все функции-члены.
2. Наследование представляет собой способность производить новый класс из существующего базового класса. Производный класс – это новый класс, а базовый класс – существующий класс. Когда вы порождаете один класс из другого (базового класса), производный класс наследует элементы базового класса.
3. Полиморфный объект представляет собой такой объект, который может изменять форму во время выполнения программы. В объектно-ориентированных языках класс является абстрактным типом данных. Полиморфизм реализуется с помощью наследования классов и виртуальных функций. Класс-потомок наследует сигнатуры методов класса-родителя, а реализация, в результате переопределения метода, этих методов может быть другой, соответствующей специфике класса-потомка. Другие функции могут работать с объектом класса-родителя, но при этом вместо него во время исполнения будет подставляться один из классов-потомков. Это называется поздним связыванием.
4. **Постановка учебно-практической задачи**

**Цель лабораторной работы**: овладение навыками работы с объектно-ориентированным программированием.

**Задачи работы:**

1. **Базовый класс треугольников**. Требуется разработать класс треугольников, задаваемых координатами вершин на плоскости, определить над объектами-треугольниками функции вычисления площади и периметра, произвести перегрузку функций, рассматривая их как унарные операции.
2. **Производный класс четырехугольников**. Используя разработанный базовый класс для построения класса выпуклых четырехугольников, задаваемых координатами вершин на плоскости, определить над объектами-четырехугольниками функции вычисления площади и периметра и произвести их перегрузку. Необходимо предусмотреть функцию проверки правильности построения четырехугольника.

При работе с системой пользователь должен иметь возможность задания количества элементов в массиве, получаемом при помощи датчика псевдослучайных чисел, и выбора алгоритма сортировки для тестирования.

Исходные данные:

Все исходные данные вводятся пользователем с клавиатуры.

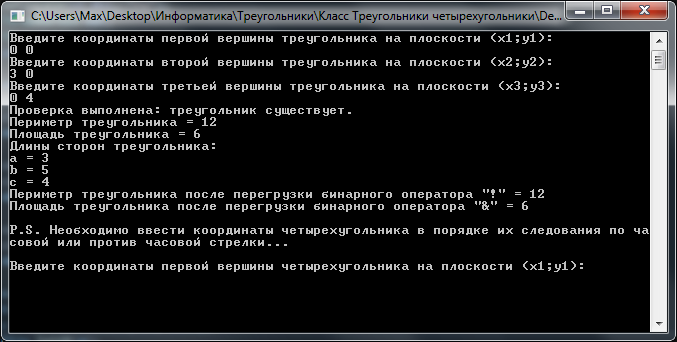
Ими являются координаты вершин треугольника/четырехугольника.

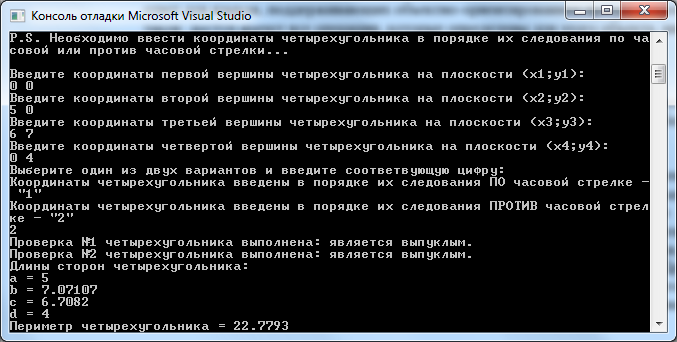
Требуемый результат:

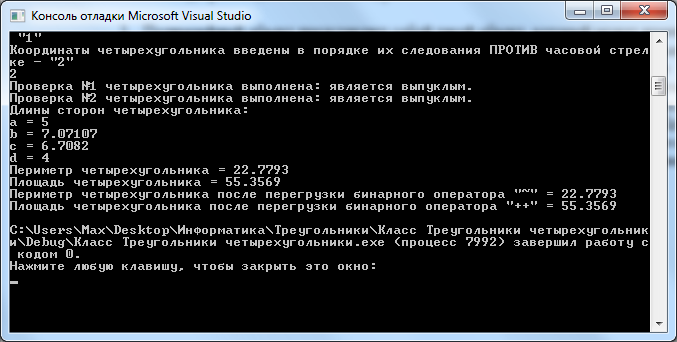
Создание двух классов (треугольники и четырехугольники), определение площади, периметра. Перегрузка бинарных операций.

Контрольный пример:

Пусть координаты треугольника A(0;0), B(3;0), C(0;4), а координаты четырехугольника A(0;0), B(5;0), C(6,7), D(0;4). (рис.0)





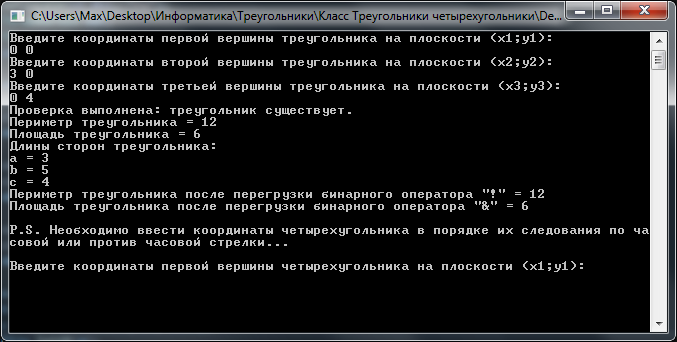


(рис. 0)

1. **Руководство пользователя**

Открываем файл «Класс Треугольники четырехугольники.exe».

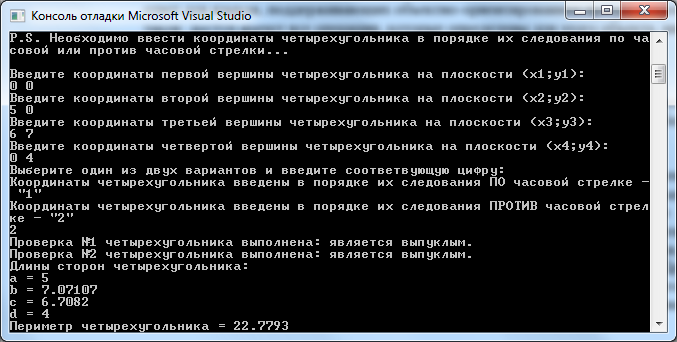
После запуска программы на экране появится надпись с просьбой ввести координаты вершины. (необходимо повторить действие несколько раз, вводя разные координаты точек на плоскости) (рис.1)



(рис.1)

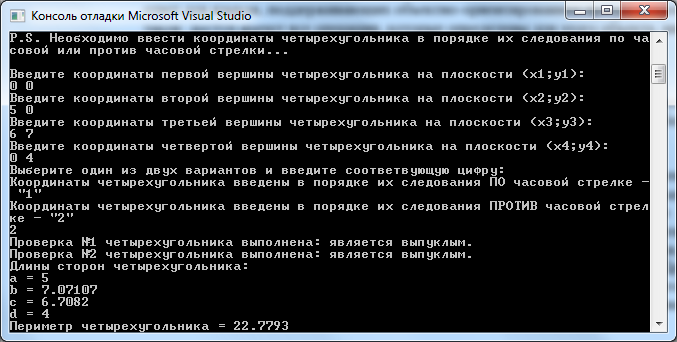
После ввода координат на консоль будет выведена площадь и периметр заданного треугольника. (При условии, что треугольник существует – в противном случае произойдет завершение программы и задачу с четырехугольником будет выполнить невозможно.)

После выполнения задачи с треугольником на экране будут появляться просьбы вводить координаты новых вершин (вершин четырехугольника) (рис.3)



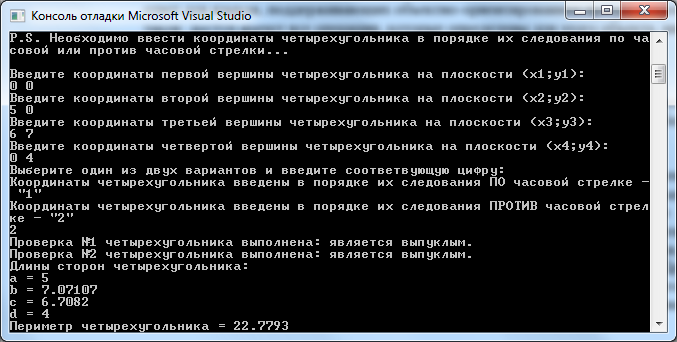
(рис.3)

Программа проверит выпуклость четырехугольника с помощью двух способов, выведет результат. Если четырехугольник выпуклый – будет подсчитана его площадь и периметр, в противном случае произойдет завершение программы.

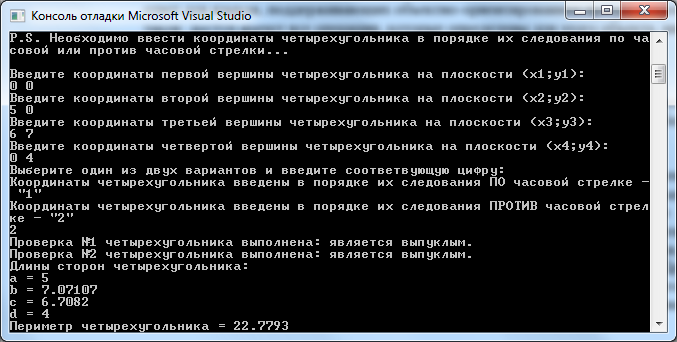


(рис.4)

Замечание! Следуйте подсказкам. (рис.5,6). Если с клавиатуры будут введены некорректные данные, программа даст возможность их перезаписать.



(рис.5)



(рис.6)

1. **Руководство программиста**
   1. **Интерфейс класса треугольник**

class ThreeVertex {

protected:

double x1, y1;

double x2, y2;

double x3, y3; //поля для хранения координат вершин

double AB, BC, AC;

double CD, AD; //поля для хранения длин сторон

public:

void SetX1(double t1);

void SetX2(double t1);

void SetX3(double t1);

void SetY1(double t2);

void SetY2(double t2);

void SetY3(double t2); //методы для присваивания координат в protected поля

void SetAB(double ab);

void SetBC(double bc);

void SetAC(double ac);

void SetCD(double cd);

void SetAD(double ad); //методы для присваивания длин сторон в protected поля

ThreeVertex() {} //конструктор по умолчанию

bool Proverka1(ThreeVertex Dlin); //метод для проверки существования треугольника

double& operator!()const;

double operator&()const;

double& operator~()const;

double operator++()const; //перегрузки соответствующих унарных операций

friend double DlinaStoroniAB(ThreeVertex ABC, ThreeVertex Dlin);

friend double DlinaStoroniBC(ThreeVertex ABC, ThreeVertex Dlin);

friend double DlinaStoroniAC(ThreeVertex ABC, ThreeVertex Dlin);

friend double DlinaStoroniAB(FourVertex ABCD);

friend double DlinaStoroniBC(FourVertex ABCD);

friend double DlinaStoroniCD(FourVertex ABCD);

friend double DlinaStoroniAD(FourVertex ABCD);

friend double DlinaStoroniAC(FourVertex ABCD); //дружественные функции для доступа соответствующей функции к полям и методам используемого класса

};

* 1. **Интерфейс класса четырехугольник**

class FourVertex :public ThreeVertex {

double x4, y4;

public:

void SetX4(double t1);

void SetY4(double t2); //функции для присваивания координат в protected поля

friend double DlinaStoroniAB(FourVertex ABCD);

friend double DlinaStoroniBC(FourVertex ABCD);

friend double DlinaStoroniCD(FourVertex ABCD);

friend double DlinaStoroniAD(FourVertex ABCD);

friend double DlinaStoroniAC(FourVertex ABCD); //дружественные функции для доступа соответствующей функции к полям и методам используемого класса

bool function1(FourVertex ABCD);//метод для определения расположения в полуплоскостях точек С и D относительно прямой AB

bool function2(FourVertex ABCD); );//метод для определения расположения в полуплоскостях точек A и D относительно прямой BC

bool function3(FourVertex ABCD); //метод для определения расположения в полуплоскостях точек A и B относительно прямой CD

bool function4(FourVertex ABCD); //метод для определения расположения в полуплоскостях точек B и C относительно прямой AD

bool vectordiag1(FourVertex ABCD);//метод для реализации проверки четырехугольника на выпуклость, используя векторное произведение векторов (№1)

bool Itog(FourVertex ABCD); //метод для реализации проверки четырехугольника на выпуклость, используя его математическое определение (№2)

};

**3.3 Переменные, объявленные в «теле» программы:**

* ThreeVertex ABC, Dlin – объекты классы треугольник;
* double t1, t2 – параметры, осуществляющие связь пользователя с программой, для временного хранения данных;
* double a,b,c,d – переменные для явного хранения длин сторон;
* double diag – переменная для хранения длины диагонали четырехугольника;
* FourVertex ABCD – объект класса четырехугольник;
  1. **Описание структуры программного комплекса**

Для удобного диалога с русскоязычным пользователем используется русификация вывода при помощи функции setlocale(LC\_ALL, "Russian");

Функции подсчета соответствующих сторон треугольника\четырехугольника:

double DlinaStoroniAB(ThreeVertex ABC, ThreeVertex Dlin);

double DlinaStoroniBC(ThreeVertex ABC, ThreeVertex Dlin);

double DlinaStoroniAC(ThreeVertex ABC, ThreeVertex Dlin);

double DlinaStoroniAB(FourVertex ABCD);

double DlinaStoroniBC(FourVertex ABCD);

double DlinaStoroniCD(FourVertex ABCD);

double DlinaStoroniAD(FourVertex ABCD);

double DlinaStoroniAC(FourVertex ABCD);

Программа использует функцию cout для вывода запросов и результатов на экран пользователя; функцию cin – для считывания данных, введенных пользователем.

Данная программа состоит из пяти файлов:

1. ThreeVertex.h - в файле хранится реализация класса треугольник;
2. FourVertex.h - в файле хранится реализация класса четырехугольник;
3. Functions.h - файл используется для хранения реализаций функций и методов;
4. TheHead.h - файл хранит прототипы функций;
5. The main fail.cpp – основной файл;

**Заключение**

В результате выполнения лабораторной работы был разработан базовый класс треугольников и производный класс четырехугольников. Выполненная программа определяет существование треугольника и производит проверку четырехугольника на выпуклость, определяет площадь и периметр существующего треугольника и выпуклого четырехугольника. Была произведена операция перегрузок таких унарных операций, как «!, ~, &, ++».

**Список литературы**

## 1. Герберт Шилдт. С++ для начинающих. Шаг за шагом

# 2. Стивен Прата. Язык программирования C++. Лекции и упражнения

## 3. Роберт Лафоре. Объектно-ориентированное программирование в С++

**Приложение**

**Код программы:**

1. **«ThreeVertex.h»**

#pragma once

#include "FourVertex.h"

#include "TheHead.h"

using namespace std;

class FourVertex;

class ThreeVertex {

protected:

double x1, y1;

double x2, y2;

double x3, y3;

double AB, BC, AC;

double CD, AD;

public:

void SetX1(double t1);

void SetX2(double t1);

void SetX3(double t1);

void SetY1(double t2);

void SetY2(double t2);

void SetY3(double t2);

void SetAB(double ab);

void SetBC(double bc);

void SetAC(double ac);

void SetCD(double cd);

void SetAD(double ad);

ThreeVertex() {}

bool Proverka1(ThreeVertex Dlin);

double& operator!()const;

double operator&()const;

double& operator~()const;

double operator++()const;

friend double DlinaStoroniAB(ThreeVertex ABC, ThreeVertex Dlin);

friend double DlinaStoroniBC(ThreeVertex ABC, ThreeVertex Dlin);

friend double DlinaStoroniAC(ThreeVertex ABC, ThreeVertex Dlin);

friend double DlinaStoroniAB(FourVertex ABCD);

friend double DlinaStoroniBC(FourVertex ABCD);

friend double DlinaStoroniCD(FourVertex ABCD);

friend double DlinaStoroniAD(FourVertex ABCD);

friend double DlinaStoroniAC(FourVertex ABCD);

};

1. **«FourVertex.h»**

#pragma once

using namespace std;

#include "ThreeVertex.h"

class FourVertex :public ThreeVertex {

double x4, y4;

public:

void SetX4(double t1);

void SetY4(double t2);

friend double DlinaStoroniAB(FourVertex ABCD);

friend double DlinaStoroniBC(FourVertex ABCD);

friend double DlinaStoroniCD(FourVertex ABCD);

friend double DlinaStoroniAD(FourVertex ABCD);

friend double DlinaStoroniAC(FourVertex ABCD);

bool function1(FourVertex ABCD);

bool function2(FourVertex ABCD);

bool function3(FourVertex ABCD);

bool function4(FourVertex ABCD);

bool vectordiag1(FourVertex ABCD);

bool Itog(FourVertex ABCD);

};

1. **«Functions.h»**

#pragma once

#include <cmath>

#include "FourVertex.h"

using namespace std;

void ThreeVertex::SetX1(double t1) {

this->x1 = t1;

}

void ThreeVertex::SetX2(double t1) {

this->x2 = t1;

}

void ThreeVertex::SetX3(double t1) {

this->x3 = t1;

}

void ThreeVertex::SetY1(double t2) {

this->y1 = t2;

}

void ThreeVertex::SetY2(double t2) {

this->y2 = t2;

}

void ThreeVertex::SetY3(double t2) {

this->y3 = t2;

}

void ThreeVertex::SetAB(double ab) {

this->AB = ab;

}

void ThreeVertex::SetBC(double bc) {

this->BC = bc;

}

void ThreeVertex::SetAC(double ac) {

this->AC = ac;

}

void ThreeVertex::SetCD(double cd) {

this->CD = cd;

}

void ThreeVertex::SetAD(double ad) {

this->AD = ad;

}

bool ThreeVertex::Proverka1(ThreeVertex Dlin) {

if ((Dlin.AB < Dlin.BC + Dlin.AC) && (Dlin.BC < Dlin.AB + Dlin.AC) && (Dlin.AC < Dlin.AB + Dlin.BC)) {

cout << "Проверка выполнена: треугольник существует." << endl;

double P, S, pp;

P = Dlin.AB + Dlin.BC + Dlin.AC;

pp = P / 2;

S = sqrt(pp \* (pp - Dlin.AB) \* (pp - Dlin.BC) \* (pp - Dlin.AC));

cout << "Периметр треугольника = " << P << endl << "Площадь треугольника = " << S << endl;

return true;

}

else {

cout << "Проверка выполнена: треугольник не существует. Программа не может быть выполнена. Выходим из программы..." << endl;

return false;

}

}

double& ThreeVertex::operator!()const {

double\* a = new double(AB + BC + AC);

return \*a;

}

double ThreeVertex::operator&()const {

double pp = 0.5 \* (AB + BC + AC);

return (sqrt(pp \* (pp - AB) \* (pp - BC) \* (pp - AC)));

}

double& ThreeVertex::operator~()const {

double\* a = new double(AB + BC + CD + AD);

return \*a;

}

double ThreeVertex::operator++()const {

double pp = 0.5 \* (AB + BC + CD + AD);

return (sqrt(pp \* (pp - AB) \* (pp - BC) \* (pp - AC)) + sqrt(pp \* (pp - CD) \* (pp - AD) \* (pp - AC)));

}

void FourVertex::SetX4(double t1) {

this->x4 = t1;

}

void FourVertex::SetY4(double t2) {

this->y4 = t2;

}

bool FourVertex::function1(FourVertex ABCD) {

double A1 = ABCD.y2 - ABCD.y1;

double B1 = ABCD.x1 - ABCD.x2;

double C1 = ABCD.x2 \* ABCD.y1 - ABCD.x1 \* ABCD.y2;

if (((A1 \* ABCD.x3 + B1 \* ABCD.y3 + C1 > 0) && (A1 \* ABCD.x4 + B1 \* ABCD.y4 + C1 > 0)) || ((A1 \* ABCD.x3 + B1 \* ABCD.y3 + C1 < 0) && (A1 \* ABCD.x4 + B1 \* ABCD.y4 + C1 < 0)))

return true;

else return false;

}

bool FourVertex::function2(FourVertex ABCD) {

double A2 = ABCD.y3 - ABCD.y2;

double B2 = ABCD.x2 - ABCD.x3;

double C2 = ABCD.x3 \* ABCD.y2 - ABCD.x2 \* ABCD.y3;

if (((A2 \* ABCD.x4 + B2 \* ABCD.y4 + C2 > 0) && (A2 \* ABCD.x1 + B2 \* ABCD.y1 + C2 > 0)) || ((A2 \* ABCD.x4 + B2 \* ABCD.y4 + C2 < 0) && (A2 \* ABCD.x1 + B2 \* ABCD.y1 + C2 < 0)))

return true;

else return false;

}

bool FourVertex::function3(FourVertex ABCD) {

double A3 = ABCD.y4 - ABCD.y3;

double B3 = ABCD.x3 - ABCD.x4;

double C3 = ABCD.x4 \* ABCD.y3 - ABCD.x3 \* ABCD.y4;

if (((A3 \* ABCD.x1 + B3 \* ABCD.y1 + C3 > 0) && (A3 \* ABCD.x2 + B3 \* ABCD.y2 + C3 > 0)) || ((A3 \* ABCD.x1 + B3 \* ABCD.y1 + C3 < 0) && (A3 \* ABCD.x2 + B3 \* ABCD.y2 + C3 < 0)))

return true;

else return false;

}

bool FourVertex::function4(FourVertex ABCD) {

double A4 = ABCD.y1 - ABCD.y4;

double B4 = ABCD.x4 - ABCD.x1;

double C4 = ABCD.x1 \* ABCD.y4 - ABCD.x4 \* ABCD.y1;

if (((A4 \* ABCD.x2 + B4 \* ABCD.y2 + C4 > 0) && (A4 \* ABCD.x3 + B4 \* ABCD.y3 + C4 > 0)) || ((A4 \* ABCD.x2 + B4 \* ABCD.y2 + C4 < 0) && (A4 \* ABCD.x3 + B4 \* ABCD.y3 + C4 < 0)))

return true;

else return false;

}

bool FourVertex::vectordiag1(FourVertex ABCD) {

char flag;

double x1 = ABCD.x3 - ABCD.x1;

double y1 = ABCD.y3 - ABCD.y1;

double x2 = ABCD.x2 - ABCD.x1;

double y2 = ABCD.y2 - ABCD.y1;

double x3 = ABCD.x4 - ABCD.x1;

double y3 = ABCD.y4 - ABCD.y1;

double x11 = ABCD.x4 - ABCD.x2;

double y11 = ABCD.y4 - ABCD.y2;

double x22 = ABCD.x1 - ABCD.x2;

double y22 = ABCD.y1 - ABCD.y2;

double x33 = ABCD.x3 - ABCD.x2;

double y33 = ABCD.y3 - ABCD.y2;

cout << "Выберите один из двух вариантов и введите соответвующую цифру:" << endl << "Координаты четырехугольника введены в порядке их следования ПО часовой стрелке - \"1\"" << endl << "Координаты четырехугольника введены в порядке их следования ПРОТИВ часовой стрелке - \"2\"" << endl;

cin >> flag;

if (flag == '1') {

if (((x1 \* y2 - x2 \* y1) > 0) && ((x1 \* y3 - x3 \* y1) < 0) && ((x11 \* y22 - x22 \* y11) < 0) && ((x11 \* y33 - x33 \* y11) > 0))

{

cout << "Проверка №1 четырехугольника выполнена: является выпуклым." << endl;

return true;

}

else {

cout << "Проверка №1 четырехугольника выполнена: НЕ является выпуклым." << endl;

return false;

}

}

else if (flag == '2') {

if (((x1 \* y2 - x2 \* y1) < 0) && ((x1 \* y3 - x3 \* y1) > 0) && ((x11 \* y22 - x22 \* y11) > 0) && ((x11 \* y33 - x33 \* y11) < 0))

{

cout << "Проверка №1 четырехугольника выполнена: является выпуклым." << endl;

return true;

}

else {

cout << "Проверка №1 четырехугольника выполнена: НЕ является выпуклым." << endl;

return false;

}

}

else

while ((flag != '2') || (flag != '1'))

cout << "Выберите один из предоставленных вариантов";

}

bool FourVertex::Itog(FourVertex ABCD) {

bool i1, i2, i3, i4, k, point;

i1 = function1(ABCD);

i2 = function2(ABCD);

i3 = function3(ABCD);

i4 = function4(ABCD);

k = vectordiag1(ABCD);

if (i1 && i2 && i3 && i4) {

cout << "Проверка №2 четырехугольника выполнена: является выпуклым." << endl;

point = true;

}

else {

cout << "Проверка №2 четырехугольника выполнена: НЕ является выпуклым. Выходим из программы...";

point = false;

}

if (point == k) {

return point;

}

}

double DlinaStoroniAB(ThreeVertex ABC, ThreeVertex Dlin) {

Dlin.AB = sqrt(pow((ABC.x2 - ABC.x1), 2) + pow((ABC.y2 - ABC.y1), 2));

return (Dlin.AB);

}

double DlinaStoroniBC(ThreeVertex ABC, ThreeVertex Dlin) {

Dlin.BC = sqrt(pow((ABC.x3 - ABC.x2), 2) + pow((ABC.y3 - ABC.y2), 2));

return (Dlin.BC);

}

double DlinaStoroniAC(ThreeVertex ABC, ThreeVertex Dlin) {

Dlin.AC = sqrt(pow((ABC.x3 - ABC.x1), 2) + pow((ABC.y3 - ABC.y1), 2));

return (Dlin.AC);

}

double DlinaStoroniAB(FourVertex ABCD) {

double a = sqrt(pow((ABCD.x2 - ABCD.x1), 2) + pow((ABCD.y2 - ABCD.y1), 2));

return (a);

}

double DlinaStoroniBC(FourVertex ABCD) {

double b = sqrt(pow((ABCD.x3 - ABCD.x2), 2) + pow((ABCD.y3 - ABCD.y2), 2));

return (b);

}

double DlinaStoroniCD(FourVertex ABCD) {

double c = sqrt(pow((ABCD.x4 - ABCD.x3), 2) + pow((ABCD.y4 - ABCD.y3), 2));

return (c);

}

double DlinaStoroniAD(FourVertex ABCD) {

double d = sqrt(pow((ABCD.x1 - ABCD.x4), 2) + pow((ABCD.y1 - ABCD.y4), 2));

return (d);

}

double DlinaStoroniAC(FourVertex ABCD) {

double diag = sqrt(pow((ABCD.x1 - ABCD.x3), 2) + pow((ABCD.y1 - ABCD.y3), 2));

return (diag);

}

1. **«TheHead.h»**

#pragma once

#include <iostream>

#include "Functions.h"

double DlinaStoroniAB(ThreeVertex ABC, ThreeVertex Dlin);

double DlinaStoroniBC(ThreeVertex ABC, ThreeVertex Dlin);

double DlinaStoroniAC(ThreeVertex ABC, ThreeVertex Dlin);

double DlinaStoroniAB(FourVertex ABCD);

double DlinaStoroniBC(FourVertex ABCD);

double DlinaStoroniCD(FourVertex ABCD);

double DlinaStoroniAD(FourVertex ABCD);

double DlinaStoroniAC(FourVertex ABCD);

1. **«The main fail.cpp»**

#include "TheHead.h"

int main() {

setlocale(LC\_ALL, "ru");

ThreeVertex ABC;

ThreeVertex Dlin;

double a, b, c, d;

double t1, t2;

cout << "Введите координаты первой вершины треугольника на плоскости (x1;y1): " << endl;

cin >> t1 >> t2;

ABC.SetX1(t1); ABC.SetY1(t2);

cout << "Введите координаты второй вершины треугольника на плоскости (x2;y2): " << endl;

cin >> t1 >> t2;

ABC.SetX2(t1); ABC.SetY2(t2);

cout << "Введите координаты третьей вершины треугольника на плоскости (x3;y3): " << endl;

cin >> t1 >> t2;

ABC.SetX3(t1); ABC.SetY3(t2);

a = DlinaStoroniAB(ABC, Dlin);

b = DlinaStoroniBC(ABC, Dlin);

c = DlinaStoroniAC(ABC, Dlin);

Dlin.SetAB(a);

Dlin.SetBC(b);

Dlin.SetAC(c);

if (Dlin.Proverka1(Dlin)) {

cout << "Длины сторон треугольника: " << endl << "a = " << a << endl << "b = " << b << endl << "c = " << c << endl;

}

else return 0;

cout <<"Периметр треугольника после перегрузки бинарного оператора \"!\" = "<< !Dlin <<endl;

cout << "Площадь треугольника после перегрузки бинарного оператора \"&\" = " << &Dlin << endl;

FourVertex ABCD;

cout << endl << "P.S. Необходимо ввести координаты четырехугольника в порядке их следования по часовой или против часовой стрелки..." << endl;

cout <<endl<< "Введите координаты первой вершины четырехугольника на плоскости (x1;y1): " << endl;

cin >> t1 >> t2;

ABCD.SetX1(t1); ABCD.SetY1(t2);

cout << "Введите координаты второй вершины четырехугольника на плоскости (x2;y2): " << endl;

cin >> t1 >> t2;

ABCD.SetX2(t1); ABCD.SetY2(t2);

cout << "Введите координаты третьей вершины четырехугольника на плоскости (x3;y3): " << endl;

cin >> t1 >> t2;

ABCD.SetX3(t1); ABCD.SetY3(t2);

cout << "Введите координаты четвертой вершины четырехугольника на плоскости (x4;y4): " << endl;

cin >> t1 >> t2;

ABCD.SetX4(t1); ABCD.SetY4(t2);

a = DlinaStoroniAB(ABCD);

b = DlinaStoroniBC(ABCD);

c = DlinaStoroniCD(ABCD);

d = DlinaStoroniAD(ABCD);

double diag = DlinaStoroniAC(ABCD);

Dlin.SetAB(a);

Dlin.SetBC(b);

Dlin.SetCD(c);

Dlin.SetAD(d);

Dlin.SetAC(diag);

if (ABCD.Itog(ABCD))

{

double Perimetr, Square;

Perimetr = a + b + c + d;

double pp = 0.5 \* Perimetr;

Square = sqrt(pp \* (pp - a) \* (pp - b) \* (pp - diag)) + sqrt(pp \* (pp - c) \* (pp - d) \* (pp - diag));

cout << "Длины сторон четырехугольника: " << endl << "a = " << a << endl << "b = " << b << endl << "c = " << c <<endl<<"d = "<<d<< endl;

cout << "Периметр четырехугольника = " << Perimetr << endl << "Площадь четырехугольника = " << Square << endl;

}

else return 0;

cout << "Периметр четырехугольника после перегрузки бинарного оператора \"~\" = " << ~Dlin << endl;

cout << "Площадь четырехугольника после перегрузки бинарного оператора \"++\" = " << ++Dlin << endl;

return 0;

}