**Министерство образования и науки**

Белгородский государственный технологический университет

им. В.Г. Шухова

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**КУРСОВАЯ РАБОТА**

по дисциплине «Основы программирования»

тема: «Нахождение площади пересечения 2-х выпуклых многоугольников»

Автор работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Новожен.Н.В

(подпись) ВТ-22

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Брусенцева В.С.

(подпись)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Белгород

2018 г.

**Оглавление**

[Введение 3](#_Toc536419757)

[Постановка задачи 3](#_Toc536419758)

[Теоретические сведения 3](#_Toc536419759)

[Обоснование выбора решения задачи. 8](#_Toc536419760)

[Демонстрационная программа. 9](#_Toc536419761)

[Блок-схема алгоритма в укрупненных блоках. 10](#_Toc536419762)

[Описание функций 12](#_Toc536419763)

[Тестовые данные 27](#_Toc536419764)

[Разбиение на модули 27](#_Toc536419765)

[Заголовочные файлы 27](#_Toc536419766)

[Заключение 28](#_Toc536419767)

[Список используемой литературы: 29](#_Toc536419768)

[Приложение 29](#_Toc536419769)

Введение

Поставленная в этой работе задача требует знания основных геометрических понятий и связана с аналитической геометрией.

Прежде чем приступить к решению задачи необходимо точно сформулировать понятие многоугольник.

Многоугольник(Полигон) - это [геометрическая](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F) фигура, обычно определяемая как часть плоскости ограниченная замкнутой не самопересекающейся [ломаной](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%BC%D0%B0%D0%BD%D0%B0%D1%8F).  
Ломанная— [геометрическая фигура](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B3%D1%83%D1%80%D0%B0_(%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F)), состоящая из [отрезков](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D1%80%D0%B5%D0%B7%D0%BE%D0%BA), последовательно соединённых своими концами.

Поиск пересечения 2-х многоугольников будет осуществляться с помощью алгоритма О ‘Рурка.

Вычисление площади пересечения будет осуществляться по формуле Гаусса.

Постановка задачи

Разработать алгоритм и написать программу на языке Си для определения

площади пересечения 2-х выпуклых многоугольников. Ввод может производится как с клавиатуры так из файла.

Теоретические сведения

Рассмотрим проблему вычисления области пересечения Р П Q двух выпуклых полигонов Р и Q. За исключением особо оговоренных случаев будем предполагать, что два полигона пересекаются невырожденно: пересечение двух ребер происходит в одной единственной точке, не являющейся вершиной какого-либо полигона. Учитывая такое предположение о невырожденности, всегда получим, что полигон Р П Q состоит из попеременных цепочек из Р и Q. Каждая пара последовательных цепочек соединяется в точке пересечения, в которой пересекаются границы полигонов Р и Q (рис. 1).

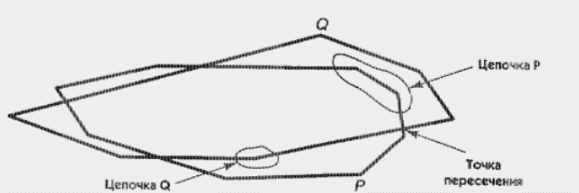


Рис. 1: Структура полигона пересечения Р П Q

Существует несколько решений этой задачи с линейной зависимостью времени выполнения от суммарного числа вершин. Описываемый здесь алгоритм обладает особым изяществом и его легко применять. Для двух заданных на входе выпуклых полигонов Р и Q алгоритм определяет окно на ребре полигона Р и еще одно окно на ребре полигона Q. Идея заключается в продвижении этих окон вдоль границ полигона по мере формирования полигона пересечения Р П Q: окна как бы проталкивают друг друга вдоль границы своих соответствующих полигонов в направлении по часовой стрелке для поиска точек пересечения ребер. Поскольку точки пересечения обнаруживаются в том порядке, в котором они располагаются вокруг полигона Р П Q, полигон пересечения оказывается сформированным, когда некоторая точка пересечения будет обнаружена вторично. В противном случае, если не будет обнаружено ни одной точки пересечения после достаточного числа итераций, то значит границы полигонов не пересекаются. В этом случае потребуется дополнительный простой тест, не содержится ли один полигон внутри другого или они вовсе не пересекаются.

Для объяснения работы оказывается весьма полезным ввести понятие серпа. На рис. 2 серпами будут шесть затененных полигонов. Каждый из них ограничен цепочкой, взятой от полигона Р, и цепочкой от полигона Q, ограниченных двумя последовательными точками пересечения. Внутренней цепочкой серпа будет та часть, которая принадлежит полигону пересечения. Отметим, что полигон пересечения окружен четным числом серпов, внутренние цепочки которых попеременно являются частями границ полигонов Р и Q.

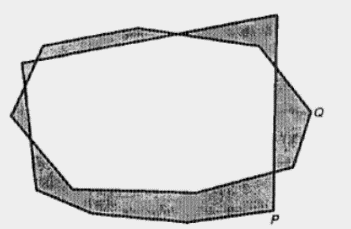


Рис. 2: Серпы, окружающие полигон пересечения

В терминах серпов алгоритм поиска полигона пересечения проходит две азы. В процессе первой фазы окно р полигона Р и окно q полигона Q перевещаются в направлении по движению часовой стрелки до тех пор, пока они не будут установлены на ребрах, принадлежащих одновременно одному тому же серпу. Каждое окно начинает свое движение с произвольной позиции. Здесь для краткости будем использовать один и тот же символ р для обозначения как окна полигона Р, так и ребра в этом окне. Тогда термин "начало р" будет относиться к точке начала ребра в окне полигона P, команда "продвинуть р" будет означать, что необходимо переместить окно полигона Р на следующее ребро. Аналогичным образом буквой q будем обозначать как окно полигона Q, так и ребро в окне. Иногда ребра р и q будем читать текущими ребрами.

Во время фазы 2 р и q продолжают перемещаться в направлении по ча-эвой стрелке, но на этот раз они двигаются в унисон от одного серпа к еднему серпу. Перед тем как любое окно перейдет из текущего серпа в седний, ребра р и q пересекутся в точке пересечения, соединяющей оба ерпа. Полигон пересечения строится именно во время второй фазы. Перед каждым перемещением р конечная точка ребра р заносится в полигон перечения, если ребро р принадлежит внутренней цепочке текущего серпа Аналогичным образом перед перемещением q фиксируется конечная точка ребра q, если q принадлежит внутренней цепочке текущего серпа. При кая дом пересечении ребер р и q точка пересечения, в которой они пересекаются, записывается в полигон пересечения.

Для принятия решения, какое из окон должно перемещаться, в алгоря ме используется правило перемещения. Это правило основано на следующв замечаниях: говорят, что ребро а нацелено на ребро b, если бесконечная прямая линия, определяемая ребром b, расположена перед а (рис. 3)

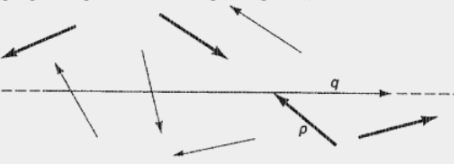


Рис. 3: Только показанные толстыми линиями ребра нацелены на ребро q, остальные – нет.

Ребро а нацелено на b, если выполняется одно из условий:

а х b > 0 и точка a.dest не лежат справа от b или

а х b < 0 и точка a.dest не лежат слева от b.

Если ребра а и b коллинеарны, то ребро а нацелено на b, если конечна точка a.dest не лежит после b. Это обстоятельство используется для тоге чтобы продвинуть а вместо b, когда два ребра пересекаются вырожденно более, чем в одной точке. Позволяя а "догонять" b, мы обеспечиваем, что ни одна точка пересечения не будет пропущена.

Возвратимся к обсуждению правил перемещения. Они сформулированы таким образом, чтобы не пропустить следующую точку пересечения. Правила отличают текущее ребро, которое может содержать следующую точку пересечения, от текущего ребра, которое возможно не может содержать следующей точки пересечения, причем в этом случае окно переносится вполне безопасно. Правила перемещения различают четыре ситуации, показанные на рис. 4. Здесь ребро а считается находящимся вне ребра b, если конечная точка a.dest расположена слева от b.

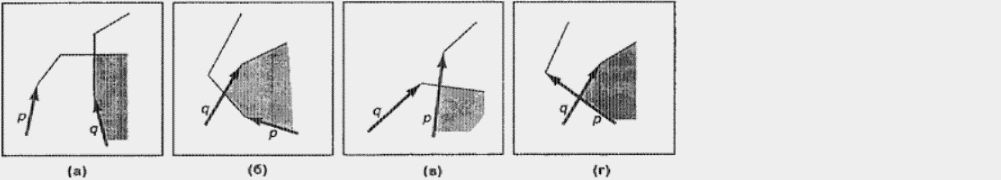


Рис. 4: Четыре правила перемещения: (а)-продвинуть р; б - продвинуть р,- в — продвинуть р; г — продвинуть р

1. р и q нацелены друг на друга: перемещается окно, соответствующее тому ребру (р или q), которое находится снаружи другого. В ситуации рис. 4а должно быть перенесено окно на ребре р. Следующая точка пересечения не может лежать на р, поскольку ребро р находится вне полигона пересечения.
2. р нацелено на q, но q не нацелено на р: конечная концевая точка ребра р заносится в полигон пересечения, если р не находится снаружи q и затем окно р переносится. На рис. 46 ребро р не может содержать следующей точки пересечения (хотя оно может содержать некоторую точку пересечения, если р находится не снаружи от q). На рис. показана ситуация, в которой ребро р, окно которого должно быть перенесено, не находится снаружи от ребра q.
3. q нацелено на р, но р не нацелено на q: конечная концевая точка ребра q заносится в полигон пересечения, если q не находится снаружи от р, после чего переносится окно q (рис. 6.14в). Этот случай симметричен предыдущему. На рис. показана ситуация, при которой ребро q, окно которого должно быть перенесено, находится снаружи от ребра Р-
4. р и q не нацелены друг на друга: переносится то окно, которое относится к ребру, расположенному снаружи от другого. Согласно рис. 4 необходимо перенести окно р, поскольку ребро р находится снаружи от ребра q.

Работа алгоритма показана на рис. 5. Каждое ребро имеет метку i, если оно обрабатывается на шаге i (у некоторых ребер показана двойная метка, поскольку они обрабатываются дважды). Два начальных ребра имеют метку 0. На этом рисунке фаза 2 (когда два текущих ребра оказывал принадлежащими одному и тому же серпу) начинается после трех итераций

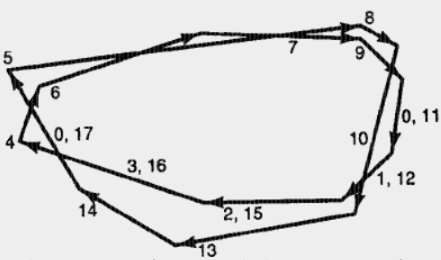
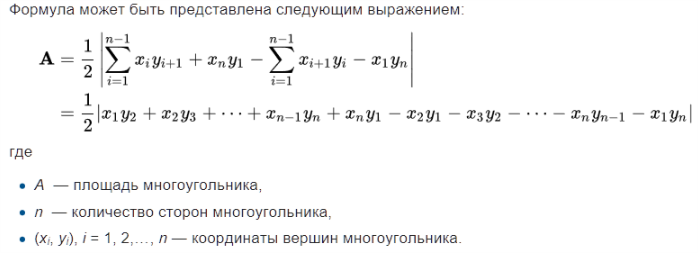


Рис. 5: Поиск полигона пересечения. Для ребра указана метка i, если оно обрабатывается на шаге i. Два начальных ребра обозначены меткой О

Если после выполнения 2(|Р| + |Q|) итераций не будет обнаружено ни одной точки пересечения, то основной цикл завершается, поскольку это означает, что границы полигона не пересекаются. Последующие обращения к функции pointlnPolygon производятся с целью обнаружения ситуаций Р С Q, Q П Р или Р П Q = 0. В противном случае, если найдена некоторая точка пересечения iPnt, то алгоритм продолжает построение полигона пересечения R и останавливается только после того, как точка iPnt будет обнаружена вторично.

Площадь полигона найдем по формуле Гаусса



Обоснование выбора решения задачи.

Решение поставленной задачи заключается в следующем:   
Ввод исходных данных производится из файла указанного пользователем или же из консоли. Точки многоугольников записываются последовательно по часовой стрелки. Если количество точек меньше 3, программа завершается т.к. площадь пресечения равна нулю, иначе начинается поиск пересечения многоугольников.

Точки представляют собой структуру

***Точка***

**typedef struct point {double x,y,z;} point\_3d ;**

**typedef point\_3d basetype;**

***Вектор хранится в этой же структуре***

***Полигон***

***typedef struct element \* Elptr;***

***struct element{basetype Date; Elptr Link; };***

***typedef struct LList { Elptr start,ptr;unsigned size; }List;***

Поиск т. пересечения(если отрезки перескаются) определяет функция crossingPoint(a,b,c,d,&iPnt).Где, ab вектор полигона P ,а cd вектор полигона Q.

Процедура advance продвигает текущее ребро полигона А, представляющее либо Р, либо Q. Эта же процедура заносит конечную концевую точку ребра х в полигон пересечения R, если А находится внутри другого полигона и \* не была последней точкой, записанной в R.

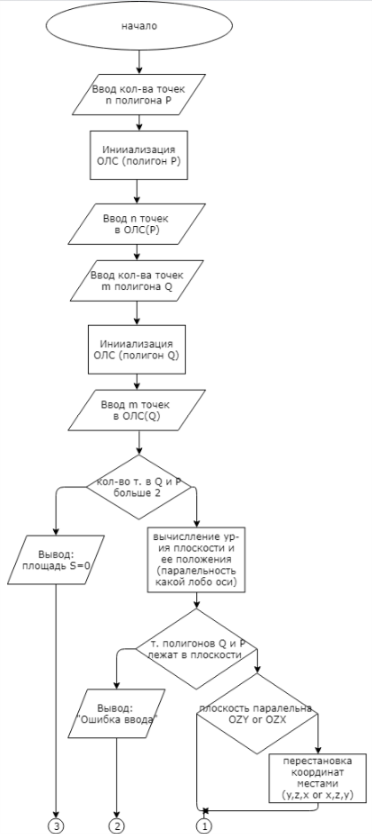
Функция aimsAt возвращает значение TRUE, если и только если ребр а нацелено на ребро b. Параметр aclass указывает на положение конечно точки a.dest относительно ребра b.

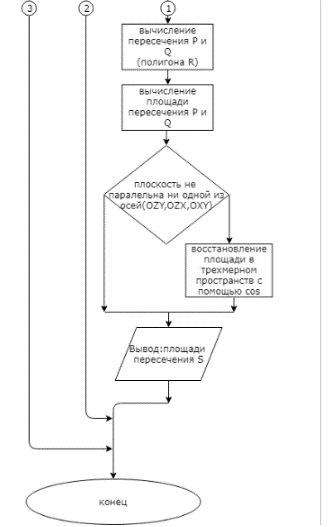
Программа выполняется за факториальное время (O(n+m)).

Демонстрационная программа.

Представляет собой программу считывающую название файла с исходными данными указанными в формате x y z, где x , y ,z соответствующие координаты точки в декартовой системе координат. При успешном нахождении площади фигуры площадь пересечения, а так же сохраняет результат в файл по запросу пользователя.

Блок-схема алгоритма в укрупненных блоках.





Описание структур данных

#include <math.h>

#define LISTOK 0 //Ошибок нет

#define LISTEMPLY 1 //Полигон пуст

#define LISTNOTMEM 2 //Нехватка памяти

#define EPSILON 0.001//Точность сравнения

#define TRUE 1

#define FALSE 0

#define M 4 //расмер массива для ур-ия плоскости

extern int err\_polig;

enum {OX=100,OY,OZ,OXYZ};//положение плоскости

enum {LEFT, RIGHT, BEYOND, BEHIND, BETWEEN, ORIGIN, DESTINATION};

// СЛЕВА, СПРАВА, ВПЕРЕДИ, ПОЗАДИ, МЕЖДУ, НАЧАЛО, КОНЕЦ

enum { COLLINEAR=66, PARALLEL, SKEW, SKEW\_CROSS, SKEW\_NO\_CROSS };

enum{UNKNOWN=20, P\_IS\_INSIDE, Q\_IS\_INSIDE};

extern int ListError;//Переменная ошибок

typedef struct point {double x,y,z;} point\_3d ;//точка

typedef point\_3d basetype;

typedef struct element \* Elptr;

struct element{basetype Date; Elptr Link; };

typedef struct LList { Elptr start,ptr;unsigned size; }List;

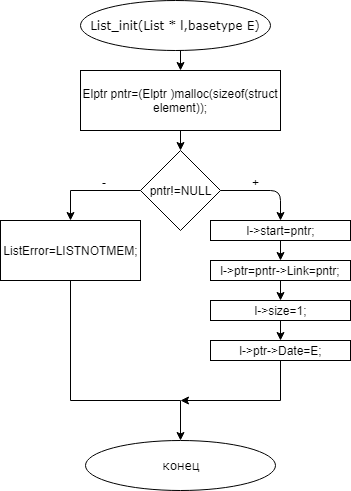
//Полигон,size кол-во т.,start -1 т.,ptr текущая т.

Описание функций

**Спецификация функции List\_init:**

void List\_init(List \* l ,basetype E);

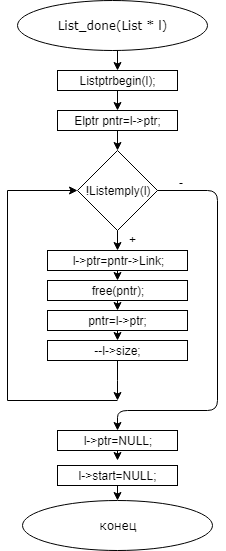
Назначение: Инициализация ОЛС и запись в указатель на структуру l т. E



**Спецификация функции List\_done:**

void List\_done(List \* l );

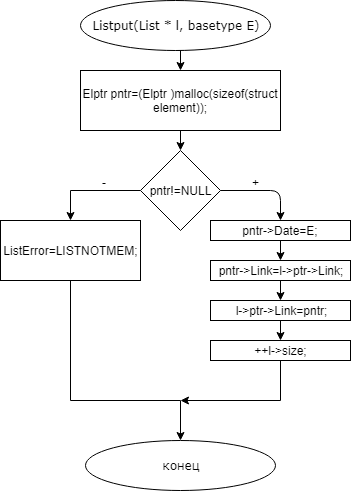
Назначение: Очистка ОЛС по указателю l



**Спецификация функции Listput:**

void Listput(List \* l, basetype E);

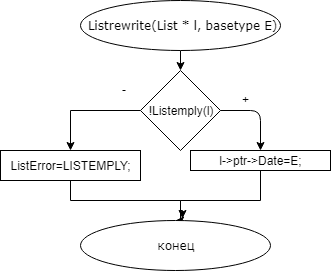
Назначение: запись в указатель на структуру l т. E



**Спецификация функции Listrewrite:**

void Listrewrite(List \* l, basetype E);

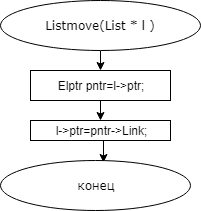
Назначение: перезапись в указатель на структуру l т. E



**Спецификация функции Listmove:**

void Listmove(List \* l );

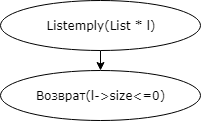
Назначение: Передвигает l->ptr по ОЛС



**Спецификация функции Listemply:**

int Listemply(List \* l);

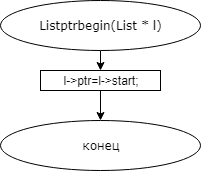
Назначение: Возвращает 1если ОЛС пуст



**Спецификация функции Listptrbegin:**

void Listptrbegin(List \* l);

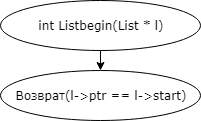
Назначение: Передвигает l->ptr в начало ОЛС



**Спецификация функции Listbegin:**

int Listbegin(List \* l);

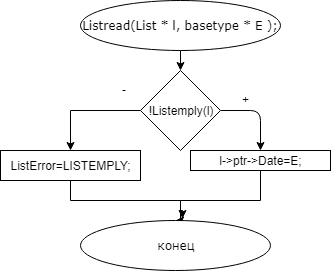
Назначение: Возвращает 1 если l->ptr указывает на начало



**Спецификация функции Listread:**

void Listread(List \* l, basetype \* E );

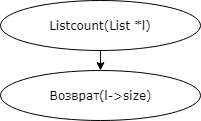
Назначение: Считыает в указатель на E значение по адресу l->ptr



**Спецификация функции Listcount:**

unsigned Listcount(List \*l);

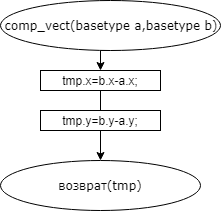
Назначение: Возвращает размер ОЛС



**Спецификация функции comp\_vect:**

basetype comp\_vect(basetype a,basetype b);

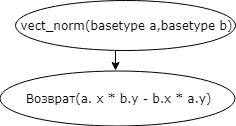
Назначение: вычисление вектора ab



**Спецификация функции vect\_norm**

double vect\_norm(basetype a,basetype b);

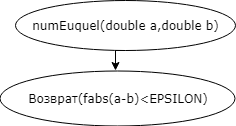
Назначение: Возвращает скалярное произведение вектора a и b на плоскости



**Спецификация функции numEuquel**

int numEuquel(double a,double b);

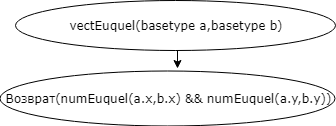
Назначение: Возвращает TRUE если числа равны



**Спецификация функции vectEuquel**

int vectEuquel(basetype a,basetype b);

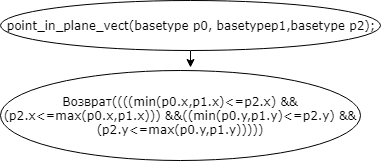
Назначение: Возвращает TRUE если векторы равны



**Спецификация функции point\_in\_plane\_vect**

int point\_in\_plane\_vect(basetype p0, basetype p1,basetype p2);

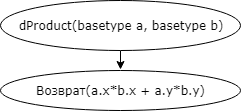
Назначение: Возвращает TRUE если проекция точки p2 лежит на вектре pop1



**Спецификация функции dProduct**

double dProduct(basetype a, basetype b);

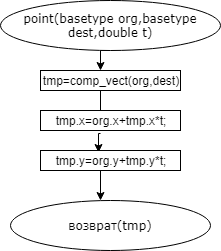
Назначение: возвращает скалярное троизведение в двумерном пространстве



**Спецификация функции point**

basetype point(basetype org,basetype dest,double t);

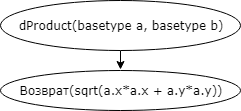
Назначение: возвращает вычесленую точки по параметру t



**Спецификация функции length\_v**

double length\_v(basetype a)

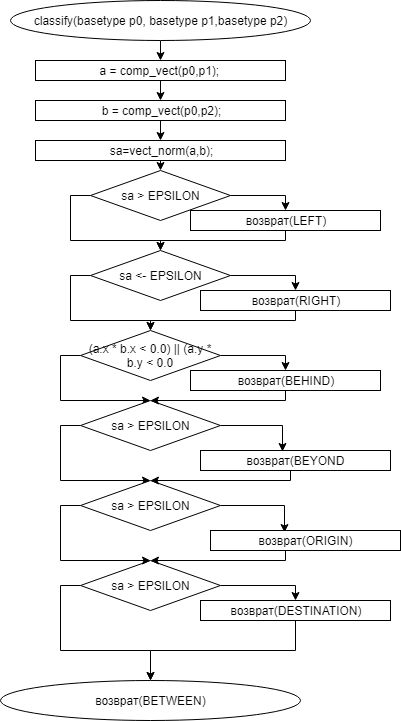
Назначение: возвращает длинну вектора а



**Спецификация функции classify**

int classify(basetype p0, basetype p1,basetype p2);

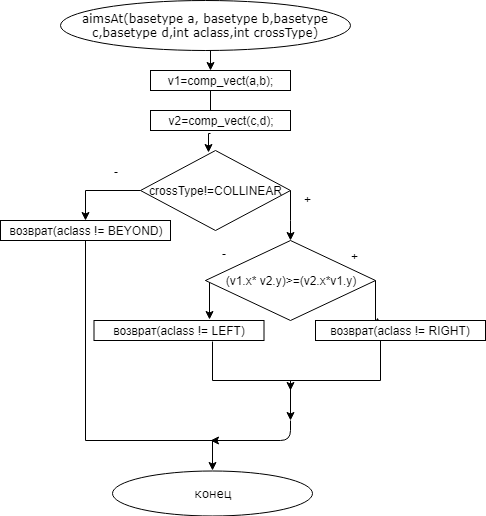
Назначение: возвращает роложение точки p2 относительно отрезка (p0,p1)



**Спецификация функции aimsAt**

int aimsAt(basetype a, basetype b,basetype c,basetype d,int aclass,int crossType);

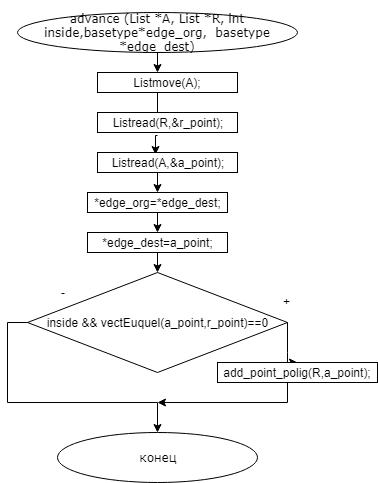
Назначение: возврат TRUE если ребро ab нацелено на cd



**Спецификация функции advance**

void advance (List \*A, List \*R, int inside,basetype \*edge\_org, basetype \*edge\_dest);

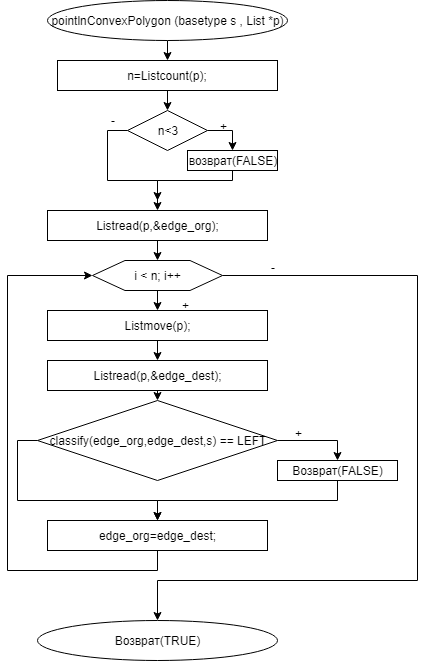
Назначение: Продвигает текущее ребро полигона по указателю A и заносит конечную концевую точку ребра в полигон пересечения R



**Спецификация функции pointlnConvexPolygon**

int pointlnConvexPolygon (basetype s , List \*p)

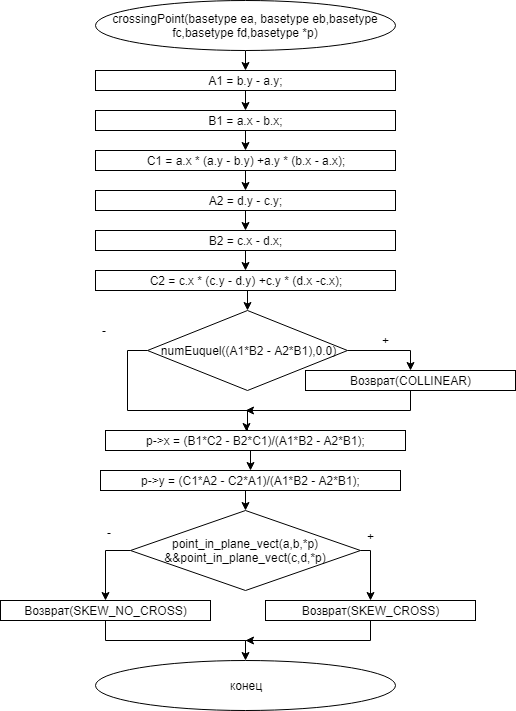
Назначение: возврат TRUE если точка s внутри полигона



**Спецификация функции crossingPoint**

int crossingPoint(basetype ea, basetype eb,basetype fc,basetype fd,basetype \*p);

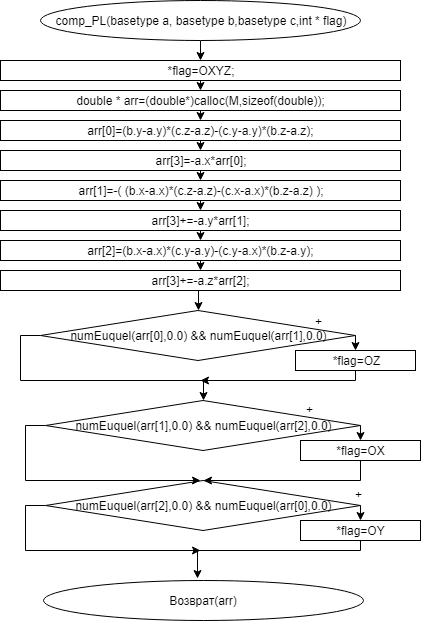
Назначение: возвращает положение ребер относительно друг друга(|| or skew or not skew) и если возможно, саписывает по указателю \*p т. пересечения



Спецификация функции

double \* comp\_PL(basetype ea, basetype eb,basetype fc,int \* flag);

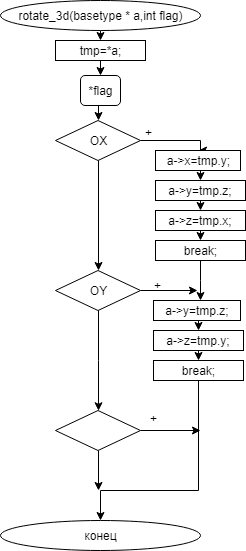
Назначение:возвращает указатель на массив [4] в кот. находится уравнение плоскости, в первенную flag возвр. значение оси паралельной оси координат



**Спецификация функции rotate\_3d**

void rotate\_3d(basetype \* a,int flag);

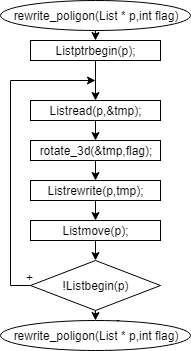
Назначение: меняет координаты местами в зависитости от значения flag



**Спецификация функции rewrite\_poligon**

void rewrite\_poligon(List \* p,int flag)

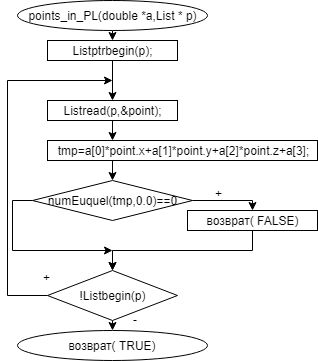
Назначение: меняет координаты местами в полигоне по адресу \*p в зависитости от значения flag



**Спецификация функции points\_in\_PL**

int points\_in\_PL(double \*a,List \* p)

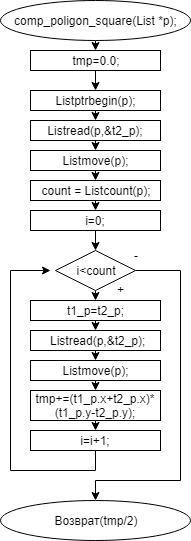
Назначение: возврат TRUE если все точки полигона p принадлежат плоскости хранящейся по указателю \*a



**Спецификация функции comp\_poligon\_square**

double comp\_poligon\_square(List \*p);

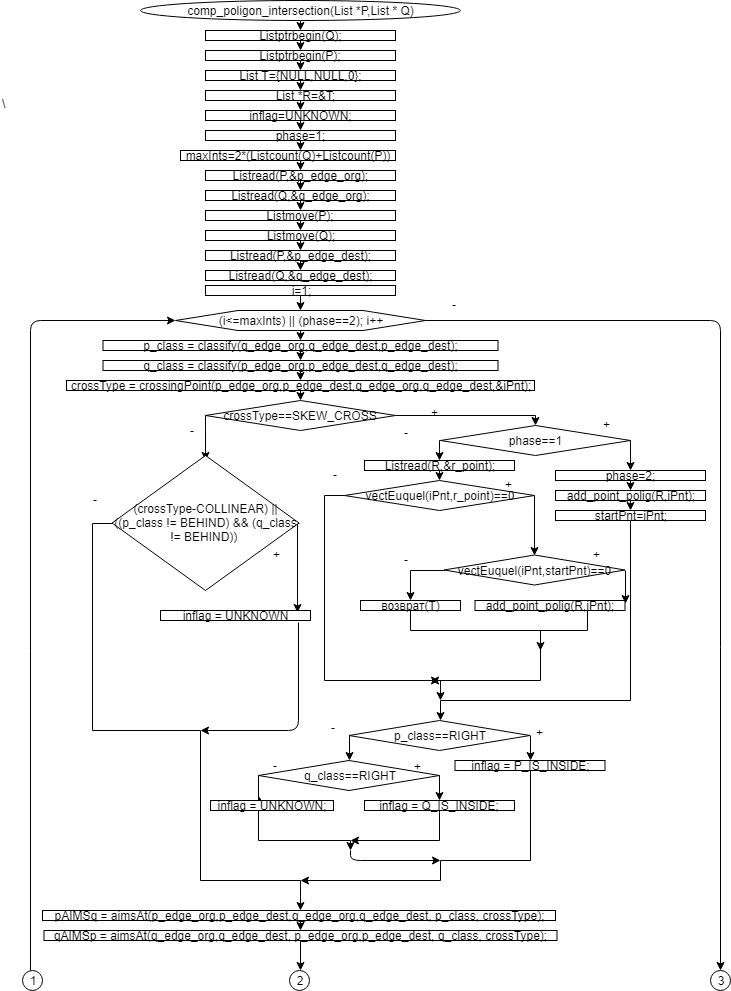
Назначение: возвращаем площадь полигона p

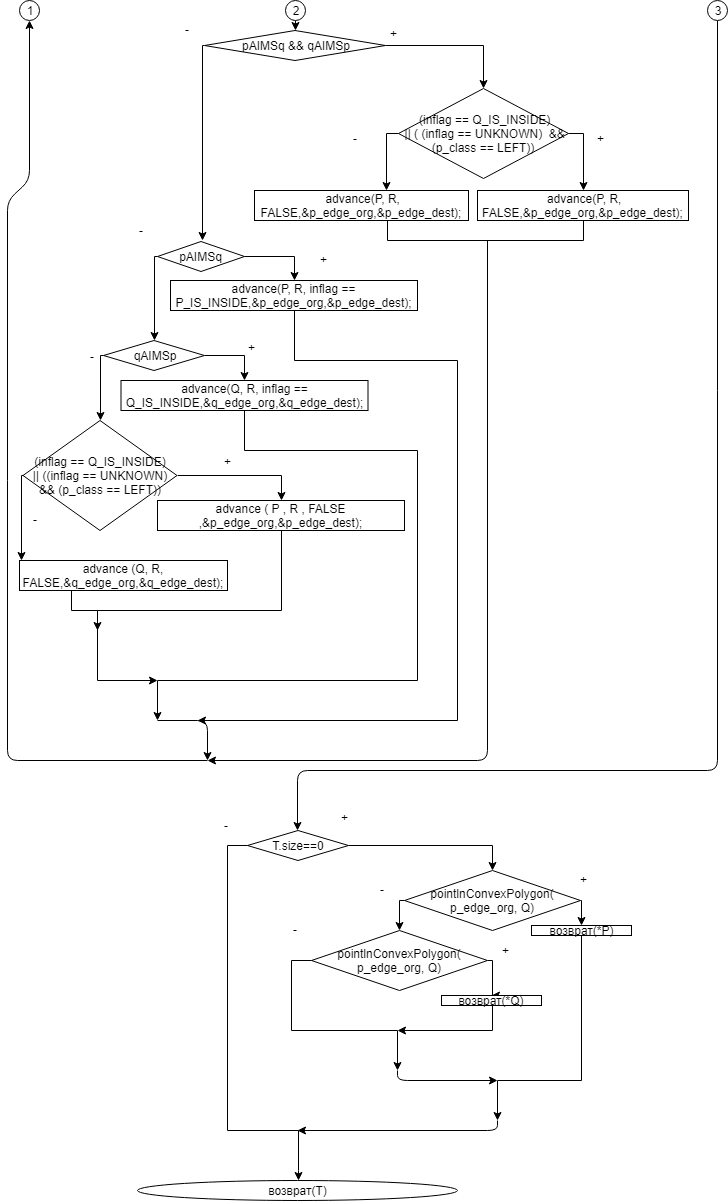


**Спецификация функции comp\_poligon\_intersection**

List comp\_poligon\_intersection(List \*P,List \* Q)

Назначение: возвращает полигон пересечения двух полигонов P и Q



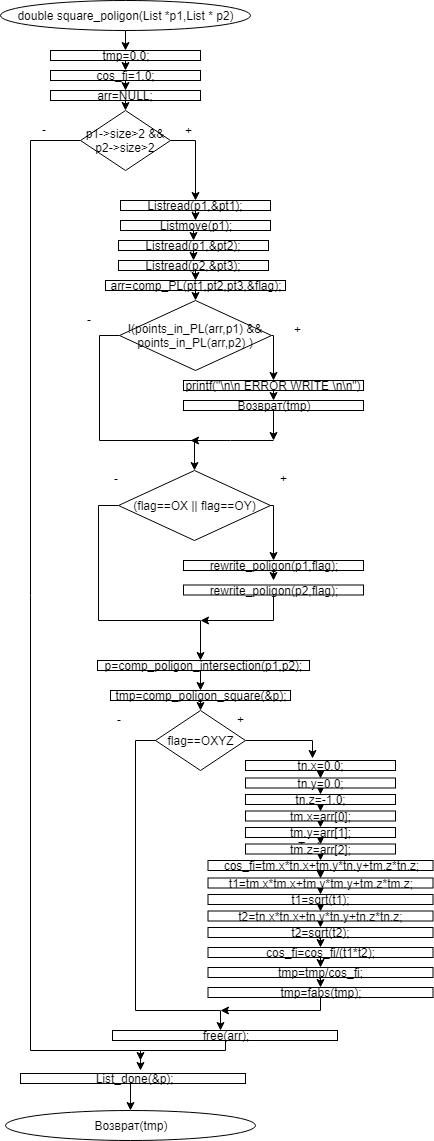


**Спецификация функции square\_poligon**

double square\_poligon(List \*p1,List \* p2)

Назначение: возвращает площадь полигона пересечения,

пересечения двух полигонов P и Q



Тестовые данные

|  |  |  |
| --- | --- | --- |
| Исходные данные | Полученный результат | Визуализация |
| 2  2.0 2.0 0.0  2.0 0.0 0.0  4  -1.0 2.0 0.0  1.0 2.0 0.0  1.0 0.0 0.0  -1.0 0.0 0.0 | Площадь = 0,0 |  |
| 4  2.0 2.0 0.0  2.0 0.0 0.0  0.0 0.0 0.0  0.0 2.0 0.0  4  -1.0 2.0 0.0  1.0 2.0 0.0  1.0 0.0 0.0  -1.0 0.0 0.0 | Площадь = 2,0 |  |
| 4  2.0 2.0 2.0  2.0 0.0 0.0  0.0 0.0 0.0  0.0 2.0 2.0  4  -1.0 2.0 2.0  1.0 2.0 2.0  1.0 0.0 0.0  -1.0 0.0 0.0 | Площадь = 2,8284 |  |

Разбиение на модули

Демонстрационная программа использует 3 модуля, модуль функций для работы с ОЛС (LIST\_K.h), модуль для решения задачи (poligon.h) и модуль для работы с файлами (fileio.h).

Заголовочные файлы

**Заголовочный файл LIST\_K.h**

**#ifndef LIST\_K\_H\_INCLUDED**

**#define LIST\_K\_H\_INCLUDED**

**#include <stdlib.h>**

**#include <stdio.h>**

**#include <string.h>**

**#define LISTOK 0**

**#define LISTEMPLY 1**

**#define LISTNOTMEM 2**

**extern int ListError;**

**typedef struct point {double x,y,z;} point\_3d ;**

**typedef point\_3d basetype;**

**typedef struct element \* Elptr;**

**struct element{basetype Date; Elptr Link; };**

**typedef struct LList { Elptr start,ptr;unsigned size; }List;**

**void List\_init(List \* l ,basetype E);**

**void List\_done(List \* l );**

**void Listput(List \* l, basetype E);**

**void Listrewrite(List \* l, basetype E);**

**void Listmove(List \* l );**

**int Listemply(List \* l);**

**void Listptrbegin(List \* l);**

**int Listbegin(List \* l);**

**void Listread(List \* l, basetype \* E );**

**unsigned Listcount(List \*l);**

**#endif // LIST\_K\_H\_INCLUDED**

**Заголовочный файл fileio.h**

#ifndef FILEIO\_H\_INCLUDED

#define FILEIO\_H\_INCLUDED

#include <stdio.h>

#include <stdlib.h>

#include"LIST\_K.h"

int fprint\_poligon(FILE \*f,List \*p1);

int fscan\_poligon(FILE \*f ,List \*p1);

#endif // FILEIO\_H\_INCLUDED

**Заголовочный файл poligon.h**

#ifndef POLIGON\_H\_INCLUDED

#define POLIGON\_H\_INCLUDED

#include <stdio.h>

#include <stdlib.h>

#include"LIST\_K.h"

#include"fileio.h"

#include <math.h>

#define EPSILON 0.001

#define TRUE 1

#define FALSE 0

#define M 4 //расмер массива для ур-ия плоскости

extern int err\_polig;

enum {OX=100,OY,OZ,OXYZ};

enum {LEFT, RIGHT, BEYOND, BEHIND, BETWEEN, ORIGIN, DESTINATION};

// СЛЕВА, СПРАВА, ВПЕРЕДИ, ПОЗАДИ, МЕЖДУ, НАЧАЛО, КОНЕЦ

enum { COLLINEAR=66, PARALLEL, SKEW, SKEW\_CROSS, SKEW\_NO\_CROSS };

enum{UNKNOWN=20, P\_IS\_INSIDE, Q\_IS\_INSIDE};

basetype comp\_vect(basetype a,basetype b);//вычисление вектора

double vect\_norm(basetype a,basetype b);//скалярное произведение

int numEuquel(double a,double b);//числа равны

int vectEuquel(basetype a,basetype b);//векторы равны

double min(double x, double y);

double max(double x, double y);

int point\_in\_plane\_vect(basetype p0, basetype p1,basetype p2);//проекция точки лежит на вектре

int classify(basetype p0, basetype p1,basetype p2);//где точка p2 относительно отрезка (p0,p1)

double dProduct(basetype a, basetype b);//скалярное троизведение

basetype point(basetype org,basetype dest,double t);//вычесление точки по параметру t

void add\_point\_polig(List \*p,basetype t);//добавить точку в полигон

int aimsAt(basetype a, basetype b,basetype c,basetype d,int aclass,int crossType);//возврат TRUE если ребро ab нацелено на cd

void advance (List \*A, List \*R, int inside,basetype \*edge\_org, basetype \*edge\_dest);

int pointlnConvexPolygon (basetype s , List \*p);// возврат TRUE если точка s внутри полигона

double length\_v (basetype a);//длинна вектора

int crossingPoint(basetype ea, basetype eb,basetype fc,basetype fd,basetype \*p);

//возвращает положение ребер относительно друг друга и если возможно,точку пересечения

double \* comp\_PL(basetype ea, basetype eb,basetype fc,int \* flag);

//возвращает указатель на массив [4] в кот находится уравнение плоскости

// в первенную flag возвр значение оси паралельной оси координат

int points\_in\_PL(double \*a,List \* p);//т. прин плоскости а

void rotate\_3d(basetype \* a,int flag);//меняет координаты местами

void rewrite\_poligon(List \* p,int flag);//меняет координаты местами во всем полигоне

double square\_poligon(List \*p1,List \* p2);//поиск пересечения полигонов

List comp\_poligon\_intersection(List \*p1,List \* p2);//вычисление полигона пересечения

double comp\_poligon\_square(List \*p);//вычисление площади полигона

#endif // POLIGON\_H\_INCLUDED

Заключение

В ходе выполнения курсовой работы были повторен материал курса “Алгебра и геометрия” и также получены новые знания в сфере аналитической геометрии.

Получены навыки работы с геоиетрическими фигурами, а также новые умения в сфере разработки алгоритмов по поставленным прикладным задачам и создание программ по созданным алгоритмам на языке программирования C, состоящих из нескольких модулей.

Список используемой литературы:

1. Нахождение пересечения и объединения геометрических объектов [Электронный ресурс] [URL: http://algolist.manual.ru/maths/geom/intersect/convex\_intersect.php](URL:https://ru.wikipedia.org/wiki/Стратегия_вычисления)
2. Структуры геометрических данных [Электронный ресурс]

<URL:http://algolist.manual.ru/maths/geom/datastruct.php\>

1. Локализация точки в выпуклом многоугольнике [Электронный ресурс]

[URL: https://habr.com/post/144571/#compgeom](URL:https://ru.wikipedia.org/wiki/Ломанная)

1. Нахождение точки пересечения 2 отрезков[Электронный ресурс]

[URL http://forum.algolist.ru/algorithm-geometry/2453-nahojdenie-tochki-peresecheniia-2-otrezkov.html](URL:https://ru.wikipedia.org/wiki/Ломанная)

1. Формула площади Гаусса [Электронный ресурс]

[URL https://ru.wikipedia.org/wiki/Формула\_площади\_Гаусса](URL:https://ru.wikipedia.org/wiki/Ломанная)

Приложение

**Исходный код fileio.с**

#include"fileio.h"

int fscan\_poligon(FILE \*f,List \*p1)

{

int i=0;

basetype point1;

fscanf(f,"%i",&i);

fscanf(f,"%lf %lf %lf ",&point1.x,&point1.y,&point1.z);

List\_init(p1,point1);

--i;

while(i>0)

{

fscanf(f,"%lf %lf %lf ",&point1.x,&point1.y,&point1.z);

Listput(p1,point1);

Listmove(p1);

--i;

}

return 0;

}

int fprint\_poligon(FILE \*f,List \*p1)

{

basetype point1;

Listptrbegin(p1);

int i=0, n=Listcount(p1);

fprintf(f,"%i \n",n);

while(n>i)

{

Listread(p1,&point1);

Listmove(p1);

fprintf(f,"x=%.3lf y=%.3lf z=%.3lf\n",point1.x,point1.y,point1.z);

++i;

}

return 0;

}

**Исходный код LIST\_K.с**

#include"LIST\_K.h"

int ListError=LISTOK;//=0

void List\_init(List \* l,basetype E)

{

Elptr pntr=(Elptr )malloc(sizeof(struct element));

if(pntr!=NULL) // память выделена

{

l->start=pntr;

l->ptr=pntr->Link=pntr;// делаум фиктивный указатель (сам на себя)

l->size=1;

l->ptr->Date=E;

}

else

ListError=LISTNOTMEM;

}

void List\_done(List \*l)

{

Listptrbegin(l);

Elptr pntr=l->ptr;

while(!Listemply(l))

{

l->ptr=pntr->Link;

free(pntr);

pntr=l->ptr;

--l->size;

}

l->ptr=NULL;l->start=NULL;

}

void Listput(List \* l, basetype E)

{

Elptr pntr=(Elptr)malloc(sizeof(struct element));//выделяем память

if(pntr!=NULL)

{

pntr->Date=E;

pntr->Link=l->ptr->Link;//Формируем указатель

l->ptr->Link=pntr;//модифицируем указатель

++l->size;

}

else ListError=LISTNOTMEM;

}

void Listrewrite(List \* l, basetype E)

{

if(!Listemply(l)) //список не пуст

{

l->ptr->Date=E;

}

else ListError=LISTEMPLY;

}

void Listmove(List \* l )

{

Elptr pntr=l->ptr;

l->ptr=pntr->Link;//двигаемся на 1 шаг ро списку

}

int Listemply(List \* l)

{

return l->size<=0; //список пуст

}

int Listbegin(List \* l)

{

return l->ptr==l->start; //мы на first элементе

}

void Listptrbegin(List \* l)

{

l->ptr=l->start; //в начасо списка

}

void Listread(List \* l, basetype \* E )

{

if(!Listemply(l)) //список не пуст

\*E=l->ptr->Date;

else ListError=LISTEMPLY;

}

unsigned Listcount(List \*l)

{

return l->size;

}

**Исходный код poligon.с**

#include"poligon.h"

int err\_polig=0;

basetype comp\_vect(basetype a,basetype b)

{

basetype tmp;

tmp.x=b.x-a.x;

tmp.y=b.y-a.y;

return tmp;

}

double vect\_norm(basetype a,basetype b)

{

return (a. x \* b.y - b.x \* a.y);

}

int numEuquel(double a,double b){return fabs(a-b)<EPSILON;}

int vectEuquel(basetype a,basetype b)

{

return ( numEuquel(a.x,b.x) && numEuquel(a.y,b.y) );

}

double max(double x, double y)

{

return (x < y)? y:x;

}

double min(double x, double y)

{

return (x < y)? x:y;

}

int point\_in\_plane\_vect(basetype p0, basetype p1,basetype p2)//проекция точки лежит на вектре

{

return( (((min(p0.x,p1.x)<=p2.x) && (p2.x<=max(p0.x,p1.x))) &&((min(p0.y,p1.y)<=p2.y) && (p2.y<=max(p0.y,p1.y)))) );

}

double dProduct(basetype a, basetype b)//скалярное троизведение

{

return (a.x\*b.x + a.y\*b.y );

}

double length\_v(basetype a)

{

return sqrt(a.x\*a.x + a.y\*a.y);

}

int classify(basetype p0, basetype p1,basetype p2)//где точка p2 относительно отрезка p0p1

{

basetype a = comp\_vect(p0,p1); //p1 - p0;

basetype b = comp\_vect(p0,p2);//p2 - p0;

//находим вектор нормали с

double sa=vect\_norm(a,b);//(a. x \* b.y - b.x \* a.y)

if (sa > EPSILON)//>0 правая тройка

return LEFT;

if (sa < -EPSILON)//<0 левая тройка

return RIGHT;

if ((a.x \* b.x < 0.0) || (a.y \* b.y < 0.0))

return BEHIND;

if (length\_v(a) < length\_v(b))

return BEYOND;

if (vectEuquel(p0 , p2))

return ORIGIN;

if (vectEuquel(p1 , p2))

return DESTINATION;

//if(point\_in\_vect

return BETWEEN;

}

basetype point(basetype org,basetype dest,double t)//

{

basetype tmp=comp\_vect(org,dest);

tmp.x=org.x+tmp.x\*t;

tmp.y=org.y+tmp.y\*t;

return tmp;//(org + t \* (dest - org));

}

void add\_point\_polig(List \* p,basetype t)//добавить точку в полигон

{

if(p->start!=NULL){ Listput(p,t); Listmove(p); }

else List\_init(p,t);

}

int aimsAt(basetype a, basetype b,basetype c,basetype d,int aclass,int crossType)

{

basetype v1=comp\_vect(a,b);

basetype v2=comp\_vect(c,d);

if(crossType!=COLLINEAR)

{ // if ab нацелено на вектор cd

if((v1.x\* v2.y)>=(v2.x\*v1.y))return (aclass != RIGHT);

else return (aclass != LEFT);

}

else return (aclass != BEYOND);

}

void advance (List \*A, List \*R, int inside, basetype \*edge\_org, basetype \*edge\_dest)

{

basetype r\_point,a\_point;

Listmove(A);

Listread(R,&r\_point);

Listread(A,&a\_point);

\*edge\_org=\*edge\_dest;

\*edge\_dest=a\_point;

if(inside && vectEuquel(a\_point,r\_point)==0 )add\_point\_polig(R,a\_point);

}

int pointlnConvexPolygon(basetype s , List \*p)

{

basetype edge\_org,edge\_dest;

unsigned n=Listcount(p);

if(n<3)return FALSE;

//print\_p(s);

Listread(p,&edge\_org);

for (int i = 0; i < n; i++)

{

Listmove(p);

//print\_p(edge\_org);

Listread(p,&edge\_dest);

//print\_p(edge\_dest);

if (classify(edge\_org,edge\_dest,s) == LEFT) return FALSE;

edge\_org=edge\_dest;

}

return TRUE;

}

int crossingPoint(basetype a, basetype b,basetype c,basetype d,basetype \*p)

{

double A1, B1, C1, A2, B2, C2;

//вычислим коэффициенты прямых Ax+By+C=0

A1 = b.y - a.y;

B1 = a.x - b.x;

C1 = a.x \* (a.y - b.y) +a.y \* (b.x - a.x);

A2 = d.y - c.y;

B2 = c.x - d.x;

C2 = c.x \* (c.y - d.y) +c.y \* (d.x -c.x);

//проверим, параллельны ли они?

if (numEuquel((A1\*B2 - A2\*B1),0.0))

return COLLINEAR;

//кооординаты точки пересечения

p->x = (B1\*C2 - B2\*C1)/(A1\*B2 - A2\*B1);

p->y = (C1\*A2 - C2\*A1)/(A1\*B2 - A2\*B1);

//проверим, находится ли точка пересечения прямых на отрезках

if (point\_in\_plane\_vect(a,b,\*p) &&point\_in\_plane\_vect(c,d,\*p))

return SKEW\_CROSS;

else

return SKEW\_NO\_CROSS;

}

List comp\_poligon\_intersection(List \*P,List \* Q)

{

Listptrbegin(Q);

Listptrbegin(P);

List T={NULL,NULL,0};

List \*R=&T;

basetype iPnt,startPnt,r\_point;

basetype p\_edge\_org,q\_edge\_org,p\_edge\_dest,q\_edge\_dest;

int inflag=UNKNOWN;

int phase=1;

int maxInts=2\*(Listcount(Q)+Listcount(P));

Listread(P,&p\_edge\_org);

Listread(Q,&q\_edge\_org);

Listmove(P);

Listmove(Q);

Listread(P,&p\_edge\_dest);

Listread(Q,&q\_edge\_dest);

for (int i = 1; (i<=maxInts) || (phase==2); i++)

{

int p\_class = classify(q\_edge\_org,q\_edge\_dest,p\_edge\_dest);

int q\_class = classify(p\_edge\_org,p\_edge\_dest,q\_edge\_dest);

int crossType = crossingPoint(p\_edge\_org,p\_edge\_dest,q\_edge\_org,q\_edge\_dest,&iPnt);

if(crossType==SKEW\_CROSS)

{

if(phase==1)

{

phase=2;

add\_point\_polig(R,iPnt);

startPnt=iPnt;

}

else

{

Listread(R,&r\_point);

if(vectEuquel(iPnt,r\_point)==0)

{if(vectEuquel(iPnt,startPnt)==0)add\_point\_polig(R,iPnt);

else return T;

}//else return T;

}

if (p\_class==RIGHT) inflag = P\_IS\_INSIDE;

else if (q\_class==RIGHT) inflag = Q\_IS\_INSIDE;

else inflag = UNKNOWN;

}

else if ( (crossType-COLLINEAR) || ((p\_class != BEHIND) && (q\_class != BEHIND)) )inflag = UNKNOWN;

int pAIMSq = aimsAt(p\_edge\_org,p\_edge\_dest,q\_edge\_org,q\_edge\_dest, p\_class, crossType);

int qAIMSp = aimsAt(q\_edge\_org,q\_edge\_dest, p\_edge\_org,p\_edge\_dest, q\_class, crossType);

if (pAIMSq && qAIMSp)

{

if ( (inflag == Q\_IS\_INSIDE) || ( (inflag == UNKNOWN) && (p\_class == LEFT)))

advance(P, R, FALSE,&p\_edge\_org,&p\_edge\_dest);

else

advance (Q, R, FALSE,&q\_edge\_org,&q\_edge\_dest);

}else if(pAIMSq)advance(P, R, inflag == P\_IS\_INSIDE,&p\_edge\_org,&p\_edge\_dest);

else if(qAIMSp)advance(Q, R, inflag == Q\_IS\_INSIDE,&q\_edge\_org,&q\_edge\_dest);

else if((inflag == Q\_IS\_INSIDE) || ((inflag == UNKNOWN) && (p\_class == LEFT)))// ||

advance ( P , R , FALSE ,&p\_edge\_org,&p\_edge\_dest);

else advance (Q, R, FALSE,&q\_edge\_org,&q\_edge\_dest);

}

if(T.size==0)

{

if (pointlnConvexPolygon( p\_edge\_org, Q) )return \*P;

else if (pointlnConvexPolygon(q\_edge\_org, P) )return \*Q;

}

return T;

}

double comp\_poligon\_square(List \*p)

{

double tmp=0.0;

basetype t1\_p,t2\_p;

Listptrbegin(p);

Listread(p,&t2\_p);

Listmove(p);

int i=0,count = Listcount(p);

while( i++<count)

{

t1\_p=t2\_p;

Listread(p,&t2\_p);

Listmove(p);

tmp+=(t1\_p.x+t2\_p.x)\*(t1\_p.y-t2\_p.y);//площадь

}

return fabs(tmp/2);

}

double square\_poligon(List \*p1,List \* p2)

{

int flag=OXYZ;

double tmp=0.0,cos\_fi=1.0,\* arr=NULL;

List p;

if(p1->size>2 && p2->size>2)

{

basetype pt1,pt2,pt3;

Listread(p1,&pt1);

Listmove(p1);

Listread(p1,&pt2);

Listread(p2,&pt3);

arr=comp\_PL(pt1,pt2,pt3,&flag);

if( !(points\_in\_PL(arr,p1) && points\_in\_PL(arr,p2) )){printf("\n\n ERROR WRITE \n\n");return tmp;}

if(flag==OX || flag==OY)rewrite\_poligon(p1,flag),rewrite\_poligon(p2,flag);

p=comp\_poligon\_intersection(p1,p2);

tmp=comp\_poligon\_square(&p);// 1==-OZ 2==-OY 3==-OX

if(flag==OXYZ)

{

double t1,t2;

basetype tm,tn;

tn.x=0.0;tn.y=0.0;tn.z=-1.0;//ур-ие плоскости XY

tm.x=arr[0];tm.y=arr[1];tm.z=arr[2];

cos\_fi=tm.x\*tn.x+tm.y\*tn.y+tm.z\*tn.z;

t1=tm.x\*tm.x+tm.y\*tm.y+tm.z\*tm.z;

t1=sqrt(t1);

t2=tn.x\*tn.x+tn.y\*tn.y+tn.z\*tn.z;

t2=sqrt(t2);

cos\_fi=cos\_fi/(t1\*t2);

tmp=tmp/cos\_fi;

tmp=fabs(tmp);

}

free(arr);

}

List\_done(&p);

return tmp;

}

double \* comp\_PL(basetype a, basetype b,basetype c,int \* flag)

{

\*flag=OXYZ;

double \* arr=(double\*)calloc(M,sizeof(double));

arr[0]=(b.y-a.y)\*(c.z-a.z)-(c.y-a.y)\*(b.z-a.z);

arr[3]=-a.x\*arr[0];

arr[1]=-( (b.x-a.x)\*(c.z-a.z)-(c.x-a.x)\*(b.z-a.z) );

arr[3]+=-a.y\*arr[1];

arr[2]=(b.x-a.x)\*(c.y-a.y)-(c.y-a.x)\*(b.z-a.y);

arr[3]+=-a.z\*arr[2];

if(numEuquel(arr[0],0.0) && numEuquel(arr[1],0.0))\*flag=OZ;//плоскость || XOY

if(numEuquel(arr[1],0.0) && numEuquel(arr[2],0.0))\*flag=OX;//плоскость || ZOY

if(numEuquel(arr[2],0.0) && numEuquel(arr[0],0.0))\*flag=OY;//плоскость || XOY

return arr;

}

void rotate\_3d(basetype \*a,int flag)

{

basetype tmp=\*a;

switch(flag)

{

case OX:a->x=tmp.y;a->y=tmp.z;a->z=tmp.x;break;

case OY:a->y=tmp.z;a->z=tmp.y;break;

default :;

}

}

void rewrite\_poligon(List \* p,int flag)

{

Listptrbegin(p);

basetype tmp;

do

{

Listread(p,&tmp);

rotate\_3d(&tmp,flag);

Listrewrite(p,tmp);

Listmove(p);

}while( !Listbegin(p));

}

int points\_in\_PL(double \*a,List \* p)

{

basetype point;

double tmp;

Listptrbegin(p);

do

{

Listread(p,&point);

tmp=a[0]\*point.x+a[1]\*point.y+a[2]\*point.z+a[3];

if(numEuquel(tmp,0.0)==0)return FALSE;

}while(!Listbegin(p));

return TRUE;

}

**Исходный код main.с**

#include <stdio.h>

#include <stdlib.h>

#include"LIST\_K.h"

#include"fileio.h"

#include"poligon.h"

#include <math.h>

#define N\_NAME 100

int main()

{

char name[N\_NAME];

List p1={NULL,NULL,0},p2={NULL,NULL,0};

int flag=1;

printf("write poligon input and output in files=1 or terminals=0 \n");

scanf("%i",&flag);

FILE \*f=stdin,\*d=stdout;

if(flag)

{

fgets(name,N\_NAME,stdin);

f=fopen(name,"r+");

}

fscan\_poligon(f,&p1);

fscan\_poligon(f,&p2);

if(flag)

{

fgets(name,N\_NAME,stdin);

d=fopen(name,"w+");

}

fprintf(d,"\n S=%f\n",square\_poligon(&p1,&p2));

fprint\_poligon(d,&p1);

fprint\_poligon(d,&p2);

List\_done(&p1);

List\_done(&p2);

printf("ERR %i \n",ListError);

return 0;

}

