**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Использование массивов для решения геометрической задачи.

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1309 |  | Ищенко Д.О. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2021

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студентка Ищенко Д.О. | | |
| Группа 1309 | | |
| Тема работы: Использование массивов для решения геометрической задачи | | |
| Исходные данные:  Дан набор координат точек на плоскости. Найти среди них точки, являющиеся вершинами фигуры, содержащей наибольшее количество заданных точек. Фигура представляет собой прямоугольник с полуокружностями на противоположных друг другу сторонах (эти стороны образуют диаметры полуокружностей). Полуокружности располагаются таким образом, что площадь всей фигуры больше площади исходного прямоугольника. | | |
| Содержание пояснительной записки:  Содержание, введение, анализ задания, описание решения задачи, интерфейсы взаимодействия с пользователем, текст программы, результат работы программы и заключение.  Предполагаемый объем пояснительной записки:  Не менее 5 страниц. | | |
| Дата выдачи задания: 16.11.2021 | | |
| Дата сдачи реферата: 22.12.2021 | | |
| Дата защиты реферата: 22.12.2021 | | |
| Студентка |  | Ищенко Д.О. |
| Преподаватель |  | Калмычков В.А. |

**Аннотация**

Задача нахождения фигуры решается перебором всех возможных комбинаций пяти вершин и проверкой каждой комбинации на выполнение набора условий. Этот набор позволяет однозначно определить, образуют ли данные точки искомую фигуру. Задача нахождения количества точек, принадлежащих данной фигуре представляет собой две подзадачи: о принадлежности точки окружности и о принадлежности точки прямоугольнику. Первая подзадача решается проверкой выполнения нестрогого неравенства уравнения окружности, вторая – с использованием утверждения о равенстве площади прямоугольника и суммы площадей треугольников, образованных соединением вершин прямоугольника с точкой, находящейся внутри него и его сторонами.

В ходе написания программы были задействованы структуры, как способ внутреннего хранения данных. Не смотря на это, написанная в ходе выполнения курсовой работы программа не является объектно-ориентированной и использует функции как основной способ разделения логики решения глобальной задачи.

Все взаимодействие с пользователем происходит через три файла: файл входных данных, файл выходных данных, файл протокола программы.

**Summary**

The problem of finding a shape is solved by iterating all possible combinations of five vertices and checking each combination for the fulfillment of a set of conditions. This set allows you to determine whether the given points form the desired shape. The problem of finding the number of points belonging to a given figure consists of two subtasks: the belonging of a point to a circle and the belonging of a point to a rectangle. The first subtask is solved by checking the fulfillment of the non-strict inequality of the equation of the circle, the second - using the statement about the equality of the area of a rectangle and the sum of the areas of triangles formed by connecting the vertices of a rectangle with a point located inside it and its sides.

During the writing of the program, structures were involved as a way of internal data storage. Despite this, the program written during the term paper is not object-oriented and uses functions as the main way to separate the logic for solving a global problem.

All interaction with the user occurs through three files: the input data file, the output data file, the program protocol file.

**содержание**

[Введение 5](#_Toc92708438)

[1. Анализ задания 6](#_Toc92708439)

[1.1. Формулировка задания. 6](#_Toc92708440)

[1.2. Выявление неточностей. 6](#_Toc92708441)

[1.3. Контрольный пример 8](#_Toc92708442)

[2. Описание решения задачи 10](#_Toc92708443)

[2.1. Математическая постановка решения. 10](#_Toc92708444)

[2.2. Формат внутреннего представления данных. 12](#_Toc92708445)

[2.3. Описание программной реализации решения. 13](#_Toc92708446)

[3. Интерфейсы взаимодействия с пользователем 21](#_Toc92708447)

[3.1. Описание формата файлов. 21](#_Toc92708448)

[3.2. Обработка некорректных входных данных. 21](#_Toc92708449)

[4. Текст программы 24](#_Toc92708450)

[5. Результат работы программы 29](#_Toc92708451)

[Заключение 34](#_Toc92708452)

# 

# Введение

Целью курсовой работы является написание программы, решающей геометрическую задачу нахождения фигур и выбора из них той, которая содержит наибольшее количество точек. Входными данными являются координаты точек, выходными - координаты вершин найденной фигуры. Фигура представляет собой прямоугольник с полуокружностями на противоположных друг другу сторонах. Данную задачу удобно разделить на две подзадачи: нахождение фигуры и нахождение точек, ей принадлежащих. В разделе «Математическая постановка задачи» описывается математические подходы к решению данной задачи, а в разделе «Описание программной реализации» применение математического решения в прикладном, программном варианте. Описание взаимодействия программы с пользовательскими данными описано в разделе «Интерфейсы взаимодействия с пользователем», а в подразделе «Результат работы программы» содержится анализ выходных данных программы.

# Анализ задания

## Формулировка задания.

Дан набор координат точек на плоскости. Найти среди них точки, являющиеся вершинами фигуры, содержащей наибольшее количество заданных точек. Фигура представляет собой прямоугольник с полуокружностями на противоположных друг другу сторонах (эти стороны образуют диаметры полуокружностей). Полуокружности располагаются таким образом, что площадь всей фигуры больше площади исходного прямоугольника (рис. 1).

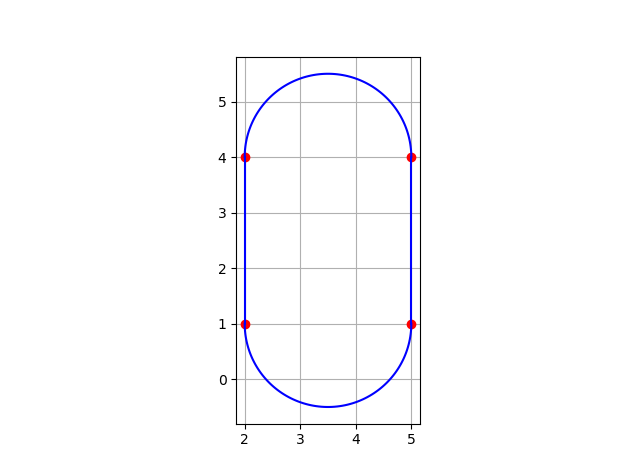


рис.

## Выявление неточностей.

Фигура, представленная в задании, образовывается в общем случае четырьмя точками. Но при этом невозможно однозначно интерпретировать её вид, имея только эти четыре точки (на рис.1 и рис.2 изображены две фигуры, имеющие различный вид, но одинаковые координаты точек).

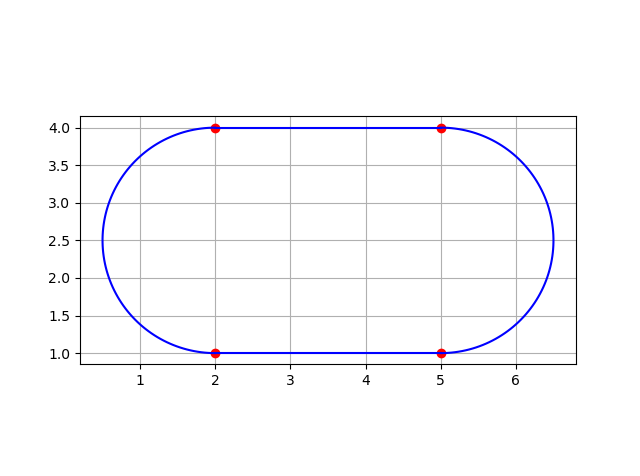


рис.

Поэтому для однозначной интерпретации фигуры будем брать ещё одну точку из набора данных, которая должна представлять собой центр одной из полуокружностей и лежать на середине одной из сторон прямоугольника (рис. 3).

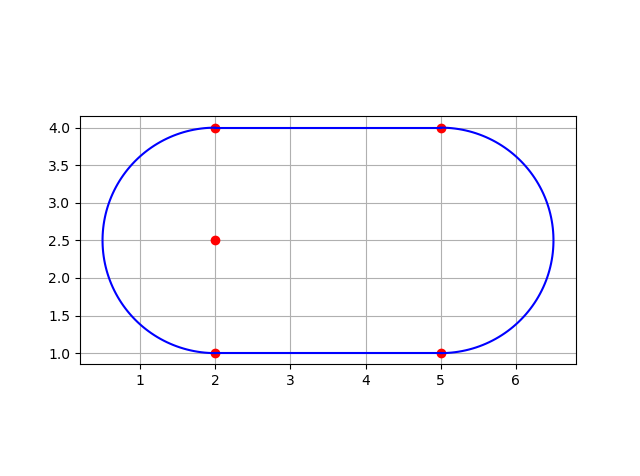


рис.

Ещё одним неоднозначным моментом является вопрос о включении в множество принадлежащих фигуре точек тех, которые лежат на периметре фигуры. В данной работе, такие точки учитываются, при подсчете.

Также данную фигуру можно рассматривать, как прямоугольник и две окружности (а не полуокружности). В таком случае подзадача проверки принадлежности точки фигуре сильно упрощается, а смысл задания в целом остается прежним. Тогда фигура с рис. 3 примет вид фигуры на рис. 4.

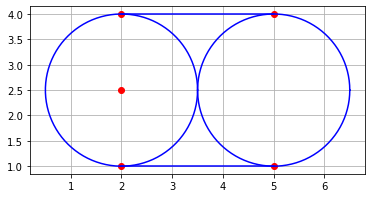


рис.

## Контрольный пример

Дан набор точек (рис. 5):

{ (6.0:8.0), (-4.0:-5.0), (-4.0:0.0), (5.0:5.0), (0.5:5.0), (5.0:-15.0), (5.0:-15.0), (5.0:-5.0), (9.0:1.0), (12.0:3.0), (-4.0:5.0), (-5.0:-5.0), (-5.0:-5.0), (1.0:0.0), (0.0:1.0), (5.0:7.0), (5.0:-3.0), (2.0:-8.0), (9.0:0.0), (9.0:0.0), (0.0:0.0), (2.0:2.0), (5.0:-5.0), (4.0:4.0), (9.0:-1.0), (9.0:-1.0), (9.0:-1.0), (9.0:-1.0)}

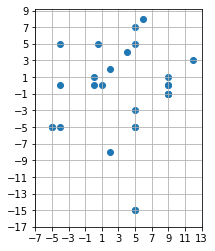


рис.

Для данного набора можно построить три фигуры, удовлетворяющие заданию.

Фигура 1 имеет вершины в точках {(5 : 5), (0.5 : 5), (-4 : 5), (-4 : -5), (5 : -5)} (рис. 6) и включает в себя 13 точек.

Фигура 2 имеет вершины в точках {(-4 : -5), (-4 : 0), (-4 : 5), (5 : 5), (5 : -5)} (рис. 7) и включает в себя 15 точек.

Фигура 3 имеет вершины в точках {(0 : 0), (2 : 2), (4 : 4), (9 : -1), (5 : -5)} (рис. 8) и включает в себя 8 точек.

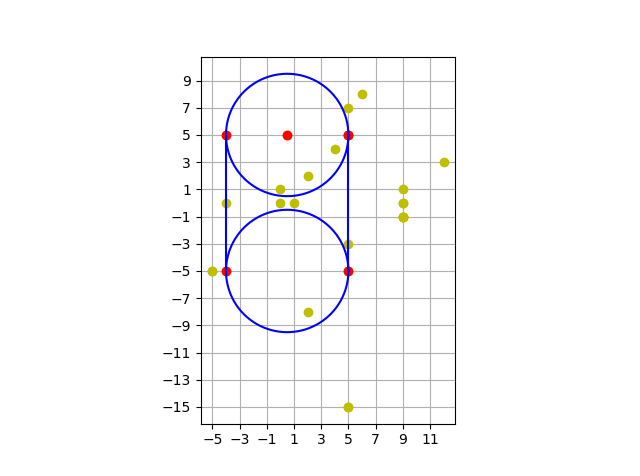


рис.

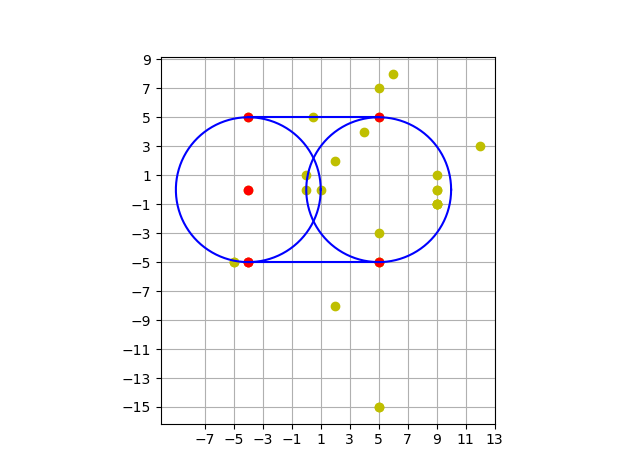


рис.

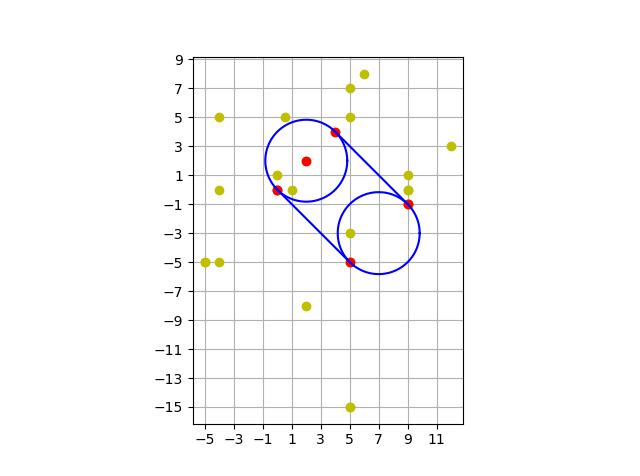


рис.

Наибольшее количество точек в себя включает фигура 2, поэтому вершины этой фигуры и будут ответом на задание.

# Описание решения задачи

## Математическая постановка решения.

*Дано:* xi,yi 0 <= i < n

*Найти:* координаты фигуры, включающей в себя наибольшее количество точек. Искомая фигура представляет собой прямоугольник, на середине одной из сторон которого лежит одна из точек. Эту фигуру задают пять точек.

*Решение:*

Множество из пяти координат точек (xi,yi) образует искомую фигуру, когда:

1. В нем есть три коллинеарные точки (лежащие на одной прямой).
2. Эти три точки образуют отрезок, разделенный одной из точек пополам (отрезок AC, где B принадлежит AC, и AB=BC).
3. Все пять точек являются различными (не совпадают).
4. Оставшиеся две точки (D, E) образуют отрезок, равный по длине AC, и CD = AE
5. CD перпендикулярен DE

Последний пункт можно проверить так:

Найдем координаты точки пересечения диагоналей предполагаемого прямоугольника S (пусть будут sx и sy).

sx = (ax+dx)/2 {1}

sy = (ay+dy)/2

где (ax, ay) и (dx,dy) это координаты точек A и B соответственно. Тогда, если **AS = CS = SD** можно построить окружность с радиусом AS. И точка С будет образовывать с точками A и D угол, опирающийся на диаметр окружности (рис. 9). По теореме Фалеса об угле, опирающемся на диаметр окружности угол DCA прямой. Учитывая это и пункт 4) можно утверждать, что ACDE – прямоугольник. А точка B лежит на середине стороны AC и точки A, B, C, D, E – образуют искомую фигуру.

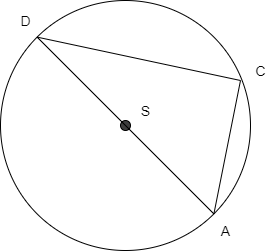


рис.

Чтобы понять, принадлежит точка фигуре, нужно проверить, принадлежит ли она хотя бы одной из окружностей фигуры или прямоугольнику.

Проверить принадлежность окружности можно по уравнению окружности:

где (x,y) – координаты точки. Для первой окружности r = AB, (x0,y0) = (bx,by). А для второй необходимо найди координаты центра второй окружности, аналогично {1} (fx,fy). Тогда r = DF, (x0,y0) = (fx,fy).

Проверить принадлежность некоторой точки К прямоугольнику можно выполнением равенства S1+S2+S3+S4 = S. Где S = площадь ACDE, S1 – площадь ACK, S2 – площадь CKD, S3 – площадь DKE, S4 – площадь EKA (рис. 10).

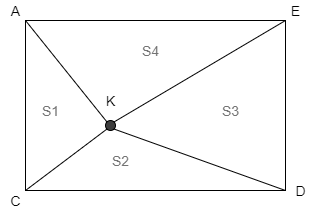


рис.

Зная координаты вершин, площадь треугольника можно найти по формуле Герона, предварительно вычислив длины его сторон.

Проверив таким образом каждую точку из данного набора, можно найти количество точек, принадлежащих фигуре. Повторив это для каждой найденной фигуры, найдем ту, для которой это количество будет максимальным. Координаты её вершины будут являться ответом на задание.

## Формат внутреннего представления данных.

Для хранения множества координат точек в программе используется динамически выделенный массив с названием points пользовательского типа coord. Структура coord включает в себя два публичных поля: double x, double y. Размер массива n определяется функцией get\_n (см. описание в подразделе 3.2) и фактически означает количество строк, в начале которых через пробел находятся два числа. При этом, поскольку считывание элементов не происходит, невозможно найти дубликаты вершин. Это означает, что фактический размер используемой памяти может быть меньше выделенного. На окончание массива будет указывать (points + n).

Для удобства работы с найденными вершинами фигур вводится ещё один пользовательский тип – fig. Он содержит публичные поля:

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Значение по умолчанию | Назначение |
| first | coord | {0,0} | Координаты первой точки (А в мат. постановке задачи). |
| second | coord | {0,0} | Координаты второй точки (B) – данного центра одной из окружностей. |
| third | coord | {0,0} | Координаты третьей точки (С). |
| four | coord | - | Координаты четвертой точки (D). |
| five | coord | - | Координаты пятой точки (E). |
| center | coord | - | Координаты центра окружности (точка пересечения диагоналей прямоугольника) (S). |
| sub\_center | coord | - | Координаты центра второй окружности (F). |
| inner\_points\_count | int | 5 | Количество точек, принадлежащих фигуре. |

**Про значения по умолчанию**. Координаты первой и второй точек фигуры имеют одинаковые значения по умолчанию, это используется для проверки ненахождения фигуры с помощью функции is\_empty. Поскольку точно известно, что ни одна вершина фигуры не может совпадать с другой, равенство первых двух точек будет указывать на то, что значение по умолчанию не было изменено, а значит, фигура не найдена. Счетчик же inner\_points\_count равен по умолчанию пяти, так как для любой фигуры минимум пять точек, её образующих, будут ей принадлежать. (при подсчете не учитываются точки center и sub\_center, так как они не входят в исходный набор точек, а являются вспомогательными.

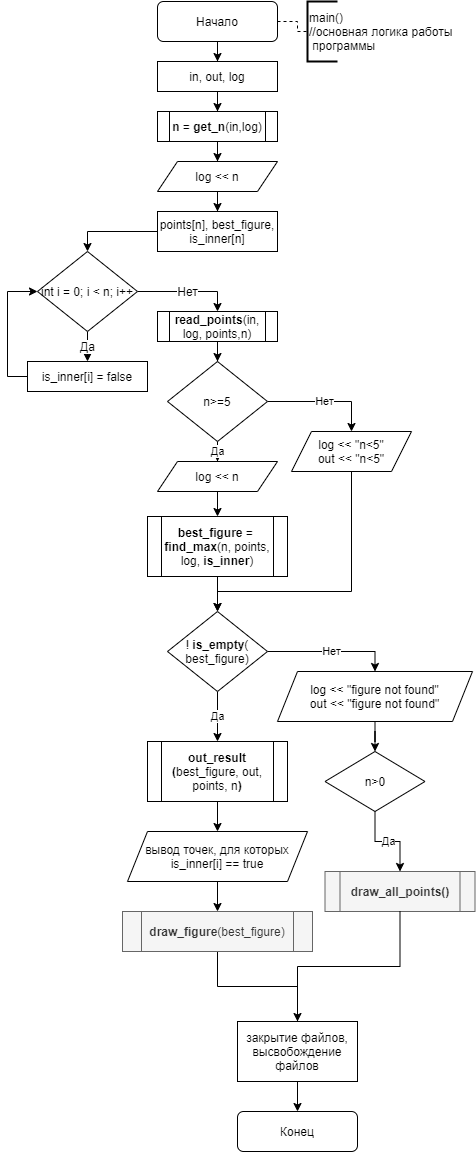
Также для определения набора точек, принадлежащих фигуре, используется динамический массив is\_inner типа bool размерностью n. Значение для i ячейки этого массива определяет, принадлежит ли i точка из массива points фигуре.

## Описание программной реализации решения.

Далее приведены описания и блок-схемы всех функций программы. Начнем описание программной реализации с функции **main.**

В таблице представлены все локальные переменных функции main.

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| in | fstream | Файл входных данных |
| out | fstream | Файл выходных данных |
| log | fstream | Файл протокола |
| n | int | Количество значащих элементов в массиве points и is\_inner |
| points | coord\* | Указатель на начало динамического массива всех считанных точек |
| best\_figure | fig | Данные фигуры, отвечающей условию задачи |
| is\_inner | bool\* | Указатель на начало динамического массива-маски для массива points. Отражает, входит ли каждая конкретная точка в фигуру best\_figure. |

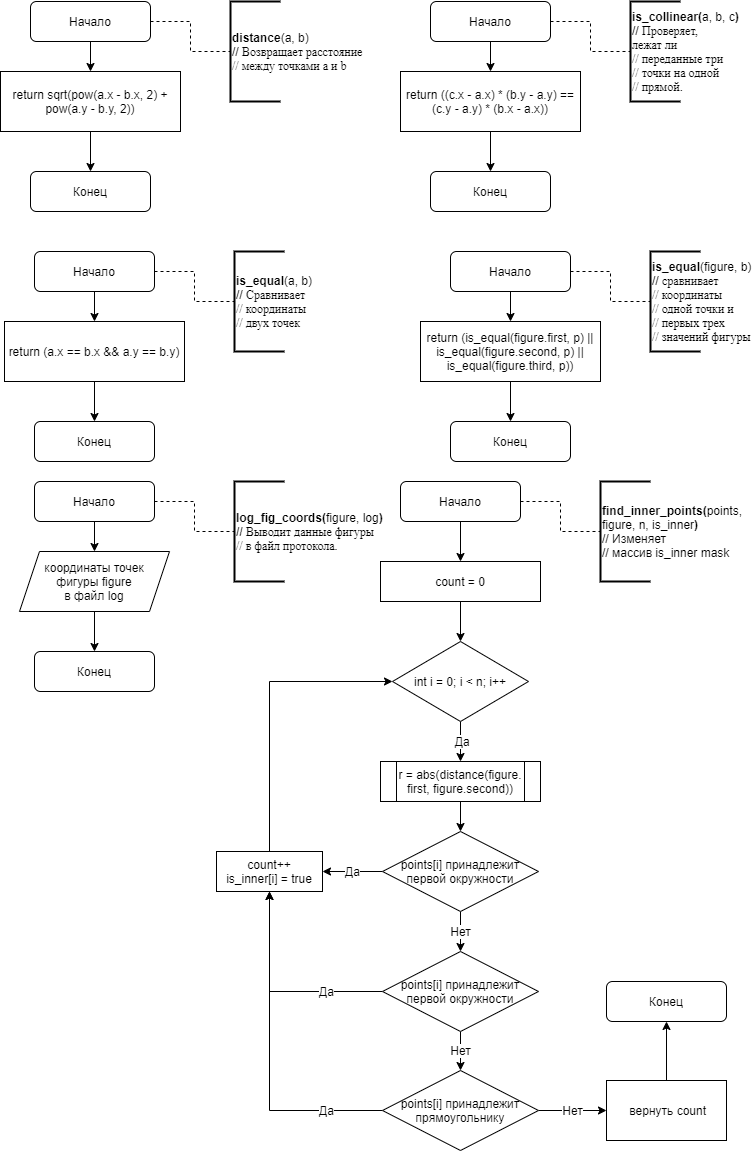


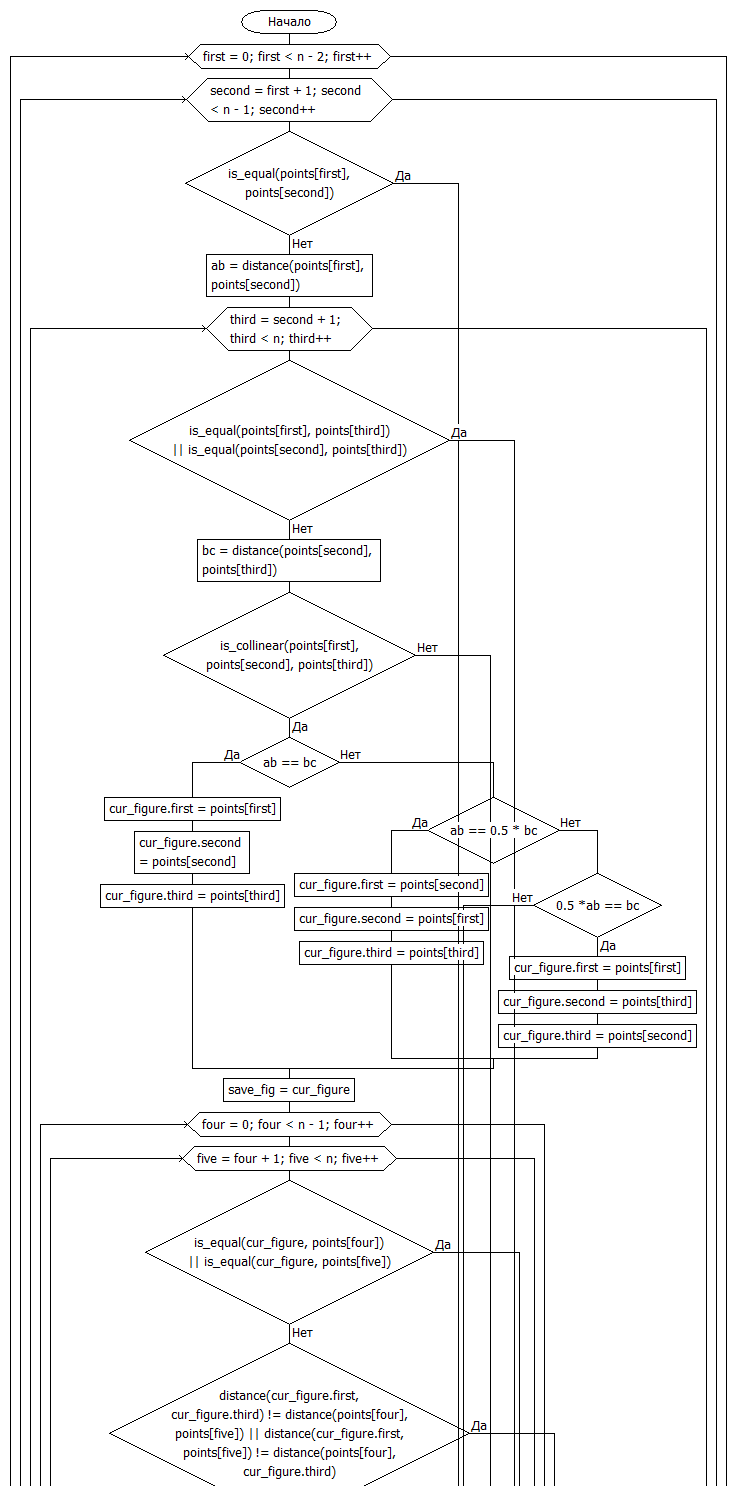
Далее следует описание функций, в порядке их вызова в функции main, кроме функций get\_n и read\_point и всех, вызываемых ими (их описание см. в подразделе 3.2)

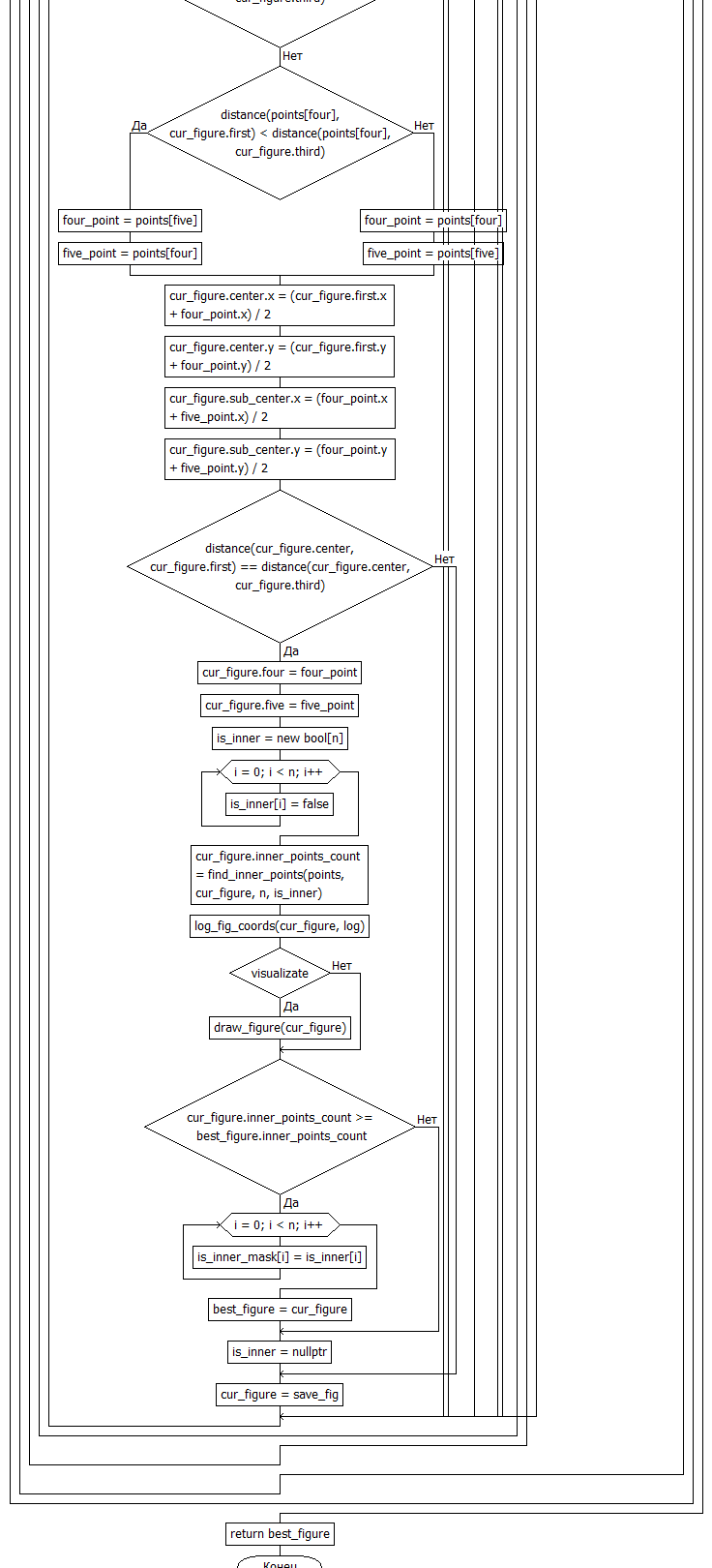
Функция **find\_max** вызывает функции distance, is\_equal, is\_collinear, find\_inner\_points, log\_fig\_coords. Их описание в см. таблице ниже.

Функция find\_max является одной из ключевых функций программы. Кратко опишем её принцип работы. Сначала с помощью вложенных циклов третьей степени происходит перебор всех комбинаций трех точек с поиском среди них коллинеарных (лежащих на одной прямой) причем таких, которые образуют отрезок, поделенный одной из точек пополам. Эта точка будет являться центром окружности (точкой B из математической постановки). Потом к этому набору точек с помощью ещё двух вложенных циклов ищутся такие, которые образуют с уже имеющимися тремя точками прямоугольник. Далее происходит поиск точек, входящих в данную фигуру с помощью функции find\_inner\_points. И по найденному количеству точек происходит сравнение с найденным предыдущим максимум и в соответствии с этим, этот локальный максимум обновляется.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Аргументы | Возвращаемое значение | Внешние эффекты | Назначение |
| Find\_max | int n, coord\* points, std::fstream& log,  bool\* is\_inner\_mask | Наилучшая найденная фигура типа fig. | Вывод значений в файл протокола, изменяет массив is\_inner mask | Находит среди точек points фигуры, выводит их в файл протокола, считает количество внутренних точек, выбирает фигуру с наибольшим количеством внутренних точек и возвращает её. |
| is\_equal | coord& a, coord& b | bool | - | Сравнивает координаты двух точек или сравнивает координаты одной точки и первых трех значений фигуры. |
| fig& figure, coord& p |
| is\_collinear | coord& a, coord& b, coord& c | bool | - | Проверяет, лежат ли переданные три точки на одной прямой. |
| log\_fig coords | fig figure, fstream& log | - | Вывод в файл протокола | Выводит данные фигуры в файл протокола. |
| find inner points | coord\* points, fig & figure, int n, bool \*is\_inner | int | Изменяет массив is\_inner mask |  |
| distance | coord a, coord b | double | - | Возвращает расстояние между точками а и b |

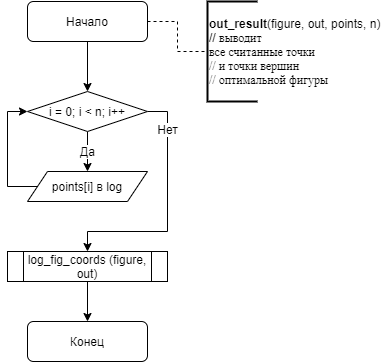


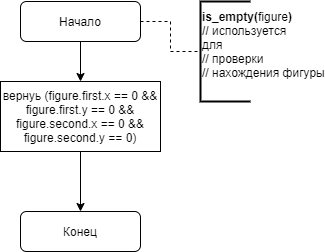




Далее следует описание оставшихся вспомогательных функций is\_empty, out\_result. Функция is\_empty используется для проверки нахождения фигуры. Если две точки фигуры совпадают – то она не была изменена, а значит, не была найдена функцией find\_max. Функция out\_result выводит все считанные точки и точки вершин оптимальной фигуры.

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Аргументы | Возвращаемое значение | Внешние эффекты |
| out\_result | const fig& figure, std::fstream &out, coord\*points, int n | - | Вывод в выходной файл out |
| is\_empty | const fig& figure | bool | - |





# Интерфейсы взаимодействия с пользователем

## Описание формата файлов.

Выходной файл *output.txt* содержит:

1. Все считанные точки
2. Координаты искомой фигуры
3. Координаты точек, принадлежащих этой фигуре.

Файл протокола *log.txt* содержит:

1. Сообщения о некорректных строках во входном файле
2. Количество строк, значения которых будут считаны
3. Список всех считанных координат точек
4. Сообщения о найденных дубликатах
5. Реально считанное количество точек n
6. Координаты всех найденных фигур и количество принадлежащих им точек.

Файл входных данных *input.txt* содержит набор координат xi, yi (по два числа в каждой строке).

## Обработка некорректных входных данных.

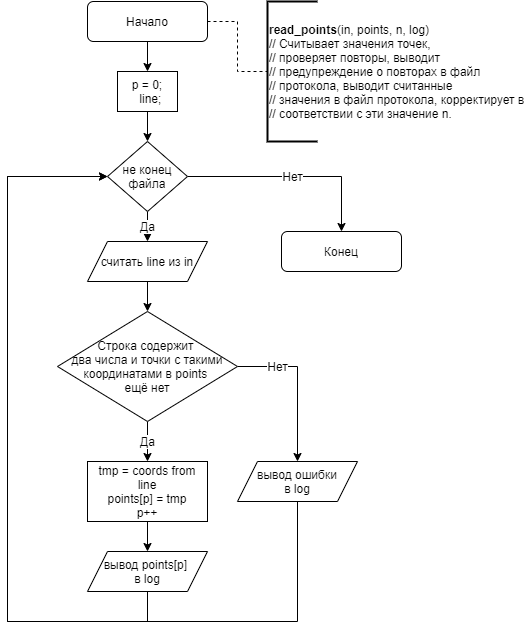
Обработкой считывания данных в программе занимаются 4 функции check\_line, get\_n, is\_in, read\_points.

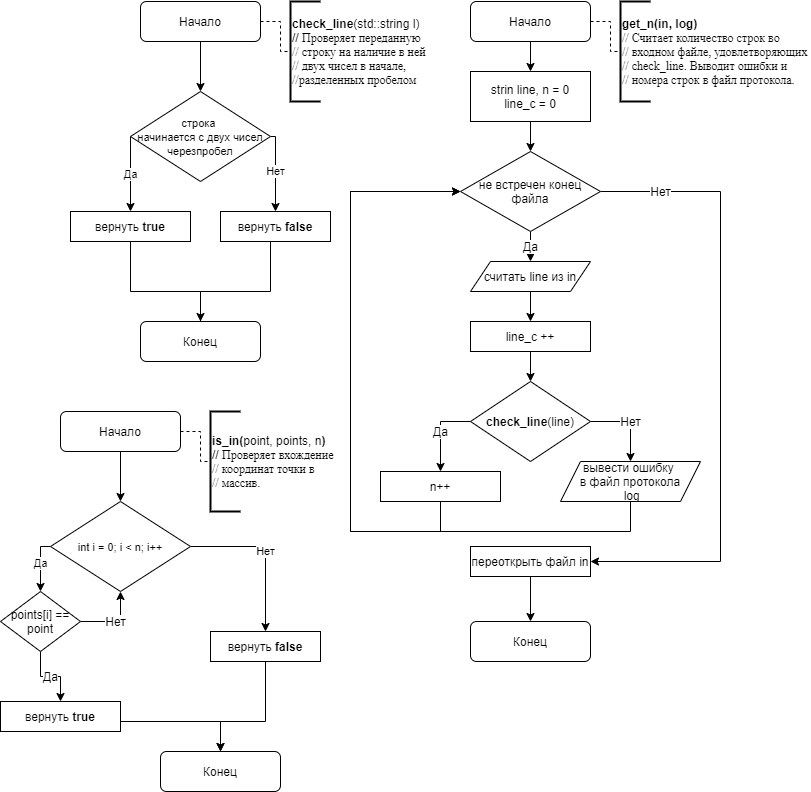
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Назначение | Параметры | Внешние эффекты | Возвращаемое значение |
| check\_line | Проверяет переданную строку на наличие в ней двух чисел в начале, разделенных пробелом | std::string l | - | Логическое значение (bool) |
| get\_n | Считает количество строк во входном файле, удовлетворяющих check\_line. Выводит ошибки и номера строк в файл протокола. | std::fstream& in, std::fstream& log | Возможен вывод в файл протокола | Количество строк (int) |
| read\_points | Считывает значения точек, проверяет повторы, выводит предупреждение о повторах в файл протокола, выводит считанные значения в файл протокола, корректирует в соответствии с эти значение n. | std::fstream& in, coord\* points, int &n, std::fstream &log | Изменяет n, изменяет значения массива points, выводит информацию в файл протокола, выводит информацию во вспомогательный файл. | - |
| is\_in | Проверяет вхождение координат точки в массив. | coord point, coord\* points, const int n | - | Логическое значение (bool) |

Данная логика работы позволяет обрабатывать некорректный ввод, например:

1. Пустые строки входного файла игнорируются.
2. Строки, не содержащие в начале два числа через пробел, игнорируются.
3. Строки, содержащие два числа через пробел и ненужную информацию после этого, обрабатываются. Числа считываются, остальное отбрасывается.

Реальное количество считанных данных может быть меньше, выделенного изначально размера массива из-за встречающихся дубликатов, при этом размерность массива n корректируется и не позволяет выйти за пределы заполненной части массива.





# Текст программы

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

struct coord {

double x;

double y;};

struct fig {

coord first = { 0,0 };

coord second = { 0,0 };

coord third = { 0,0 };

coord four;

coord five;

coord center;

coord sub\_center;

int inner\_points\_count = 5;};

bool check\_line(std::string l) {

bool has\_first\_n = false, has\_one\_whitespace = false, has\_second\_digit = false,

has\_two\_whitespaces = true;

for (int ch = 0; ch < l.size(); ch++) {

if ((isdigit(l[ch]) || l[ch] == '-' || l[ch] == '.') && !has\_first\_n) {

has\_first\_n = true;

}

else if ((isdigit(l[ch]) || l[ch] == '-' || l[ch] == '.') && has\_one\_whitespace && !has\_second\_digit) { has\_second\_digit = true; }

else if (l[ch] == ' ' && has\_first\_n && !has\_one\_whitespace) {

has\_one\_whitespace = true; }

else if (l[ch] == ' ' && has\_first\_n && has\_one\_whitespace && !has\_second\_digit) {

has\_one\_whitespace = false; }

else if (has\_first\_n && has\_one\_whitespace && has\_second\_digit && (!isdigit(l[ch]) && l[ch] != '-' && l[ch] != '.')) {

if (l[ch] == ' ' || l[ch] == '\n') {

has\_two\_whitespaces = true; }

else {

has\_two\_whitespaces = false;

break; } } }

return (has\_first\_n && has\_one\_whitespace && has\_second\_digit && has\_two\_whitespaces);}

int get\_n(std::fstream& in, std::fstream& log) {

std::string line = "aa";

int n = 0;

int line\_c = 0;

while (!in.eof()) {

std::getline(in, line);

line\_c++;

if (check\_line(line))

n++;

else

log << "format error in input file in " + std::to\_string(line\_c) + " line\n"; }

in.close();

in.open("../../input.txt");

return n;}

bool is\_in(coord point, coord\* points, const int n) {

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

if (point.x == points[i].x && point.y == points[i].y) {

return true; } }

return false;}

void read\_points(std::fstream& in, coord\* points, int &n, std::fstream &log) {

int p = 0;

std::string line;

log << "read points: \n";

while (!in.eof()) {

std::getline(in, line);

if (check\_line(line)) {

int first\_space\_ptr = -1, second\_space\_ptr = -1;

for (int i = 0; i < line.size(); i++) {

if (line[i] == ' ') {

if (first\_space\_ptr == -1)

first\_space\_ptr = i;

else if (second\_space\_ptr == -1) {

second\_space\_ptr = i;

break; } }

else if (line[i] == '\n') {

second\_space\_ptr = i;

break; } }

coord tmp{ stod(line.substr(0, first\_space\_ptr)) ,stod(line.substr(first\_space\_ptr, second\_space\_ptr)) };

if (!is\_in(tmp, points, n)) {

points[p].x = tmp.x;

points[p].y = tmp.y;

log << points[p].x << " " << points[p].y << '\n';

p++; }

else {

log << "found doubles " << std::to\_string(tmp.x) << " : " << std::to\_string(tmp.y)<<'\n';

n--; } } }

std::fstream f("../../beauty\_points.txt", std::fstream::out | std::fstream::trunc);

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

f << points[i].x << " " << points[i].y << '\n'; }}

double distance(const coord a, const coord b) {

double d = sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));

return d;}

bool is\_equal(const coord& a, const coord& b) {

return (a.x == b.x && a.y == b.y);}

bool is\_equal(const fig& figure, const coord& p) {

return (is\_equal(figure.first, p) || is\_equal(figure.second, p) || is\_equal(figure.third, p));}

bool is\_collinear(const coord& a, const coord& b, const coord& c) {

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

void draw\_all\_points() {

system("python ../../visualisation/draw\_points.py");}

void draw\_figure(const fig& cur\_figure) {

std::string args = std::to\_string(cur\_figure.first.x) + ' ' + std::to\_string(cur\_figure.first.y) + ' ' +

std::to\_string(cur\_figure.second.x) + ' ' + std::to\_string(cur\_figure.second.y) + ' ' +

std::to\_string(cur\_figure.third.x) + ' ' + std::to\_string(cur\_figure.third.y) + ' ' +

std::to\_string(cur\_figure.four.x) + ' ' + std::to\_string(cur\_figure.four.y) + ' ' +

std::to\_string(cur\_figure.five.x) + ' ' + std::to\_string(cur\_figure.five.y);

std::string command = "python ../../visualisation/draw\_fig\_with\_points.py " + args;

system(command.c\_str());}

void print\_fig\_coords(fig figure) {

std::cout << "1: " << figure.first.x << " : " << figure.first.y << "\n" <<

"2: " << figure.second.x << " : " << figure.second.y << "\n" <<

"3: " << figure.third.x << " : " << figure.third.y << "\n" <<

"4: " << figure.four.x << " : " << figure.four.y << "\n" <<

"5: " << figure.five.x << " : " << figure.five.y << "\n" <<

"sub\_c: " << figure.sub\_center.x << " : " << figure.sub\_center.y << "\n\n";}

void log\_fig\_coords(fig figure, std::fstream& log) {

log<< "figure has "<<figure.inner\_points\_count<< " inner points\n" << "1: " << figure.first.x << " : " << figure.first.y << "\n" <<

"2: " << figure.second.x << " : " << figure.second.y << "\n" <<

"3: " << figure.third.x << " : " << figure.third.y << "\n" <<

"4: " << figure.four.x << " : " << figure.four.y << "\n" <<

"5: " << figure.five.x << " : " << figure.five.y << "\n" <<

"sub\_c: " << figure.sub\_center.x << " : " << figure.sub\_center.y << "\n\n";}

unsigned find\_inner\_points(const coord\* points, const fig & figure, int n, bool \*is\_inner) {

unsigned count = 0;

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

double r = abs(distance(figure.first, figure.second));

if (sqrt(pow(points[i].x - figure.second.x, 2) + pow(points[i].y - figure.second.y, 2)) <= r) {

count++;

is\_inner[i] = true; }

else if (sqrt(pow(points[i].x - figure.sub\_center.x, 2) + pow(points[i].y - figure.sub\_center.y, 2)) <= r) {

count++;

is\_inner[i] = true; } else {

double a = distance(figure.first, figure.third),

b = distance(figure.first, figure.five);

double S = a \* b;

double d1 = distance(points[i], figure.first),

d2 = distance(points[i], figure.third),

d3 = distance(points[i], figure.four),

d4 = distance(points[i], figure.five);

double p1 = (d1 + a + d2) / 2, p2 = (d2 + b + d3) / 2, p3 = (d3 + a + d4) / 2,

p4 = (d4 + d1 + b) / 2;

double s1 = sqrt(p1 \* (p1 - a) \* (p1 - d1) \* (p1 - d2)),

s2 = sqrt(p2 \* (p2 - b) \* (p2 - d3) \* (p2 - d2)),

s3 = sqrt(p3 \* (p3 - a) \* (p3 - d4) \* (p3 - d3)),

s4 = sqrt(p4 \* (p4 - b) \* (p4 - d1) \* (p4 - d4));

if (s1 + s2 + s3 + s4 == S) {

count++;

is\_inner[i] = true; } } }

return count;}

void log\_not\_found(const fig& figure, std::fstream& log) {

log<<"points "<< figure.first.x << ":" << figure.first.y <<" "<< figure.second.x << ":" <<

figure.second.y << " " << figure.third.x << ":" << figure.third.y << " " <<

figure.four.x << ":" << figure.four.y << " " << figure.five.x << ":" << figure.five.y << " not a figure\n";}

fig find\_max(int n, coord\* points, std::fstream& log, bool\*is\_inner\_mask,bool visualizate=false) {

fig best\_figure;

for (int first = 0; first < n - 2; first++) {

for (int second = first + 1; second < n - 1; second++) {

if (is\_equal(points[first], points[second]))

continue;

double ab = distance(points[first], points[second]);

for (int third = second + 1; third < n; third++) {

if (is\_equal(points[first], points[third]) || is\_equal(points[second], points[third]))

continue;

double bc = distance(points[second], points[third]);

fig cur\_figure;

if (is\_collinear(points[first], points[second], points[third])) {

if (ab == bc) {

cur\_figure.first = points[first];

cur\_figure.second = points[second];

cur\_figure.third = points[third]; }

else if (ab == 0.5 \* bc) {

cur\_figure.first = points[second];

cur\_figure.second = points[first];

cur\_figure.third = points[third]; }

else if (0.5 \* ab == bc) {

cur\_figure.first = points[first];

cur\_figure.second = points[third];

cur\_figure.third = points[second]; }

else continue; }

else continue;

fig save\_fig = cur\_figure;

for (int four = 0; four < n - 1; four++) {

for (int five = four + 1; five < n; five++) {

if (is\_equal(cur\_figure, points[four]) || is\_equal(cur\_figure, points[five]))

continue;

coord four\_point, five\_point;

if (distance(cur\_figure.first, cur\_figure.third) != distance(points[four], points[five]) ||

distance(cur\_figure.first, points[five]) != distance(points[four], cur\_figure.third)) {

cur\_figure.four = points[five];

cur\_figure.five = points[four];

log\_not\_found(cur\_figure, log);

continue; }

if (distance(points[four], cur\_figure.first) < distance(points[four], cur\_figure.third)) {

four\_point = points[five];

five\_point = points[four]; }

else {

four\_point = points[four];

five\_point = points[five]; }

cur\_figure.center.x = (cur\_figure.first.x + four\_point.x) / 2;

cur\_figure.center.y = (cur\_figure.first.y + four\_point.y) / 2;

cur\_figure.sub\_center.x = (four\_point.x + five\_point.x) / 2;

cur\_figure.sub\_center.y = (four\_point.y + five\_point.y) / 2;

if (distance(cur\_figure.center, cur\_figure.first) == distance(cur\_figure.center, cur\_figure.third )) {

cur\_figure.four = four\_point;

cur\_figure.five = five\_point;

bool\* is\_inner = new bool[n];

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

is\_inner[i] = false; }

cur\_figure.inner\_points\_count = find\_inner\_points(points, cur\_figure, n, is\_inner);

log\_fig\_coords(cur\_figure, log);

if (visualizate)

draw\_figure(cur\_figure);

if (cur\_figure.inner\_points\_count >= best\_figure.inner\_points\_count) {

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

is\_inner\_mask[i] = is\_inner[i]; }

best\_figure = cur\_figure; }

delete[] is\_inner;

is\_inner = nullptr; }

else {

log\_not\_found(cur\_figure, log); }

cur\_figure = save\_fig; } } } } }

return best\_figure;}

bool is\_empty(const fig& figure) {

return (figure.first.x == 0 && figure.first.y == 0 && figure.second.x == 0 && figure.second.y == 0);}

void out\_result(const fig& figure, std::fstream &out, coord\*points, int n) {

out << "read points: \n";

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

out << points[i].x << " : " << points[i].y << '\n'; }

log\_fig\_coords(figure, out);}

int main(){

std::fstream in("../../input.txt");

std::fstream out("../../output.txt", std::fstream::out | std::fstream::trunc);

std::fstream log("../../log.txt", std::fstream::out | std::fstream::trunc);

int n = get\_n(in,log);

log << "number of matching strings = " << n << '\n';

bool VISUALIZATE = false;

coord\* points = new coord[n];

read\_points(in, points, n, log);

fig best\_figure;

bool\* is\_inner = new bool[n];

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

is\_inner[i] = false;

if (n >= 5) {

log << "n = " << n << '\n';

best\_figure = find\_max(n, points, log, is\_inner, VISUALIZATE); }

else {

log << "points count (n) < 5: " << n << '\n';

out << "points count (n) < 5: " << n << '\n';}

if (!is\_empty(best\_figure)) {

out\_result(best\_figure, out, points,n);

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

if (is\_inner[i]) {

out << points[i].x << " : " << points[i].y << "\n"; } }

draw\_figure(best\_figure); }

else {

log << "figure not found" <<'\n';

out << "figure not found" <<'\n';

if (n > 0) {

draw\_all\_points(); } }

in.close(); out.close(); log.close();

delete[] points; points = nullptr;

delete[] is\_inner; is\_inner = nullptr;}

# Результат работы программы

По набору точек (содержимое входного файла):

6 8

-4 -5

-4 0

5 5

0.5 5

5 -15

5 -5

9 1

12 3

-4 5

-5 -5

1 0

0 1

5 7

5 -3

2 -8

9 0

0 0

2 2

5 -5

4 4

9 -1

Программа формирует файл протокола вида (большая часть строк пропущена (…) и содержит сообщение о том, что определенный набор точек не составляет фигуру):

format error in input file in 7 line

format error in input file in 13 line

format error in input file in 20 line

number of matching strings = 22

read points:

6 8

-4 -5

-4 0

5 5

0.5 5

5 -15

5 -5

9 1

12 3

-4 5

-5 -5

1 0

0 1

5 7

5 -3

2 -8

9 0

0 0

2 2

found doubles 5.000000 : -5.000000

4 4

9 -1

n = 21

points -4:-5 -4:0 -4:5 5:5 6:8 not a figure

(…)

points -4:-5 -4:0 -4:5 5:-15 5:5 not a figure

figure has 15 inner points

1: -4 : -5

2: -4 : 0

3: -4 : 5

4: 5 : 5

5: 5 : -5

sub\_c: 5 : 0

points -4:-5 -4:0 -4:5 9:1 5:5 not a figure

(…)

points 5:5 0.5:5 -4:5 5:-15 -4:-5 not a figure

figure has 13 inner points

1: 5 : 5

2: 0.5 : 5

3: -4 : 5

4: -4 : -5

5: 5 : -5

sub\_c: 0.5 : -5

points 5:5 0.5:5 -4:5 9:1 -4:-5 not a figure

(…)

points 0:0 2:2 4:4 9:0 5:-5 not a figure

figure has 8 inner points

1: 0 : 0

2: 2 : 2

3: 4 : 4

4: 9 : -1

5: 5 : -5

sub\_c: 7 : -3

points 0:0 2:2 4:4 12:3 9:1 not a figure

(…)

points 0:0 2:2 4:4 9:-1 9:0 not a figure

Сначала программа выводит сообщения об ошибках и номер строки, где произошла ошибка. Строка «number of matching strings = 22» означает, что функция get\_n нашла 8 строк, которые начинаются с двух чисел через пробел. Далее идет список всех считанных точек, с указанием дубликатов (рис. 11). Потом указывается фактическое количество точек («n = 21»). Далее идет список всех перебираемых фигур, с указанием является ли данное сочетание точек фигурой или нет. В первом случае так же указывается количество точек, которые входят в эту фигуру. Первая найденная фигура содержит 15 точек (рис. 12), вторая содержит 13 точек (рис. 13) и третья – 8 точек (рис. 14).

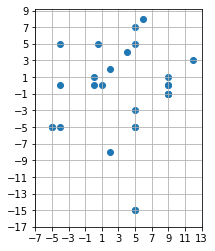


рис.

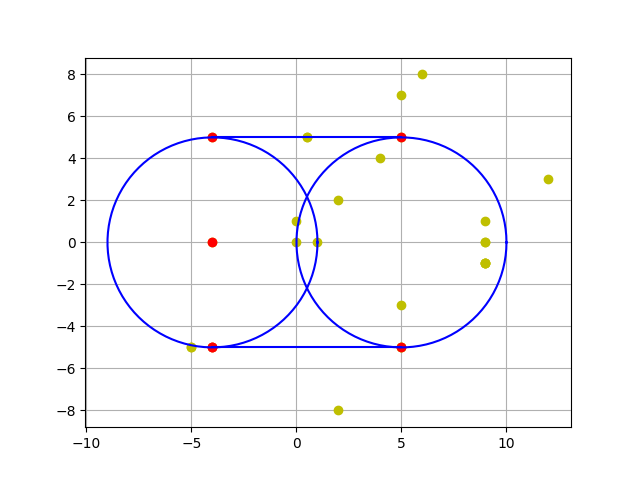


рис.

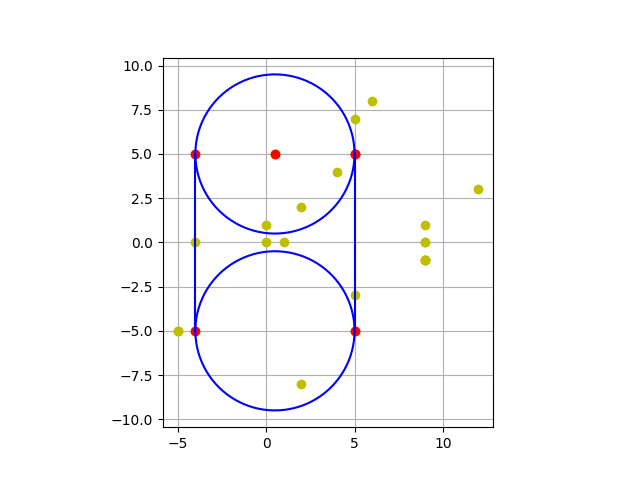


рис.

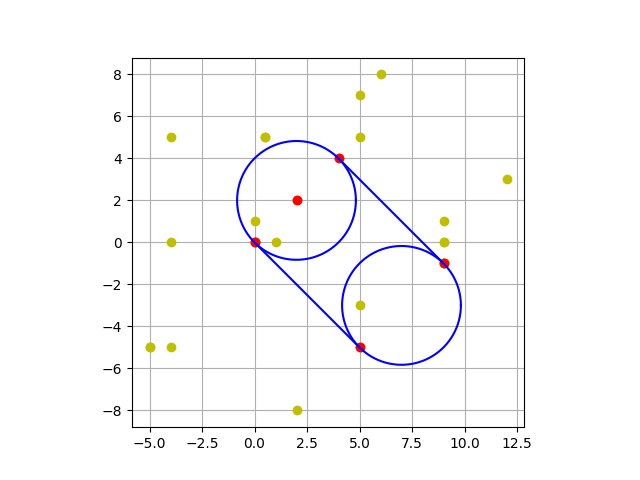


рис.

В выходной файл идет информация о считанных точках, фактическом значении n, оптимальная найденная фигура (рис. 12), количество точек, принадлежащих ей, её координаты и координаты точек, которые ей принадлежат:

read points:

6 : 8

-4 : -5

-4 : 0

5 : 5

0.5 : 5

5 : -15

5 : -5

9 : 1

12 : 3

-4 : 5

-5 : -5

1 : 0

0 : 1

5 : 7

5 : -3

2 : -8

9 : 0

0 : 0

2 : 2

4 : 4

9 : -1

figure has 15 inner points

1: -4 : -5

2: -4 : 0

3: -4 : 5

4: 5 : 5

5: 5 : -5

sub\_c: 5 : 0

-4 : -5

-4 : 0

5 : 5

0.5 : 5

5 : -5

9 : 1

-4 : 5

1 : 0

0 : 1

5 : -3

9 : 0

0 : 0

2 : 2

4 : 4

9 : -1

# Заключение

В ходе выполнения курсовой работы была написана программа, решающая поставленную задачу. Программа состоит из множества функций, основной структурой данных является динамический массив, так же были задействованы структуры, как способ удобного хранения координат точек и как способ хранения вершин фигуры. Математическая часть курсовой работы базируется на формуле окружности в общем виде, теореме Фалеса, формуле Герона. Для перебора комбинаций пяти точек пришлось обратиться к большой вложенности циклов (пятой степени). Для оптимизации работы такой вложенности не происходит перебор всех комбинаций, а сначала идет поиск отрезка поделенного пополам, а далее ищутся две недостающие точки. Программа корректно обрабатывает различные ситуации, такие как нехватка данных, ненахождение фигуры, пустые строки и лишние символы в исходных данных.