**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

Курсовая РАБОТА

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1302 |  | Новиков Г.В. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2021

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Новиков Г.В. | | |
| Группа 1302 | | |
|  | | |
| Исходные данные:  Рассматривая точки пересечения прямых, проходящих через любую пару из заданных N точек на плоскости, определить принадлежность кругу радиуса R: в) с центром в точках из N заданных максимального количества точек пересечения. | | |
| Содержание пояснительной записки:  Исходная формулировка, особенности задания, контрольный пример, формальная постановка задания, формат хранения данных, ограничения, макет ввода и вывода, средства обеспечения ввода и вывода, алгоритм решения, программа, результаты, выводы, список используемых материалов. | | |
|  | | |
| Дата выдачи задания: 23.11.2021 | | |
| Дата сдачи: 17.12.2021 | | |
| Дата защиты: 17.12.2021 | | |
| Студент |  | Новиков Г.В. |
| Преподаватель |  | Калмычков В.А. |

**Аннотация**

В курсовую работу входит программа, которая обрабатывает и корректирует двумерный массив из файла и выполняет индивидуальное задание. Программа выполняется по средствам некоторых математических формул, таких как: построение прямых по координатам заданных точек, нахождение точек пересечения прямых, проверка на принадлежность точек кругу радиуса R с заданным центром. В результате программа выводит в файл откорректированный двумерный массив, основные действия программы и ответ на заданное индивидуальное задание.

**Summary**

The course work includes a program that processes and corrects a two-dimensional array from a file and performs an individual task. The program is executed by means of some mathematical formulas, such as: plotting straight lines from the coordinates of given points, finding lines intersection points, points belonging to circle with radius R and given center check. As a result, the program outputs to the file the corrected two-dimensional array, the main actions of the program and the answer to the given individual task.

Оглавление

[Исходная формулировка 5](#_Toc90620857)

[Особенности задания 5](#_Toc90620858)

[Контрольный пример 5](#_Toc90620859)

[Формальная постановка задания 5](#_Toc90620860)

[Формат хранения данных 5](#_Toc90620861)

[Ограничения, обусловленные выполнением на компьютере 6](#_Toc90620862)

[Макет ввода и вывода 6](#_Toc90620863)

[Средства обеспечения ввода и вывода 7](#_Toc90620864)

[Алгоритм решения 8](#_Toc90620865)

[Программа 11](#_Toc90620866)

[Результаты работы программы 14](#_Toc90620867)

[Выводы о проделанной работе 15](#_Toc90620868)

[Список использованных источников 16](#_Toc90620869)

# **Исходная формулировка**

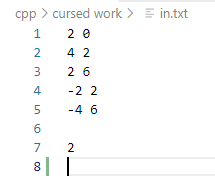
Рассматривая точки пересечения прямых, проходящих через любую пару из заданных N точек на плоскости, определить принадлежность кругу радиуса R: в) с центром в точках из N заданных максимального количества точек пересечения.

# **Особенности задания**

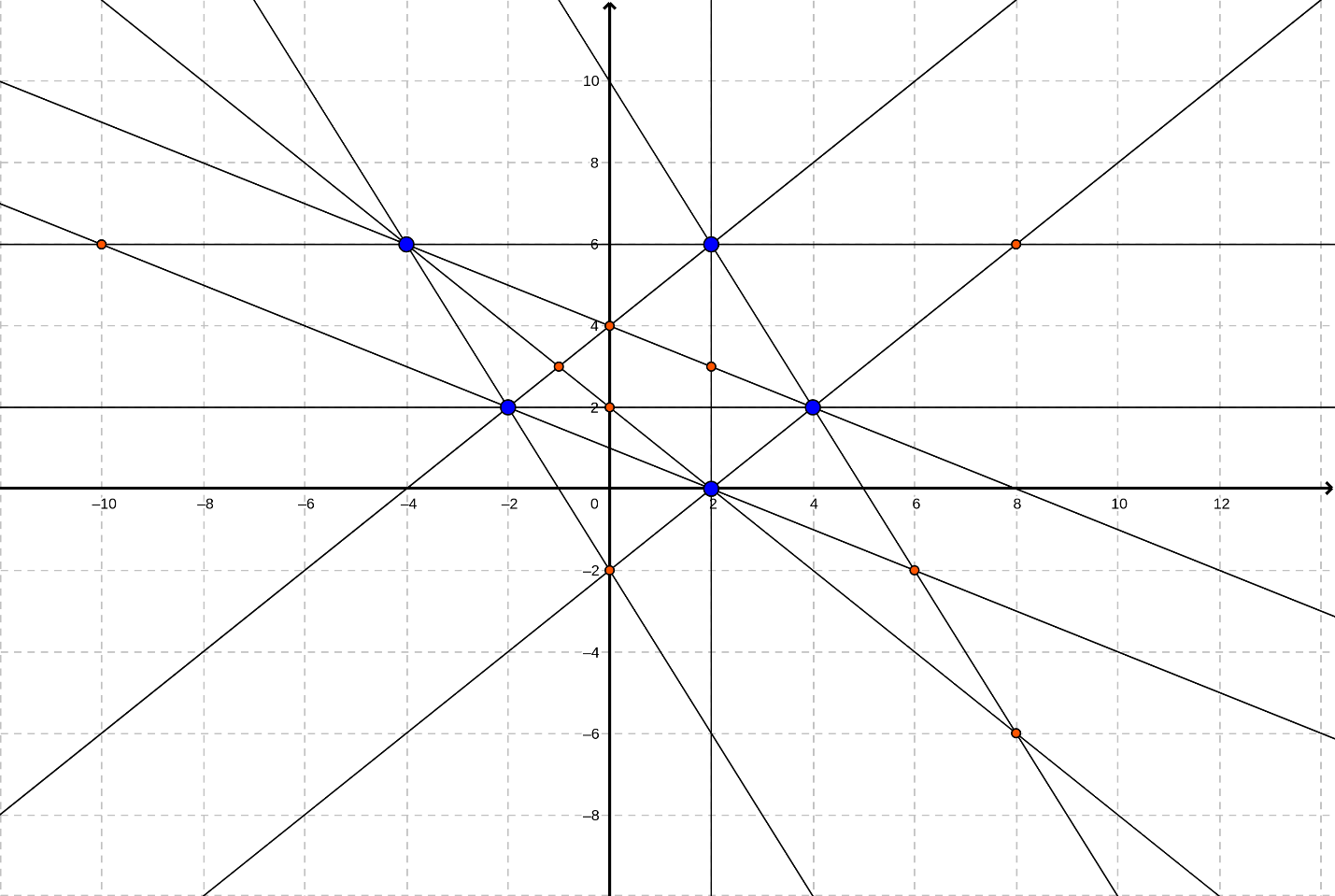
Обработка двумерного массива с использованием функций и с организацией файлового ввода/вывода.

# **Контрольный пример**

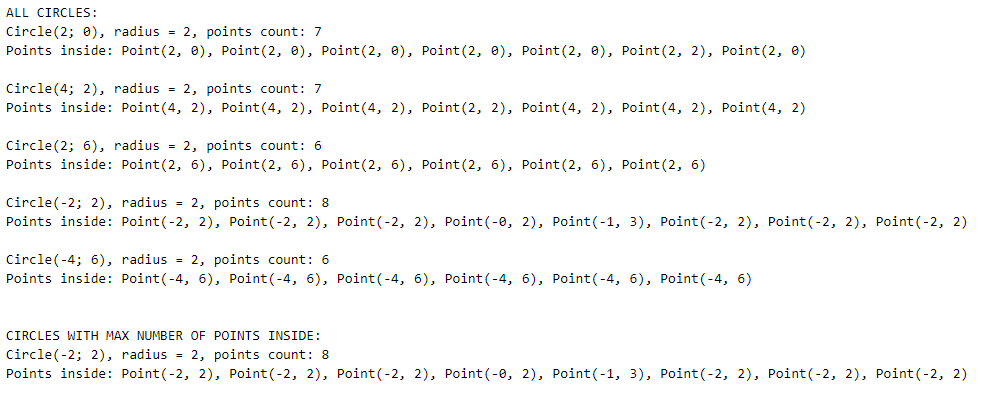
In.txt:



Как это выглядит на плоскости:



Результат (без вывода промежуточных значений):



# **Формальная постановка задания**

*Дано*:

*Найти*: окружность радиуса R с центром в одной из входных точек, содержащую максимальное количество точек пересечения прямых, проведенных через входные точки.

*Способ решения*: найти прямые, проведенные через точки, найти точки пересечения прямых, перебрать входные точки: для каждой точки создавать круг с центром в этой точке и радиусом R, считать количество точек пересечения внутри круга. Результатом будут круги с максимальным количеством точек пересечения внутри.

# **Формат хранения данных**

|  |  |  |
| --- | --- | --- |
| Переменная | Тип | Назначение |
| filename | string | Имя входного файла |
| n | int | Количество исходных точек |
| radius | double | Радиус |
| init\_points | \*Point | Исходные точки |
| number\_of\_lines | int | Количество прямых |
| lines | \*Line | Прямые |
| number\_of\_intersections | int | Количество пересечений |
| intersections | \*Intersection | Пересечения |
| circles | \*Circle | Круги |
| max\_point\_count | int | Количество искомых кругов |
| circles\_max | \*Circle | Искомые круги |
| line | string | Строка для ввода с помощью getline() |
| ss | stringstream | Помогает при вводе |
| line\_els\_count | int | Количество элементов в строке |
| file | fstream | Переменная файла |
| pretext | string | Текст, выводимый перед выводом значений в функциях вывода |

# **Ограничения, обусловленные выполнением на компьютере**

Тип double предоставляет доступ к вещественным числам в диапазоне от +/- 1.7E-308 до 1.7E+308.

# **Макет ввода и вывода**

|  |  |
| --- | --- |
| Приветствие | Author: Novikov G.  Group: 1302  Start date: 3.12.2021  End date: 8.12.2021  Cursed work |
| Ввод | d…d  ………….  d…d  d…d |
| Вводимые точки | INPUT POINTS:  d…d d…d  ………….  d…d d…d |
| Прямые | LINES:  y = d…dx + d…d или x = d…d  ………….  y = d…dx + d…d или x = d…d |
| Точки пересечения | INTERSECTION POINTS:  d…d d…d или Point does not exist  ………….  d…d d…d или Point does not exist |
| Дополнительная информация | Points: d…d  Lines: d…d  Intersections: d…d  Radius: d…d |
| Все круги | ALL CIRCLES:  Circle(d…d; d…d), radius = d…d, points count: d…d  Points inside: Point(d…d, d…d),… , Point(d…d, d…d)  ………….  Circle(d…d; d…d), radius = d…d, points count: d…d  Points inside: Point(d…d, d…d),… , Point(d…d, d…d) |
| Искомые круги | CIRCLES WITH MAX NUMBER OF POINTS INSIDE:  Circle(d…d; d…d), radius = d…d, points count: d…d  Points inside: Point(d…d, d…d),… , Point(d…d, d…d)  ………….  Circle(d…d; d…d), radius = d…d, points count: d…d  Points inside: Point(d…d, d…d),… , Point(d…d, d…d) |

# Средства обеспечения ввода и вывода

Средство обеспечения ввода: getline() (библиотека string)

Средство обеспечения вывода в консоль: cout << (библиотека iostream)

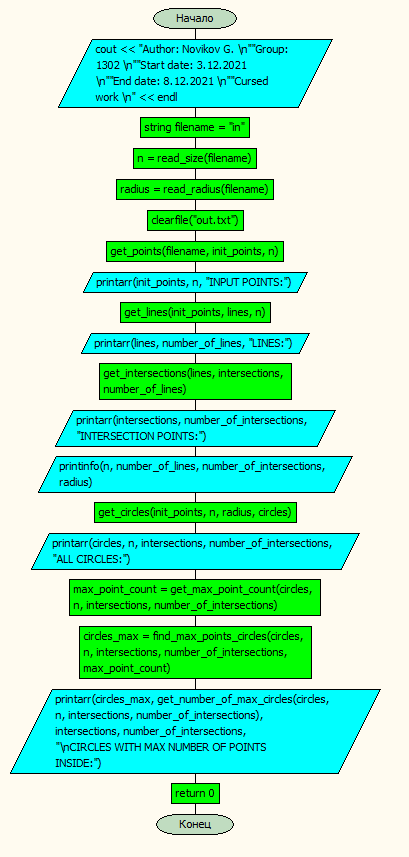
Средство обеспечения вывода в файл: file << (библиотека fstream)

# **Алгоритм решения**

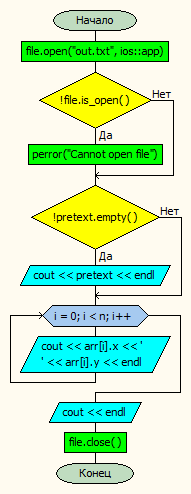
Функции:

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Аргументы** | **Тип возвращаемого значения** | **Назначение** |
| main | int argc, char const \*argv[] | int | Исполнение программы |
| clearfile | string filename | void | Удаление данных в выходном файле |
| printarr | Circle\* arr, int n, Intersection\* intersections, int intersections\_n, string pretext = "" | void | Вывод массива(работает с разными типами данных массива) |
| printarr | Intersection\* arr, int n, string pretext = "" |
| printarr | Line\* arr, int n, string pretext = "" |
| printarr | Point\* arr, int n, string pretext = "" |
| printinfo | int points\_n, int lines\_n, int intersections\_n, double radius | void | Вывод доп. информации |
| find\_max\_points\_circles | Circle\* circles, int n, Intersection\* intersections, int intersection\_n, int max\_point\_count | Circle\* | Находит искомые круги |
| get\_number\_of\_max\_circles | Circle\* circles, int n, Intersection\* intersections, int intersection\_n | int | Возвращает количество искомых кругов |
| get\_max\_point\_count | Circle\* circles, int n, Intersection\* intersections, int intersection\_n | int | Возвращает максимальное количество точек внутри круга |
| get\_circles | Point\* init\_points, int n, double radius, Circle\* circles | void | Создает массив кругов |
| get\_intersections | Line\* lines, Intersection\* intersections, int n | Intersection\* | Создает массив пересечений |
| get\_lines | Point\* points, Line\* lines, int n | Line\* | Создает массив прямых |
| get\_points | string filename, Point\* arr, int n | Point\* | Создает массив точек |
| read\_radius | string filename | double | Считывает из файла радиус |
| read\_size | string filename | int | Считает количество точек во входном файле |

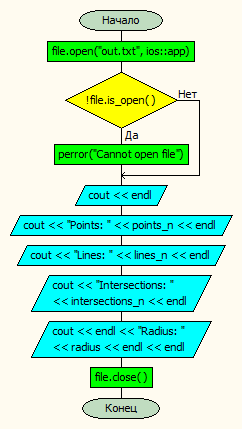
main



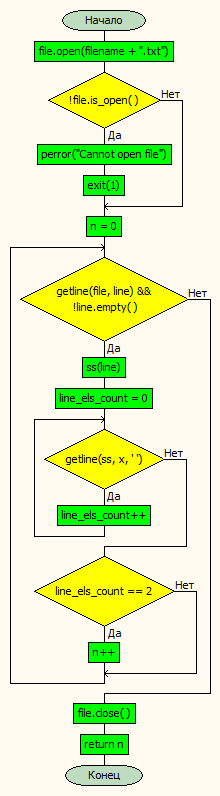
printarr



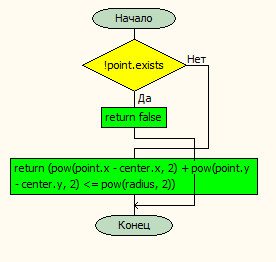
printinfo



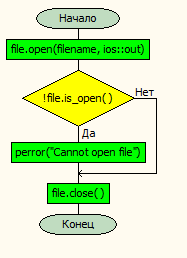
read\_radius



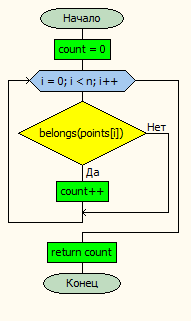
Belongs



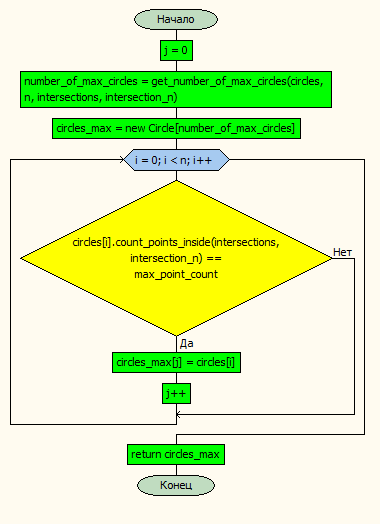
clearfile



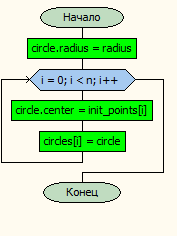
count\_points\_inside



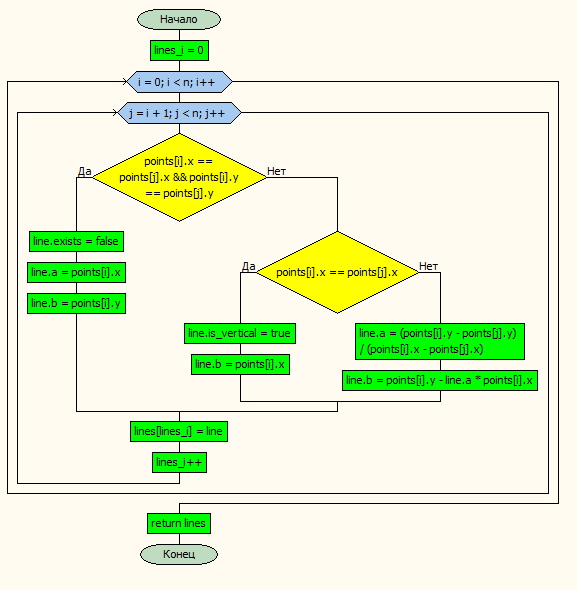
find\_max\_points\_circles



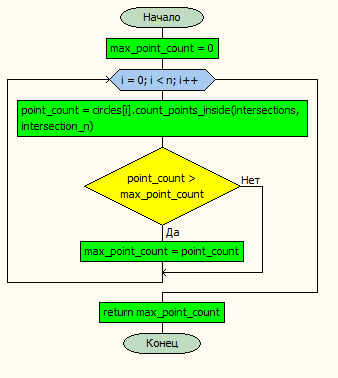
get\_circles



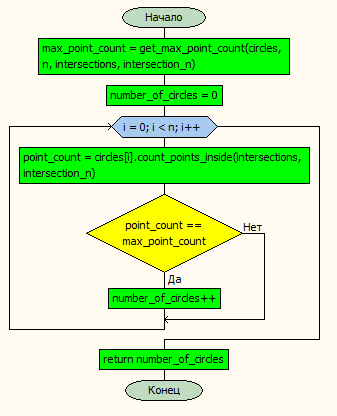
get\_lines



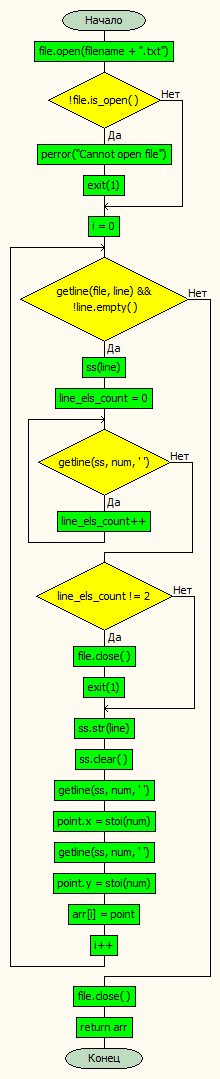
get\_max\_point\_count



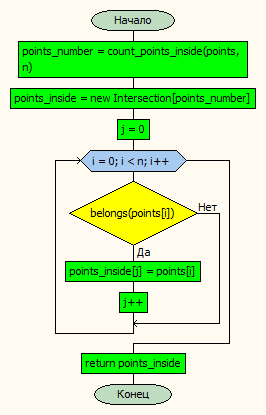
get\_number\_of\_max\_circles



get\_points



get\_points\_inside



# **Программа**

#include <iostream>

#include <fstream>

#include <sstream>

#include <iomanip>

#include <math.h>

using namespace std;

struct Point {

double x;

double y;

};

struct Intersection : Point {

bool exists = true;

bool whole\_x = false;

bool whole\_y = false;

bool whole\_line = false;

};

struct Circle {

Point center;

double radius;

bool belongs(Intersection point) {

if (!point.exists) return false;

return (pow(point.x - center.x, 2) + pow(point.y - center.y, 2) <= pow(radius, 2));

}

int count\_points\_inside(Intersection\* points, int n) {

int count = 0;

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

if (belongs(points[i])) count++;

}

return count;

}

Intersection\* get\_points\_inside(Intersection\* points, int n) {

int points\_number = count\_points\_inside(points, n);

Intersection\* points\_inside = new Intersection[points\_number];

int j = 0;

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

if (belongs(points[i])) {

points\_inside[j] = points[i];

j++;

}

}

return points\_inside;

}

};

struct Line {

// normal: y = ax + b

// vertical: x = b

double a;

double b;

bool is\_vertical = false;

bool exists = true;

};

int read\_size(string filename) {

ifstream file;

file.open(filename + ".txt");

if (!file.is\_open()) {

perror("Cannot open file");

exit(1);

}

int n = 0;

string line;

while (getline(file, line) && !line.empty()) {

stringstream ss(line);

string x;

int line\_els\_count = 0;

while (getline(ss, x, ' ')) {

line\_els\_count++;

}

if (line\_els\_count == 2) {

n++;

}

}

file.close();

return n;

}

double read\_radius(string filename) {

ifstream file;

file.open(filename + ".txt");

if (!file.is\_open()) {

perror("Cannot open file");

exit(1);

}

double radius;

string line;

while (getline(file, line) && !line.empty()) {}

while (getline(file, line) && line.empty()) {}

stringstream ss(line);

string x;

int line\_els\_count = 0;

while (getline(ss, x, ' ')) {

line\_els\_count++;

}

if (line\_els\_count != 1) {

cerr << "Radius line must contain only one number - radius" << endl;

file.close();

exit(1);

}

radius = stod(line);

file.close();

return radius;

}

Point\* get\_points(string filename, Point\* arr, int n) {

ifstream file;

file.open(filename + ".txt");

if (!file.is\_open()) {

perror("Cannot open file");

exit(1);

}

int i = 0;

string line;

string num;

while (getline(file, line) && !line.empty()) {

Point point;

stringstream ss(line);

string num;

int line\_els\_count = 0;

while (getline(ss, num, ' ')) {

line\_els\_count++;

}

if (line\_els\_count != 2) {

cerr << "get\_points: Point line must contain 2 numbers" << endl;

file.close();

exit(1);

}

ss.str(line);

ss.clear();

getline(ss, num, ' ');

point.x = stoi(num);

getline(ss, num, ' ');

point.y = stoi(num);

arr[i] = point;

i++;

}

file.close();

return arr;

}

Line\* get\_lines(Point\* points, Line\* lines, int n) {

int lines\_i = 0;

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

for (int j = i + 1; j < n; j++) {

Line line;

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

line.exists = false;

line.a = points[i].x;

line.b = points[i].y;

} else if (points[i].x == points[j].x) {

line.is\_vertical = true;

line.b = points[i].x;

} else {

line.a = (points[i].y - points[j].y) / (points[i].x - points[j].x);

line.b = points[i].y - line.a \* points[i].x;

}

lines[lines\_i] = line;

lines\_i++;

}

}

return lines;

}

Intersection\* get\_intersections(Line\* lines, Intersection\* intersections, int n) {

int intersections\_i = 0;

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

for (int j = i + 1; j < n; j++) {

Intersection point;

if (!lines[i].exists || !lines[j].exists) {

point.exists = false;

} else if(lines[i].is\_vertical && lines[j].is\_vertical) { // both lines are x = b

if (lines[i].b == lines[j].b) { // same vertical line

point.x = lines[i].b;

point.whole\_y = true;

point.whole\_line = true;

} else { // parallel lines

point.exists = false;

}

} else if (lines[i].is\_vertical) { // one vertical line

point.x = lines[i].b;

point.y = lines[j].a \* lines[i].b + lines[j].b;

} else if(lines[j].is\_vertical) { // one vertical line

point.x = lines[j].b;

point.y = lines[i].a \* lines[j].b + lines[i].b;

} else if (lines[i].a == lines[j].a) {

if (lines[i].b == lines[j].b) {

point.whole\_line = true; // same line

} else {

point.exists = false; // parallel line

}

} else if (lines[i].a == 0 && lines[j].a == 0) { // y = b

if (lines[i].b == lines[j].b) { // same horizonral line

point.y = lines[i].b;

point.whole\_x = true;

point.whole\_line = true;

} else { // parallel lines

point.exists = false;

}

} else {

point.x = (lines[j].b - lines[i].b) / (lines[i].a - lines[j].a);

point.y = lines[i].a \* point.x + lines[i].b;

}

point.x = roundf(point.x \* 1000000) / 1000000;

point.y = roundf(point.y \* 1000000) / 1000000;

intersections[intersections\_i] = point;

intersections\_i++;

}

}

return intersections;

}

void get\_circles(Point\* init\_points, int n, double radius, Circle\* circles) {

Circle circle;

circle.radius = radius;

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

circle.center = init\_points[i];

circles[i] = circle;

}

}

int get\_max\_point\_count(Circle\* circles, int n, Intersection\* intersections, int intersection\_n) {

int max\_point\_count = 0;

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

int point\_count = circles[i].count\_points\_inside(intersections, intersection\_n);

if (point\_count > max\_point\_count) {

max\_point\_count = point\_count;

}

}

return max\_point\_count;

}

int get\_number\_of\_max\_circles(Circle\* circles, int n, Intersection\* intersections, int intersection\_n) {

int max\_point\_count = get\_max\_point\_count(circles, n, intersections, intersection\_n);

int number\_of\_circles = 0;

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

int point\_count = circles[i].count\_points\_inside(intersections, intersection\_n);

if (point\_count == max\_point\_count) {

number\_of\_circles++;

}

}

return number\_of\_circles;

}

Circle\* find\_max\_points\_circles(Circle\* circles, int n, Intersection\* intersections, int intersection\_n, int max\_point\_count) {

int j = 0;

int number\_of\_max\_circles = get\_number\_of\_max\_circles(circles, n, intersections, intersection\_n);

Circle\* circles\_max = new Circle[number\_of\_max\_circles];

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

if (circles[i].count\_points\_inside(intersections, intersection\_n) == max\_point\_count) {

circles\_max[j] = circles[i];

j++;

}

}

return circles\_max;

}

void printinfo(int points\_n, int lines\_n, int intersections\_n, double radius) {

fstream file;

file.open("out.txt", ios::app);

if (!file.is\_open()) {

perror("Cannot open file");

}

cout << endl;

cout << "Points: " << points\_n << endl;

cout << "Lines: " << lines\_n << endl;

cout << "Intersections: " << intersections\_n << endl;

cout << endl << "Radius: " << radius << endl << endl;

file << endl;

file << "Points: " << points\_n << endl;

file << "Lines: " << lines\_n << endl;

file << "Intersections: " << intersections\_n << endl;

file << endl << "Radius: " << radius << endl << endl;

file.close();

}

void printarr(Point\* arr, int n, string pretext = "") {

fstream file;

file.open("out.txt", ios::app);

if (!file.is\_open()) {

perror("Cannot open file");

}

if (!pretext.empty()) {

cout << pretext << endl;

file << pretext << endl;

}

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

cout << arr[i].x << ' ' << arr[i].y << endl;

file << arr[i].x << ' ' << arr[i].y << endl;

}

cout << endl;

file << endl;

file.close();

}

void printarr(Line\* arr, int n, string pretext = "") {

fstream file;

file.open("out.txt", ios::app);

if (!file.is\_open()) {

perror("Cannot open file");

}

if (!pretext.empty()) {

cout << pretext << endl;

file << pretext << endl;

}

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

if (!arr[i].exists) {

cout << "Line does not exist" << endl;

file << "Line does not exist" << endl;

} else if (arr[i].is\_vertical) {

cout << setprecision(4) << "x = " << arr[i].b << endl;

file << setprecision(4) << "x = " << arr[i].b << endl;

} else {

cout << setprecision(4) << "y = " << arr[i].a << "x + " << arr[i].b << endl;

file << setprecision(4) << "y = " << arr[i].a << "x + " << arr[i].b << endl;

}

}

cout << endl;

file << endl;

file.close();

}

void printarr(Intersection\* arr, int n, string pretext = "") {

fstream file;

file.open("out.txt", ios::app);

if (!file.is\_open()) {

perror("Cannot open file");

}

if (!pretext.empty()) {

cout << pretext << endl;

file << pretext << endl;

}

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

if (!arr[i].exists) {

cout << "Point does not exist" << endl;

file << "Point does not exist" << endl;

} else if (arr[i].whole\_line) {

if (arr[i].whole\_x) {

cout << "y = " << arr[i].y << " x = (-inf, +inf)" << endl;

file << "y = " << arr[i].y << " x = (-inf, +inf)" << endl;

} else if (arr[i].whole\_y) {

cout << "y = (-inf, +inf) x = " << arr[i].x << endl;

file << "y = (-inf, +inf) x = " << arr[i].x << endl;

} else {

cout << "y = (-inf, +inf) x = (-inf, +inf)" << endl;

file << "y = (-inf, +inf) x = (-inf, +inf)" << endl;

}

} else {

cout << setprecision(4) << arr[i].x << ' ' << arr[i].y << endl;

file << setprecision(4) << arr[i].x << ' ' << arr[i].y << endl;

}

}

file.close();

}

void printarr(Circle\* arr, int n, Intersection\* intersections, int intersections\_n, string pretext = "") {

fstream file;

file.open("out.txt", ios::app);

if (!file.is\_open()) {

perror("Cannot open file");

}

if (!pretext.empty()) {

cout << pretext << endl;

file << pretext << endl;

}

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

int point\_count = arr[i].count\_points\_inside(intersections, intersections\_n);

Intersection\* points\_inside = arr[i].get\_points\_inside(intersections, intersections\_n);

cout << "Circle(" << arr[i].center.x << "; " << arr[i].center.y << "), radius = " << arr[i].radius << ", points count: " << point\_count << endl;

file << "Circle(" << arr[i].center.x << "; " << arr[i].center.y << "), radius = " << arr[i].radius << ", points count: " << point\_count << endl;

cout << "Points inside: ";

file << "Points inside: ";

for (int j = 0; j < point\_count; j++) {

cout << "Point(" << points\_inside[j].x << ", " << points\_inside[j].y << ")";

file << "Point(" << points\_inside[j].x << ", " << points\_inside[j].y << ")";

if (j != point\_count - 1) {

cout << ", ";

file << ", ";

} else {

cout << "\n" << endl;

file << "\n" << endl;

}

}

}

file.close();

}

void clearfile(string filename) {

fstream file;

file.open(filename, ios::out);

if (!file.is\_open()) {

perror("Cannot open file");

}

file.close();

}

int main(int argc, char const \*argv[]) {

cout << "Author: Novikov G. \n"

"Group: 1302 \n"

"Start date: 3.12.2021 \n"

"End date: 8.12.2021 \n"

"Cursed work \n" << endl;

const string filename = "in";

int n = read\_size(filename);

double radius = read\_radius(filename);

clearfile("out.txt");

Point\* init\_points = new Point[n];

get\_points(filename, init\_points, n);

printarr(init\_points, n, "INPUT POINTS:");

int number\_of\_lines = n \* (n - 1) / 2;

Line\* lines = new Line[number\_of\_lines];

get\_lines(init\_points, lines, n);

printarr(lines, number\_of\_lines, "LINES:");

int number\_of\_intersections = number\_of\_lines \* (number\_of\_lines - 1) / 2;

Intersection\* intersections = new Intersection[number\_of\_intersections];

get\_intersections(lines, intersections, number\_of\_lines);

printarr(intersections, number\_of\_intersections, "INTERSECTION POINTS:");

printinfo(n, number\_of\_lines, number\_of\_intersections, radius);

Circle\* circles = new Circle[n];

get\_circles(init\_points, n, radius, circles);

printarr(circles, n, intersections, number\_of\_intersections, "ALL CIRCLES:");

int max\_point\_count = get\_max\_point\_count(circles, n, intersections, number\_of\_intersections);

Circle\* circles\_max = find\_max\_points\_circles(circles, n, intersections, number\_of\_intersections, max\_point\_count);

printarr(circles\_max, get\_number\_of\_max\_circles(circles, n, intersections, number\_of\_intersections), intersections, number\_of\_intersections, "\nCIRCLES WITH MAX NUMBER OF POINTS INSIDE:");

delete[] init\_points;

delete[] lines;

delete[] intersections;

delete[] circles;

return 0;

}

# **Результаты работы программы**

# **Выводы о проделанной работе****Список использованных источников**

1. Описание методических указаний / сост.: Калмычков В.А. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2021.

2. Описание составления отчета по курсовой работе / СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2021.