**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: **«**Обработка текстов. Реализация длинной арифметики на C++**»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3352 |  | Рябов В.А. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Рябов В.А. | | |
| Группа 3352 | | |
| Тема работы: Создание обучающей системы, предназначенная для школьников 1–2-го классов средней школы, которая позволяет выполнять базовые арифметические операции с контролем хода вычислений и полученных результатов над целыми n-разрядными положительными числами (n < 80). | | |
| Исходные данные:  Дан список чисел, список запросов для этих чисел. | | |
| |  | | --- | | Содержание пояснительной записки: | | | |
| Предполагаемый объем пояснительной записки:  Не менее 30 страниц. | | |
| Дата выдачи задания: 20.02.2024 | | |
| Дата сдачи реферата: 05.2024 | | |
| Дата защиты реферата: .05.2024 | | |
| Студент |  | Рябов В.А. |
| Преподаватель |  | Калмычков В.А. |

**Аннотация**

В данной курсовой работе требуется реализовать обучающую систему для школьников 1-2 класса. Система должна демонстрировать базовые операции сложения, вычитания, умножения, деления чисел. Все числа имеют длину не более 80 символов и хранятся в списках в виде линейных связанных списков целых чисел. Методами создания программы стали изученные в ходе семестра возможности языка программирования C++, а именно классы, структуры и списки.

**Summary**

In this course work, it is required to implement a training system for schoolchildren in grades 1-2. The system must demonstrate basic operations of addition, subtraction, multiplication, and division of numbers. All numbers are no more than 80 characters long and are stored in lists as linear lists of integers. The methods for creating the program were the capabilities of the C++ programming language studied during the semester, namely classes, structures and lists.**содержание**

|  |  |
| --- | --- |
|  | |
|  | Введение. | | 5 |
| 1. | Исходная формулировка. | | 5 |
| 2. | Математическая постановка задачи. | | 5 |
| 3. | Особенности реализации на компьютере. | | 8 |
| 4. | Контрольные примеры. | | 9 |
| 4.1 | Пример №1. | | 9 |
| 4.2 | Пример №2. | | 10 |
| 4.3 | Пример №3. | | 11 |
| 4.4 | Пример №4. | | 12 |
| 5. | Организация диалога с пользователем. | | 12 |
| 6. | Переменные в программе. | | 14 |
| 7. | Функции программы. | | 15 |
| 8. | Описание алгоритма. | | 20 |
| 9. | Алгоритм решения. | | 20 |
| 10. | Программа. | | 30 |
| 11. | Пример работы программы. | | 46 |
|  | Выводы. | | 48 |

**Введение.**

Целью работы является реализация программы на языке С++ с использованием знаний, полученных в ходе 1 и 2 семестра, обучится использованию связанных списков, а также вложенных связанных списков.

**Исходная формулировка задания.**

Обучающая система, предназначенная для школьников 1–2-го классов средней школы должна позволять демонстрировать выполнение арифметических операций (сложение, вычитание, умножение и деление) над целыми n-разрядными положительными числами (n < 80) выполнять арифметические операции с контролем хода вычислений и полученных результатов.

**Математическая постановка задачи.**

В работе необходимо проведение арифметических операций над длинными числами. Так как числа достаточно длинные, то они хранятся в списках char по 5 символов в каждом блоке. Это удобно для считывания и хранения, но не для выполнения операций. Поэтому, числа переводятся в массивы цифр типа int.

Рассмотрим методы сложения и вычитания. Реализация их похожа. Для начала списки будет необходимо отзеркалить, как показано на рисунке:

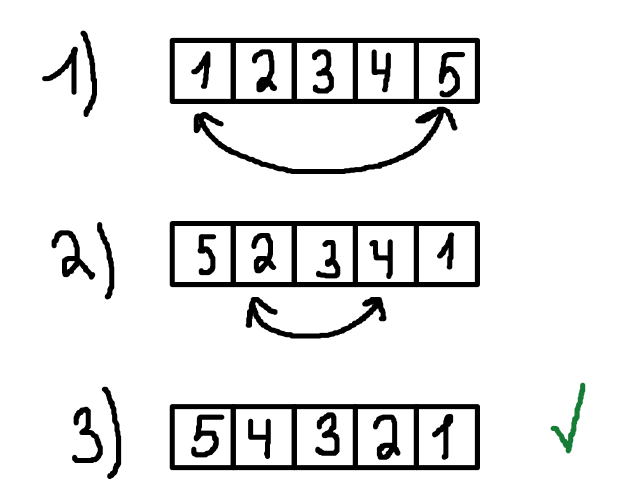


Рис. 1 – Переворачивание списка.

Далее, мы будем складывать числа по разрядам прибавлять остаток, если число + остаток больше 9 то в ответ пишем последнюю цифру если нет то просто число + остаток. Сохраняем остаток.

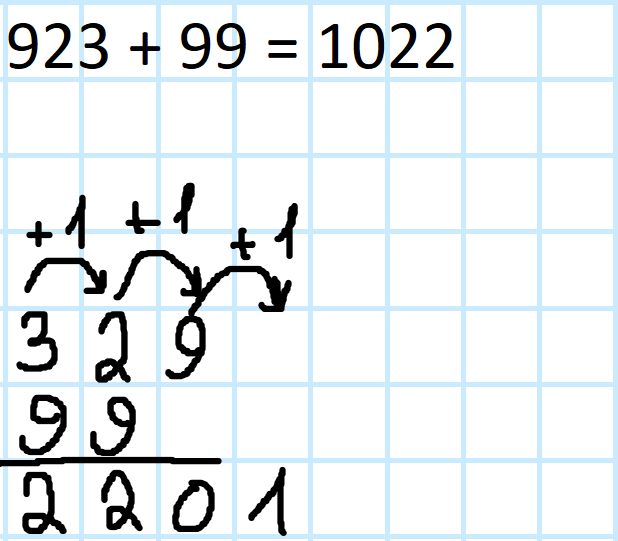


Рис. 2 – Реализация сложения в программе.

Перед вычитанием мы убеждаемся, что вычитаем из большего меньшее. Вычитание реализуется аналогично, только мы будем вместо сложения цифр их вычитать. Если получим отрицательное число, то к результату прибавим 10, а из следующего разряда вычитаемого вычитаем 1.

Теперь рассмотрим умножение. Мы выделим память под результат, он будет равен сумме размеров множителей. Будем умножать цифры второго множителя на 1 добавлять остаток и то, что у нас находится в i+j разряде. Так как результат мы будем сразу складывать, то вывод “лесенкой” у нас не получится, поэтому перед умножением мы выведем лесенку, просто умножив цифру у второго множителя на первый множитель, не забудем сделать отступ.

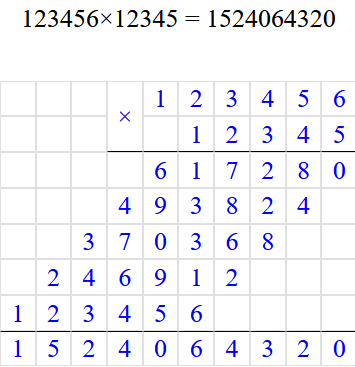


Рис. 3 – Умножение лесенкой.

Рассмотрим деление. Сразу обработаем случаи, когда делитель = 0. Когда делитель больше делимого просто запишем 0. Когда числа равны запишем 1.

Далее, из делимого с начала будем брать по цифре, сохранять ее, затем проверять, больше ли она делителя.

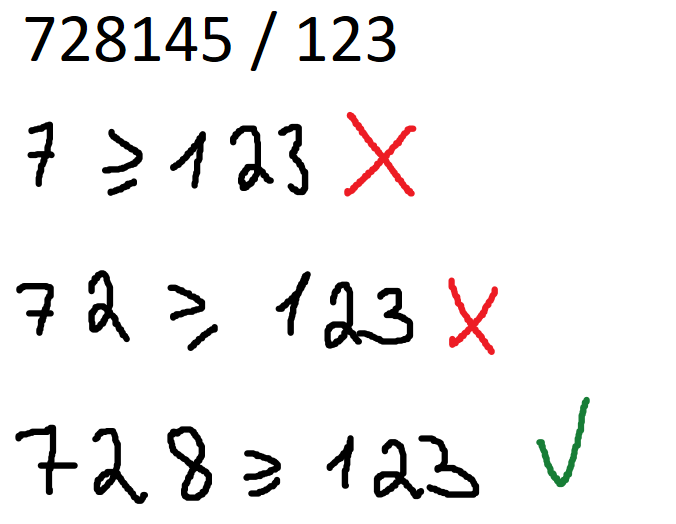


Рис. 4 – Подбор.

Когда число найдено, начинаем умножать делитель на числа от 1 до 9 пока не найдем равное или число большее.

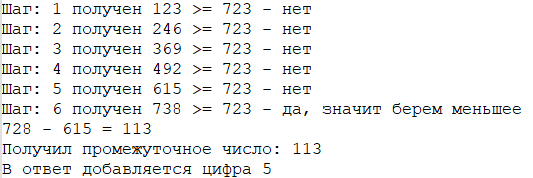


Рис. 5 – Процесс деления.

И так далее. Не забудем потом, что когда мы сносим сразу две цифры сразу, то в результат нужно добавить 0. Когда сносить будет нечего, мы прекратим деление и получим результат.

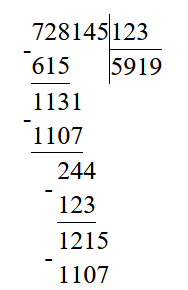


Рис. 6 – Деление столбиком.

**Особенности реализации на компьютере.**

В задаче необходимо работать с файлом, в котором изначально находится двумерная матрица размером Nx2. Размер матрицы нигде не указан, программа будет читать файл до самого конца. Возможны ошибки пользователя при вводе данных. Если пользователь по ошибке добавит n-ное количество пробелов или табуляций между элементами, оставит пустые строчки, напишет что-то текстом, введет вместо 2 значений одно или больше чем два, тогда программа обработает эти ошибки, сообщит о них в файл протокола, в конечном итоге пропустив ненужные строки программа корректно отобразит исходный результат.

В итоге, после всех действий в программе будут установлены действительные размеры n и m, а также массив с позициями начала строк.

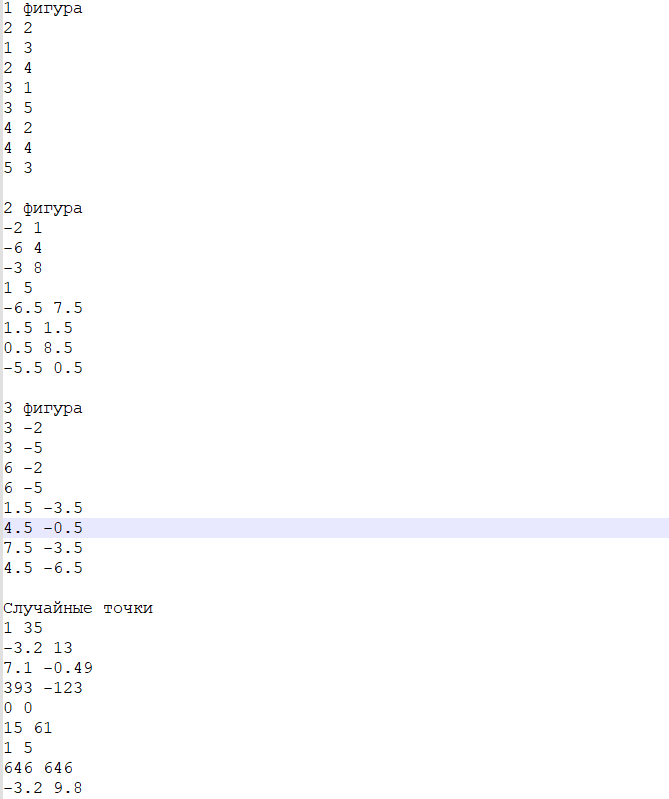
Когда файл точек будет прочитан и занесен в массив, он будет отсортирован методом базовой сортировки и в нем будут удалены все дубликаты, обычным методом вынесения лишних элементов в конец массива и изменения его длины.

**Контрольные примеры.**

Пример №1.

В данном примере имеются три фигуры, одна из которых наклонена на несколько градусов, две другие не имеют наклона. Также, у одной фигуры имеется повторяющаяся точка c координатами (1;5), фигура с красным цветом пересекается с фигурой желтого цвета.

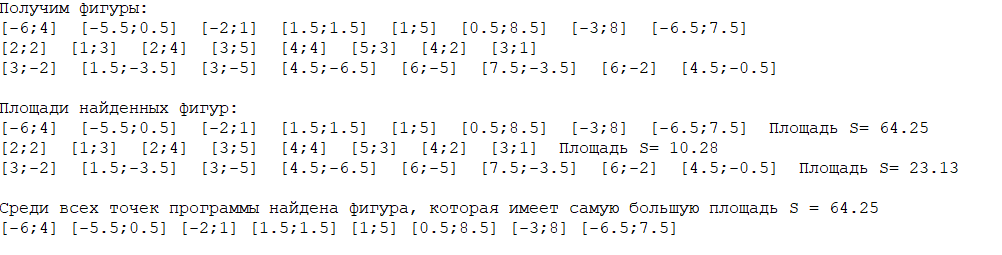
Исходный файл:



Графическое представление:



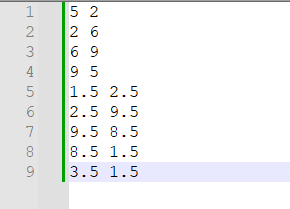
Итог:



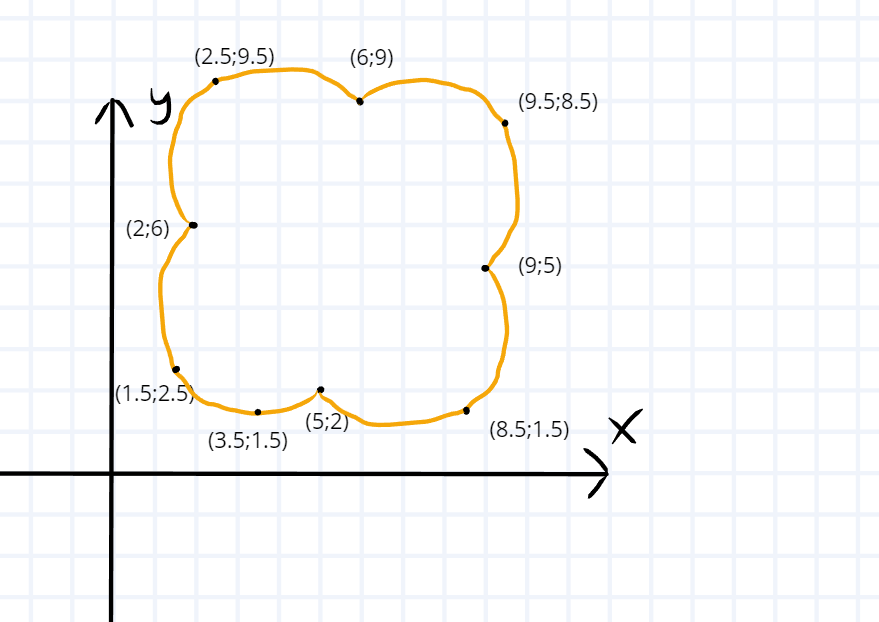
Пример №2

Точка на окружности необязательно должна лежать именно в ее центре. Ранее рассматривались примеры, где точка на окружности лежит именно на ее центре. Это было сделано для удобства, но точка может находиться где угодно, главное, чтобы она была на этой окружности. Рассмотрим пример, когда на одной дуге находятся сразу две точки.

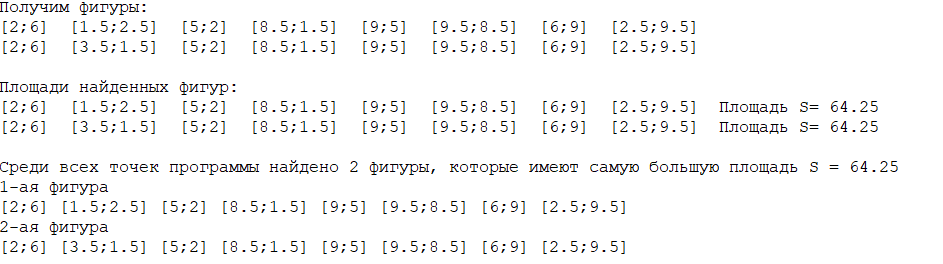
Исходный файл:



Графическое представление:



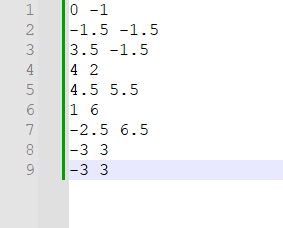
Итог:



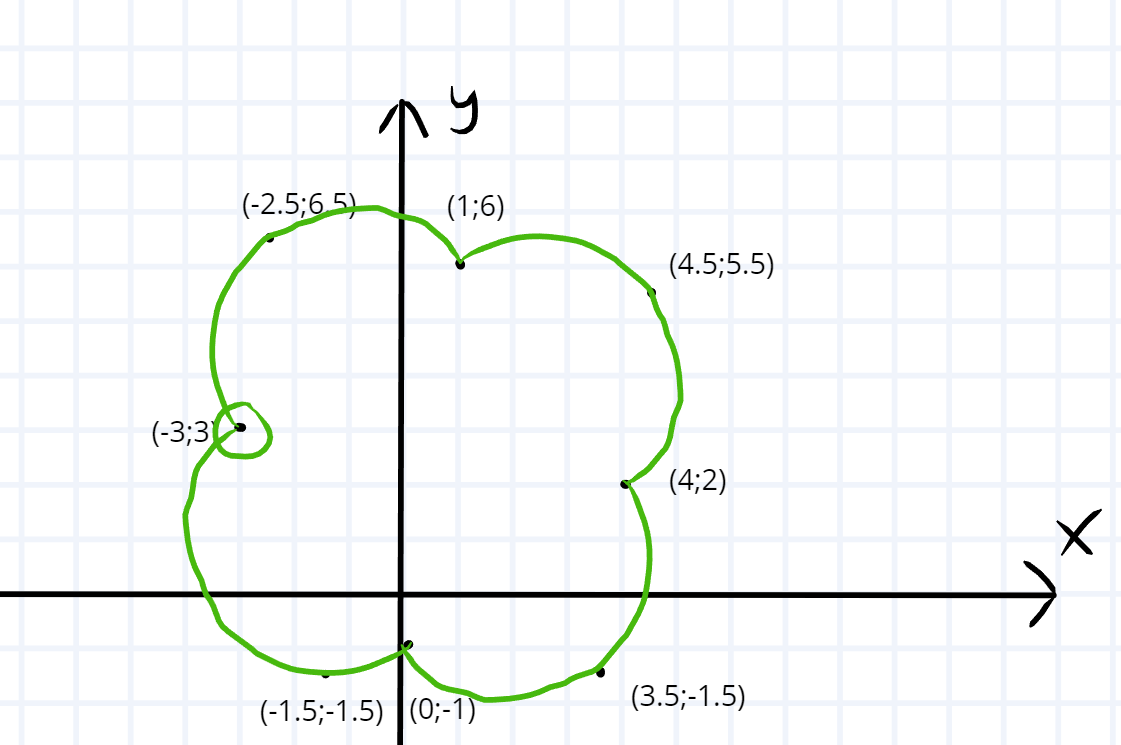
Пример №3

Теперь рассмотрим пример, когда одна фигура имеет несколько повторяющихся точек. В нашем примере точка (-3;3) повторяется два раза.

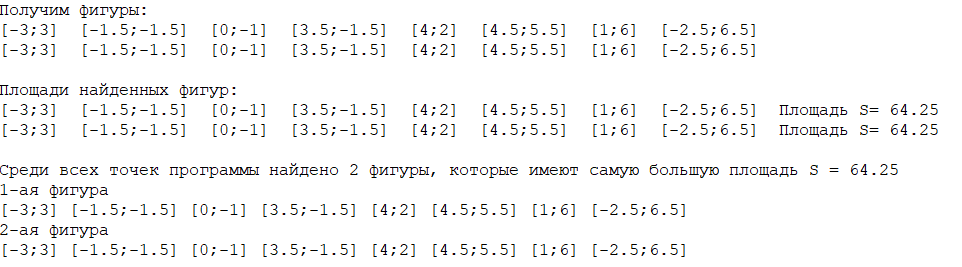
Исходный файл:



Графическое представление:



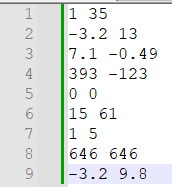
Итог:



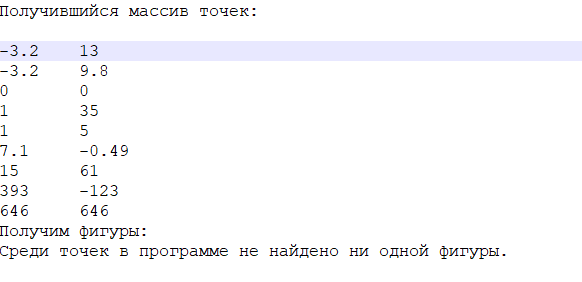
Пример №4

В файле отсутствуют фигуры.

Исходный файл.



Итог:



**Организация диалога с пользователем.**

Диалог с пользователем происходит через файл. В файле Points.txt пользователь вводит координаты точек, что будет происходить, если точки будут введены некорректно рассказано в другом блоке.

Таблица 2 – Пример организации UI в файле вывода.

|  |  |
| --- | --- |
| 1. Приветствие | Курсовая работа. Рябов Вадим. 3352.  Задание 8.1 Дано N произвольных точек на плоскости. Найти среди них точки, являющиеся вершинами (образующими) фигур, обладающих следующими свойствами : 1) фигура с максимальной площадью |
| 2. Исходный массив точек. | Получившийся массив точек:  -6.5 7.5  -6 4  -5.5 0.5  -3 8  -2 1  0.5 8.5  1 5  1.5 2.5  1.5 1.5  2 6  2.5 9.5  3.5 1.5  5 2  6 9  8.5 1.5  9 5  9.5 8.5 |
| 3. Вывод полученных фигур. | Получим фигуры:  [-6;4] [-5.5;0.5] [-2;1] [1.5;1.5] [1;5] [0.5;8.5] [-3;8] [-6.5;7.5]  [2;6] [1.5;2.5] [5;2] [8.5;1.5] [9;5] [9.5;8.5] [6;9] [2.5;9.5]  [2;6] [3.5;1.5] [5;2] [8.5;1.5] [9;5] [9.5;8.5] [6;9] [2.5;9.5] |
| 4. Вывод фигур и их площадей. | Площади найденных фигур:  [-6;4] [-5.5;0.5] [-2;1] [1.5;1.5] [1;5] [0.5;8.5] [-3;8] [-6.5;7.5] Площадь S= 64.25  [2;6] [1.5;2.5] [5;2] [8.5;1.5] [9;5] [9.5;8.5] [6;9] [2.5;9.5] Площадь S= 64.25  [2;6] [3.5;1.5] [5;2] [8.5;1.5] [9;5] [9.5;8.5] [6;9] [2.5;9.5] Площадь S= 64.25 |
| Вывод о найденных точках. | Среди всех точек программы найдено 3 фигуры, которые имеют самую большую площадь S = 64.25  1-ая фигура  [-6;4] [-5.5;0.5] [-2;1] [1.5;1.5] [1;5] [0.5;8.5] [-3;8] [-6.5;7.5]  2-ая фигура  [2;6] [1.5;2.5] [5;2] [8.5;1.5] [9;5] [9.5;8.5] [6;9] [2.5;9.5]  3-ая фигура  [2;6] [3.5;1.5] [5;2] [8.5;1.5] [9;5] [9.5;8.5] [6;9] [2.5;9.5]  или (если фигуры отсутствуют)  Среди точек в программе не найдено ни одной фигуры. |

**Переменные в программе.**

В таблице не рассматриваются переменные счетчиков, а также временные переменные, которые не выходят за области видимости своих локальных функций.

Таблица 3 – Переменные в программе.

|  |  |  |
| --- | --- | --- |
| Тип переменной | Название | Значение |
| unsigned | n | Количество строк. |
|  | m | Количество столбцов (по стандарту равно 2). |
| fstream | file | Файл с массивом. |
|  | ofile | Файл для вывода. |
|  | protocol | Файл протокола. |
| int\* | positions | Массив позиций начала строк в файле. |
| float\*\* | points | Динамический двумерный массив хранящий точки, прочитанные в файле. |
| unsigned | org\_n | Оригинальное количество строк. Необходимо для правильного удаления массива. |
| double\*\*\* | rectangles | Массив, хранящий квадраты в функции Process |
| double\*\*\* | pt | Массив который хранит координаты фигур. |

**Функции программы.**

Таблица 4 – Функции, использованные в программе.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Имя | Назначение | Параметры | | | | Возвращаемые значения | Внешние эффекты |
| Входные | Выходные | Модифицированные | Транзитные |
| CheckFile | Начальная обработка файла для установления действительных размеров. | unsigned& n, unsigned m, fstream& f, int\* positions, fstream& protocol |  | n, file, positions |  | void |  |
| InitDynamicAr | Создание динамического массива размером m\*n. | int n, int m | new\_array |  |  | double\*\* | Выделяется память под двумерный динамический массив. |
| DeleteDynamicAr | Удаление динамического массива. | double\*\* array, int n |  | array |  | void | Память, выделенная под двумерный дин. массив очищается. |
| FillArray | Заполнение динамического массива. | double\*\* array, int n, int m, fstream& f, int\* positions, fstream& protocol |  | array |  | void |  |
| Sort\_2DM\_Array | Сортировка массива точек по возрастанию. | double\*\* arr, int n |  |  | arr | void | Массив точек становится отсортированным. |
| Print2DNArray | Вывод двумерного массива на экран и в файл. | double\*\* points, int n, int m, fstream& protocol, bool toconsole = true |  |  |  | void |  |
| Process | Основной метод нахождения квадратов. | double\*\* ar, unsigned n, unsigned m, fstream& protocol, fstream& output |  | ar | double x1, x2, x3, x4, y1, y2, y3, y4  n, ar  double k, b, k\_per, b\_ber, k4, b4  double\*\*\* rectangles,  double count  fstream& protocol, output | void | В массив rectangles заносятся координаты квадратов. В конце он удаляется. |
| Swap | Меняет местами две переменные. | double\* a1, double\* a2 |  |  | a1, a2 |  | Перменные меняются местами. |
| GetTg | Находит k | double A[2], double B[2] | k |  |  | double |  |
| GetB | Находит b | double A[2], double k | b |  |  | double |  |
| Distance | Находит расстояние между двумя точками. | double x1, double y1, double x2, double y2 | distance |  |  | double |  |
| is\_on\_line | Проверка нахождения точки на прямой. | double x, double y, double k, double b | 0/1 |  |  | bool |  |
| found\_x | Находит x по коэффициентам. | double b1, double b2, double k1, double k2 | Арифм. выр. |  |  | double |  |
| found\_y | Находит y по коэффициентам. | double x, double k, double b | Арифм. выр. |  |  | double |  |
| Init3DNArr | Cоздание 3-х мерного массива. | unsigned n, int m = 8, int h = 2 | array |  |  | double\*\*\* | Создается трехмерный массив. |
| Delete3DNArr | Удаление 3-х мерного массива. | double\*\*\* arr, unsigned n, int m = 8, int h = 2 |  |  |  | void | Удаление трехмерного массива. |
| SaveRectangle | Сохранение квадрата в массив. | int p, double A[2], double B[2], double C[2], double D[2], double\*\*\* rectangles, unsigned n |  |  |  | void | В массив сохраняются точки квадрата. |
| DeleteVoidElements | Метод, который удаляет пустые элементы в массиве квадратов | double\*\*\* arr, int count, unsigned n, int m = 8, int h = 2 | new\_arr |  | count, arr, n | double\*\*\* | Удаление оригинального массива и создание нового. |
| get\_xc | Находит сумму координат x деленную на 4. | double\*\* kv | Арифм выр. |  |  | double |  |
| get\_yc | Находит сумму координат y деленную на 4. | double\*\* kv | Арифм выр. |  |  | double |  |
| is\_on\_circle | Проверяет, находится ли точка на окружности. | double x0, double y0, double r, double x, double y | 0/1 |  |  | bool |  |
| Print3DArray | Выводит на экран трёхмерный массив. | double\*\*\* rectangles, int count, fstream& protocol,  int n = 8 |  |  |  | void | Выводит массив на экран и в файл. |
| ChangeLastArray | Изменяет массив с точками, перемещая координаты  по кругу. | double\*\*\* figures, int count, int originalc | double\*\*\* new\_array |  | count, figures, originalc | double\*\*\* | Удаляет старый массив. Новый заполняет значениями. |
| SquareFinder | Итоговый метод нахождения максимальной площади. | double\*\*\* figures, int count, fstream& protocol, fstream& output |  |  |  | void | Выводит в файлы вывода и протокола результат. |
| CirclesFounder | Основной метод нахождения точек на окружности. | double\*\* points, int n, double\*\*\* figures, int count, fstream& protocol, fstream& output |  |  | n, double xc, yc, x, y, double\*\*\* pt, int colvo | void | Создает новый динамический массив для точек. |
| CheckPeretasovka | Проверят одинаковые квадраты. | double\*\*\* rectangles, int count, double A[2], double B[2], double C[2], double D[2], int n | 1/0 |  |  | bool |  |
| main | Главная функция. |  |  |  |  | int |  |

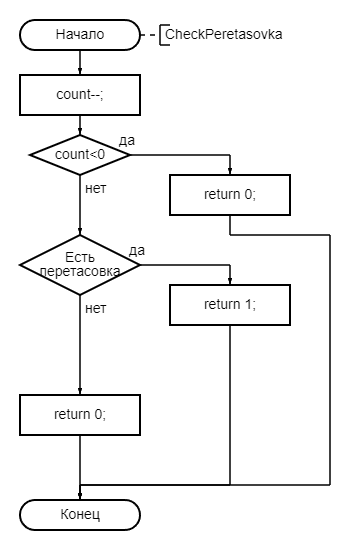
**Описание алгоритма.**

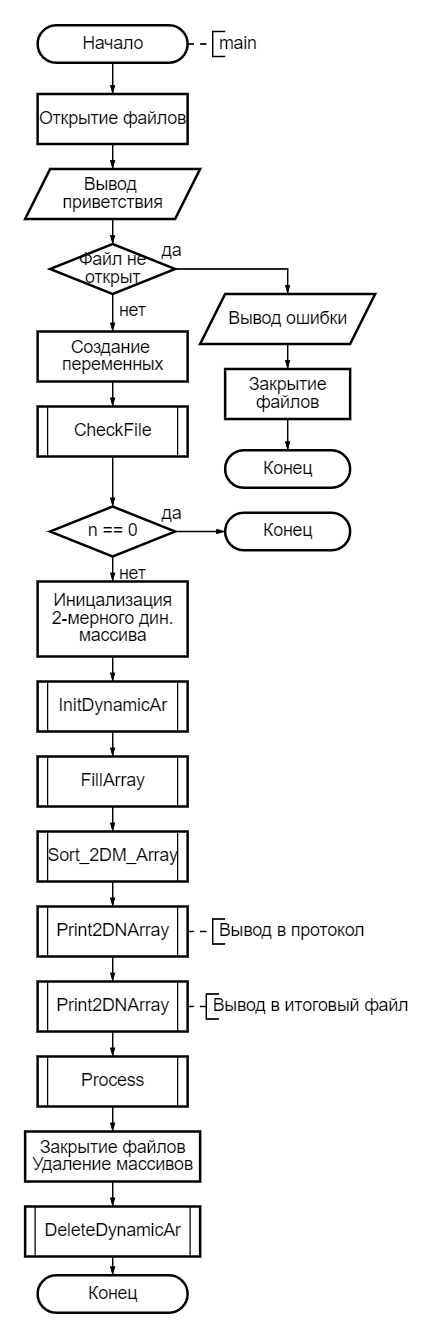
Перед тем как находить фигуры элементы из файла читаются в методе CheckFile. В нем идет сохранение позиции каждой строки в файле, если там нету ошибки и можно считать два числа. Далее метод FillArray занесет все точки из файла в двумерный массив точек.

Алгоритм представляет из себя последовательное нахождение элементов фигуры. В методе Process идет перебор точек, большое количество их проверок, и в конечном итоге сохранение всех найденных квадратов. Далее, в методе CirclesFounder происходит перебор всех точек для каждой стороны и, при успешном нахождении точек, создание массива, который хранит в себе точки по схеме – точка квадрата – точка на окружности. Далее метод SquareFinder находит площадь каждой фигуры, максимальную площадь одной или нескольких одинаковых фигур, и выводит результат в итоговый файл.

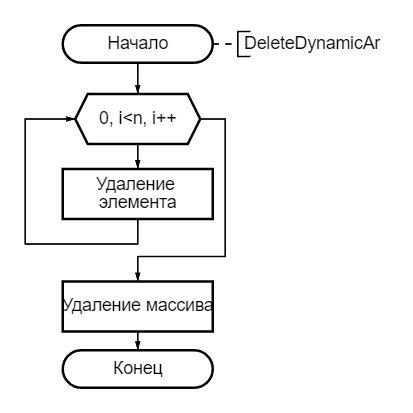
**Алгоритм решения.**

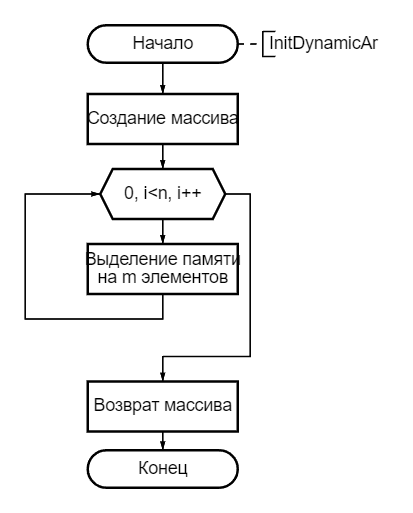
int main: CheckPeretasovka:



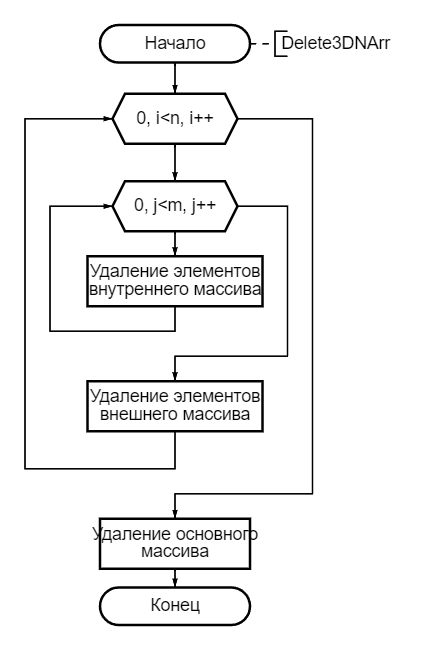
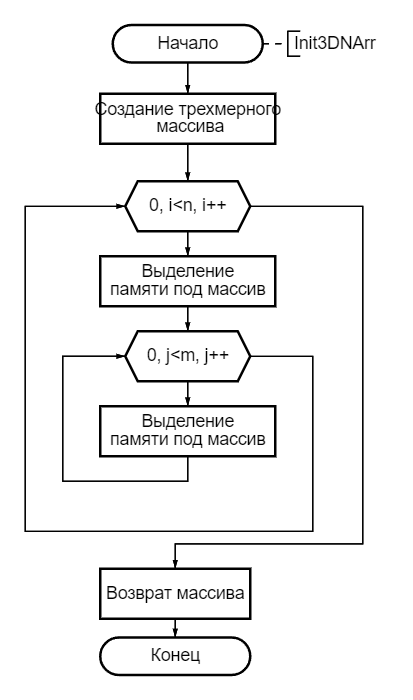


InitDynamicAr: DeleteDynamicAr:

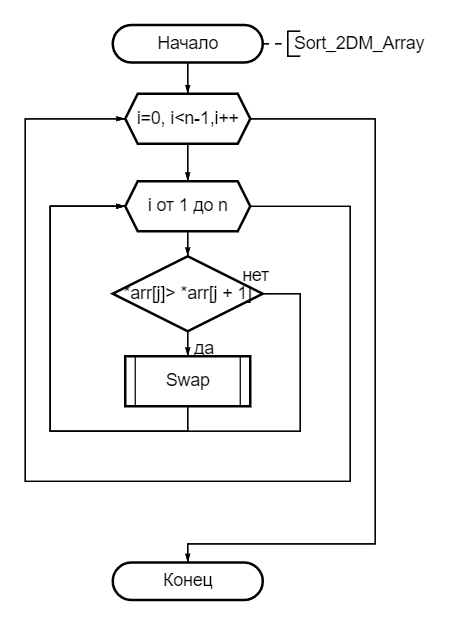
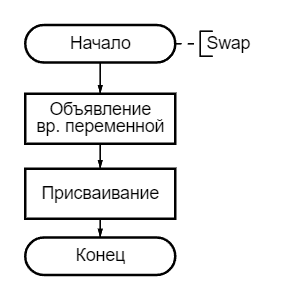




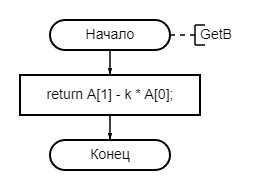
Init3DNArr: Delete3DNArr:

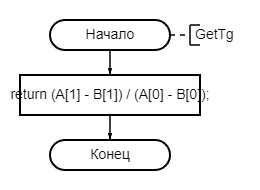


Swap: Sort\_2DM\_Array:

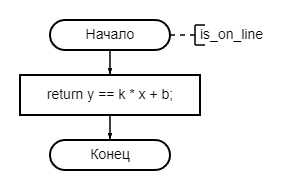
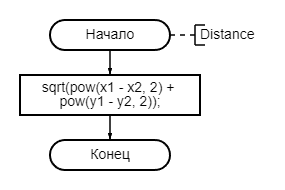


GetTg: GetB:

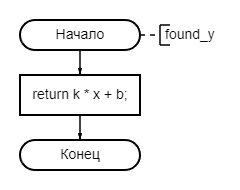
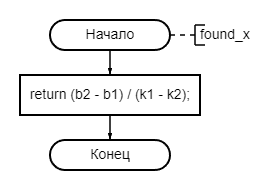




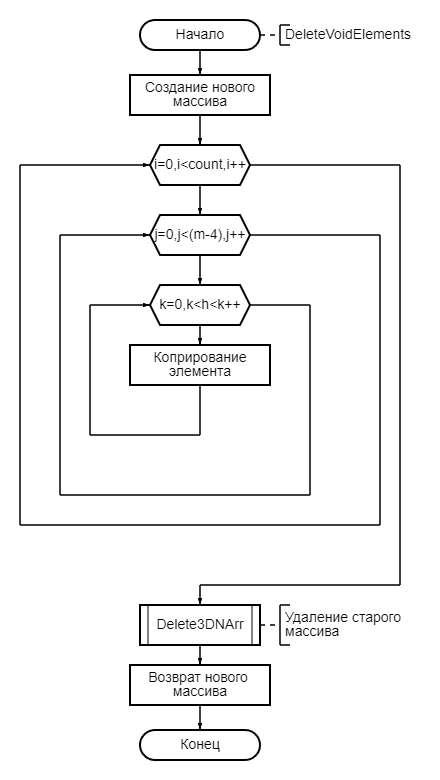
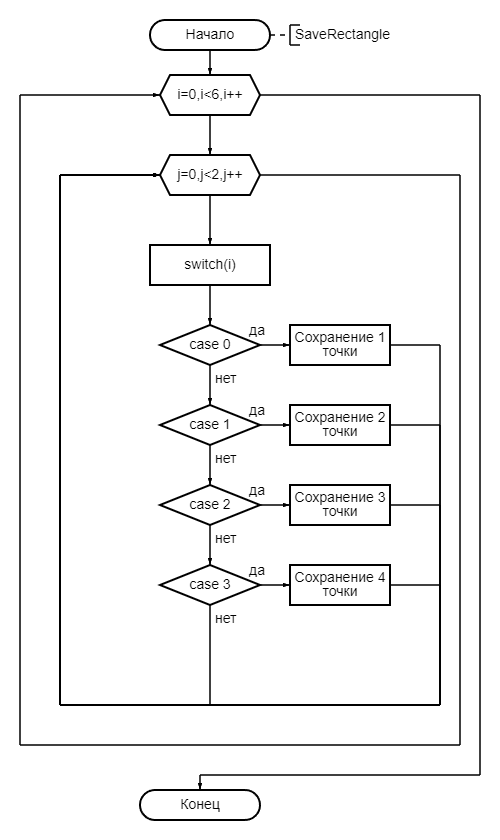
Distance: is\_on\_line:



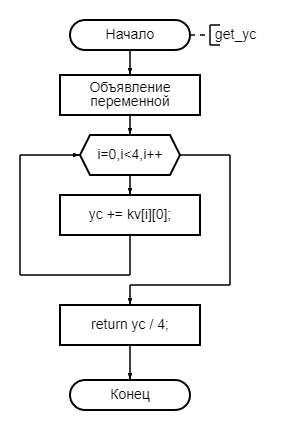
found\_x: found\_y:

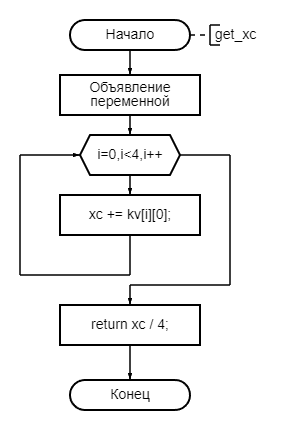


SaveRectangle: Delete3DNArr:

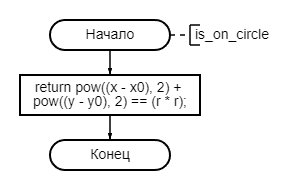


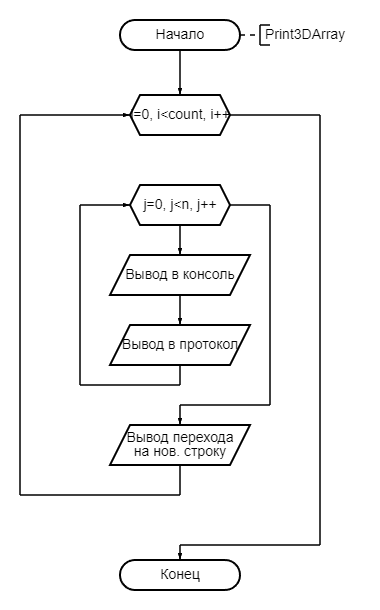
get\_xc: get\_yc:



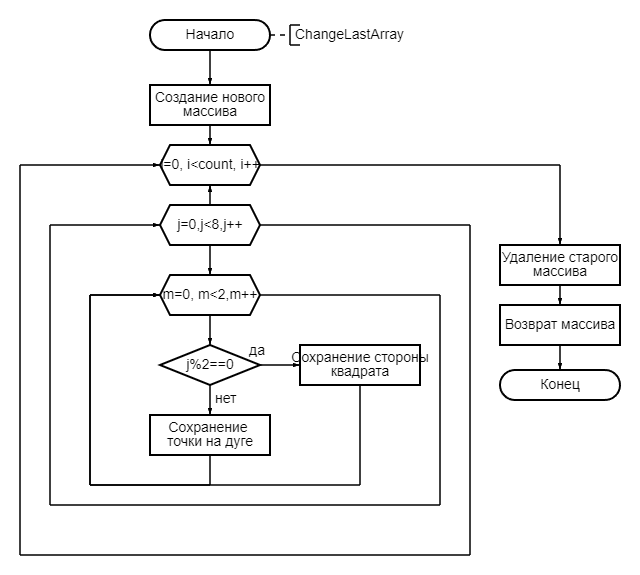


Print3DArray: is\_on\_circle:

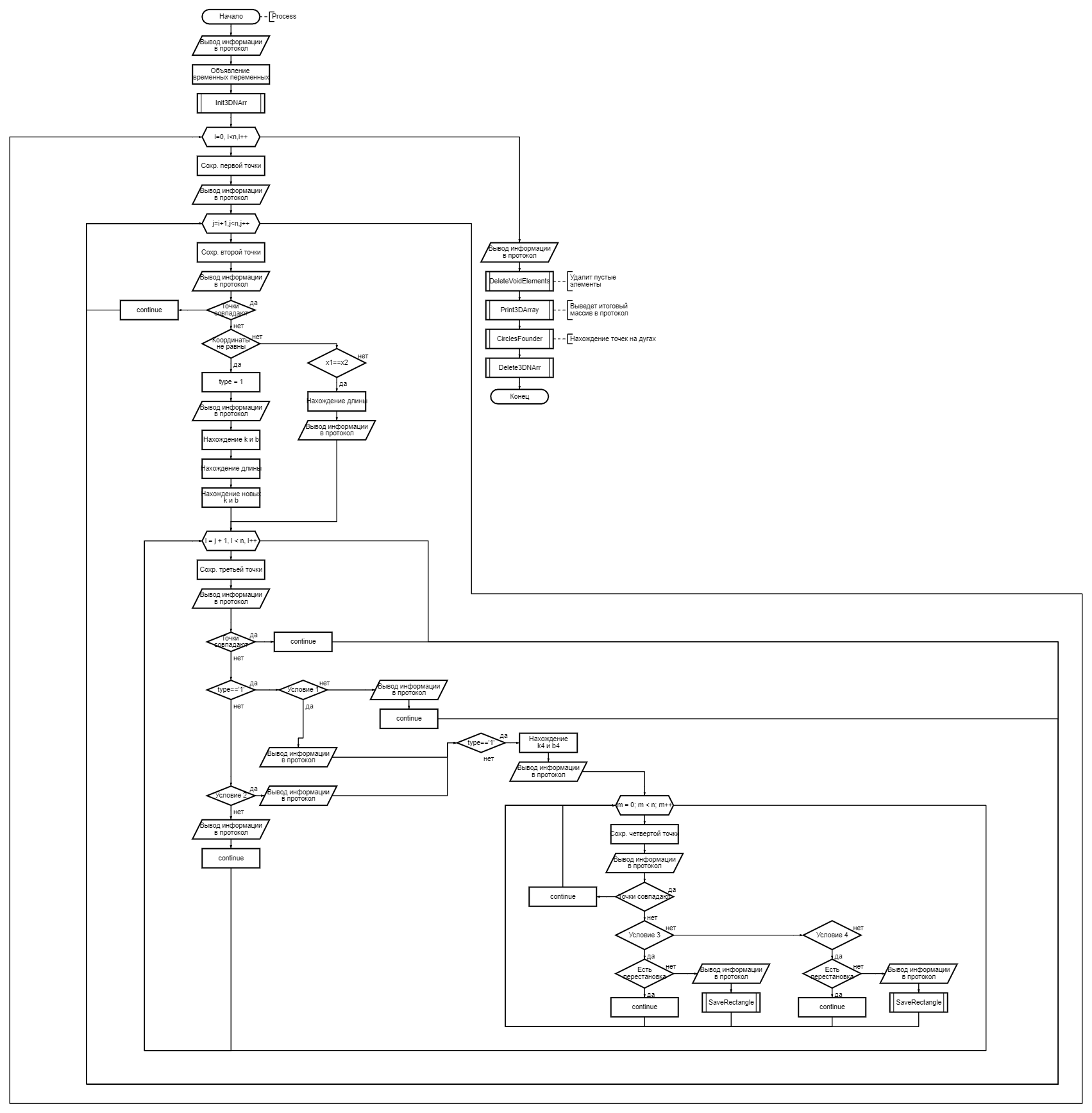
****



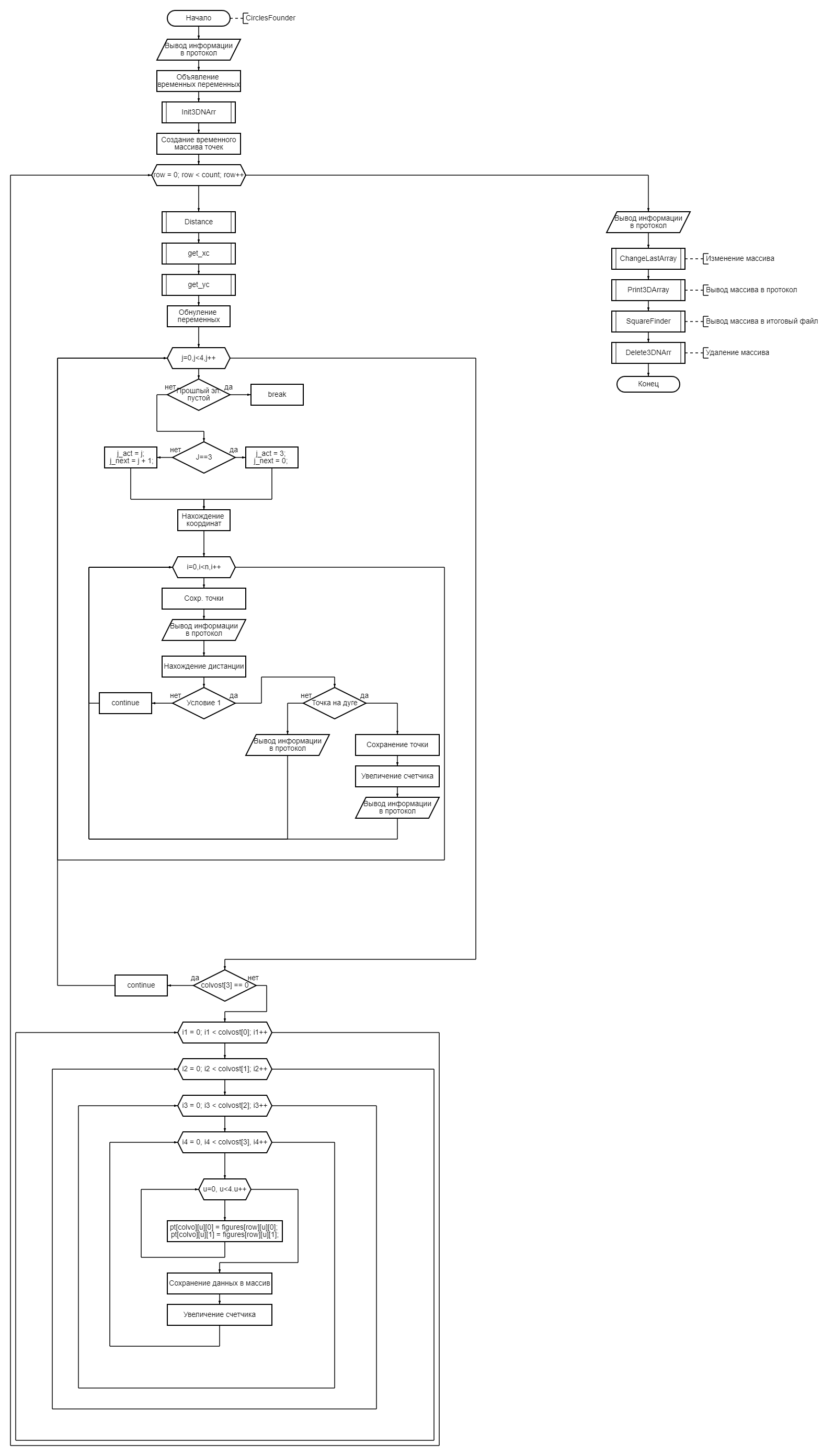
ChangeLastArray:



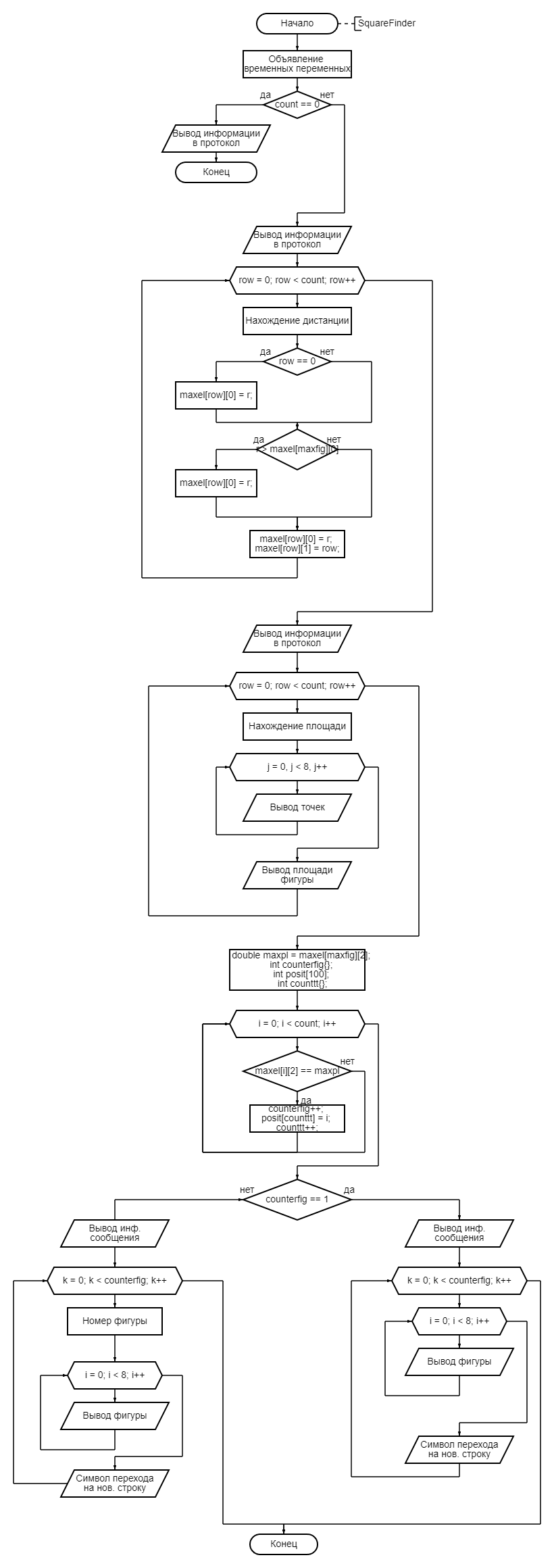
Process:



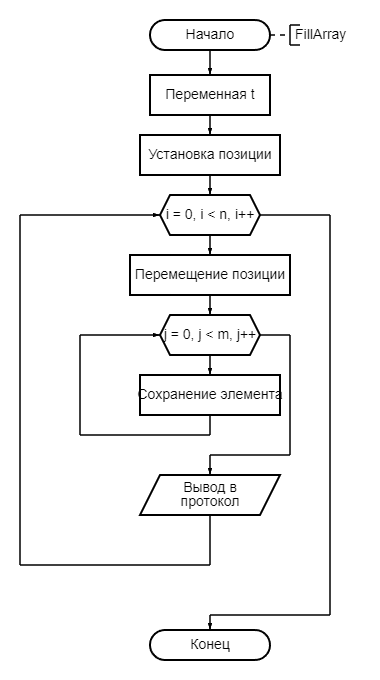
CirclesFounder:

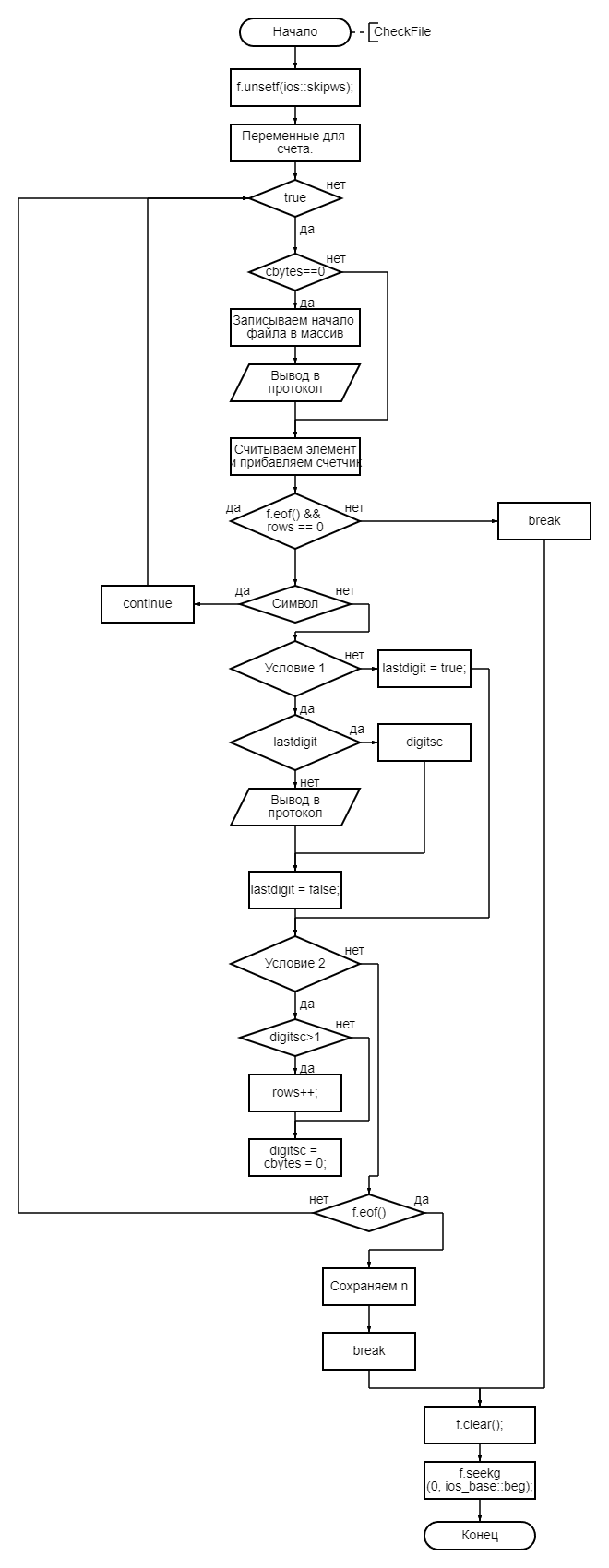


SquareFinder:



CheckFile: FillArray:





**Программа.**

#include <iostream>

#include <fstream>

#include <ctime>

#include <math.h>

#include <iomanip>

/\*

1. Обработка в файл с протоколом.

2. Проверить везде удаление массивов

3. Если нет фигур, надо обработать.

\*/

using namespace std;

void CheckFile(unsigned&, unsigned, fstream&, int\* positions, fstream&);

void FillArray(double\*\*, int n, int m, fstream&, int\*, fstream&);

void Sort\_2DM\_Array(double\*\* arr, int n);

void Process(double\*\*, unsigned, unsigned, fstream&, fstream&);

double\*\* InitDynamicAr(int n, int m)

{

double\*\* array = new double\*[n];

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

{

array[i] = new double[m];

}

return array;

}

void DeleteDynamicAr(double\*\* array, int n)

{

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

{

delete[] array[i];

}

delete[] array;

}

void Print2DNArray(double\*\* points, int n, int m, fstream& protocol, bool toconsole = true)

{

if(toconsole)

cout << endl;

protocol << endl;

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

{

for (int j = 0; j < m; j++)

{

if(toconsole)

cout << setw(8) << left << setprecision(8) << points[i][j];

protocol << setw(8) << left << setprecision(8) << points[i][j];

}

if(toconsole)

cout << endl;

protocol << "\n";

}

}

int main()

{

setlocale(0, "Russian");

fstream file("Points.txt", ios::in); // Файл с координатами точек.

fstream protocol("Protocol.txt", ios::out); // Файл протокола.

fstream ofile("Output.txt", ios::out); // Файл на вывод данных

ofile << "Курсовая работа. Рябов Вадим. 3352.\n"

<< "Задание 8.1 Дано N произвольных точек на плоскости. Найти среди них точки, являющиеся вершинами\n"

<< "(образующими)фигур, обладающих следующими свойствами : 1) фигура с максимальной площадью\n";

if (!file.is\_open())

{

protocol << "Ошибка. Исходный файл не найден.";

protocol.close(); ofile.close();

return 0;

}

protocol << "[main] Файлы Points.txt, Protocol.txt, Output.txt успешно открыты.\n";

int\* positions = new int[10000];

protocol << "[main] Создан массив позиций на 10000 элементов.\n";

unsigned n{}; unsigned m = 2;

protocol << "[main] Производится первоначальная проверка файла.\n";

CheckFile(n, m, file, positions, protocol); // Проверка файла на ввод

unsigned org\_n = n;

if (n==0)

{

protocol << "[main] Точки отсутсвуют. Конец программы. \n";

return 0;

}

double\*\* points = InitDynamicAr(n,m); // объявляем массив точек

protocol << "[main] Создан массив точек на " << n << " элементов.\n";

FillArray(points, n, m, file, positions, protocol); // заполняем массив точек

protocol << "[main] Массив точек заполнен успешно.\n";

Sort\_2DM\_Array(points, n);

protocol << "[main] Массив точек успешно отсортирован по возрастанию.\n";

//DelDuplicates(points, n, protocol);

protocol << "[main] Массив точек готов к дальнейшему использованию.\nВыведем получившиеся точки на экран:";

Print2DNArray(points, n, 2, protocol);

ofile << "\nПолучившийся массив точек: \n";

Print2DNArray(points, n, 2, ofile, false);

protocol << "[main] Выполним основной метод программы. \n";

Process(points, n, m, protocol, ofile); // выполняем метод проверки точек

protocol << "[main] Удаляем массивы, закрываем файлы, программа успешно завершена.\n";

protocol.close(); ofile.close(); file.close();

delete[] positions;

DeleteDynamicAr(points, org\_n);

}

void Swap(double\* a1, double\* a2)

{

double tmp;

tmp = a1[0]; a1[0] = a2[0]; a2[0] = tmp;

tmp = a1[1]; a1[1] = a2[1]; a2[1] = tmp;

}

void Sort\_2DM\_Array(double\*\* arr, int n)

{

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

{

for (int j = 0; j < n-i-1; j++)

{

if (\*arr[j]> \*arr[j + 1])

{

Swap(arr[j], arr[j + 1]);

}

}

}

}

double GetTg(double A[2], double B[2]) // находит коэффициент к

{

return (A[1] - B[1]) / (A[0] - B[0]);

}

double GetB(double A[2], double k) // находит коэффициент b

{

return A[1] - k \* A[0];

}

double Distance(double x1, double y1, double x2, double y2) // расстояние между точками

{

return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));

}

bool is\_on\_line(double x, double y, double k, double b) // проверка на нахождение точки на прямой

{

return y == k \* x + b;

}

double found\_x(double b1, double b2, double k1, double k2)

{

return (b2 - b1) / (k1 - k2);

}

double found\_y(double x, double k, double b)

{

return k \* x + b;

}

// создание и удаление 3-х мерного массива квадратов

double\*\*\* Init3DNArr(unsigned n, int m = 8, int h = 2)

{

double\*\*\* three\_dn\_arr = new double\*\* [n];

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

{

three\_dn\_arr[i] = new double\* [m];

for (int j = 0; j < m; j++)

{

three\_dn\_arr[i][j] = new double[h] {};

}

}

return three\_dn\_arr;

}

void Delete3DNArr(double\*\*\* arr, unsigned n, int m = 8, int h = 2)

{

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

{

for (int j = 0; j < m; j++)

delete [] arr[i][j];

delete[] arr[i];

}

delete[] arr;

}

// Метод для сохранения квадрата в массив

void SaveRectangle(int p, double A[2], double B[2], double C[2], double D[2], double\*\*\* rectangles, unsigned n)

{

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 2; j++)

{

switch (i)

{

case 0:

rectangles[p][i][j] = A[j];

break;

case 1:

rectangles[p][i][j] = B[j];

break;

case 2:

rectangles[p][i][j] = C[j];

break;

case 3:

rectangles[p][i][j] = D[j];

break;

}

}

}

}

// Метод, который удаляет пустые элементы в массиве квадратов

double\*\*\* DeleteVoidElements(double\*\*\* arr, int count, unsigned n, int m = 8, int h = 2)

{

double\*\*\* new\_arr = Init3DNArr(count);

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

{

for (int j = 0; j < (m-4); j++)

{

for (int k = 0; k < h; k++)

{

new\_arr[i][j][k] = arr[i][j][k]; // копирование элемента

}

}

}

Delete3DNArr(arr, n);

return new\_arr;

}

double get\_xc(double\*\* kv)

{

double xc{};

for (int i = 0; i < 4; i++)

{

xc += kv[i][0];

}

return xc / 4;

}

double get\_yc(double\*\* kv)

{

double yc{};

for (int i = 0; i < 4; i++)

{

yc += kv[i][1];

}

return yc / 4;

}

bool is\_on\_circle(double x0, double y0, double r, double x, double y)

{

return pow((x - x0), 2) + pow((y - y0), 2) == (r \* r);

}

void Print3DArray(double\*\*\* rectangles, int count, fstream& protocol, int n = 8)

{

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

{

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

{

cout << "[" << rectangles[i][j][0] << ";" << rectangles[i][j][1] << "] ";

protocol << "[" << rectangles[i][j][0] << ";" << rectangles[i][j][1] << "] ";

}

cout << endl;

protocol << endl;

}

}

double\*\*\* ChangeLastArray(double\*\*\* figures, int count, int originalc)

{

double\*\*\* new\_array = Init3DNArr(count);

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

{

for (int j = 0; j < 8; j++)

{

for (int m = 0; m < 2; m++)

{

if (j % 2 == 0)

{

new\_array[i][j][m] = figures[i][j/2][m];

}

else

{

new\_array[i][j][m] = figures[i][j+(3-(j/2))][m];

}

}

}

}

Delete3DNArr(figures, originalc);

return new\_array;

}

// Нахождение фигуры с максимальной площадью

void SquareFinder(double\*\*\* figures, int count, fstream& protocol, fstream& output)

{

double maxel[100][3]{};

int maxfig{};

double r{};

if (count == 0)

{

protocol << "[SquareFinder] Среди точек в программе не найдено ни одной фигуры. \n";

output << "Среди точек в программе не найдено ни одной фигуры. \n";

cout << "[SquareFinder] Среди точек в программе не найдено ни одной фигуры. \n";

return;

}

protocol << "[SquareFinder] Переберем все фигуры и найдем максимальную площадь: \n";

for (int row = 0; row < count; row++)

{

r = Distance(figures[row][0][0], figures[row][0][1], figures[row][2][0], figures[row][2][1]);

if (row == 0)

{

maxel[row][0] = r;

}

if (r > maxel[maxfig][0])

{

maxfig = row;

}

maxel[row][0] = r;

maxel[row][1] = row;

}

protocol << "\n[SquareFinder] Площади найденных фигур: \n";

output << "\nПлощади найденных фигур: \n";

cout << "\n[SquareFinder] Площади найденных фигур: \n";

double square;

for (int row = 0; row < count; row++)

{

square = maxel[row][0] \* maxel[row][0] \* 2.57;

maxel[row][2] = square;

for (int j = 0; j < 8; j++)

{

cout << "[" << figures[row][j][0] << ";" << figures[row][j][1] << "] ";

protocol << "[" << figures[row][j][0] << ";" << figures[row][j][1] << "] ";

output << "[" << figures[row][j][0] << ";" << figures[row][j][1] << "] ";

}

cout << "Площадь S= " << square << endl;

protocol << "Площадь S= " << square << endl;

output << "Площадь S= " << square << endl;

}

double maxpl = maxel[maxfig][2];

int counterfig{};

int posit[100];

int counttt{};

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

{

if (maxel[i][2] == maxpl)

{

counterfig++;

posit[counttt] = i;

counttt++;

}

}

if (counterfig == 1)

{

output << endl << "Среди всех точек программы найдена фигура, которая имеет самую большую площадь S = " << maxpl << "\n";

protocol << endl << "Среди всех точек программы найдена фигура, которая имеет самую большую площадь S = " << maxpl << "\n";

cout << endl << "Среди всех точек программы найдена фигура, которая имеет самую большую площадь S = " << maxpl << "\n";

for (int i = 0; i < 8; i++)

{

protocol << "[" << figures[maxfig][i][0] << ";" << figures[maxfig][i][1] << "] ";

cout << "[" << figures[maxfig][i][0] << ";" << figures[maxfig][i][1] << "] ";

output << "[" << figures[maxfig][i][0] << ";" << figures[maxfig][i][1] << "] ";

}

}

else

{

output << endl << "Среди всех точек программы найдено " << counterfig << " фигуры, которые имеют самую большую площадь S = " << maxpl << "\n";

protocol << endl << "Среди всех точек программы найдена фигура, которая имеет самую большую площадь S = " << maxpl << "\n";

cout << endl << "Среди всех точек программы найдено " << counterfig << " фигуры, которые имеют самую большую площадь S = " << maxpl << "\n";

for (int k = 0; k < counterfig; k++)

{

output << k + 1 << "-ая фигура\n";

for (int i = 0; i < 8; i++)

{

protocol << "[" << figures[posit[k]][i][0] << ";" << figures[posit[k]][i][1] << "] ";

cout << "[" << figures[posit[k]][i][0] << ";" << figures[posit[k]][i][1] << "] ";

output << "[" << figures[posit[k]][i][0] << ";" << figures[posit[k]][i][1] << "] ";

}

output << endl;

}

}

}

void CirclesFounder(double\*\* points, int n, double\*\*\* figures, int count, fstream& protocol, fstream& output)

{

protocol << "[CirclesFounder] Найдем все окружности для массива.\n";

double x, y;

double xc, yc;

double r, rc;

double xt, yt; // temp coordinates polovini storoni

int colvost[4] = {};

double points\_on\_side[4][100][2]{};

double\*\*\* pt = Init3DNArr(n);

int j\_act, j\_next; // для перебора сторон квадрата

int colvo{};

for (int row = 0; row < count; row++)

{

r = Distance(figures[row][0][0], figures[row][0][1], figures[row][1][0], figures[row][1][1]); //находим сторону

xc = get\_xc(figures[row]); yc = get\_yc(figures[row]); // находим координаты центра

protocol << "[CirclesFounder] Рассматриваем 1 квадрат. Сторона - " << r << " Координаты центра - [" << xc << ";" << yc << "]\n";

fill\_n(colvost, 4, 0);

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 100; j++)

{

fill\_n(points\_on\_side[i][j], 2, 0);

}

}

for (int j = 0; j < 4; j++)

{

if (j != 0 && colvost[j - 1] == 0)

{

protocol << "[CirclesFounder] Не нашли все точки, завершаем цикл. \n";

break;

}

//координаты точек, небходимых к перебору

if (j == 3)

{

j\_act = 3; j\_next = 0;

}

else

{

j\_act = j; j\_next = j + 1;

}

// координаты точки на стороне

xt = (figures[row][j\_act][0] + figures[row][j\_next][0]) / 2;

yt = (figures[row][j\_act][1] + figures[row][j\_next][1]) / 2;

protocol << "[CirclesFounder] Координаты точки на стороне - [" << xt << ";" << yt <<"]\n";

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

{

x = points[i][0]; y = points[i][1];

protocol << "[CirclesFounder] Рассмотрим точку - [" << x << ";" << y << "]\n";

rc = Distance(xc, yc, x, y); // расстояние от точки до центра

if (rc <= (sqrt((r \* r) + (r \* r)) / 2) || rc > r)

{

protocol << "[CirclesFounder] Точка слишком далеко от стороны. \n";

continue;

}

if (is\_on\_circle(xt, yt, (r/2), x, y))

{

protocol << "[CirclesFounder] Точка подходит под все условия, сохраняем ее. \n";

points\_on\_side[j][colvost[j]][0] = x;

points\_on\_side[j][colvost[j]][1] = y;

colvost[j]++;

}

else

{

protocol << "[CirclesFounder] Точка не лежит на четверти окружности. \n";

}

}

}

if (colvost[3] == 0)

{

protocol << "[CirclesFounder] Не нашли все точки, завершаем цикл. \n";

continue;

}

for (int i1 = 0; i1 < colvost[0]; i1++)

{

for (int i2 = 0; i2 < colvost[1]; i2++)

{

for (int i3 = 0; i3 < colvost[2]; i3++)

{

for (int i4 = 0; i4 < colvost[3]; i4++)

{

for (int u = 0; u < 4; u++)

{

pt[colvo][u][0] = figures[row][u][0];

pt[colvo][u][1] = figures[row][u][1];

}

pt[colvo][4][0] = points\_on\_side[0][i1][0];

pt[colvo][4][1] = points\_on\_side[0][i1][1];

pt[colvo][5][0] = points\_on\_side[1][i2][0];

pt[colvo][5][1] = points\_on\_side[1][i2][1];

pt[colvo][6][0] = points\_on\_side[2][i3][0];

pt[colvo][6][1] = points\_on\_side[2][i3][1];

pt[colvo][7][0] = points\_on\_side[3][i4][0];

pt[colvo][7][1] = points\_on\_side[3][i4][1];

colvo++;

protocol << "[CirclesFounder] Сохраняем в массив фигуру. \n";

}

}

}

}

}

cout << endl;

protocol << "[CirclesFounder] Нашли ровно " << colvo << " фигур. Изменим порядок точек в фигурах и выведем их на экран. \n";

pt = ChangeLastArray(pt, colvo, n);

protocol << "[CirclesFounder] Получим фигуры: \n";

Print3DArray(pt, colvo, protocol);

output << "Получим фигуры: \n";

Print3DArray(pt, colvo, output);

protocol << "[CirclesFounder] Выполним итоговый метод для нахождения максимальной площади. \n";

SquareFinder(pt, colvo, protocol, output);

Delete3DNArr(pt, colvo);

}

bool CheckPeretasovka(double\*\*\* rectangles, int count, double A[2], double B[2], double C[2], double D[2], int n)

{

count--;

if (count < 0)

return 0;

if ((rectangles[count][0][0] == A[0] && rectangles[count][0][1] == A[1]) && (rectangles[count][1][0] == D[0] && rectangles[count][1][1] == D[1]) && (rectangles[count][2][0] == C[0] && rectangles[count][2][1] == C[1]) && (rectangles[count][3][0] == B[0] && rectangles[count][3][1] == B[1]))

return 1;

return 0;

}

void Process(double\*\* ar, unsigned n, unsigned m, fstream& protocol, fstream& output) //главный метод

{

protocol << "[Process] Выполняется главный метод. \n";

// S = a^2 \* 2.57

char type{}; // переменная типа

int count = 0;

double x1, y1, x2, y2, x3, y3, x4, y4; // Переменные для перебора координат.

double k, b, k\_per, b\_per, r{}; // Переменные для перебора коэффициентов прямых

double k4, b4; // для 4 точки

double xc, yc, kc1, bc1, kc2, bc2; // для центра квадрата

double\*\*\* rectangles = Init3DNArr(n);

for (int i = 0; i < n; i++) // взятие 1 точки

{

x1 = ar[i][0]; y1 = ar[i][1]; // Сохраняем координаты

protocol << "[Process] Берем 1 точку с координатами - [" << x1 << ";" << y1 << "]\n";

for (int j = i + 1; j < n; j++) // взятие 2 точки

{

x2 = ar[j][0]; y2 = ar[j][1]; // Сохраняем координаты

protocol << "[Process] Берем 2 точку с координатами - [" << x2 << ";" << y2 << "]\n";

if (x1 == x2 && y1 == y2) continue; // 2 одинаковых точки

if (x1 != x2 && y1 != y2) // 1 случай, когда прямая имеет наклон

{

type = '1';

protocol << "[Process] Прямая имеет наклон, находим ее угол наклона к и коэффициент b. \n";

k = GetTg(ar[i], ar[j]);

b = GetB(ar[j], k);

protocol << "[Process] K = " << k << " B = " << b << "\n";

// Находим k и b новой прямой

k\_per = -1 / k;

b\_per = y2 - k\_per \* x2;

protocol << "[Process] Для новой прямой K = " << k\_per << " B = " << b\_per << "\n";

// Высчитываем ддину прямой

r = Distance(x1, y1, x2, y2);

}

else if (x1 == x2) // если переменная вида x=c то расстояние считаем как разность y

{

type = '2';

protocol << "[Process] Прямая не имеет наклон, находим ее длину. \n";

r = abs(y2 - y1);

}

protocol << "[Process] Длина прямой " << r << endl;

for (int l = j + 1; l < n; l++) // 3 точка

{

x3 = ar[l][0]; y3 = ar[l][1];

protocol << "[Process] Берем 3 точку с координатами - [" << x3 << ";" << y3 << "]\n";

// Проверяем совпадение точек

if (x1 == x3 && y1 == y3 || x2 == x3 && y2 == y3) continue;

if (type == '1')

{

if (is\_on\_line(x3, y3, k\_per, b\_per) && r == Distance(x2, y2, x3, y3))

{

cout << endl << "[Process] Случай 1:\n" << "Точка A - [" << x1 << ";" << y1 << "] \nТочка B - [" << x2 << ";" << y2 << "] \nТочка C - [" << x3 << ";" << y3 << "]" << endl;

protocol << "[Process] Случай 1:\n" << "Точка A - [" << x1 << ";" << y1 << "] \nТочка B - [" << x2 << ";" << y2 << "] \nТочка C - [" << x3 << ";" << y3 << "]" << endl;

}

else

{

protocol << "[Process] 3 точка не подходит под условия, пропускаем её. \n";

continue;

}

}

else if (type == '2')

{

if (y3 == y2 && r == Distance(x2, y2, x3, y3))

{

cout << endl << "[Process] Случай 2:\n" << "Точка A - [" << x1 << ";" << y1 << "] \nТочка B - [" << x2 << ";" << y2 << "] \nТочка C - [" << x3 << ";" << y3 << "]" << endl;

protocol << "[Process] Случай 2:\n" << "Точка A - [" << x1 << ";" << y1 << "] \nТочка B - [" << x2 << ";" << y2 << "] \nТочка C - [" << x3 << ";" << y3 << "]" << endl;

}

else

{

protocol << "[Process] 3 точка не подходит под условия, пропускаем её. \n";

continue;

}

}

// Составим уравнение прямой, двигая основную прямую

if (type == '1')

{

k4 = k;

b4 = GetB(ar[l], k4);

protocol << "[Process] Координаты новой прямой для проверки 4 точки: . \n";

protocol << "[Process] K = " << k4 << " B = " << b4 << "\n";

}

for (int m = 0; m < n; m++) // берем четвертую точку

{

x4 = ar[m][0]; y4 = ar[m][1];

protocol << "[Process] Берем 4 точку с координатами - [" << x4 << ";" << y4 << "]\n";

// Проверяем совпадение точек

if ((x4 == x1 && y4 == y1) || (x4 == x2 && y4 == y2) || (x4 == x3 && y4 == y3)) continue;

if (type == '1' && is\_on\_line(x4, y4, k4, b4) && r == Distance(x3, y3, x4, y4) && r == Distance(x1, y1, x4, y4))

{

if (CheckPeretasovka(rectangles, count, ar[i], ar[j], ar[l], ar[m], n))

continue;

cout << " 1 Точка D - [" << x4 << "; " << y4 << "]" << endl;

protocol << " [Process] 1 Случай, найдена точка D - [" << x4 << "; " << y4 << "]" << endl;

SaveRectangle(count++, ar[i], ar[j], ar[l], ar[m],rectangles, n);

}

else if (type == '2' && x3 == x4 && y4 == y1)

{

if (CheckPeretasovka(rectangles, count, ar[i], ar[j], ar[l], ar[m], n))

continue;

cout << " 2 Точка D - [" << x4 << "; " << y4 << "]" << endl;

protocol << " [Process] 2 Случай, найдена точка D - [" << x4 << "; " << y4 << "]" << endl;

SaveRectangle(count++, ar[i], ar[j], ar[l], ar[m], rectangles, n);

}

}

}

}

}

protocol << "[Process] В итоге мы нашли " << count << " квадратов. Изменим размер массива.\n";

rectangles = DeleteVoidElements(rectangles, count, n);

// Вывод массива с квадратами.

protocol << "\n\n[Process] Выведем получившийся массив:\n";

Print3DArray(rectangles, count, protocol, 4);

protocol << "[Process] Выполним метод поиска окружностей.\n";

CirclesFounder(ar, n, rectangles, count, protocol, output);

protocol << "[Process] Удалим массив.\n";

Delete3DNArr(rectangles, count);

}

void CheckFile(unsigned& n, unsigned m, fstream& f, int\* positions, fstream& protocol)

{

f.unsetf(ios::skipws);

//Переменные для счета

char s; // Временная переменная для вывода

unsigned digitsc = 0; //Кол-во цифр в строке

bool lastdigit = true; //Был ли последний симол - цифрой

unsigned cbytes = 0; // Общее число байтов

unsigned rows = 0; // Полученное кол-во строк

positions[0] = (int)f.tellg();

while (true)

{

if (cbytes == 0) // Записываем начало файла в массив

{

positions[rows] = (int)f.tellg();

protocol << "[checkfile] Сохраняем позицию в массив. Строка -"<< rows << " Позиция - " << positions[rows] << "\n";

}

//счетчик чисел в строке

f >> s; cbytes++;

if (f.eof() && rows == 0)

{

break;

}

if (s > 58 && s < 126 || s < 0)

{

protocol << "[checkfile] Обнаружена буква, пропускаем.\n";

continue;

}

// Счет чисел в строке

if (s == ' ' || s == '\n' || f.eof() || s == '\t')

{

if (lastdigit)

{

digitsc++;

protocol << "[checkfile] Засчитана цифра.\n";

}

else

{

protocol << "[checkfile] Цифра не засчитана.\n";

}

lastdigit = false;

}

else

{

lastdigit = true;

}

// Сохранение позиции

if (s == '\n' || f.eof())

{

// основные вычисления

if (digitsc>1)

{

rows++;

protocol << "[checkfile] Переход на следующую строку.\n";

}

digitsc = cbytes = 0;

}

// Условие выхода

if (f.eof())

{

//n = rows < n ? rows : n;

protocol << "[checkfile] Достигли конца файла, завершаем выполненние.\n";

n = rows;

break;

}

}

protocol << "[checkfile] Получили в конечном итоге " << "n - " << n << endl;

f.clear();

f.seekg(0, ios\_base::beg);

}

void FillArray(double\*\* array, int n, int m, fstream& f, int\* positions, fstream& protocol)

{

double t;

f.setf(ios::skipws);

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

{

f.seekg(positions[i], ios::beg);

for (int j = 0; j < m; j++)

{

f >> t;

array[i][j] = t;

}

protocol << "[FillArray] успешно заполнили" << i << "-ую строку.\n";

}

}

**Пример работы программы.**

Пример №1

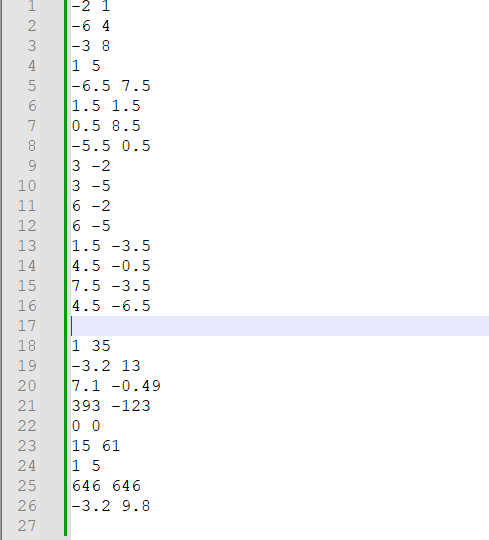


Рис. 8 – Исходный файл

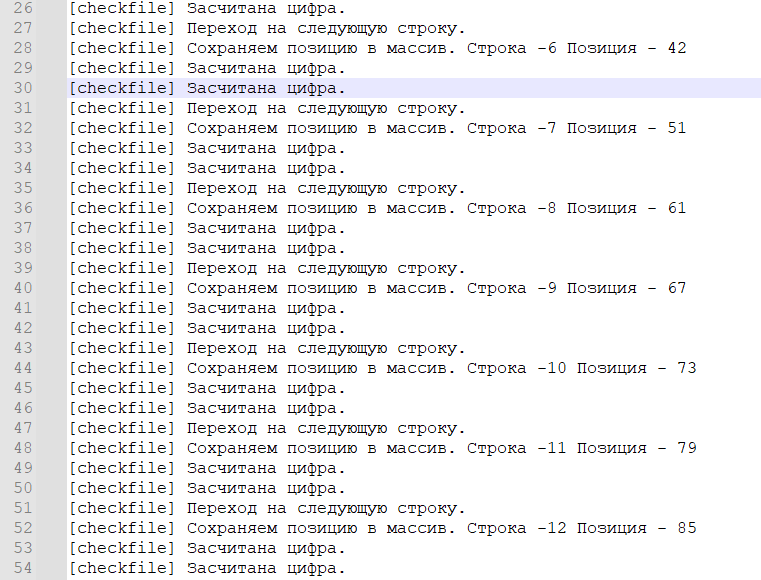
****

Рис. 9 – Фрагмент протокола

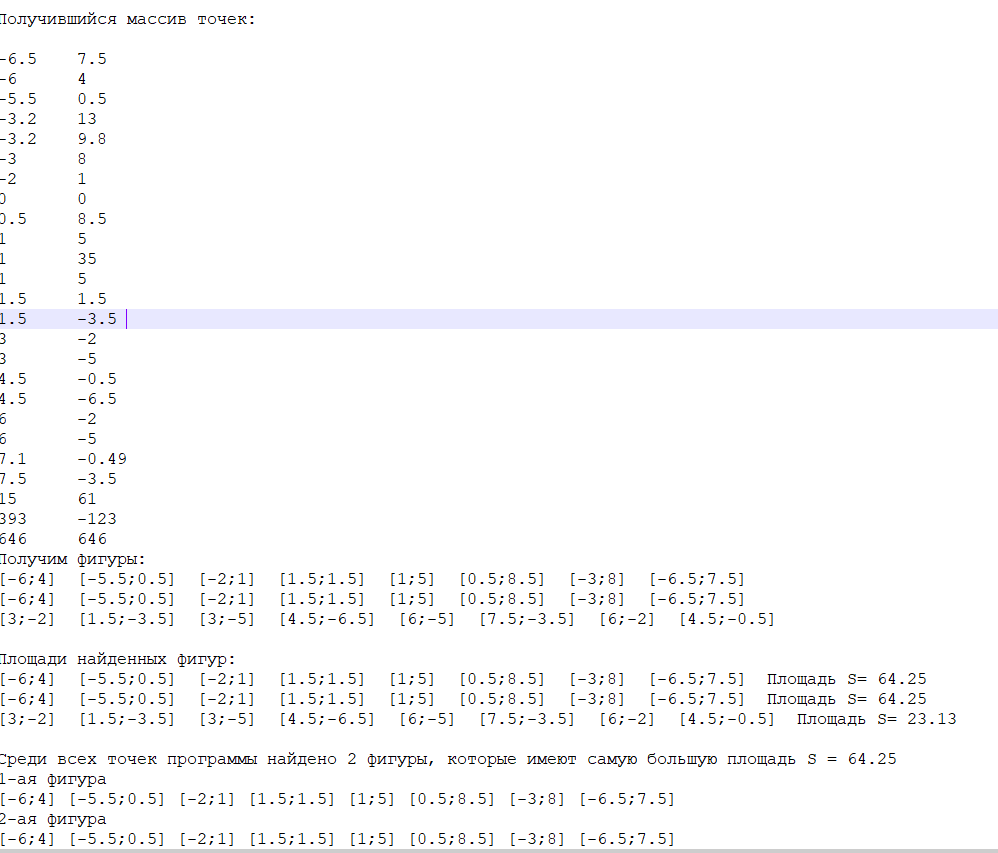


Рис. 10 - Итог

Пример №2

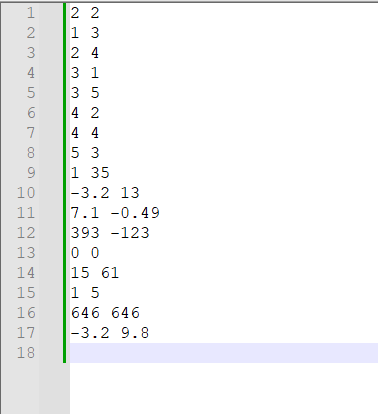


Рис. 11 – Исходный файл

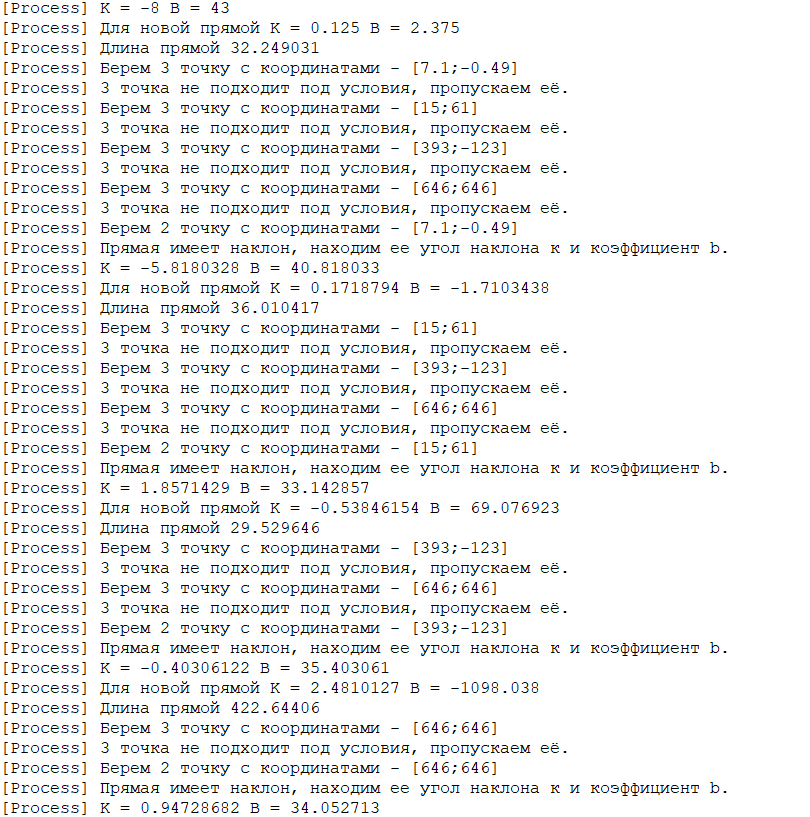


Рис. 12 – Фрагмент протокола

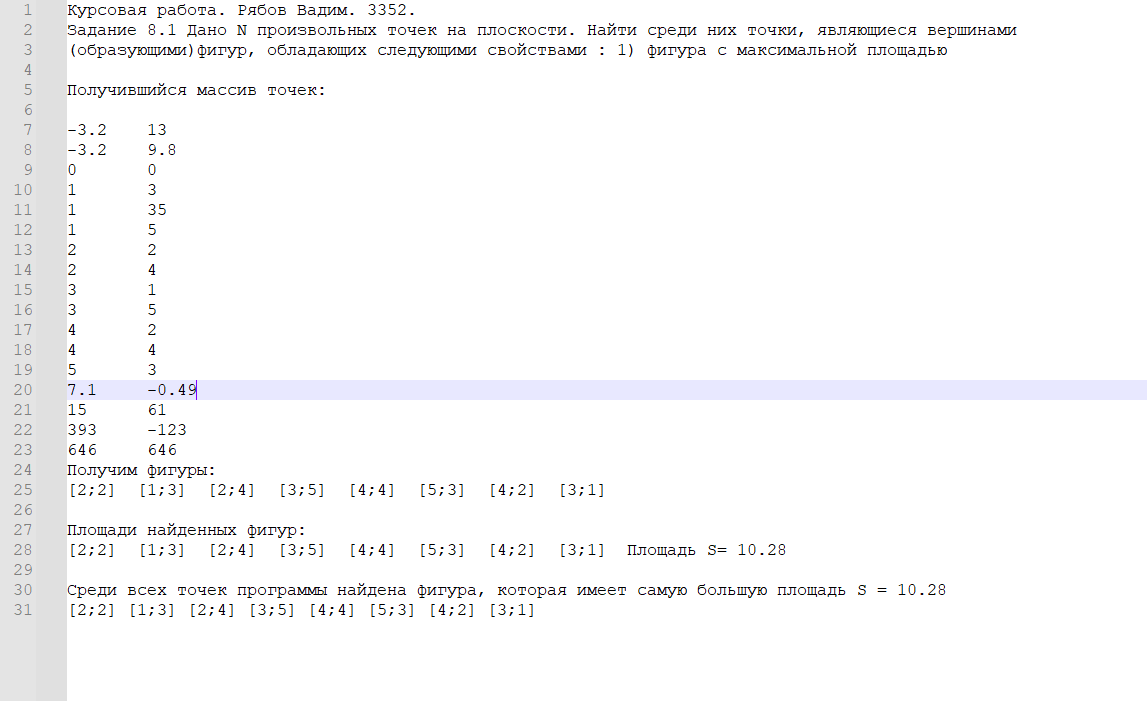


Рис. 13 - Итог

**Выводы.**

В ходе курсовой работы я закрепил все знания языка C++ полученные мною в первом семестре, научился решать сложные задачи с использованием методов и указателей.