|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Двумерный массив»** | |
|  | |
| Выполнил студент группы ИКБО-09-22 | Гришин А. В. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

# **Цель работы**

Получение навыков по определению двумерного массива для структуры данных задачи получение навыков по разработке алгоритмов операций на двумерном массиве в соответствии с задачей.

1. **Постановка задачи**
2. Разработать программу по обработке данных, представленных в задаче матрицей и реализованной в программе двумерным (многомерным) статическим массивом.
3. Максимально допустимые размеры статического массива установить через глобальные константы Rows, Cols.
4. Если тип элемента массива не определен в варианте, определить массив для хранения значений целого типа.
5. Разработать функции по реализации алгоритмов заполнения матрицы: с клавиатуры, датчиком случайных чисел. Разработать функции по реализации алгоритма вывода массива на экран построчно.
6. Выполнить декомпозицию задачи варианта, разработать алгоритм решения. Реализовать функцию, выполняющую задачу и отладить ее.
7. Разработать программу, демонстрирующую работу всех функций.
8. Разработать программу по обработке данных, представленных в задаче матрицей и реализованной в программе двумерным (многомерным) динамическим массивом.
9. Размеры массива должны определяться пользователем.
10. Двумерный массив определить как двойной указатель и выполнить его создание операцией new.
11. Разработать функции по реализации алгоритмов заполнения матрицы: с клавиатуры, датчиком случайных чисел. Разработать функции по реализации алгоритма вывода массива на экран построчно.
12. Выполнить декомпозицию задачи варианта, разработать алгоритм решения. Реализовать функцию, выполняющую задачу и отладить ее.
13. Разработать программу, демонстрирующую работу всех функций.
14. Разработать программу по задаче варианта с реализацией данных задачи с применением контейнера vector библиотеки STL.
15. Реализовать структуру хранения данных на основе шаблона <vector>, размеры определить при вводе с клавиатуры.
16. Разработать функции: заполнение структуры хранения исходных данных, вывода структуры хранения.
17. Выполнить декомпозицию задачи варианта, разработать алгоритм решения. Реализовать функцию, выполняющую задачу и отладить ее.
18. Разработать программу, демонстрирующую работу всех функций.
19. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Персональный вариант – 7.

Таблица 1. Задание варианта для задач 1-2.

|  |
| --- |
| Дана квадратная матрица. Вывести ее элементы, обходя матрицу по часовой стрелке, начиная первой строки. |

Таблица 2. Задание варианта для задачи 3.

|  |
| --- |
| Даны два множества точек на плоскости. Из первого множества выбрать три различные точки так, чтобы треугольник с вершинами в этих точках, содержал (строго внутри себя) равное количество точек первого и второго множества. |

# **Решение**

* 1. **Краткая теория по теме**

Структура данных – это совокупность элементов, связанных друг с другом некоторыми отношениями. Для представления таких данных в программе, используются структуры хранения в оперативной памяти, они хранят значения элементов и отношения между элементами. Элементом структуры хранения является ячейка памяти, предназначенная для хранения значения, а способ соединения ячеек между собой определяет вид структуры хранения: линейная или нелинейная.

Двумерный массив предназначен для представления структур данных элементы которых подчиняются бинарным (2-арным) отношениям. Каждый элемент структуры данных располагается в массиве в ячейке в соответствии с двумя параметрами (их еще называют координатами) - номер строки и номер столбца. Например, для хранения матрицы размером 3×4 очень удобно использовать двумерный массив.

Статический массив – это структура данных, которая создается во время компиляции (т.е. массиву выделяется память). Размер массива нельзя изменять во время работы программы.

Формат определения двумерного статического массива:

|  |
| --- |
| тип имя\_массива[количество\_строк][количество\_столбцов]; |

Например:

|  |
| --- |
| const int rows = 3;  const int cols = 3;  int arr[rows][cols]; //для статического массива размер должен быть константой |

Операции можно выполнять только над элементом массива, допустимые базовым типом массива. Доступ к элементам массива осуществляется по индексу элемента, указав при этом индекс строки элемента.

Определение статического массива arr[rows][cols], где rows, cols – константные значения, можно разделить на составные части: arr[rows] – можно прочитать так: arr – указатель на массив из трех элементов, а скобка [cols] указывает на элемент массива из трех.

Динамический массив – это структура данных, которая создается во время выполнения программы. Данный тип массива отличается от статического тем, что его изначальный размер можно задать в ходе выполнения программы. Динамический массив в языке С++ определяется через указатель. Для выделения динамической памяти под массив мы используем оператор new или функцию malloc().

Формат определения двумерного динамического массива:

|  |
| --- |
| int\*\* arr = new int\* [количество\_строк]; |

Такая запись называется косвенным указателем, т.к. arr является указателем на ячейку, которая так же является указателем. Двойной указатель используется для определения динамического двумерного массива.

Класс вектора стандартной библиотеки C++ — это шаблон класса для контейнеров последовательностей. Вектор хранит элементы заданного типа в линейном расположении и обеспечивает быстрый случайный доступ к любому элементу. Для работы с вектором необходимо подключить заголовок <vector>.

Если считать, что одна строка — это vector<тип\_данных>, то двумерный массив — это вектор элементов типа vector<тип\_данных>, то есть его нужно объявлять, как:

|  |
| --- |
| vector<vector<тип\_данных>> название\_вектора |

Вектор является предпочтительным контейнером для последовательности, так как он удобнее и быстрее, чем тот же динамический массив.

* 1. **. Дополнительные функции**

Для упрощения работы над реализацией программы и повышения читаемости кода введем дополнительные функции, а именно для: вывода матрицы (двумерного массива) на экран для первой и второй задач; вывода точек, на которых строится треугольник, и заполнения вектора точками с координатами для третьей задачи.

|  |
| --- |
| void MatrixOutput(auto arr, int size)  {  for (int i = 0; i < size; i++)  {  for (int k = 0; k < size; k++)  {  cout.width(3);  cout << arr[i][k] << " ";  }  cout << endl;  }  }  void PointsFilling(vector<vector<int>>& points, int size)  {  for (int i = 0; i < size; i++)  {  for (int k = 0; k < 2; k++)  {  points[i].resize(2);  cin >> points[i][k];  }  }  }  void PrintPoints(vector<vector<int>> points)  {  for (int i = 0; i < points.size(); i++)  {  for (int j = 0; j < points[i].size(); j++)  cout << points[i][j] << " ";  cout << '\n';  }  } |

* 1. **. Выполнение первой задачи. Вывести элементы матрицы, обходя её по часовой стрелке, начиная с первой строки.**

Для статического и динамического массивов задача реализуется одинаково с помощью универсального типа auto, который сам определяет тип данных переменной. Для реализации на статическом массиве, мы создаем данный двумерный массив с размерностью по умолчанию, равной 50, для строк и столбцов, и просим пользователя ввести размерность матрицы. Далее выбирается метод заполнения матрицы: самостоятельно или автоматически, с помощью генератора случайных чисел.

Для самостоятельного заполнения матрицы вызывается функция MatrixFillSelf(), принимающая в качестве аргументов сам массив и его размерность. В данной функции матрица заполняется пользователем с клавиатуры.

|  |
| --- |
| void MatrixFillSelf(auto arr, int size)  {  for (int i = 0; i < size; i++)  {  for (int k = 0; k < size; k++)  {  cin >> arr[i][k];  }  }  } |

При выборе автоматического заполнения, вызывается функция MatrixFillAuto(), которая также принимает массив и ее размерность, где матрица заполняется генератором случайных чисел.

|  |
| --- |
| void MatrixFillAuto(auto arr, int size)  {  srand(time(0));  for (int i = 0; i < size; i++)  {  for (int k = 0; k < size; k++)  {  arr[i][k] = 1 + rand() % 99;  }  }  } |

Далее выводится сама матрица и вызывается функция MatrixCircle() для вывода всех элементов матрицы по часовой стрелке с помощью циклов, начиная с первой строки.

|  |
| --- |
| void MatrixCircle(auto arr, int size)  {  int a = 0, b = 1, c = 2;  for (int i = 0; i < size; i++) {  for (int j = a; j < size - a; j++) {  cout << arr[a][j] << " ";  }  for (int j = b; j < size - a; j++) {  cout << arr[j][size - b] << " ";  }  for (int j = size - c; j >= a; j--) {  cout << arr[size - b][j] << " ";  }  for (int j = size - c; j > a; j--) {  cout << arr[j][a] << " ";  }  a++;  b++;  c++;  }  } |

* 1. **Выполнение второй задачи. Вывести элементы матрицы, обходя её по часовой стрелке, начиная с первой строки.**

Для выполнения второй задачи нам необходимо для начала создать динамический массив. Для этого мы определяем массив как двойной указатель и создаем его с помощью операции new. Далее мы задаем размерность каждого ряда (вложенного массива), используя проход основному массиву с помощью цикла со счетчиком for.

|  |
| --- |
| int\*\* arr = new int\* [size];  for (int i = 0; i < size; i++)  arr[i] = new int[size]; |

Для реализации данной задачи, мы используем те же функции, что и для первой задачи, так как данные функции принимают массив как параметр с типом данных auto, то есть программа сама определяет, с каким типом данных приходит параметр в функцию. Вследствие функция становится универсальной, так как может принимать аргументы с разным типом данных, не нарушающих работу функции.

* 1. **Выполнение третьей задачи. Из первого множества выбрать три различные точки так, чтобы треугольник с вершинами в этих точках, содержал (строго внутри себя) равное количество точек первого и второго множества.**

Для реализации данной задачи, нам необходимо использовать два вектора для хранения множества точек на плоскости. Однако у точек есть координаты, и их тоже надо хранить, поэтому будем использовать двумерный вектор (вектор векторов). Поэтому мы создаем его и задаем размерность, равную количеству, которое ввел пользователь, точек.

|  |
| --- |
| vector<vector<int>> points;  points.resize(size); |

Далее пользователь заполняет двумерным вектором координат точек, и начинается выполнение задачи - происходит вызов функции TriangleFill(), которая создаёт треугольник по заданным точкам, полученным в функции main() программы. После этого возвращает координаты подходящего треугольника. В качестве аргументов в функцию передается три вектора точек, взятых из первого двумерного вектора.

|  |
| --- |
| vector<vector<int>> TriangleFill(vector<int> points1, vector<int> points2, vector<int> points3) {  vector<vector<int>> triangle;  triangle.resize(3);  for (int i = 0; i < 3; i++) {  triangle[i].resize(2);  }  triangle[0][0] = points1[0];  triangle[0][1] = points1[1];  triangle[1][0] = points2[0];  triangle[1][1] = points2[1];  triangle[2][0] = points3[0];  triangle[2][1] = points3[1];  return triangle;  } |

В функции мы создаем вектор векторов triangle для хранения точек, составляющих треугольник. Далее мы задаём координаты вершин треугольника, используя точки из аргументов функции.

В ходе работы циклов функции main() мы задаём различные точки из первого вектора, таким образом строя треугольник.

Далее, после создания треугольника, мы начинаем проверку на нахождение точек внутри треугольника, используя функцию PointCheck(). По специальной формуле мы получим результат, входит ли данная точка или нет. В качестве аргументов функция принимает вектор треугольника и координаты X и Y точки.

|  |
| --- |
| bool PointCheck(vector<vector<int>> triangle, int x, int y) {  int x1 = triangle[0][0];  int x2 = triangle[1][0];  int x3 = triangle[2][0];  int y1 = triangle[0][1];  int y2 = triangle[1][1];  int y3 = triangle[2][1];  if (((x1 - x) \* (y2 - y1) - (x2 - x1) \* (y1 - y)) > 0 && ((x2 - x) \* (y3 - y2) - (x3 - x2) \* (y2 - y)) > 0 && ((x3 - x) \* (y1 - y3) - (x1 - x3) \* (y3 - y)) > 0) {  return true;  }  else {  return false;  }  } |

Далее, по условию задачи, нам необходимо проверить, равно ли количество находящихся в треугольнике точек из первого вектора количеству точек из второго вектора. Используя циклы, проходимся по каждой точке и считаем их количество отдельно для первого вектора и отдельно для второго в переменные c1 и c2 соответственно.

Если подходящий треугольник найден, то выводим его координаты, а также возвращаем значение 0. В ином случае, выводим сообщение об ошибке поиска треугольника.

|  |
| --- |
| for (int a = 0; a < points1.size(); a++) {  for (int b = 0; b < points1.size(); b++) {  for (int c = 0; c < points1.size(); c++) {  if (points1[a] != points1[b] && points1[b] != points1[c] && points1[a] != points1[c]) {  auto triangle = TriangleFill(points1[a], points1[b], points1[c]);  int c1 = 0, c2 = 0;  for (int i = 0; i < size1; i++) {  if (PointCheck(triangle, points1[i][0], points1[i][1])) {  c1++;  }  }  for (int i = 0; i < size2; i++) {  if (PointCheck(triangle, points2[i][0], points2[i][1])) {  c2++;  }  }  if (c1 == c2 && c1 != 0 && c2 != 0) {  cout << "Подходящий треугольник: " << endl;  PrintPoints(triangle);  return 0;  }  }  }  }  }  cout << "Не найдено подходящего треугольника." << endl; |

* 1. **Описание работы программы и её интерфейса**

При запуске программы пользователь выбирает номер задачи, в каждом из которых используются разные типы хранилищ данных. В программе реализовано бесконечное меню для всех задач, за исключением третьей (в случае успешной работы программы).

Если пользователь выбрал первую или вторую задачу, то пользователя просят ввести размер для рядов и столбцов квадратной матрицы. Причем размер должен быть больше единицы и, специально для статического массива, меньше или равно 50 – максимальное допустимое значение для рядов и столбцов матрицы для статического массива.

При выборе первой задачи создается двумерный статический массив с размерностью по умолчанию для рядов и столбцов матрицы, равной 50. Далее идет выбор метода реализации заполнения матрицы: вручную или автоматически, и вывод всех ячеек матрицы по часовой стрелке.

Для второй задачи все происходит также, только вместо статического используется двумерный динамический массив, объявленный как двойной указатель.

Реализация третьей задачи происходит на двумерном векторе. Пользователь задает количество точек, которое должно быть больше 3-х, так как треугольник можно построить только на трёх точках, не лежащих на одной прямой. После пользователь вводит координаты точек, и программа выводит все данные, требуемые в условии задачи.

Далее представлен интерфейс программы для первой и второй задач (рис. 1) и для третьей задачи (рис. 2).

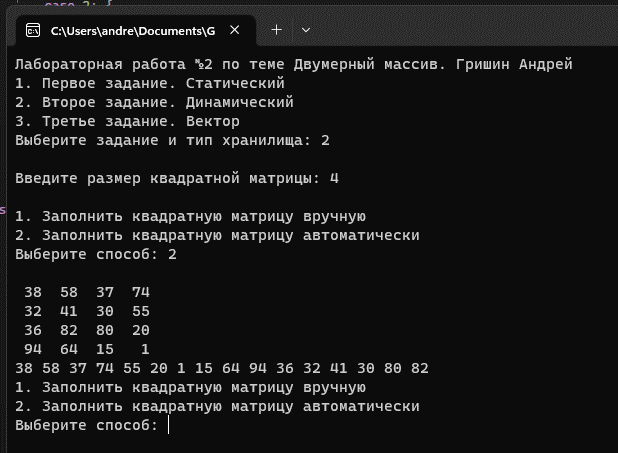


Рисунок 1. Интерфейс программы для первой и второй задач

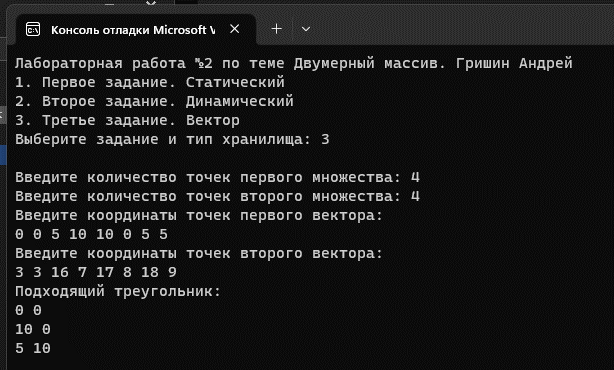


Рисунок 2. Интерфейс программы для третьей задачи

1. **Тестирование**

Протестируем работу программы для всех трёх задач на двумерных статическом, динамическом массивах и векторе по отдельности.

Рассмотрим выполнение первой задачи на статическом двумерном массиве с ручным (рис. 3) и автоматическим (рис. 4) заполнением матрицы.

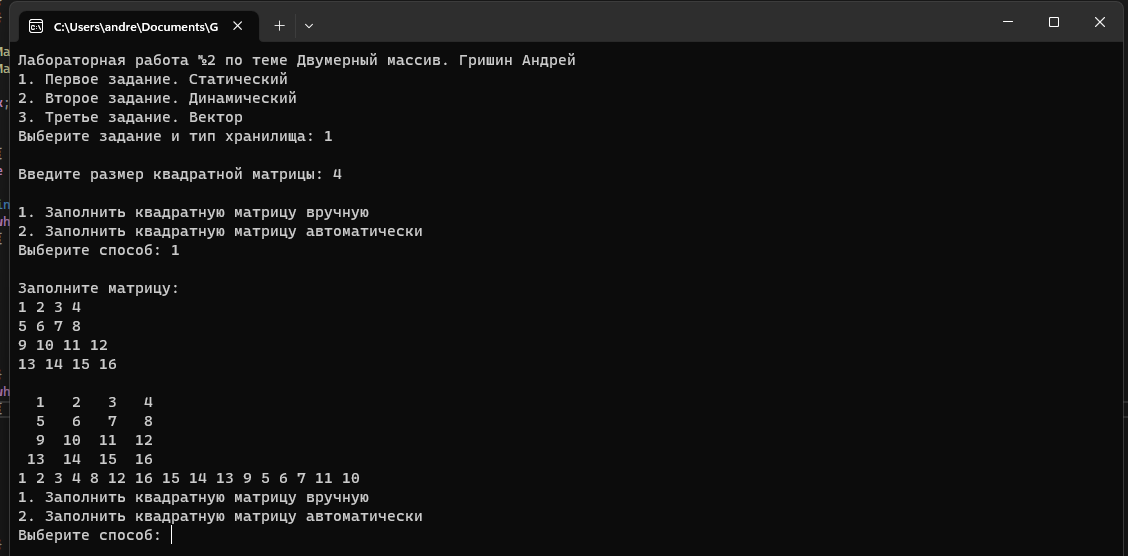


Рисунок 3. Тестирование первой задачи

с ручным заполнением матрицы

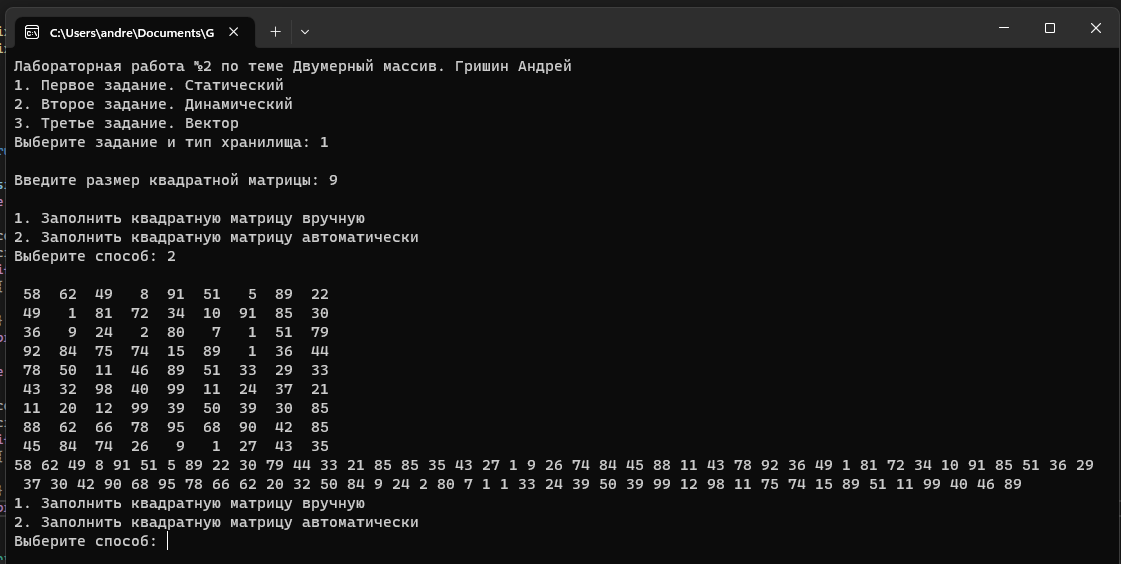


Рисунок 4. Тестирование первой задачи

с автоматическим заполнением матрицы

Тестирование первой задачи завершено успешно. Протестируем работу программы для второй задачи на двумерном динамическом массиве с помощью ручного (рис. 5) и автоматического (рис. 6) заполнения матрицы.

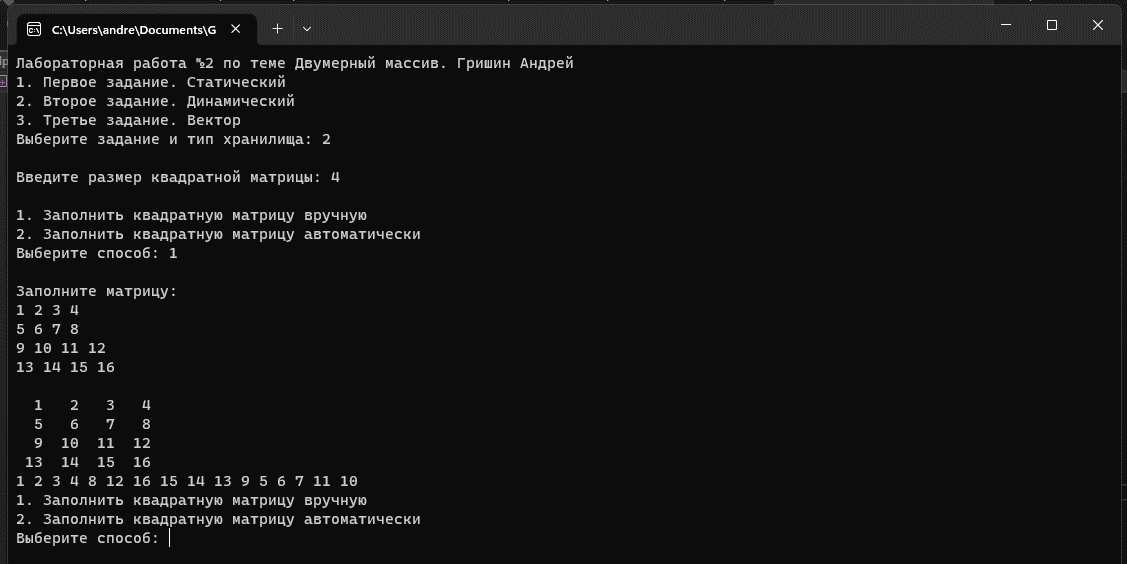


Рисунок 5. Тестирование второй задачи

с ручным заполнением матрицы

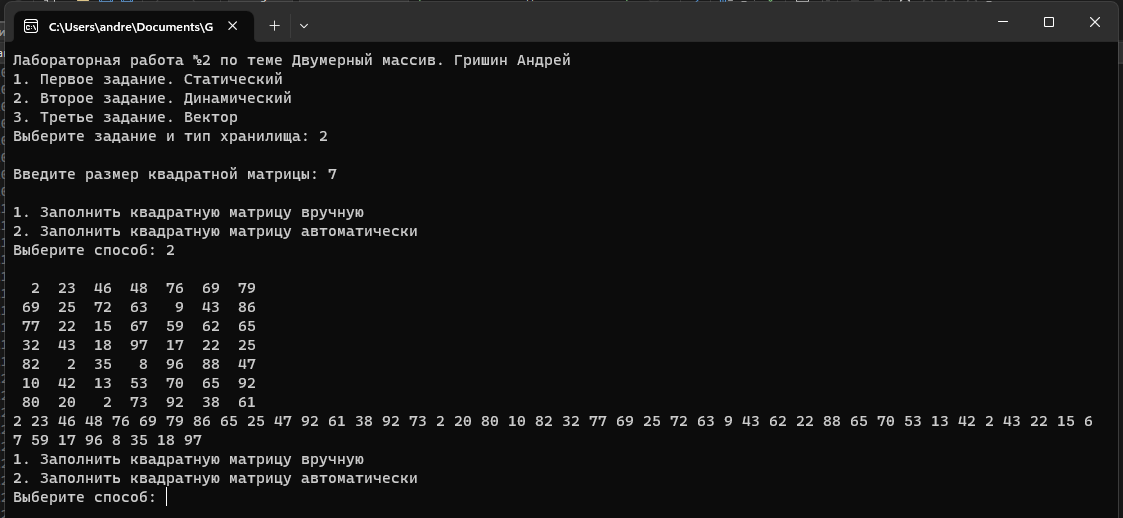


Рисунок 6. Тестирование второй задачи

с автоматическим заполнением матрицы

Тестирование второй задачи завершено успешно. Для тестирования третьей задачи на векторе, помимо результатов тестирования, приведем доказательства корректности выполнения нашей программы. Для начала проведем тестирование программы при корректных (рис. 7) и ошибочных (рис. 8) входных данных.

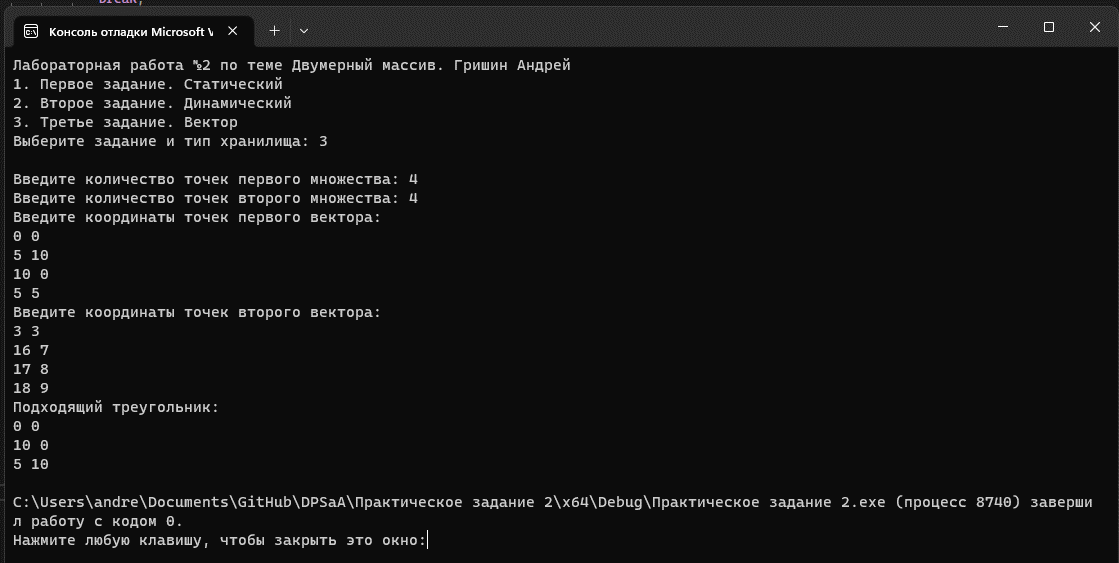


Рисунок 7. Тестирование третьей задачи при корректных входных данных

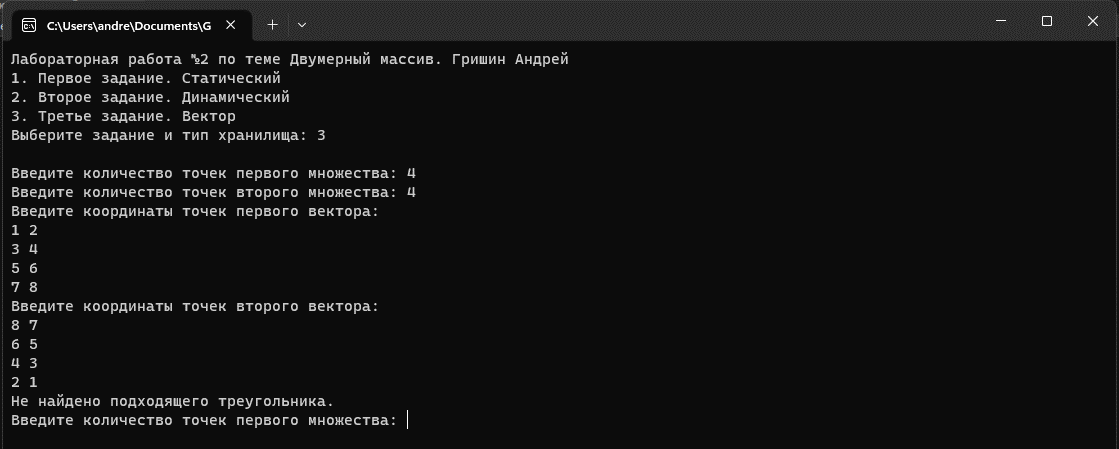


Рисунок 8. Тестирование третьей задачи при ошибочных входных данных

Тестирование прошло успешно. Однако мы не можем быть уверены в результатах работы программы для данной задачи, ведь нет наглядного доказательства корректности выполнения. Для этого воспользуемся материалом из интернета, а именно сайтом с построением геометрических фигур (рис. 9) для решения данной задачи.

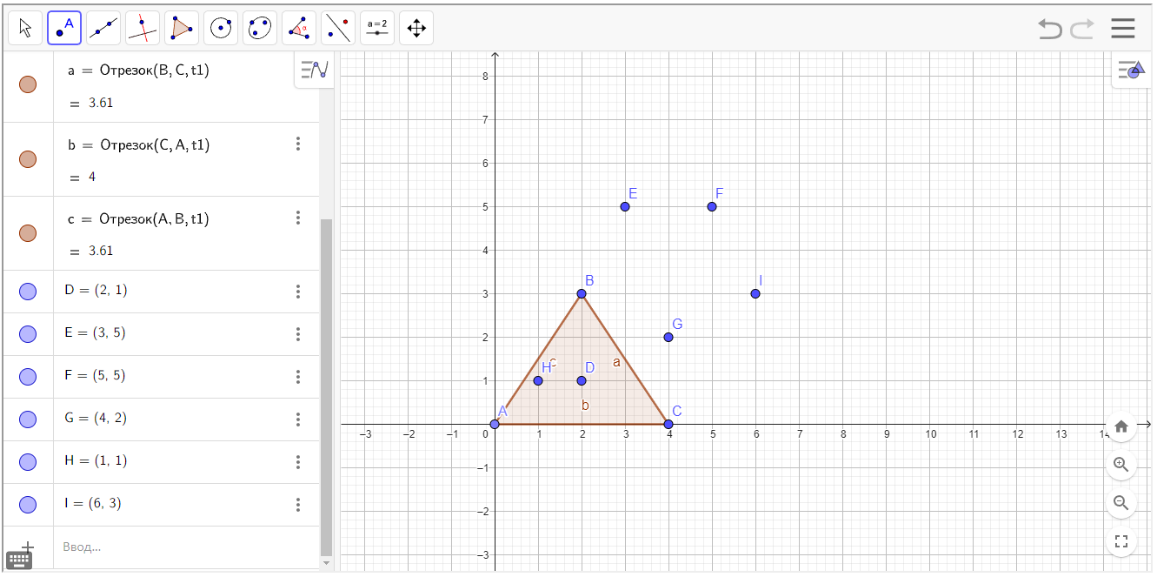


Рисунок 9. Наглядное решение задачи используя координатную плоскость

Возьмем координаты из данного чертежа и используем их в качестве входных данных. Тестирование с данными из чертежа изображено на рисунке 10.

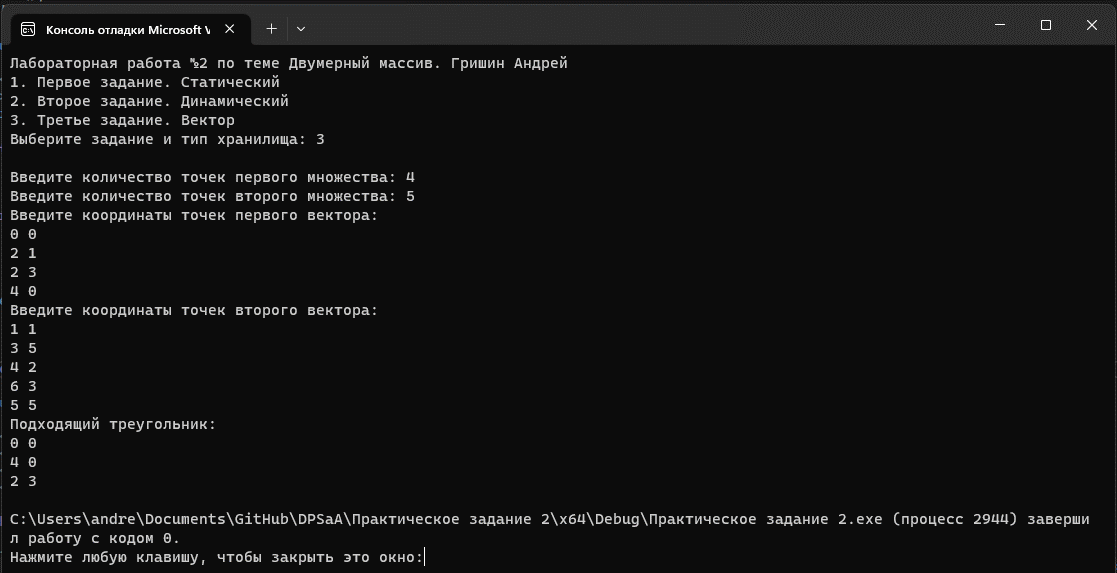


Рисунок 10. Тестирование третьей задачи с данными из чертежа (рис. 9)

Данное тестирование третьей задачи подтверждает корректность работы программы.

1. **Вывод**

В результате выполнения работы я приобрел навыки по определению двумерного массива и вектора для структуры данных задачи и для разработки алгоритмов операций над двумерным массивом и вектором в языке программирования C++.

1. **Исходный код программы**

|  |
| --- |
| #include <iostream>  #include <vector>  #define Rows 50  #define Cols 50  using namespace std;  void MatrixFillSelf(auto arr, int size)  {  for (int i = 0; i < size; i++)  {  for (int k = 0; k < size; k++)  {  cin >> arr[i][k];  }  }  }  void MatrixFillAuto(auto arr, int size)  {  srand(time(0));  for (int i = 0; i < size; i++)  {  for (int k = 0; k < size; k++)  {  arr[i][k] = 1 + rand() % 99;  }  }  }  void MatrixOutput(auto arr, int size)  {  for (int i = 0; i < size; i++)  {  for (int k = 0; k < size; k++)  {  cout.width(3);  cout << arr[i][k] << " ";  }  cout << endl;  }  }  void MatrixCircle(auto arr, int size)  {  int a = 0, b = 1, c = 2;  for (int i = 0; i < size; i++) {  for (int j = a; j < size - a; j++) {  cout << arr[a][j] << " ";  }  for (int j = b; j < size - a; j++) {  cout << arr[j][size - b] << " ";  }  for (int j = size - c; j >= a; j--) {  cout << arr[size - b][j] << " ";  }  for (int j = size - c; j > a; j--) {  cout << arr[j][a] << " ";  }  a++;  b++;  c++;  }  }  void PointsFilling(vector<vector<int>>& points, int size)  {  for (int i = 0; i < size; i++)  {  for (int k = 0; k < 2; k++)  {  points[i].resize(2);  cin >> points[i][k];  }  }  }  void PrintPoints(vector<vector<int>> points)  {  for (int i = 0; i < points.size(); i++)  {  for (int j = 0; j < points[i].size(); j++)  cout << points[i][j] << " ";  cout << '\n';  }  }  vector<vector<int>> TriangleFill(vector<int> points1, vector<int> points2, vector<int> points3) {  vector<vector<int>> triangle;  triangle.resize(3);  for (int i = 0; i < 3; i++) {  triangle[i].resize(2);  }  triangle[0][0] = points1[0];  triangle[0][1] = points1[1];  triangle[1][0] = points2[0];  triangle[1][1] = points2[1];  triangle[2][0] = points3[0];  triangle[2][1] = points3[1];  return triangle;  }  bool PointCheck(vector<vector<int>> triangle, int x, int y) {  int x1 = triangle[0][0];  int x2 = triangle[1][0];  int x3 = triangle[2][0];  int y1 = triangle[0][1];  int y2 = triangle[1][1];  int y3 = triangle[2][1];  if (((x1 - x) \* (y2 - y1) - (x2 - x1) \* (y1 - y)) > 0 && ((x2 - x) \* (y3 - y2) - (x3 - x2) \* (y2 - y)) > 0 && ((x3 - x) \* (y1 - y3) - (x1 - x3) \* (y3 - y)) > 0) {  return true;  }  else {  return false;  }  }  int vibor1() {  int menuChoice;  cout << "1. Первое задание. Статический" << endl;  cout << "2. Второе задание. Динамический" << endl;  cout << "3. Третье задание. Вектор" << endl;  cout << "Выберите задание и тип хранилища: "; cin >> menuChoice;  cout << endl;  return menuChoice;  }  int main()  {  // Меню выбора типа массива  setlocale(LC\_ALL, "Russian");  cout << "Лабораторная работа №2 по теме Двумерный массив. Гришин Андрей" << endl;  int size;  int num = vibor1();  if (num == 1 || num == 2)  {  while (true)  {  cout << "Введите размер квадратной матрицы: ";  cin >> size;  if (size > 1)  {  if (num == 1 && size > Rows)  {  continue;  }  break;  }  }  }  int taskChoice;  switch (num)  {  case 1: {  int arr[Rows][Cols];  while (true)  {  cout << "\n1. Заполнить квадратную матрицу вручную" << endl;  cout << "2. Заполнить квадратную матрицу автоматически" << endl;  cout << "Выберите способ: "; cin >> taskChoice;  cout << endl;  switch (taskChoice)  {  case 1: {  cout << "Заполните матрицу: " << endl;  MatrixFillSelf(arr, size);  cout << endl;  break;  }  case 2: {  MatrixFillAuto(arr, size);  break;  }  }  MatrixOutput(arr, size);  MatrixCircle(arr, size);  }  break;  }  case 2: {  int\*\* arr = new int\* [size];  for (int i = 0; i < size; i++)  arr[i] = new int[size];  while (true)  {  cout << "\n1. Заполнить квадратную матрицу вручную" << endl;  cout << "2. Заполнить квадратную матрицу автоматически" << endl;  cout << "Выберите способ: "; cin >> taskChoice;  cout << endl;  switch (taskChoice)  {  case 1: {  cout << "Заполните матрицу: " << endl;  MatrixFillSelf(arr, size);  cout << endl;  break;  }  case 2: {  MatrixFillAuto(arr, size);  break;  }  }  MatrixOutput(arr, size);  MatrixCircle(arr, size);  }  break;  }  case 3: {  while (true)  {  int size1, size2;  while (true)  {  cout << "Введите количество точек первого множества: ";  cin >> size1;  if (size1 < 3)  {  continue;  }  break;  }  while (true)  {  cout << "Введите количество точек второго множества: ";  cin >> size2;  if (size2 < 3)  {  continue;  }  break;  }  vector<vector<int>> points1;  vector<vector<int>> points2;  points1.resize(size1);  points2.resize(size2);  int pointsCounter = 0;  cout << "Введите координаты точек первого вектора: " << endl;  PointsFilling(points1, size1);  cout << "Введите координаты точек второго вектора: " << endl;  PointsFilling(points2, size2);    for (int a = 0; a < points1.size(); a++) {  for (int b = 0; b < points1.size(); b++) {  for (int c = 0; c < points1.size(); c++) {  if (points1[a] != points1[b] && points1[b] != points1[c] && points1[a] != points1[c]) {  auto triangle = TriangleFill(points1[a], points1[b], points1[c]);  int c1 = 0, c2 = 0;  for (int i = 0; i < size1; i++) {  if (PointCheck(triangle, points1[i][0], points1[i][1])) {  c1++;  }  }  for (int i = 0; i < size2; i++) {  if (PointCheck(triangle, points2[i][0], points2[i][1])) {  c2++;  }  }  if (c1 == c2 && c1 != 0 && c2 != 0) {  cout << "Подходящий треугольник: " << endl;  PrintPoints(triangle);  return 0;  }  }  }  }  }  cout << "Не найдено подходящего треугольника." << endl;  }  break;  }  default:  system("cls");  }  } |