Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение высшего образования

«Омский государственный технический университет»

|  |  |
| --- | --- |
| Факультет (институт) | *Информационных технологий и компьютерных систем* |
|  |  |
| Кафедра | *Прикладная математика и фундаментальная информатика* |
|  |  |

**Расчетно–графическая работа**

|  |  |
| --- | --- |
| по дисциплине | ***Алгоритмизация и программирование*** |
|  |  |
| на тему | Программная реализация задач |

Пояснительная записка

|  |  |
| --- | --- |
| **Шифр проекта** | 020–РГР–02.03.02–№ 1 – ПЗ |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Студента** | | Байбулова Ильяса Ерланович | | | | | |
|  |  |  |  | | фамилия, имя, отчество полностью | | | | | |
|  |  |  | Курс | 1 |  | Группа | | ФИТ–242 | | |
|  |  |  |  |  |  | |  |  | |  |
|  | | | **Направление (специальность)** | | | | | ***02.03.02*** | | |
|  | | | Фундаментальная информатика и информационные технологии | | | | | | | |
|  |  |  | код, наименование | | | | | | | |
|  |  |  | Руководитель | | ***ст. преподаватель*** | | | | | |
|  |  |  | ученая степень, звание | | | | | |
|  |  |  | ***Федотова И.В.*** | | | | | | | |
|  |  |  | фамилия, инициалы | | | | | | | |
|  |  |  | Выполнил | |  | | | | | |
|  |  |  | дата, подпись студента | | | | | |
|  |  |  |  | | | | | | | |
|  |  |  | **Работа защищена с количеством баллов** | | | | | |  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | дата, подпись руководителя |  |  |  |

Омск 2025

**Содержание**

[Введение 3](#_Toc200071487)

[Постановка задачи “Многоугольник” 4](#_Toc200071488)

[Ход решения задачи “Многоугольник” 5](#_Toc200071489)

[Постановка задачи “Расстояние между городами” 9](#_Toc200071490)

[Ход решения задачи “Расстояние между городами” 10](#_Toc200071491)

[Постановка задачи “СуперКОНЬ” 12](#_Toc200071492)

[Ход решения задачи “СуперКОНЬ” 13](#_Toc200071493)

[Постановка задачи “Города” 16](#_Toc200071494)

[Ход решения задачи “Города” 17](#_Toc200071495)

[Заключение 20](#_Toc200071496)

[Список используемой литературы 21](#_Toc200071497)

# **Введение**

В настоящее время для решения задач применяется язык программирования C#. Этот современный объектно-ориентированный язык был разработан компанией Microsoft. C# создавался с целью предоставить разработчикам мощный и гибкий инструмент для создания разнообразных приложений, включая простые консольные программы, сложные веб-приложения, мобильные решения и игровые проекты. Благодаря своей универсальности C# стал одним из наиболее востребованных языков программирования в мире. Синтаксис C# отличается понятностью, что способствует его доступности для изучения. Для работы с данным языком использовалась среда разработки Visual Studio, предлагающая мощные инструменты для создания кода. При решении задач были задействованы различные коллекции, циклы, массивы, строки и методы.

# **Постановка задачи “Многоугольник”**

Дано N точек (3 ≤ N ≤ 1000) с целочисленными координатами, некоторые (или все) из которых являются вершинами выпуклого многоугольника, а остальные (или ни одной) находятся внутри него. Любые три точки не лежат на одной прямой. Определить площадь выпуклого многоугольника.

Модули координат не превосходят 30000. Считается, что кроме заданных вершин, других вершин нет.

Входной файл

Первая строка содержит число точек N.

Следующие N строк содержат по два числа xi и yi, разделенных пробелом – координаты точек.

Выходной файл

Содержит площадь многоугольника с тремя знаками после запятой.

# **Ход решения задачи “Многоугольник”**

Программа принимает количество точек и их координаты, сохраняя X и Y координаты в отдельные массивы. Точки предварительно сортируются по возрастанию X координат, а при совпадении по Y координатам, что обеспечивает корректную работу алгоритма построения выпуклой оболочки. Для определения минимального выпуклого многоугольника, содержащего все точки, используется функция построения оболочки, которая анализирует взаимное расположение точек через векторное произведение, что в свою очередь позволяет определить ориентацию тройки точек относительно базового вектора и выбрать граничные точки. Полученные вершины оболочки передаются в функцию расчёта площади, где по методу шнуровки Гаусса вычисляется точная площадь многоугольника через суммирование векторных произведений координат смежных вершин, после чего итоговое значение площади выводится как результат работы программы.

Код программы представлен на рисунках ниже.

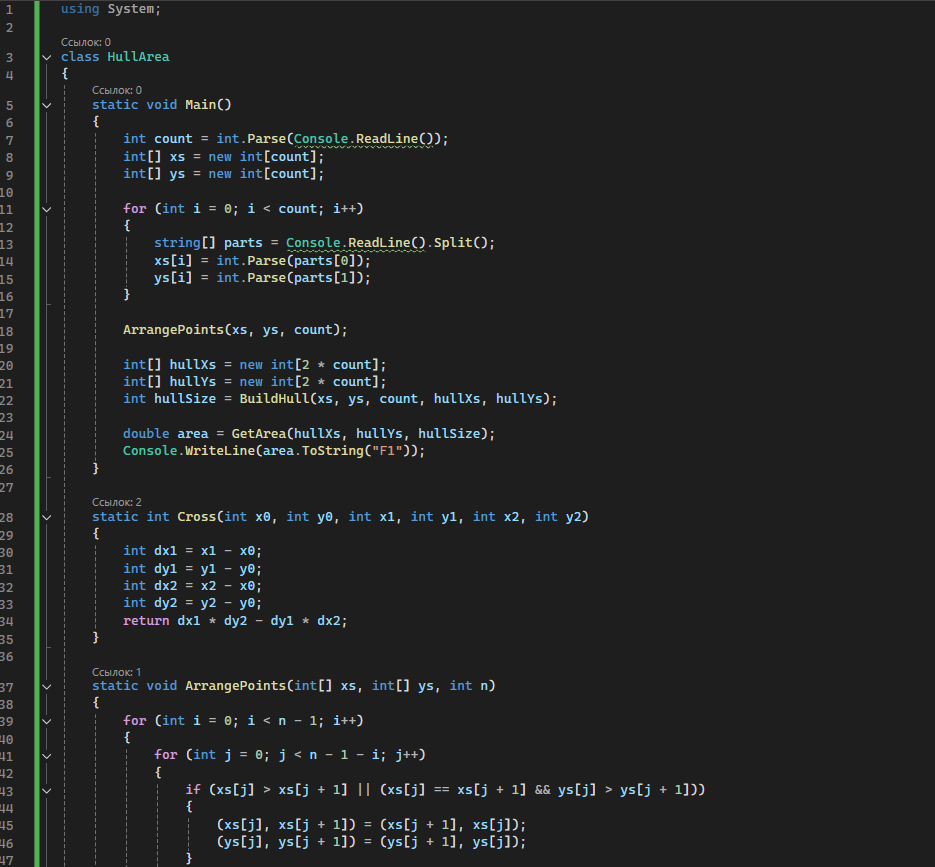


Рисунок 1 – Код программы

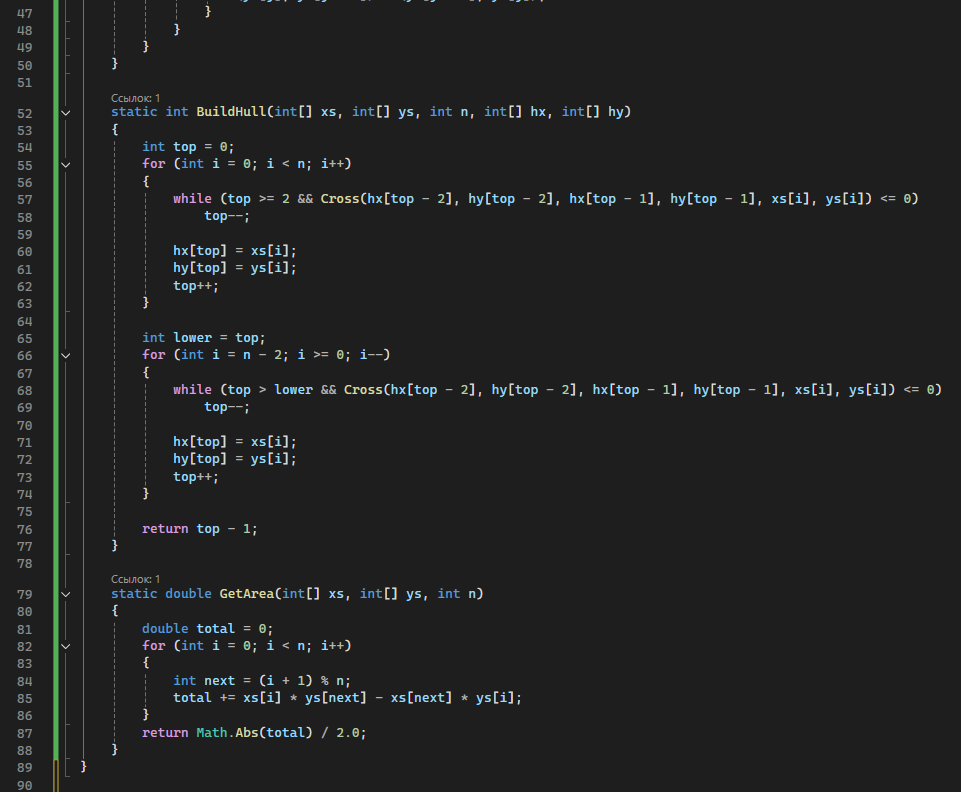


Рисунок 2 – Код программы

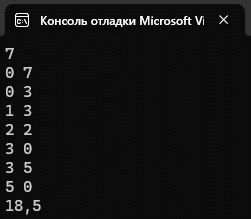


Рисунок 3 – Результат работы программы



Рисунок 4 – Результат работы программы

# **Постановка задачи “Расстояние между городами”**

На поверхности некоторой планеты, представляющей собой идеальный шар радиуса R, заданы координаты двух городов в виде двух чисел – широты и долготы.

Требуется определить минимальное расстояние по поверхности планеты между этими городами.

Примечание:

Пары (широта, долгота) уникальны.

Входной файл:

Первая строка содержит два целых числа S1 и D1 – широту и долготу первого города в градусах (-90 ≤ S1 ≤ 90; 0 ≤ D1 ≤ 359).

Вторая строка содержит два целых числа S2 и D2 – широту и долготу второго города в градусах (-90 ≤ S2 ≤ 90; 0 ≤ D2 ≤ 359).

Третья строка содержит целое число R – радиус планеты (1 ≤ R ≤ 30000).

Выходной файл:

Должен содержать одно вещественное число – минимальное расстояние между городами по поверхности планеты, выведенное с тремя знаками после запятой.

# **Ход решения задачи “Расстояние между городами”**

Программа вычисляет кратчайшее расстояние между двумя городами на сферической планете по их географическим координатам. На вход подаются три строки: в первых двух широта и долгота городов в градусах, в третьей радиус планеты. Координаты немедленно преобразуются в радианы для тригонометрических расчетов. По формуле сферической геометрии вычисляется центральный угол между точками: arccos(sin(lat1) \* sin(lat2) + cos(lat1) \* cos(lat2) \* cos(|lon1-lon2|)). Итоговое расстояние получается умножением этого угла на радиус планеты, что дает длину дуги большого круга — точное кратчайшее расстояние по поверхности сферы. Результат выводится с точностью до трех десятичных знаков.

Код программы представлен на рисунке ниже:

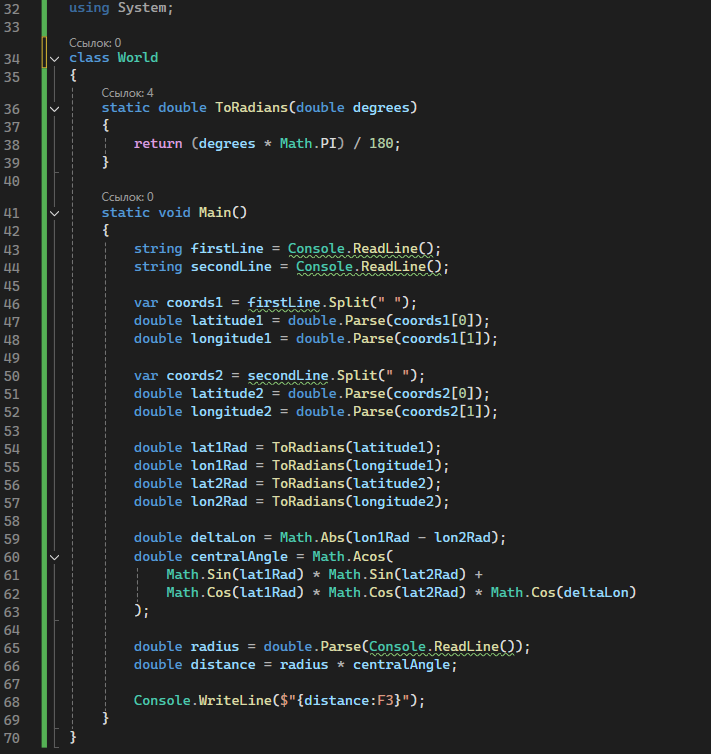


Рисунок 5 – Код программы

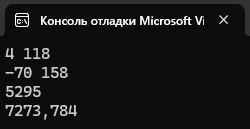


Рисунок 6 – Результат работы программы

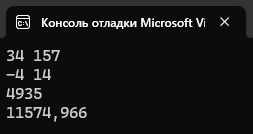


Рисунок 7 – Результат работы программы

# **Постановка задачи “СуперКОНЬ”**

На квадратной доске с клетками белого и черного цвета (расположенными как на шахматной доске) указаны две клетки.

Найти наименьшее количество ходов, которое нужно для перемещения фигуры “СуперКОНЬ” из первой указанной клетки во вторую.

“СуперКОНЬ” ходит следующим образом:

если клетка черная, то “СуперКОНЬ” ходит как обычный шахматный конь;

если клетка белая, то “СуперКОНЬ” ходит на одну клетку вверх, вниз, вправо или влево.

Клетка с номером (1,1) – черная.

Входной файл

Первая строка содержит число N (число вертикалей и горизонталей, 1 ≤ N ≤ 255).

Во второй строке, разделенные пробелом, располагаются числа X1 Y1 X2 Y2 (координаты клеток начала и окончания движения).

Выходной файл

Должен содержать единственную строку с минимальным количеством ходов, или слово “NO” (без кавычек), если “СуперКОНЬ” не может достигнуть заданной клетки.

# **Ход решения задачи “СуперКОНЬ”**

Алгоритм использует поиск в ширину для нахождения кратчайшего пути фигуры по шахматной доске размером N×N. Фигура изменяет поведение в зависимости от цвета клетки: на чёрных клетках (где сумма координат x+y чётная) она ходит как конь (8 L-образных перемещений), а на белых клетках (x+y нечётное) - как король (4 перемещения в соседние клетки по вертикали/горизонтали).

Программа начинает с чтения размера доски и координат старта/финиша. Инициализируется очередь с начальной позицией и матрицей посещений. На каждом шаге извлекается текущая позиция - если достигнута цель, возвращается количество шагов. Для непосещённых клеток определяют тип хода (конь/король), генерируют допустимые перемещения в пределах доски и добавляют новые позиции в очередь с увеличением счётчика шагов.

Если очередь опустеет без достижения цели, возвращается "NO". BFS гарантирует нахождение кратчайшего пути благодаря поуровневому обходу, а динамическое изменение типа хода реализовано через выбор соответствующего массива смещений координат в зависимости от цвета текущей клетки.

Код программы представлен ниже:

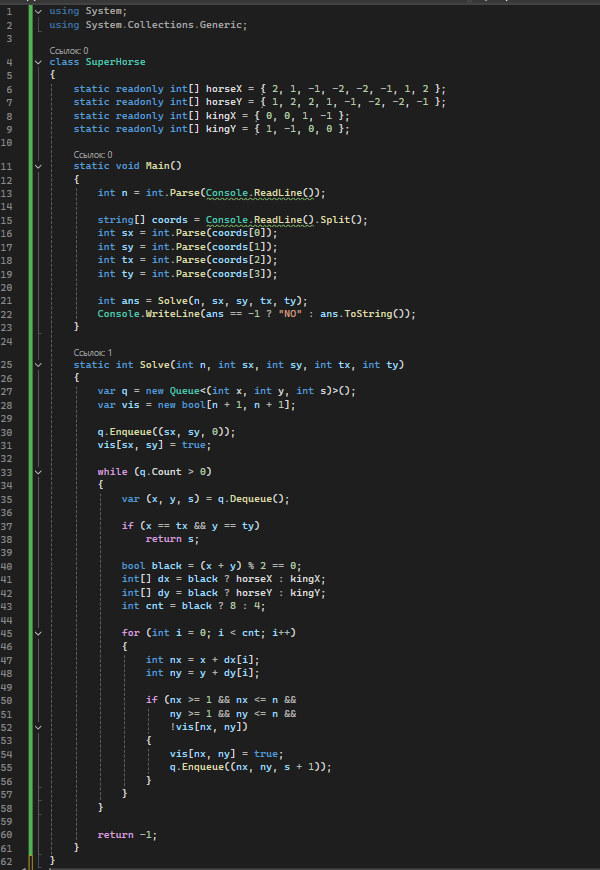


Рисунок 8 – Код программы

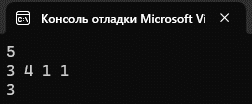


Рисунок 9 – Результат работы программы

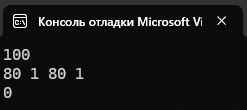


Рисунок 10 – Результат работы программы

# **Постановка задачи “Города”**

В некоторой стране население живет в городах. Города соединены дорогами. Каждая дорога начинается в одном городе и заканчивается в другом. Длины всех дорог известны. Из любого города через систему дорог можно добраться до любого другого города.

Будем называть расстоянием между двумя городами длину кратчайшего пути между ними.

Требуется найти наибольшее расстояние между городами в стране.

Входной файл

Первая строка содержит два целых числа N и M, разделенных пробелом – количество городов и дорог (2 ≤ N ≤ 400, 1 ≤ M ≤ 100000).

Следующие M строк содержат по три целых числа I, J, L – номера городов, соединенных дорогой (1 ≤ I, J ≤ N) и длина дороги (1 ≤ L ≤ 1000).

Выходной файл

Содержит единственное целое число – наибольшее расстояние между городами в стране.

# **Ход решения задачи “Города”**

Программа вычисляет диаметр графа, то есть максимальное кратчайшее расстояние между любыми двумя городами в транспортной сети. На вход подаются количество городов (n) и дорог (m), после чего последовательно считываются описания дорог в формате "городA городB длина". Инициализируется матрица расстояний, где сразу задаются нулевые расстояния между одинаковыми городами и "бесконечность" для остальных пар. При чтении каждой дороги программа автоматически сохраняет минимальную длину между соединёнными городами, учитывая неориентированность графа (дороги работают в обе стороны).

Далее применяется алгоритм Флойда-Уоршелла, который через тройную вложенность циклов перебирает все возможные пути с промежуточными городами. На каждом шаге он проверяет, сократит ли путь добавление "транзитного" города, и обновляет матрицу расстояний. В финале определяется максимальное значение в матрице (игнорируя "бесконечность" для несвязанных городов), что и представляет собой диаметр сети, или длину самого длинного кратчайшего пути.

Код программы представлен ниже:

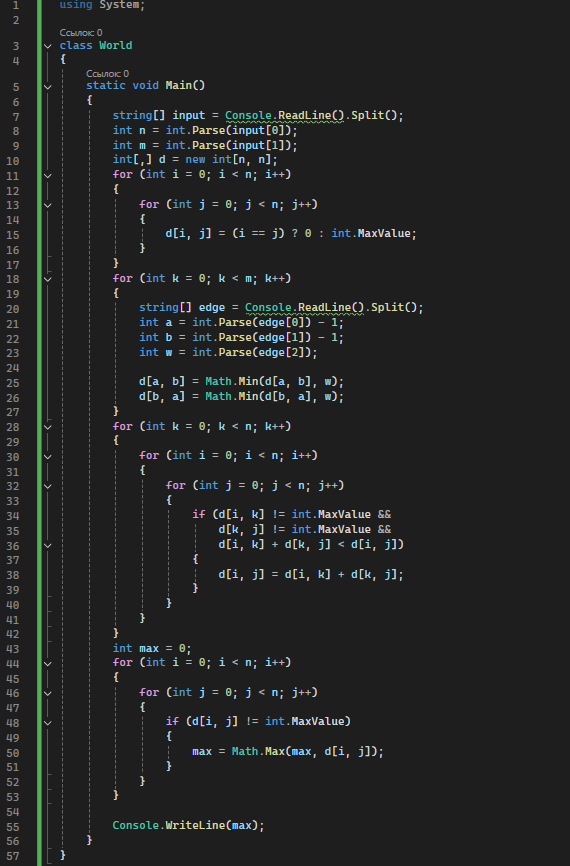


Рисунок 11 – Код программы

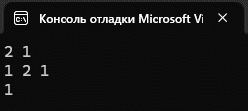


Рисунок 12 – Результат работы программы

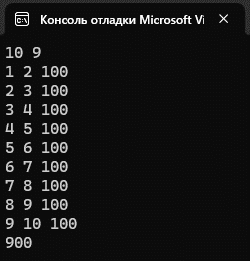


Рисунок 13 – Результат работы программы

# **Постановка задачи “Прямоугольник”**

Имеется прямоугольная область размером N×M клеток, некоторые из которых закрашены.

Требуется найти прямоугольник максимальной площади, не содержащий закрашенные клетки. Стороны прямоугольника должны быть параллельны краям прямоугольной области.

Входной файл содержит:

* в первой строке два числа N и M – число столбцов и строк прямоугольной области соответственно (1 ≤ N, M ≤ 250);
* во второй строке число K – количество закрашенных клеток (0 ≤ K ≤ (N\*M-1));
* следующие K строк содержат по 2 числа X и Y – координаты (столбец и строка соответственно) закрашенной клетки.

Выходной файл должен содержать площадь найденного прямоугольника.

# **Ход решения задачи “Прямоугольник”**

Программа получает координаты закрашенных клеток. Каждую закрашенную клетку программа помечает значением 1, а незакрашенные клетки остаются со значением 0. Создается специальный массив для отслеживания высот столбцов. Для каждой свободной клетки значение в соответствующем столбце увеличивается, а для закрашенной клетки значение столбца сбрасывается в ноль. Далее вызывается функция MaxRectangle, которая находит максимальную площадь прямоугольника. Эта функция использует стек для обработки элементов. Пока текущее значение высоты меньше значения на вершине стека, функция извлекает элементы из стека и вычисляет площадь прямоугольника. Если полученная площадь превышает текущий максимум, максимальное значение обновляется.

Код программы представлен на рисунке ниже.

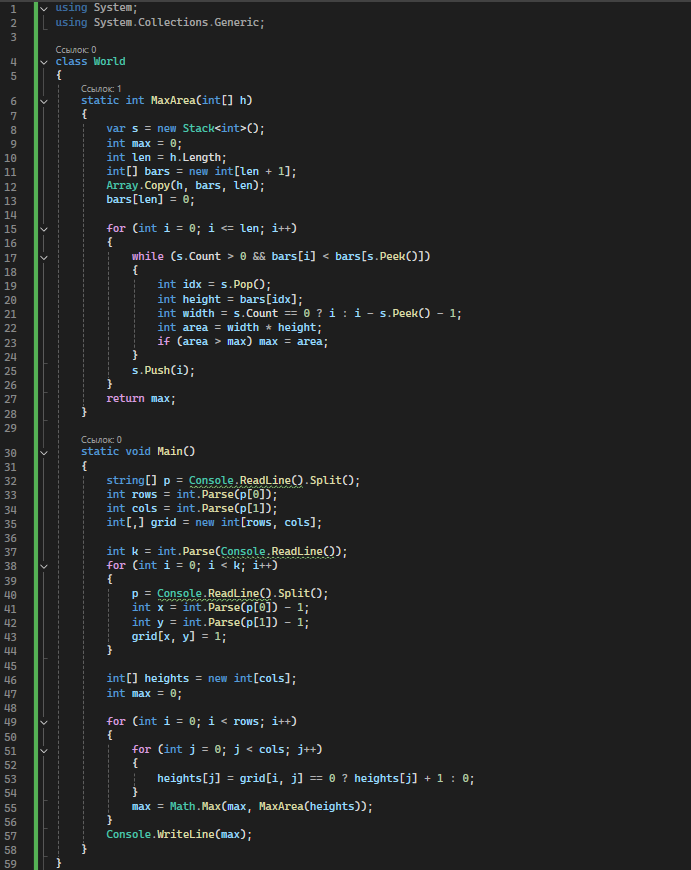


Рисунок 14 ­– Код программы

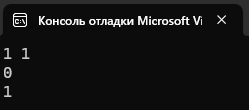


Рисунок 15 – Результат работы программы

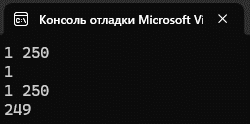


Рисунок 16 – Результат работы программы

# **Заключение**

В ходе выполнения практической работы я овладел фундаментальными принципами языка C#. Решение поставленных задач потребовало не только понимания циклов, массивов, методов и функций, но также знаний основ объектно ориентированного программирования и различных коллекций, что существенно углубило моё понимание C#. В процессе работы я освоил эффективное использование коллекций, включая списки и стеки, а также изучил классы как ключевые элементы ООП. При выполнении заданий активно применялись массивы, методы, циклы и коллекции, что обеспечило ценный практический опыт и углублённое изучение языка. Таким образом, данная работа по алгоритмизации и программированию стала значимым этапом обучения, а приобретённые навыки и знания сформировали прочную основу для дальнейшего освоения C#.

# **Список используемой литературы**

1. Metanit <https://metanit.com/sharp/tutorial/4.5.php> (дата обращения: 28.05.25)
2. Microsoft Learn <https://learn.microsoft.com/ru-ru/dotnet/api/system.collections.generic.hashset-1?view=net-8.0> (дата обращения: 28.04.25)
3. Платформа .NET и C# от А до Я <https://csharp.webdelphi.ru/stek-stack-v-c/> (дата обращения: 29.06.25)
4. Пахомов Б.И. C# для начинающих. – «БХВ-Петербург», Санкт-Петербург, 2014. – 431 с.