|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

|  |  |
| --- | --- |
| **Отчет по выполнению практического задания № 1** | |
| **Тема:** | |
| **«Оценка вычислительной сложности алгоритма»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Величко В.Д. |
|  |  |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 2](#_Toc179660470)

[2 ЗАДАНИЕ 1 3](#_Toc179660471)

[2.1 Первый алгоритм 3](#_Toc179660472)

[2.1.1 Описание математической модели 3](#_Toc179660473)

[2.1.2 Блок-схема алгоритма, доказательство корректности циклов 4](#_Toc179660474)

[2.1.3 Определение вычислительной сложности алгоритма 5](#_Toc179660475)

[2.1.4 Реализация алгоритма на C++ 6](#_Toc179660476)

[2.1.5 Тестирование 8](#_Toc179660477)

[2.2 Второй алгоритм 9](#_Toc179660478)

[2.2.1 Описание математической модели 9](#_Toc179660479)

[2.2.2 Блок-схема алгоритма, доказательство корректности циклов 10](#_Toc179660480)

[2.2.3 Определение вычислительной сложности алгоритма 10](#_Toc179660481)

[2.2.4 Реализация алгоритма на языке С++ 11](#_Toc179660482)

[2.2.5 Тестирование 12](#_Toc179660483)

[2.3 Вывод по первому заданию 13](#_Toc179660484)

[3 ЗАДАНИЕ 2 14](#_Toc179660485)

[3.1 Описание математической модели 14](#_Toc179660486)

[3.2 Блок-схема алгоритма, доказательство корректности циклов 14](#_Toc179660487)

[3.3 Определение вычислительной сложности алгоритма 17](#_Toc179660488)

[3.4 Реализация алгоритма на языке C++ 18](#_Toc179660489)

[3.5 Тестирование 20](#_Toc179660490)

[3.6 Выводы по заданию 2 21](#_Toc179660491)

[4 ВЫВОДЫ 22](#_Toc179660492)

[5 ЛИТЕРАТУРА 23](#_Toc179660493)

# 1 ЦЕЛЬ

Приобретение практических навыков:

* Эмпирическому определению вычислительной сложности алгоритмов на теоретическом и практическом уровнях;
* Выбору эффективного алгоритма решения вычислительной задачи из нескольких.
* Разработка собственного алгоритма в соответствии с задачей.

# 2 ЗАДАНИЕ 1

**Формулировка задания:** выбрать эффективный алгоритм вычислительной задачи из двух предложенных, используя теоретическую и практическую оценку вычислительной сложности каждого из алгоритмов, а также его емкостную сложность. Пусть имеется вычислительная задача:

– дан массив х из n элементов целого типа; удалить из этого массива все значения равные заданному (ключевому) key.

Удаление состоит в уменьшении размера массива с сохранением порядка следования всех элементов, как до, так и следующих после удаляемого.

## 

## 2.1 Первый алгоритм

### 2.1.1 Описание математической модели

С помощью цикла идем по массиву с первого элемента до n-ного, где n – размер массива. Если текущий элемент равен заданному значению, то смещаем все следующие значения в массиве на 1 позицию влево, тем самым заменяя и удаляя требуемый элемент. Переменную n, отвечающую за размер массива, уменьшаем на 1.

### 2.1.2 Блок-схема алгоритма, доказательство корректности циклов

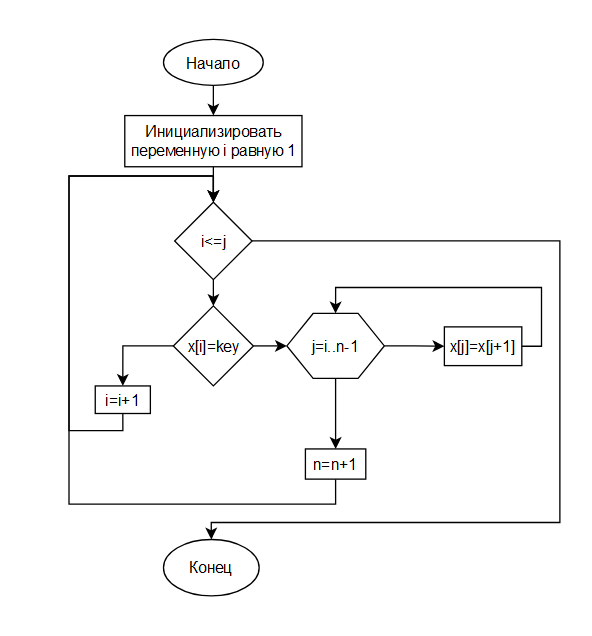


Рисунок 1 – Блок-схема для первого алгоритма

Касательно доказательства корректности циклов, можно рассмотреть инвариантные условия.

Например:

1. Для цикла while: Инвариант - до текущего индекса i включительно все элементы массива отличны от искомого элемента. Инвариант сохраняется на каждой итерации цикла.

2. Для цикла for (при нахождении искомого элемента): Инвариант - сдвиг происходит корректно, остальные элементы не меняются местами. Инвариант также поддерживается на каждой итерации.

### 2.1.3 Определение вычислительной сложности алгоритма

|  |  |  |
| --- | --- | --- |
| Номер строки | Алгоритм, записанный на псевдокоде | Количество повторений действия в  зависимости от объема входных  данных n |
| 1 | delFirstMetod(x,n,key){ |  |
| 2 | i←1 | 1 |
| 3 | while (i<=n) **do** | n+1 |
| 4 | if x[i]=key then | n |
| 5 | for j←i to n-1 do |  |
| 6 | x[j] ←x[j+1] |  |
| 7 | оd |  |
| 8 | n←n-1 | n |
| 9 | else |  |
| 10 | i←i+1 |  |
| 11 | endif |  |
| 12 | **od** |  |
| 13 | } |  |

Количество повторений действия в строке 6 представляет собой арифметическую прогрессию. Найдем ее сумму  *.*

Тогда общая вычислительная сложность алгоритма в худшем случае определяется функцией. То есть алгоритм имеет квадратичный порядок роста времени вычисления.

В лучшем случае, когда ни один элемент удалять не нужно, сложность определяется функцией T(n)=2n+2.

### 2.1.4 Реализация алгоритма на C++

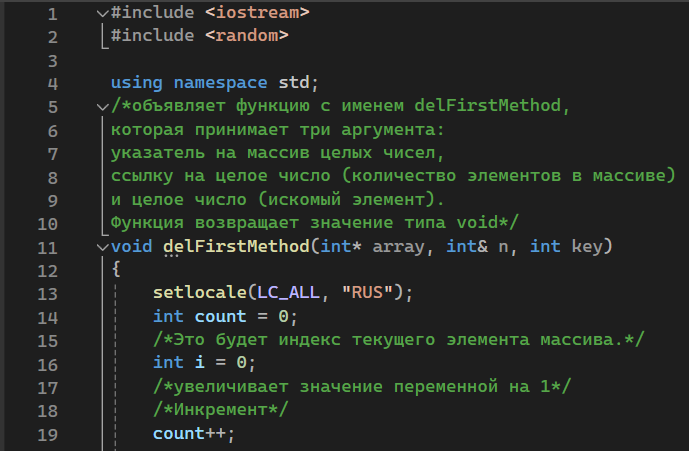


Рисунок 2.1 – Первый алгоритм



Рисунок 2.2 – Первый алгоритм

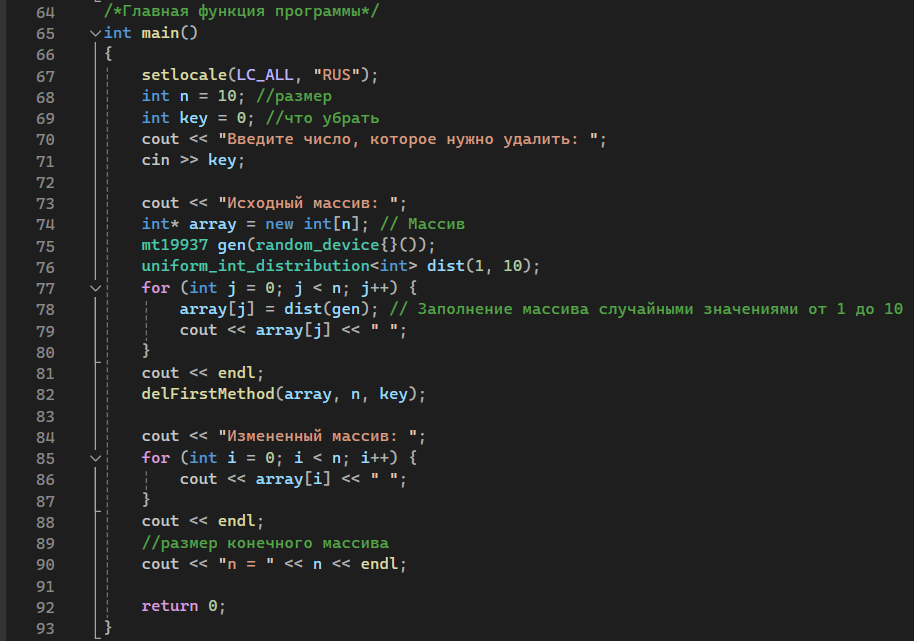


Рисунок 2.3 – Первый алгоритм

### 2.1.5 Тестирование

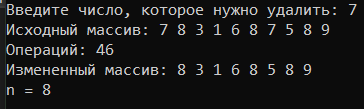


Рисунок 3.1 - Тестирование при 10 элементах, key=7. Случайная ситуация.

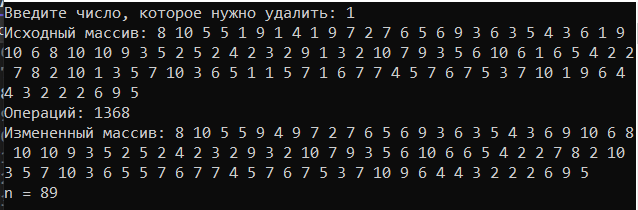


Рисунок 3.2 - Тестирование при 100 элементах, key=1. Случайная ситуация.

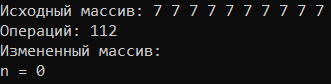


Рисунок 3.3 - Тестирование при 10 элементах и удаления всех.

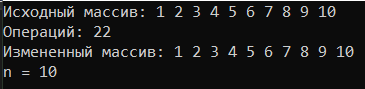


Рисунок 3.4 - Тестирование при 10 элементах и ничего не удаляется

## 2.2 Второй алгоритм

### 2.2.1 Описание математической модели

С помощью цикла идем по массиву с первого элемента до n-ного, где n – размер массива. В переменной i хранится номер рассматриваемого элемента исходного массива, в переменной j хранится номер размещаемого в данный момент элемента в конечном массиве. В j-тый элемент постоянно помещаем i-тый, но увеличиваем j на 1 (то есть размещаем теперь в следующий элемент конечного массива) только если текущий не равен искомому значению.

### 2.2.2 Блок-схема алгоритма, доказательство корректности циклов

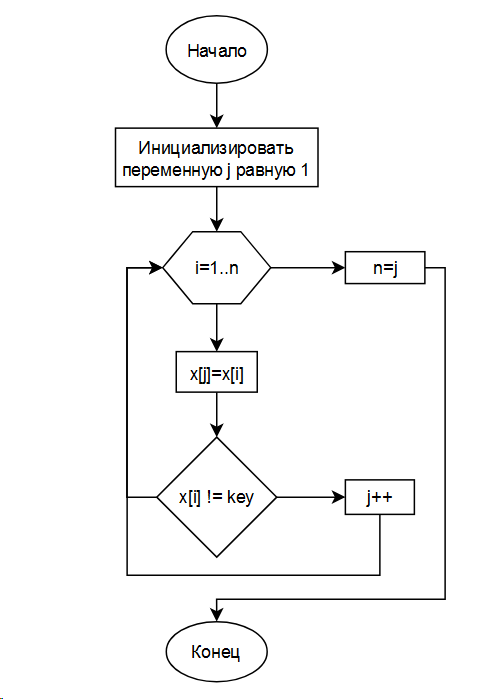


Рисунок 4 – Блок-схема второго алгоритма

Доказательство конечности цикла: при каждой итерации область неопределённости сужается на 1 элемент. До начала цикла не просмотрено n элементов, после первой итерации n-1, после второй n-2 и так далее. После n-ной итерации будет не просмотрено n-n=0 элементов, следовательно цикл завершится.

Таким образом, цикл алгоритма корректен, а значит и сам алгоритм, корректен.

### 2.2.3 Определение вычислительной сложности алгоритма

|  |  |  |
| --- | --- | --- |
| Номер строки | Алгоритм, записанный на псевдокоде | Количество повторений действия в  зависимости от объема входных  данных n |
| 1 | delOtherMetod(x,n,key){ |  |
| 2 | j←1 | 1 |
| 3 | for i←1 to n **do** | n+1 |
| 4 | x[j]=x[i]; | n |
| 5 | if x[i]!=key then | n |
| 6 | j++ | n |
| 7 | endif |  |
| 8 | **od** |  |
| 9 | **n**←j | 1 |
| 10 | } |  |

Общая вычислительная сложность алгоритма в худшем случае определяется функцией. То есть алгоритм имеет линейны1 порядок роста времени вычисления.

В лучшем случае, когда все нужно удалять, сложность определяется функцией T(n)=2n+3.

### 2.2.4 Реализация алгоритма на языке С++

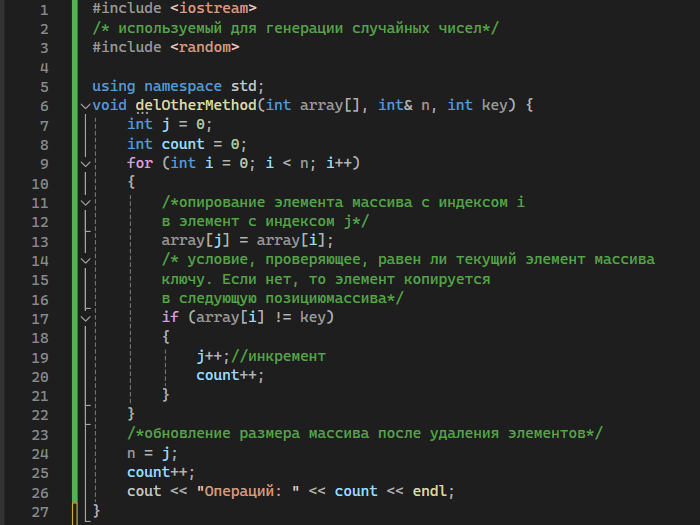


Рисунок 5.1 – Второй алгоритм

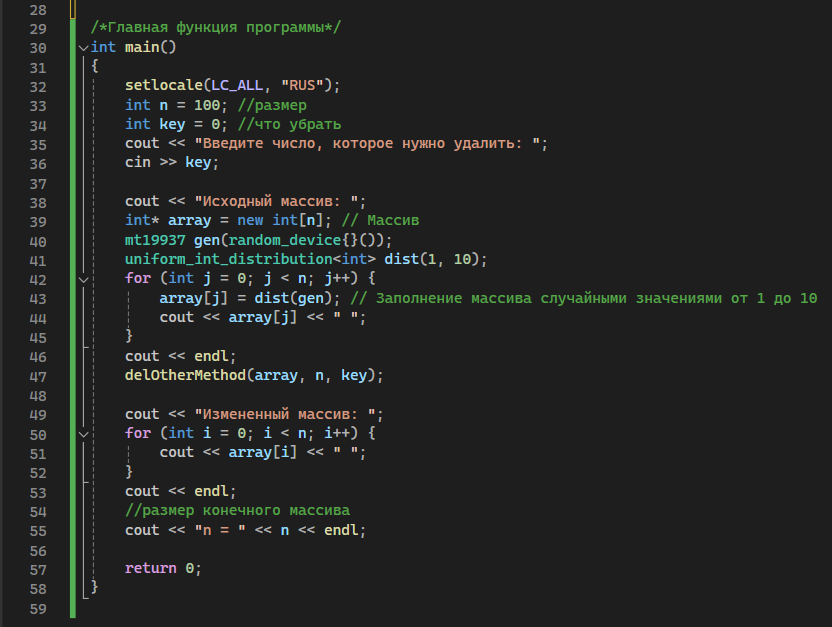


Рисунок 5.2 – Второй алгоритм

### 2.2.5 Тестирование

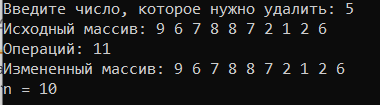


Рисунок 6.1 - Тестирование при 10 элементах. Случайная ситуация.

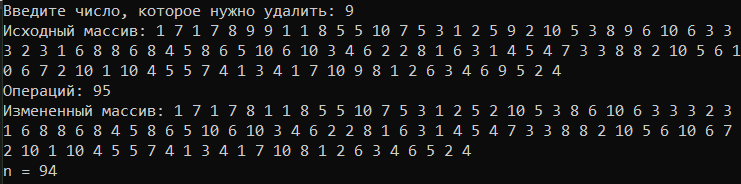


Рисунок 6.2 - Тестирование при 100 элементах, key=7. Случайная ситуация.

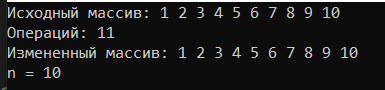


Рисунок 6.3 - Тестирование при 10 элементах и ничего не удаляется

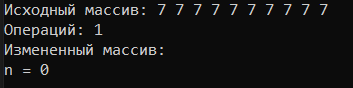


Рисунок 6.4 - Тестирование при 10 элементах и удаления всех.

## 2.3 Вывод по первому заданию

Основываясь на полученных результатах, можно сделать вывод, что второй алгоритм и в среднем и худшем случае требует намного меньше действий для выполнения.

Поскольку результаты теоретического расчета сложности практически совпадают с экспериментально полученными, можно заявить, что расчеты выполнены верно.

# 3 ЗАДАНИЕ 2

## 3.1 Описание математической модели

Математическая модель задачи представляет собой матрицу чисел MxN, с границами t, b, l, r. Цикл продолжается пока t <= b и l <= r, в каждой итерации выводятся элементы строки сверху слева направо и столбца справа сверху вниз. Границы t и r изменяются после каждой итерации.

## 3.2 Блок-схема алгоритма, доказательство корректности циклов

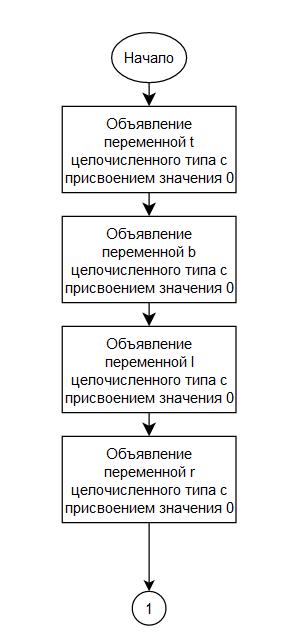


Рисунок 7.1 – Первая часть Блок-схемы

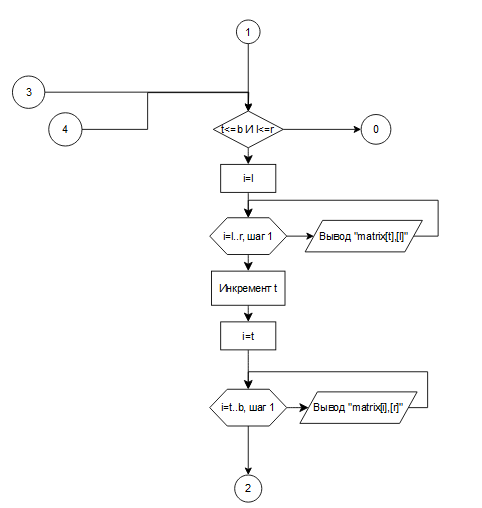


Рисунок 7.2 – Вторая часть Бок-схемы

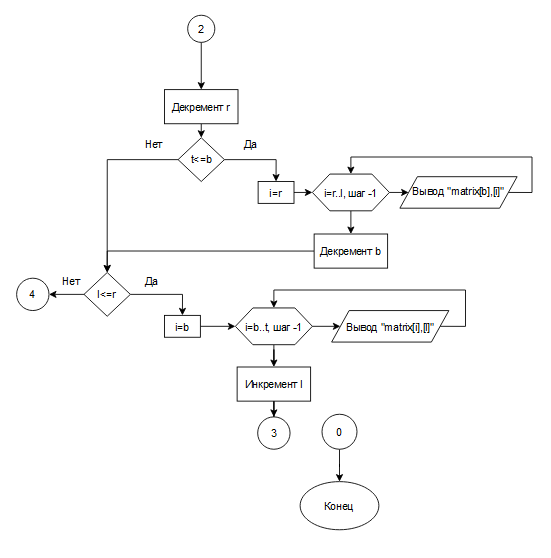


Рисунок 7.3 - Третья Блок-схема

Доказательство конечности циклов: В данной функции используется алгоритм спирали, который заключается в том, что элементы матрицы выводятся по спирали, начиная с верхнего левого угла и двигаясь вниз, затем вправо и так далее. Доказательство конечности циклов основывается на том, что размер матрицы конечен, а в каждом шаге цикла хотя бы один индекс изменяется. Поскольку индексы не могут превышать размер матрицы, то количество итераций цикла конечно и зависит от размера матрицы.

## 3.3 Определение вычислительной сложности алгоритма

|  |  |  |
| --- | --- | --- |
| Номер строки | Оператор | Количество повторений действия в  зависимости от объема входных  данных n |
| 1 | Void spiral |  |
| 2 | t = 0 | 1 |
| 3 | b = M - 1 | 1 |
| 4 | l = 0 | 1 |
| 5 | r = N - 1 | 1 |
| 6 | while (t <= b && l <= r) | n |
| 7 | for (int i = l; i <= r; i++) | n(n+1)=n^2+n |
| 8 | { |  |
| 9 | cout << matrix[t][i] << " "; | n-1 |
| 10 | } |  |
| 11 | t++; | n |
| 12 | p++; |  |
| 13 | for (int i = t; i <= b; i++) | n(n+1) |
| 14 | { |  |
| 15 | cout << matrix[i][r] << " "; | n-1 |
| 16 | } |  |
| 17 | r--; | n |
| 18 | p++; |  |
| 19 |  | n-1 |
| 20 | if (t <= b) | n |
| 21 | for (int i = r; i >= l; i--) | n(n+1) |
| 22 | { |  |
| 23 | cout << matrix[b][i] << " "; | n-1 |
| 24 | } |  |
| 25 | b--; | n |
| 26 | p++; |  |
| 27 |  | n-1 |
| 28 | if (l <= r) | n |
| 29 | for (int i = b; i >= t; i--) | n(n+1) |
| 30 | { |  |
| 31 | cout << matrix[i][l] << " "; | n-1 |
| 32 | } |  |
| 33 | l++; | n |
| 34 | p++; |  |
| 35 |  | n-1 |
| 36 |  | 1 |

Вычислительная сложность алгоритма в худшем, лучшем, среднем случае не будет отличаться своим значением функции так как при любых значениях элементов матрицы обход по спирали будут выполняться все циклы в программе, в результате чего получается квадратичная функция роста алгоритма

T(n)= 5+3n^2+3n+n-1+n-1+2n = 3n^2+6n+3

## 3.4 Реализация алгоритма на языке C++

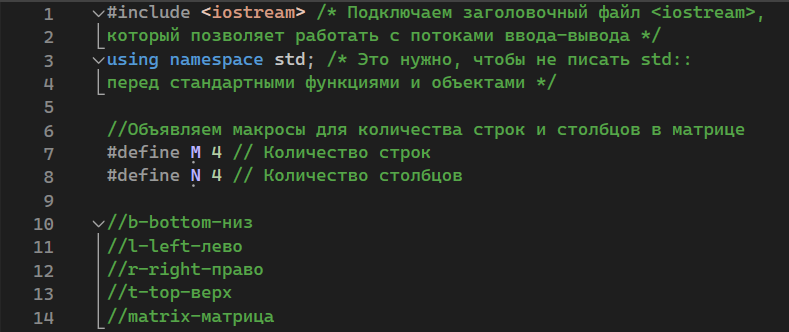


Рисунок 8.1 – Программа на С++

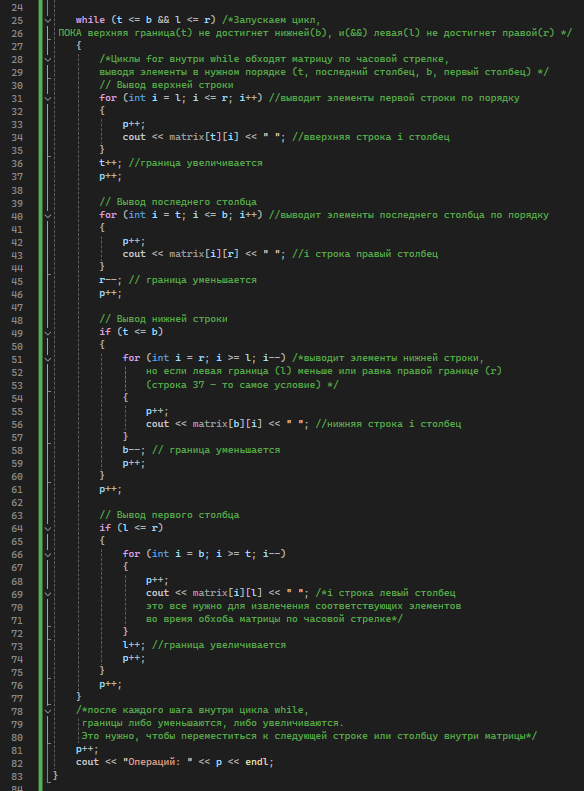


Рисунок 8.2 – Программа на C++

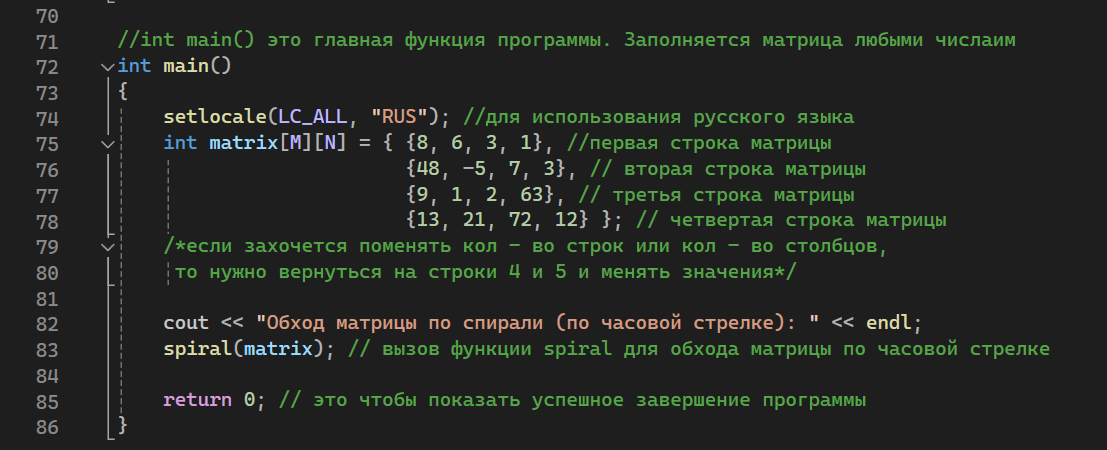


Рисунок 8.3 – Программа на C++

## 3.5 Тестирование

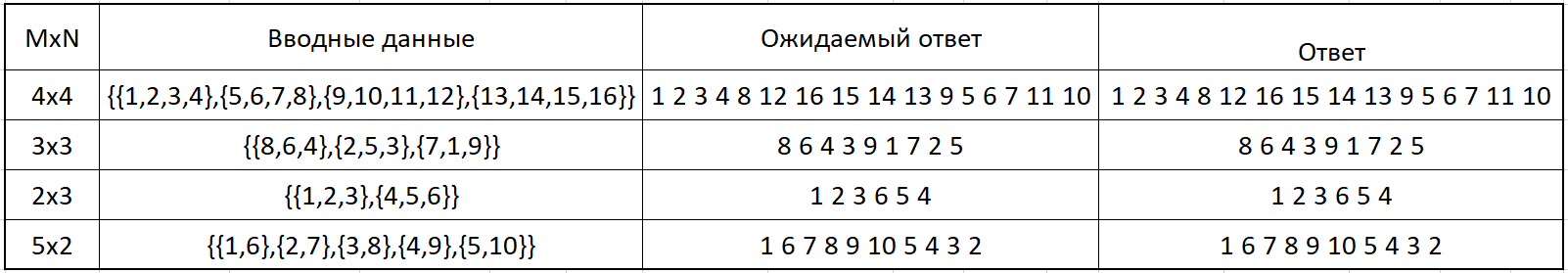


Рисунок 9 – Тестовые данные



Рисунок 10.1 – Тест для 4х4



Рисунок 10.2 – Тест для 3х3



Рисунок 10.3 – Тест для 2х3



Рисунок 10.4 – Тест для 5х2

Практическая оценка сложности алгоритма обхода матрицы по спирали для больших n будет зависеть от размеров матрицы. Наилучшая, наихудшая и средняя временная сложность алгоритма сохраняется и для различных размеров матриц.

1) Наилучший случай: В лучшем случае, когда матрица представляет собой квадратную матрицу, размером n x n, алгоритм обойдет матрицу за O(n^2) времени. В этом случае, все элементы матрицы будут пройдены без дополнительных условий.

2) Наихудший случай: В худшем случае, когда матрица представляет собой прямоугольную матрицу, размером m x n, где m ≠ n, алгоритм обойдет матрицу за O(mn) времени. В этом случае, алгоритм должен выполнить дополнительные проверки для обхода всех элементов.

3) Средний случай: - В среднем случае, для случайных матриц, также время работы алгоритма будет O(mn).

## 3.6 Выводы по заданию 2

В ходе работы был разработан алгоритм в соответствии с индивидуальным вариантом, оценена его сложность теоретическим и практическим методами. Основываясь на полученных данных, можно утверждать, что алгоритм по обходу матрицы по спирали (по часовой стрелке) имеет зависимость от размера матрицы.

# 4 ВЫВОДЫ

В ходе работы отработаны навыки определению:

* сложности алгоритмов на теоретическом и практическом уровнях;
* эффективного алгоритма решения задачи из нескольких.

Разработан собственный алгоритм решения задачи и оценена его эффективность. Тестирование подтвердило правильность решения задачи алгоритмом, а также правильность теоретического расчета вычислительной сложности алгоритмов.

# 5 ЛИТЕРАТУРА

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона, 2010.

2. Кнут Д. Искусство программирования. Тома 1-4, 1976-2013.

3. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для про-граммистов и любопытствующих, 2017.

4. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013.

5. Лафоре Р. Структуры данных и алгоритмы в Java. 2-е изд., 2013.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., 2018.

7. Скиена С. Алгоритмы. Руководство по разработке, 2011.

8. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2017.

9. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Инфор-матика и вычислительная биология, 2003.

По языку С++:

10. Страуструп Б. Программирование. Принципы и практика с использовани-ем C++. 2-е изд., 2016.

11. Павловская Т.А. C/C++. Программирование на языке высокого уровня, 2003.

12. Прата С. Язык программирования С++. Лекции и упражнения. - 6-е изд., 2012.

13. Седжвик Р. Фундаментальные алгоритмы на C++, 2001-2002

14. Хортон А. Visual C++ 2010. Полный курс, 2011.

15. Шилдт Г. Полный справочник по C++. 4-е изд., 2006.