МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

Лабораторная работа № 2

по дисциплине «Программирование»

на тему «Двумерные массивы»

Выполнили

студенты группы М3О-125БВ-24

Вариант №4

Егоров А.В.,

Федоров А.И.

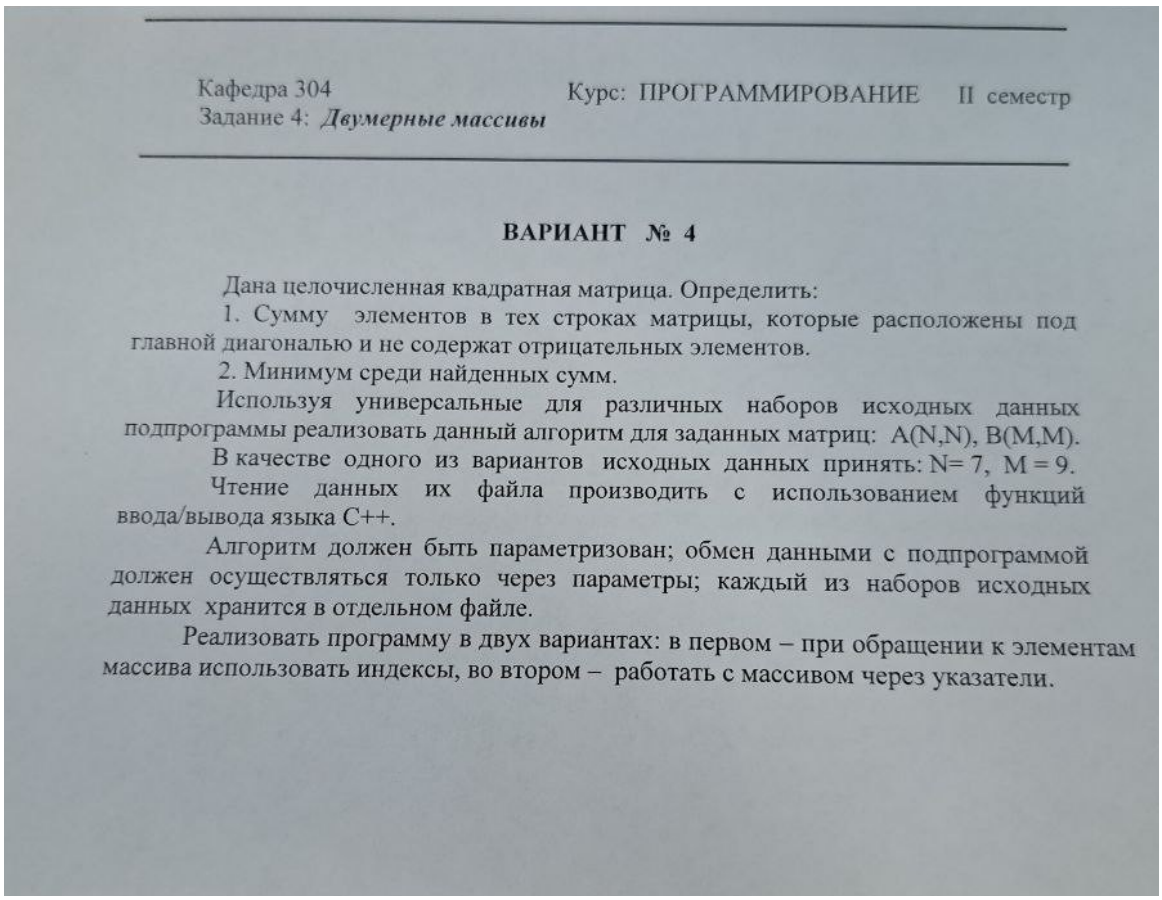
Приняли

ст. преп. каф. 304 Татаринкова Е.М.,

Москва
2025

Содержание:

Постановка задачи:



Кафедра 304 Курс: ПРОГРАММИРОВАНИЕ II семестр
Задание 4: *Двумерные массивы*

ВАРИАНТ № 4

Дана целочисленная квадратная матрица. Определить:

1. Сумму элементов в тех строках матрицы, которые расположены под главной диагональю и не содержат отрицательных элементов.
2. Минимум среди найденных сумм.

Используя универсальные для различных наборов исходных данных подпрограммы реализовать данный алгоритм для заданных матриц: A(N,N), B(M,M).

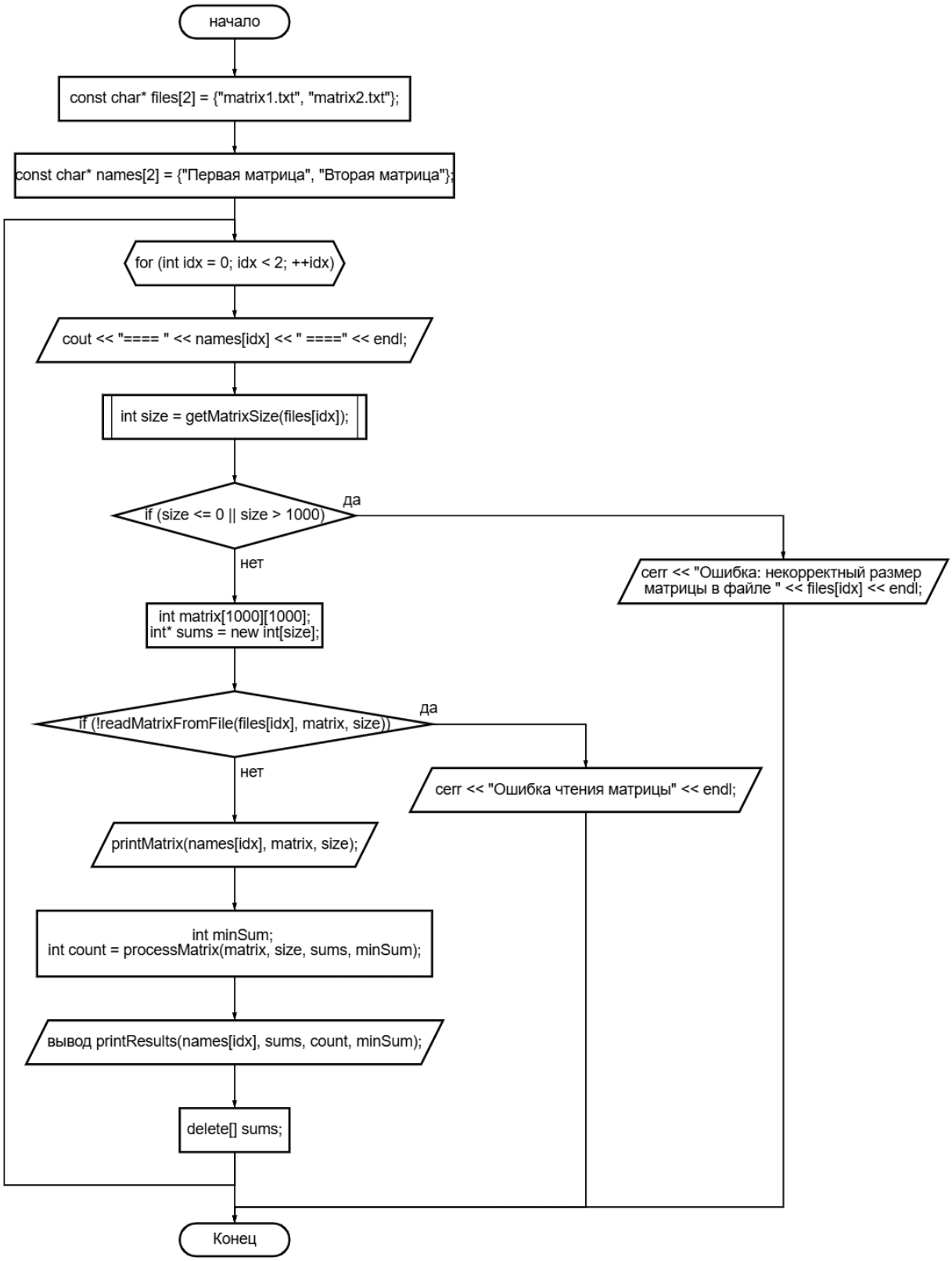
В качестве одного из вариантов исходных данных принять: N= 7, M = 9.

Чтение данных их файла производить с использованием функций ввода/вывода языка C++.

Алгоритм должен быть параметризован; обмен данными с подпрограммой должен осуществляться только через параметры; каждый из наборов исходных данных хранится в отдельном файле.

Реализовать программу в двух вариантах: в первом – при обращении к элементам массива использовать индексы, во втором – работать с массивом через указатели.

Блок-схема:


начало
const char* files[2] = {"matrix1.txt", "matrix2.txt"};
const char* names[2] = {"Первая матрица", "Вторая матрица"};
for (int idx = 0; idx < 2; ++idx)
cout << "==== " << names[idx] << " ====" << endl;
int size = getMatrixSize(files[idx]);
if (size <= 0 || size > 1000)
да
нет
cerr << "Ошибка: некорректный размер матрицы в файле " << files[idx] << endl;
int matrix[1000][1000];
int* sums = new int[size];
if (!readMatrixFromFile(files[idx], matrix, size))
да
нет
cerr << "Ошибка чтения матрицы" << endl;
printMatrix(names[idx], matrix, size);
int minSum;
int count = processMatrix(matrix, size, sums, minSum);
вывод printResults(names[idx], sums, count, minSum);
delete[] sums;
Конец

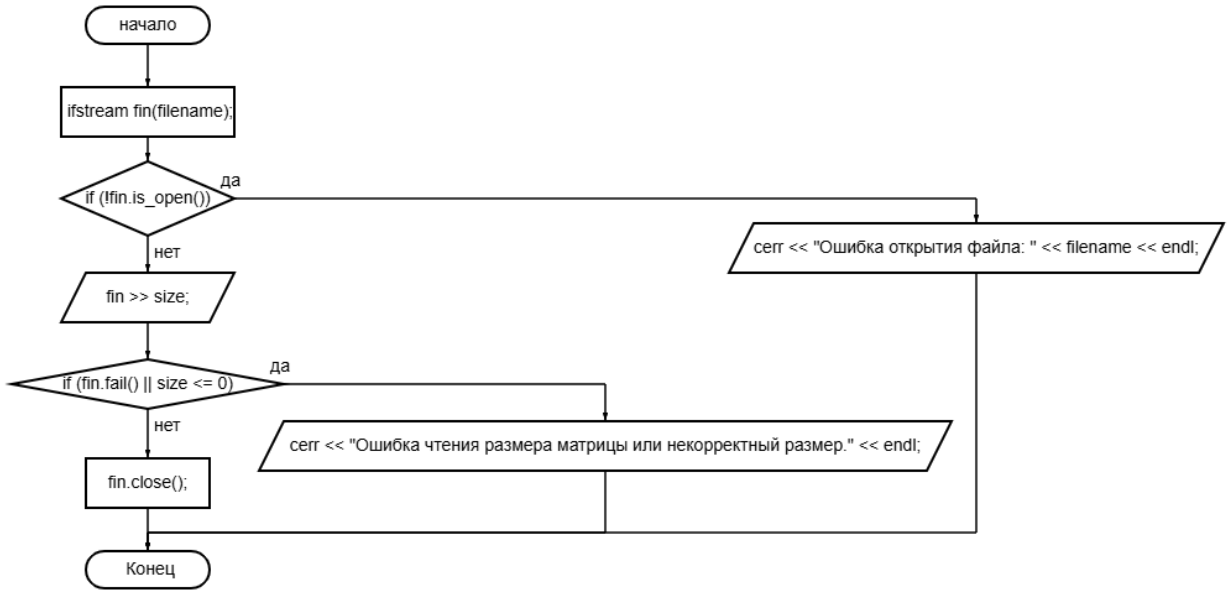
Описание функций:

Функция getMatrixSize:

1. Назначение: Получение размера квадратной матрицы из файла;
2. Прототип функции: int getMatrixSize(const char* filename);
3. Обращение: getMatrixSize(files[idx]);
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/Выхоной
getMatrixSize	int	Полкчение размера матрицы из файла	выходной
filename	const char*	Имя файла	входной

5. Блок-схема функции:

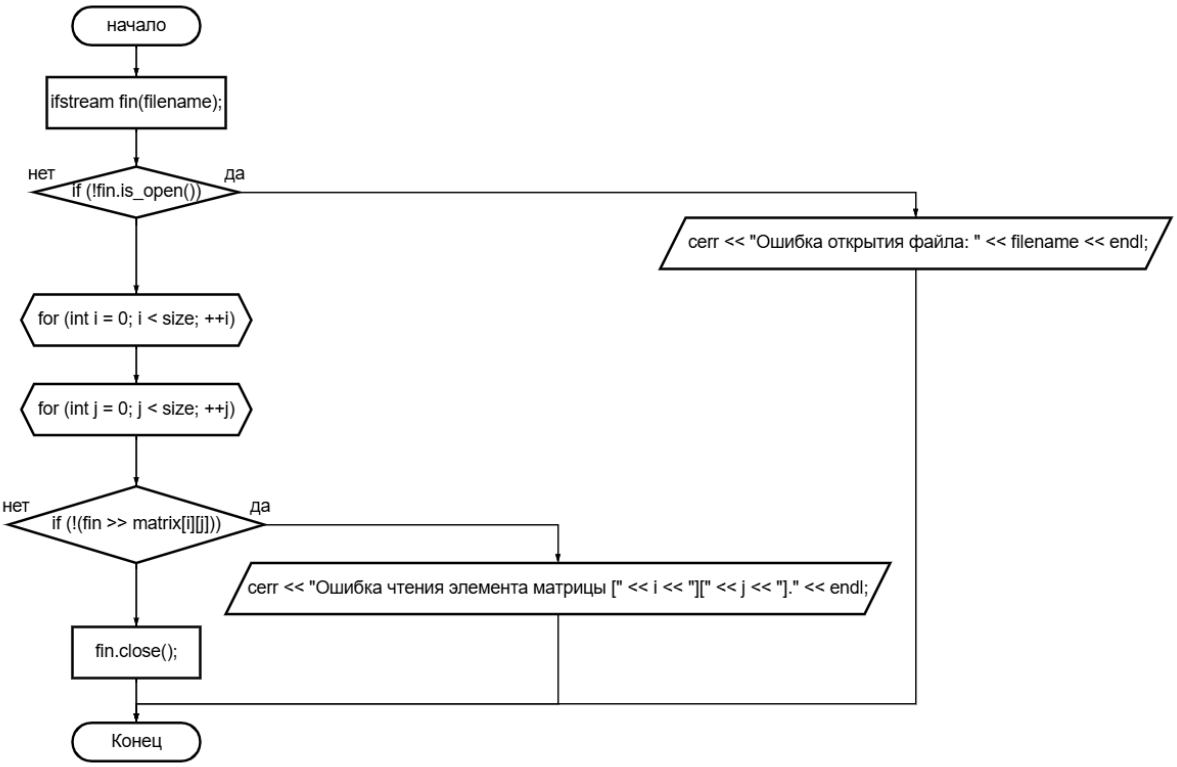


Функция readMatrixFromFile:

1. Назначение: Чтение квадратной матрицы из файла и заполнение одномерного массива;
2. Прототип функции: bool readMatrixFromFile(const char* filename, int* matrix, int size);
3. Обращение: readMatrixFromFile(files[idx], matrix, size);
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/Выхоной
readMatrixFromFile	bool	Чтение матрицы из файла и заполнение массива.	выходной
filename	const char*	Имя файла	входной
matrix	int*	Указатель на матрицу	входной
size	int	Размерность	входной

5. Блок-схема функции:

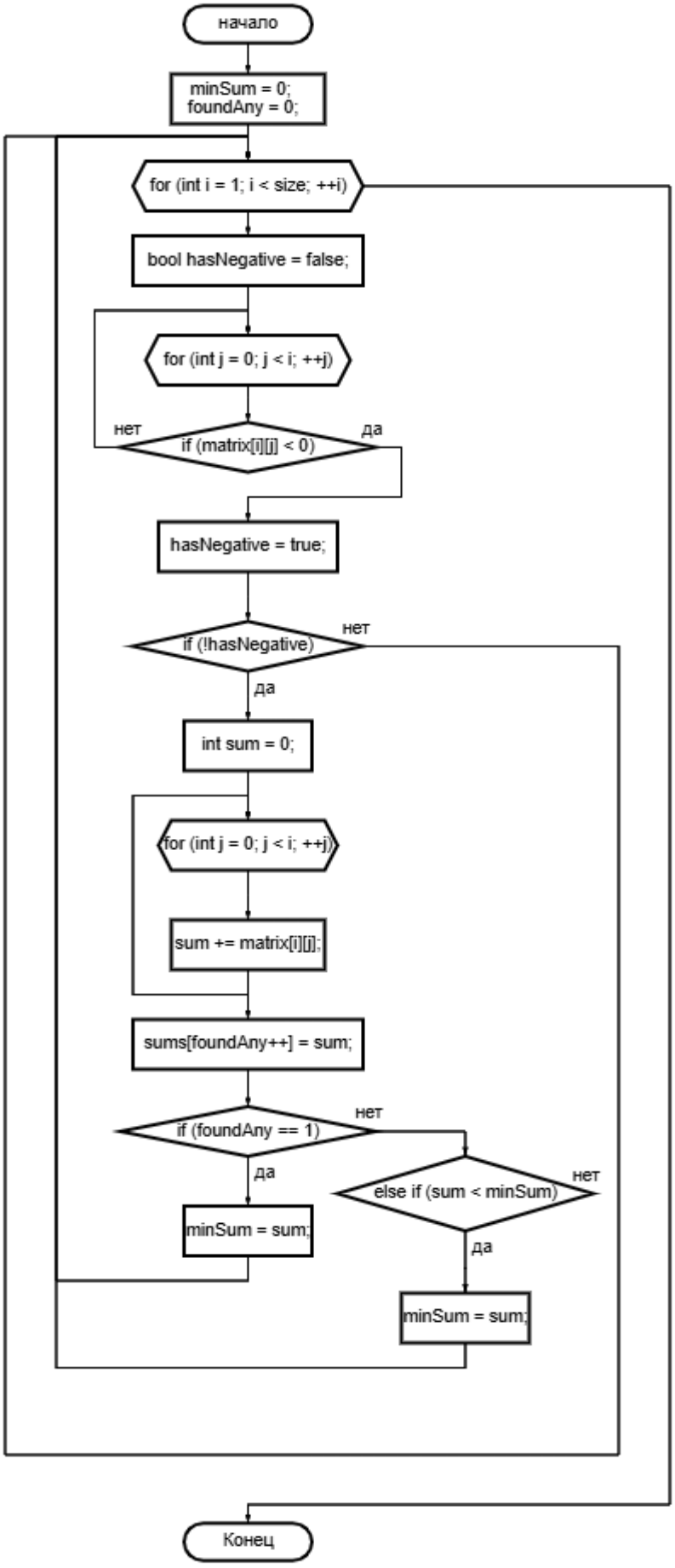


Функция processMatrix:

1. Назначение: поиск сумм в строках под главной диагональю без отрицательных элементов;
2. Прототип функции: int processMatrix(const int* matrix, int size, int* sums, int& minSum);
3. Обращение: processMatrix(matrix, size, sums, minSum);
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/Выхоной
processMatrix	int	поиск сумм в строках под главной диагональю без отрицательных элементов	выходной
matrix	const int*	указатель на массив	входной
size	int	размер	входной
sums	int*	массив для хранения сумм	входной
minSum	int&	минимум	входной

5. Блок-схема функции:

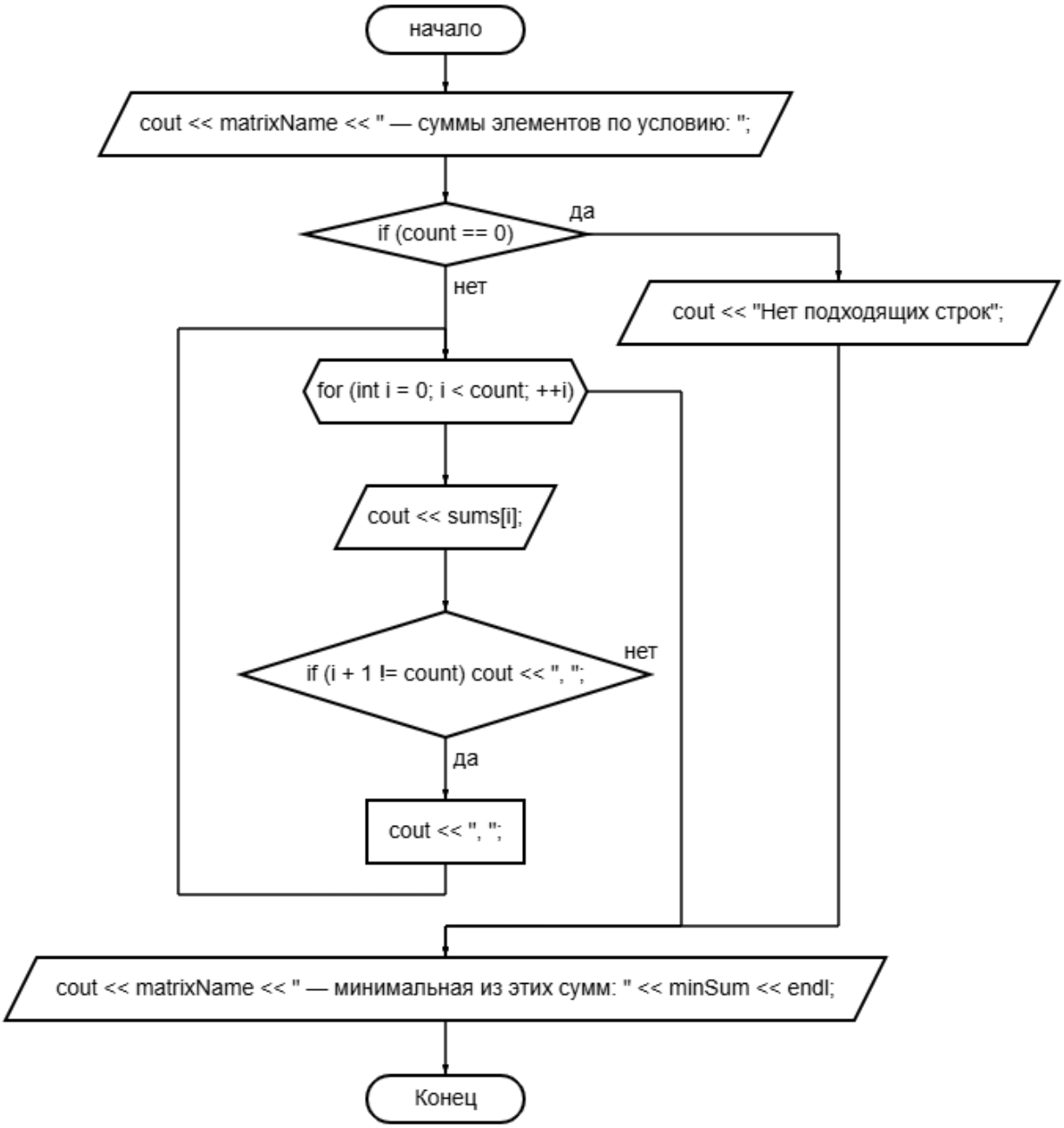


Функция printResults:

1. Назначение: Функция для вывода результатов анализа одной матрицы на экран;
2. Прототип функции: void printResults(const char* matrixName, int* sums, int count, int minSum);
3. Обращение: printResults(names[idx], sums, count, minSum);
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/Выхоной
printResults	void	Вывод результатов анализа одной матрицы на экран;	выходной
matrixName	const char*	Название матрицы	входной
sums	int*	Массив сумм	входной
count	int	Количество	входной
minSum	int	Минимум	входной

5. Блок-схема функции:

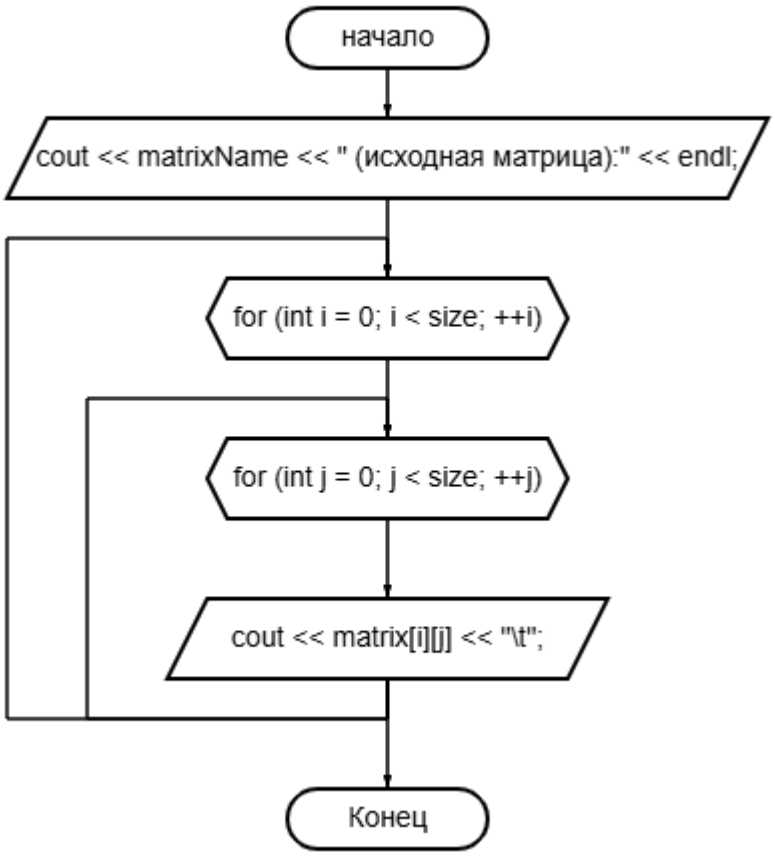


Функция printMatrix:

1. Назначение: Функция для вывода матрицы на экран;
2. Прототип функции: void printMatrix(const char* matrixName, int matrix[][1000], int size);
3. Обращение: printMatrix(names[idx], matrix, size);
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/Выхоной
printMatrix	void	Вывод матрицы на экран	выходной
matrixName	const char*	Название матрицы	входной
matrix	int	Двумерный массив	входной
size	int	Размерность	входной

5. Блок-схема функции:



Код программы:

Программа на индексах:

```
#include <iostream>
#include <fstream>
using namespace std;

// Получение размера квадратной матрицы из файла.
// Открывает файл, считывает первую строку (размер матрицы).
// Возвращает размер, если успешно, иначе -1.
int getMatrixSize(const char* filename);

// Чтение квадратной матрицы из файла и заполнение двумерного массива.
// filename – имя файла, matrix – двумерный массив, size – размерность
матрицы.
// Возвращает true при успешном чтении, иначе false.
bool readMatrixFromFile(const char* filename, int matrix[][1000], int
size);

// Обработка матрицы: поиск сумм в строках под главной диагональю без
отрицательных элементов.
// matrix – матрица, size – размер, sums – массив для хранения сумм, minSum
– минимальная из сумм.
// Возвращает количество подходящих строк.
int processMatrix(int matrix[][1000], int size, int* sums, int& minSum);

// Функция для вывода результатов анализа одной матрицы на экран.
// matrixName – название матрицы (для вывода), sums – массив сумм, count –
количество, minSum – минимум.
void printResults(const char* matrixName, int* sums, int count, int
minSum);

// Функция для вывода матрицы на экран.
// matrix – двумерный массив, size – размерность, matrixName – название
матрицы.
void printMatrix(const char* matrixName, int matrix[][1000], int size);

int main() {
    // Массив имён файлов с матрицами (два файла)
    const char* files[2] = {"matrix1.txt", "matrix2.txt"};
    // Массив имён для вывода на экран
    const char* names[2] = {"Первая матрица", "Вторая матрица"};

    // Последовательно обрабатываем оба файла с матрицами
    for (int idx = 0; idx < 2; ++idx) {
        cout << "==== " << names[idx] << " ====" << endl;

        // Получаем размерность матрицы из файла
        int size = getMatrixSize(files[idx]);
        if (size <= 0 || size > 1000) {
            cerr << "Ошибка: некорректный размер матрицы в файле " <<
files[idx] << endl;
            continue;
        }

        // Объявляем статический двумерный массив (до 1000x1000)
```

```
        int matrix[1000][1000];
        // Динамически выделяем память под массив найденных сумм (не больше
size-1, но для простоты size)
        int* sums = new int[size];

        // Читаем матрицу из файла
        if (!readMatrixFromFile(files[idx], matrix, size)) {
            cerr << "Ошибка чтения матрицы" << endl;
            delete[] sums;
            continue;
        }

        // Выводим исходную матрицу перед обработкой
        printMatrix(names[idx], matrix, size);

        // minSum — для хранения минимальной из найденных сумм
        int minSum;
        // count — количество подходящих строк (по условию задачи)
        int count = processMatrix(matrix, size, sums, minSum);

        // Выводим результаты на экран
        printResults(names[idx], sums, count, minSum);

        // Освобождаем память под массив найденных сумм
        delete[] sums;
        cout << endl;
    }
    return 0;
}

int getMatrixSize(const char* filename) {
    // Открываем файл для чтения
    ifstream fin(filename);
    if (!fin.is_open()) {
        cerr << "Ошибка открытия файла: " << filename << endl;
        return -1;
    }
    int size = -1;
    // Считываем размер матрицы (первое число в файле)
    fin >> size;
    // Проверка на ошибку чтения и корректность размера
    if (fin.fail() || size <= 0) {
        cerr << "Ошибка чтения размера матрицы или некорректный размер." <<
endl;
        fin.close();
        return -1;
    }
    fin.close();
    return size;
}

bool readMatrixFromFile(const char* filename, int matrix[][1000], int size)
{
    // Открываем файл для чтения
    ifstream fin(filename);
    if (!fin.is_open()) {
        cerr << "Ошибка открытия файла: " << filename << endl;
```

```
        return false;
    }
    int tmp;
    // Пропускаем размер матрицы
    fin >> tmp;
    // Читаем элементы матрицы по строкам
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < size; ++j) {
            if (!(fin >> matrix[i][j])) {
                cerr << "Ошибка чтения элемента матрицы [" << i << "][" << j
<< "]." << endl;
                fin.close();
                return false;
            }
        }
    fin.close();
    return true;
}

int processMatrix(int matrix[][1000], int size, int* sums, int& minSum) {
    // minSum — минимальная из найденных сумм, если нет строк — останется 0
    minSum = 0;
    // foundAny — количество строк, удовлетворяющих условию
    int foundAny = 0;
    // Проходим по всем строкам, находящимся ниже главной диагонали (i > 0)
    for (int i = 1; i < size; ++i) {
        bool hasNegative = false; // Флаг наличия отрицательного элемента в
строке
        // Проверяем только элементы под главной диагональю (j < i)
        for (int j = 0; j < i; ++j) {
            if (matrix[i][j] < 0) {
                hasNegative = true;
                break; // Если встречен отрицательный элемент — строка не
подходит
            }
        }
        // Если в строке нет отрицательных элементов — считаем сумму и
сохраняем
        if (!hasNegative) {
            int sum = 0;
            for (int j = 0; j < i; ++j)
                sum += matrix[i][j];
            sums[foundAny++] = sum;
            // Первый подходящий элемент — сразу пишем в minSum
            if (foundAny == 1) minSum = sum;
            // Для последующих — ищем минимум
            else if (sum < minSum) minSum = sum;
        }
    }
    // Если ни одной подходящей строки не найдено, minSum остаётся 0
    return foundAny;
}

void printResults(const char* matrixName, int* sums, int count, int minSum)
{
    // Выводим все найденные суммы по условию задачи
    cout << matrixName << " — суммы элементов по условию: ";
```

```
    if (count == 0) {
        cout << "Нет подходящих строк";
    } else {
        for (int i = 0; i < count; ++i) {
            cout << sums[i];
            if (i + 1 != count) cout << ", ";
        }
    }
    cout << endl;
    // Выводим минимальную из этих сумм
    cout << matrixName << " — минимальная из этих сумм: " << minSum << endl;
}

// Функция для вывода матрицы на экран
void printMatrix(const char* matrixName, int matrix[][1000], int size) {
    cout << matrixName << " (исходная матрица):" << endl;
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            cout << matrix[i][j] << "\t";
        }
        cout << endl;
    }
    cout << endl;
}
```

Программа на указателях:

```
#include <iostream>
#include <fstream>
using namespace std;

// Получение размера квадратной матрицы из файла (размер – первая строка).
// Если файл не удаётся открыть или данные некорректны, возвращает -1.
int getMatrixSize(const char* filename);

// Чтение квадратной матрицы из файла и заполнение одномерного массива
(размещённого динамически).
// filename – имя файла, matrix – указатель на массив, size – размерность.
// Возвращает true при успехе, иначе false.
bool readMatrixFromFile(const char* filename, int* matrix, int size);

// Обработка матрицы: поиск сумм в строках под главной диагональю без
отрицательных элементов.
// matrix – указатель на массив, size – размер, sums – массив для хранения
сумм, minSum – минимум.
// Возвращает количество подходящих строк.
int processMatrix(const int* matrix, int size, int* sums, int& minSum);

// Функция для вывода результатов анализа одной матрицы
// matrixName – имя для вывода, sums – массив сумм, count – количество,
minSum – минимум
void printResults(const char* matrixName, int* sums, int count, int
minSum);

int main() {
    // Массив имён файлов с матрицами (два файла)
    const char* files[2] = {"matrix1.txt", "matrix2.txt"};
    // Массив имён для вывода на экран
    const char* names[2] = {"Первая матрица", "Вторая матрица"};

    // Последовательно обрабатываем оба файла с матрицами с помощью
арифметики указателей
    for (int idx = 0; idx < 2; ++idx) {
        cout << "==== " << names[idx] << " ====" << endl;

        // Получаем размерность матрицы из файла
        int size = getMatrixSize(files[idx]);
        if (size <= 0 || size > 1000) {
            cerr << "Ошибка: некорректный размер матрицы в файле " <<
files[idx] << endl;
            continue;
        }

        // Динамически выделяем память под одномерный массив для хранения
матрицы
        int* matrix = nullptr;
        int* sums = nullptr;
        try {
            matrix = new int[size * size];
            sums = new int[size];
        } catch (bad_alloc&) {
            cerr << "Ошибка выделения памяти." << endl;
```

```
            delete[] matrix;
            delete[] sums;
            continue;
        }

        // Читаем матрицу из файла в одномерный массив
        if (!readMatrixFromFile(files[idx], matrix, size)) {
            cerr << "Ошибка чтения матрицы" << endl;
            delete[] matrix;
            delete[] sums;
            continue;
        }

        // minSum — для хранения минимальной из найденных сумм
        int minSum;
        // count — количество подходящих строк (по условию задачи)
        int count = processMatrix(matrix, size, sums, minSum);

        // Выводим результаты на экран
        printResults(names[idx], sums, count, minSum);

        // Освобождаем память под массивы
        delete[] matrix;
        delete[] sums;
        cout << endl;
    }
    return 0;
}

int getMatrixSize(const char* filename) {
    // Открываем файл для чтения
    ifstream fin(filename);
    if (!fin.is_open()) {
        cerr << "Ошибка открытия файла: " << filename << endl;
        return -1;
    }
    int size = -1;
    // Считываем размер матрицы (первое число в файле)
    fin >> size;
    // Проверка на ошибку чтения и корректность размера
    if (fin.fail() || size <= 0) {
        cerr << "Ошибка чтения размера матрицы или некорректный размер." <<
endl;
        fin.close();
        return -1;
    }
    fin.close();
    return size;
}

bool readMatrixFromFile(const char* filename, int* matrix, int size) {
    // Открываем файл для чтения
    ifstream fin(filename);
    if (!fin.is_open()) {
        cerr << "Ошибка открытия файла: " << filename << endl;
        return false;
    }
```

```
    int tmp;
    // Пропускаем размер матрицы
    fin >> tmp;
    // Заполняем одномерный массив, используя арифметику указателей
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < size; ++j) {
            if (!(fin >> *(matrix + i*size + j))) {
                cerr << "Ошибка чтения элемента матрицы [" << i << "][" << j
<< "]." << endl;
                fin.close();
                return false;
            }
        }
    fin.close();
    return true;
}

int processMatrix(const int* matrix, int size, int* sums, int& minSum) {
    // minSum — минимальная из найденных сумм, если нет строк — останется 0
    minSum = 0;
    // foundAny — количество строк, удовлетворяющих условию
    int foundAny = 0;
    // Проходим по всем строкам, находящимся ниже главной диагонали (i > 0)
    for (int i = 1; i < size; ++i) {
        bool hasNegative = false; // Флаг наличия отрицательного элемента в
строке
        // Проверяем только элементы под главной диагональю (j < i)
        for (int j = 0; j < i; ++j) {
            if (*(matrix + i*size + j) < 0) {
                hasNegative = true;
                break; // Если встречен отрицательный элемент — строка не
подходит
            }
        }
        // Если в строке нет отрицательных элементов — считаем сумму и
сохраняем
        if (!hasNegative) {
            int sum = 0;
            for (int j = 0; j < i; ++j)
                sum += *(matrix + i*size + j);
            sums[foundAny++] = sum;
            // Первый подходящий элемент — сразу пишем в minSum
            if (foundAny == 1) minSum = sum;
            // Для последующих — ищем минимум
            else if (sum < minSum) minSum = sum;
        }
    }
    // Если ни одной подходящей строки не найдено, minSum остаётся 0
    return foundAny;
}

void printResults(const char* matrixName, int* sums, int count, int minSum)
{
    // Выводим все найденные суммы по условию задачи
    cout << matrixName << " — суммы элементов по условию: ";
    if (count == 0) {
        cout << "Нет подходящих строк";
```

```
    } else {
        for (int i = 0; i < count; ++i) {
            cout << sums[i];
            if (i + 1 != count) cout << ", ";
        }
    }
    cout << endl;
    // Выводим минимальную из этих сумм
    cout << matrixName << " – минимальная из этих сумм: " << minSum << endl;
}
```

Тесты:

Корректные тесты:

Тест 1 (Индексы):

Матрица 1:

```
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
36 37 38 39 40 41 42
43 44 45 46 47 48 49
```

Матрица 2:

```
 2  1  0 -1  4  5  6  7  8
 9  2  1  0  1  2  3  4  5
 6  7  2  1  0  1  2  3  4
 5  6  7  2  1  0  1  2  3
 4  5  6  7  2  1  0  1  2
 3  4  5  6  7  2  1  0  1
 2  3  4  5  6  7  2  1  0
-1  2  3  4  5  6  7  2  1
 0 -1  2  3  4  5  6  7  2
```

```
❯ ./main_indexes
==== Первая матрица ====
Первая матрица — суммы элементов по условию: 8, 31, 69, 122, 190, 273
Первая матрица — минимальная из этих сумм: 8

==== Вторая матрица ====
Вторая матрица — суммы элементов по условию: 9, 13, 18, 22, 25, 27
Вторая матрица — минимальная из этих сумм: 9
```

Тест 2 (Индексы):

Матрица 1:

```
1 0 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24
```

Матрица 2:

```
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
```

```
❯ ./main_indexes
==== Первая матрица ====
Первая матрица — суммы элементов по условию: 5, 21, 48, 86
Первая матрица — минимальная из этих сумм: 5

==== Вторая матрица ====
Вторая матрица — суммы элементов по условию: 7, 27, 60, 106, 165
Вторая матрица — минимальная из этих сумм: 7
```

Тест 3 (Указатели):

Матрица 1:

```
12 45 78  3 26 59 34
67 88 10 41  5 77 23
31  6 90 53 28 14 66
72 19 11  7 80 38 61
49 35 17 96 20  1 69
84 62 44 30 55  9 73
16 25 18 86 33 60 47
```

Матрица 2:

```
91  4 37 22
68 13 50 11
 7 29 82 48
60  2 35 99
```

```
❯ ./main_pointers
==== Первая матрица ====
Первая матрица — суммы элементов по условию: 67, 37, 102, 197, 275, 238
Первая матрица — минимальная из этих сумм: 37

==== Вторая матрица ====
Вторая матрица — суммы элементов по условию: 68, 36, 97
Вторая матрица — минимальная из этих сумм: 36
```

Некорректные тесты:

Тест 1 (Отсутствие файла матрицы 1):

```
❯ ./main_indexes
==== Первая матрица ====
Ошибка открытия файла: matrix1.txt
Ошибка: некорректный размер матрицы в файле matrix1.txt
==== Вторая матрица ====
Вторая матрица – суммы элементов по условию: 68, 36, 97
Вторая матрица – минимальная из этих сумм: 36
```

Тест 2 (Отсутствие файла матрицы 2):

```
❯ ./main_indexes
==== Первая матрица ====
Первая матрица – суммы элементов по условию: 67, 37, 102, 197, 275, 238
Первая матрица – минимальная из этих сумм: 37

==== Вторая матрица ====
Ошибка открытия файла: matrix2.txt
Ошибка: некорректный размер матрицы в файле matrix2.txt
```

Вывод по работе:

Разработка программы завершена на том основании, что:

1. Полученный результаты совпали с ожидаемыми;
2. Считаем набор тестов полным.