Министерство образования и науки РФ
Государственное образовательное учреждение
высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

ДИНАМИЧЕСКИЕ ПЕРЕМЕННЫЕ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ С

Лабораторная работа № 12

по курсу «Программирование на ЯВУ»

Вариант № 4

Выполнил: студент группы 220601 _____ Белым А.А.
(подпись)

Проверил: к. ф.-м. н., доцент _____ Сулимова В.В.
(подпись)

Тула 2011

Цель работы

Цель работы заключается в том, чтобы научиться использовать динамические переменные. В данной работе требуется написать программу с использованием динамических переменных, в которой данные сначала из файла считываются в память, потом обрабатываются и записываются в файл.

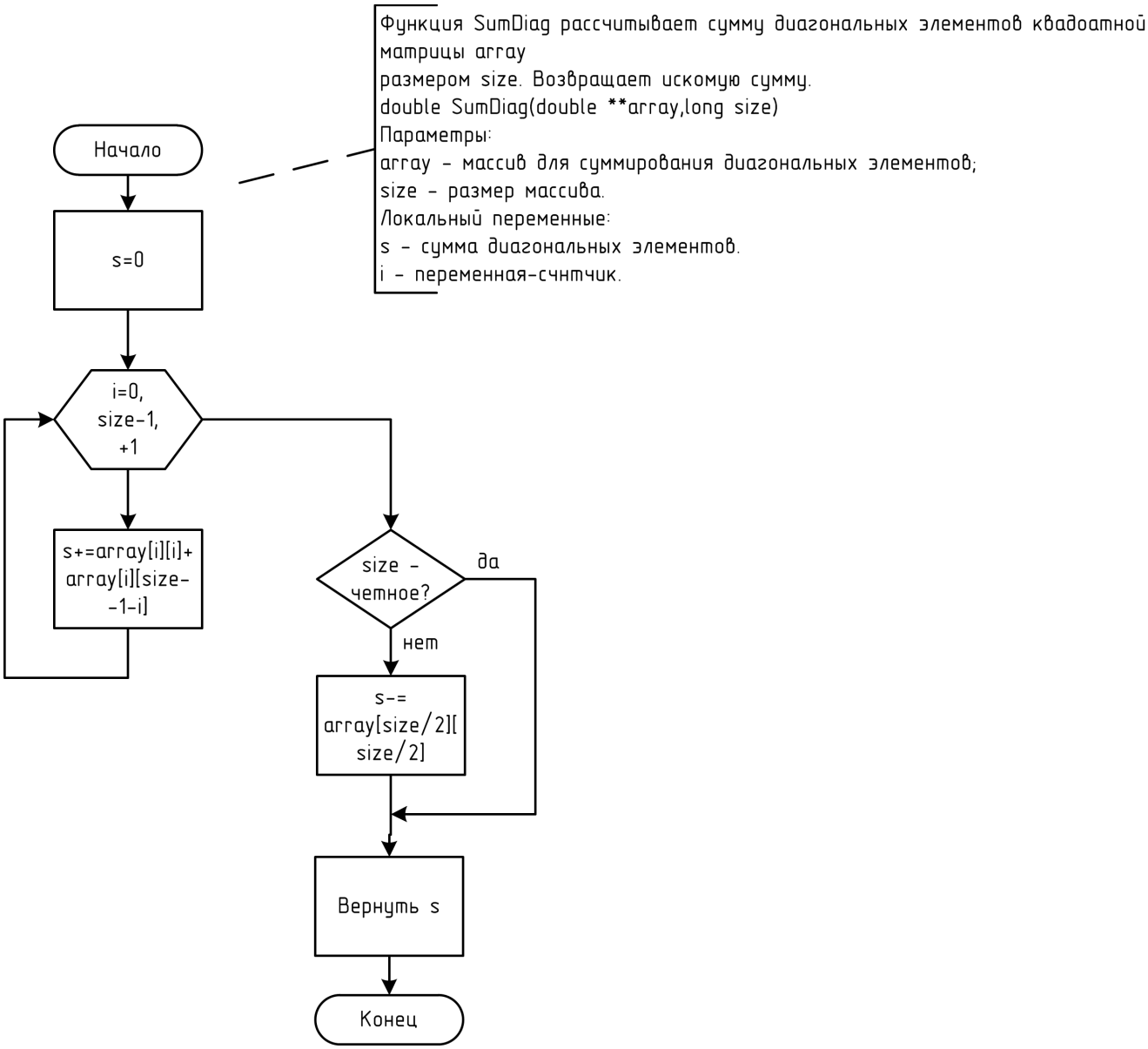
Задание

Вычислить сумму элементов, лежащих на диагоналях матрицы n x n (обратить внимание на четность-нечетность числа n). Результат работы программы должен выводиться на экран и в файл.

Теоретическая справка

1. Схема алгоритма

На рисунке 3.1 представлена схема алгоритма подсчета суммы диагональных элементов квадратной матрицы.



Рисункок 1 - Схема алгоритма подсчета суммы диагональных элементов квадратной матрицы

Схему общего алгоритма (ввод данных в матрицу, её проверка, подсчет суммы и вывод результатов) сортировки главной диагонали можно увидеть на рисунке 2.

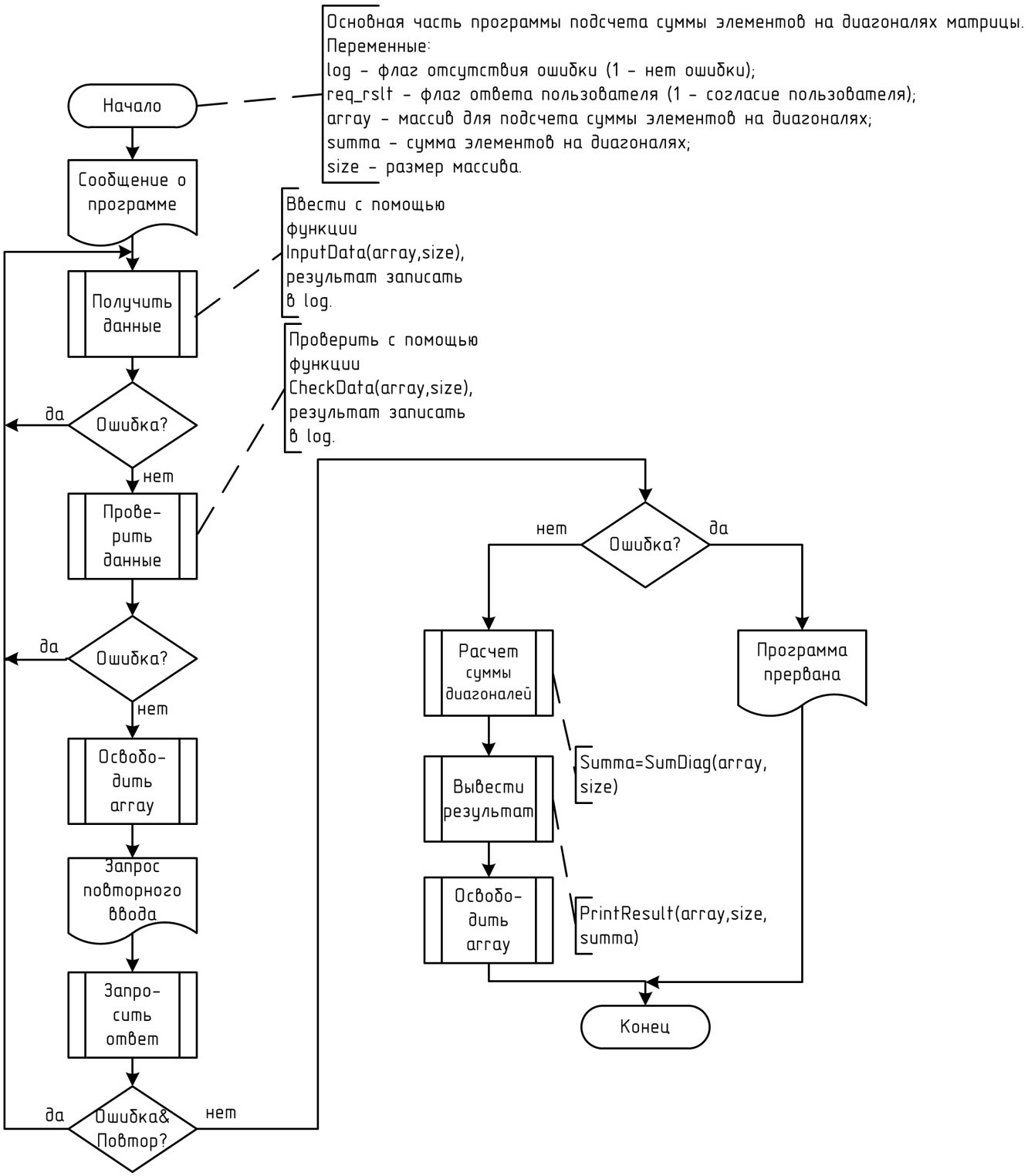


Рисунок 2 - Схема алгоритма заполнения, подсчета суммы диагональных элементов и вывода результата

Так как память под матрицу выделяется динамически, то соответственно нужны процедуры выделения памяти и освобождения памяти. Блок-схемы данных алгоритмов представлены на рисунках 3 и 4.

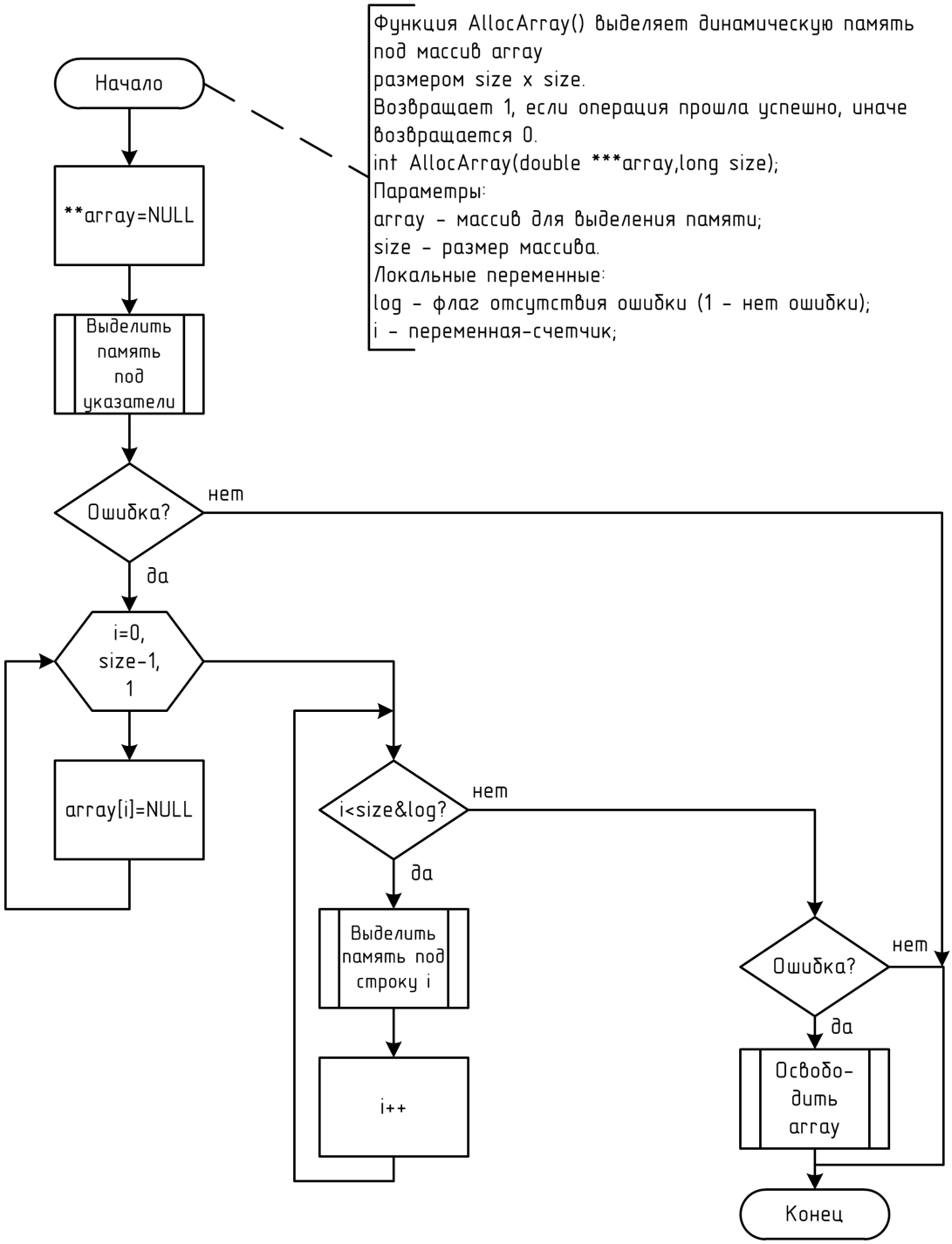


Рисунок 3 - Схема алгоритма выделения памяти под матрицу

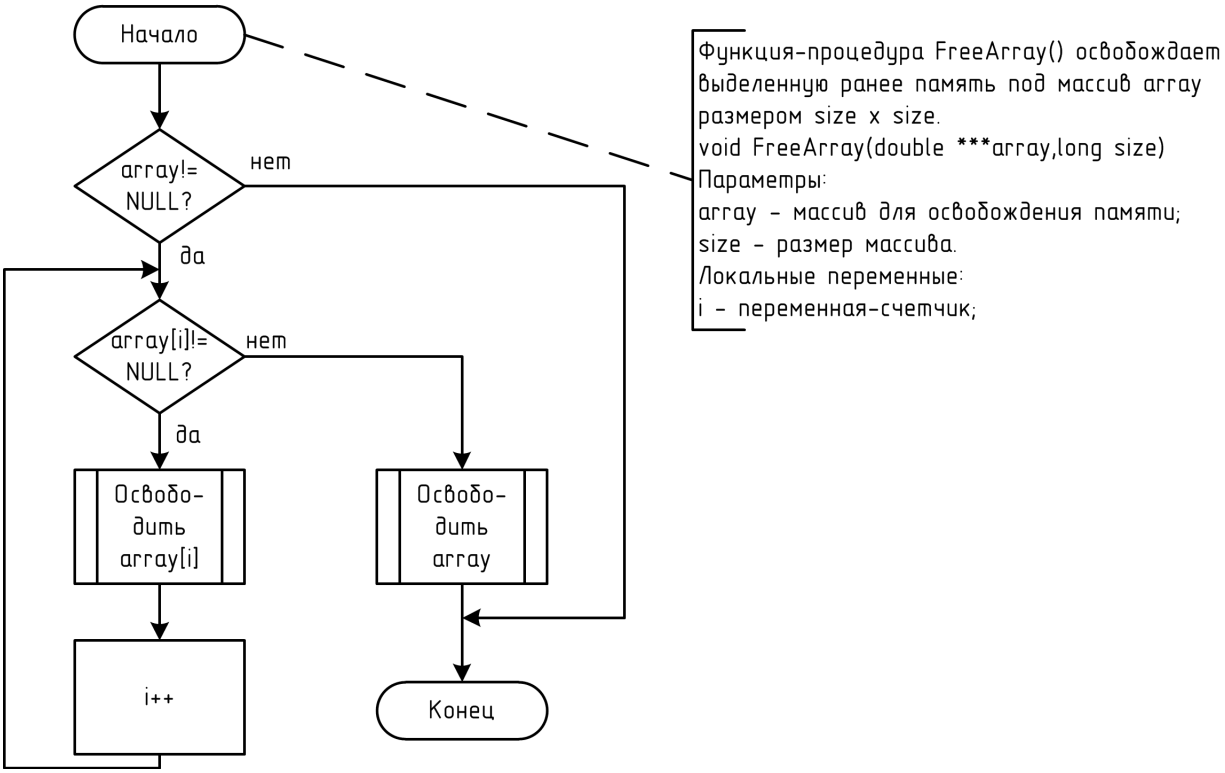
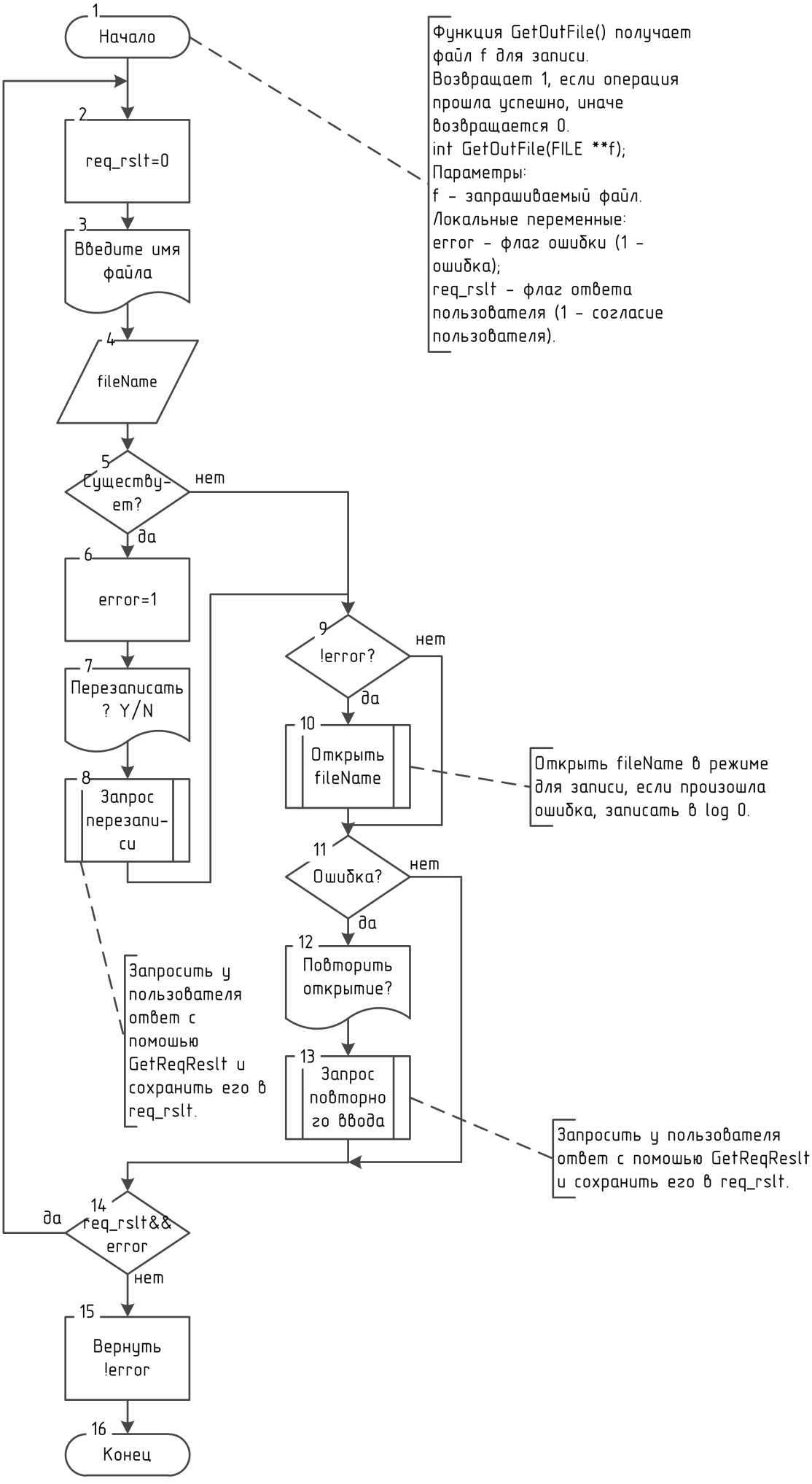


Рисунок 4 - Схема алгоритма освобождения памяти под матрицу

На рисунках 5 и 6 представлены схемы алгоритмов получения входного и выходного файла.


1
Начало
2
req_rslt=0
3
Введите имя файла
4
fileName
5
Существу-ет?
нет
да
6
error=1
7
Перезаписать ? Y/N
8
Запрос перезапи-си
9
!error?
нет
да
10
Открыть fileName
11
Ошибка?
нет
да
12
Повторить открытие?
13
Запрос повторно го ввода
14
req_rslt&& error
да
нет
15
Вернуть !error
16
Конец
Функция GetOutFile() получает файл f для записи.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
int GetOutFile(FILE **f);
Параметры:
f – запрашиваемый файл.
Локальные переменные:
error – флаг ошибки (1 – ошибка);
req_rslt – флаг ответа пользователя (1 – согласие пользователя).
Открыть fileName в режиме для записи, если произошла ошибка, записать в log 0.
Запросить у пользователя ответ с помошью GetReqReslt и сохранить его в req_rslt.
Запросить у пользователя ответ с помошью GetReqReslt и сохранить его в req_rslt.

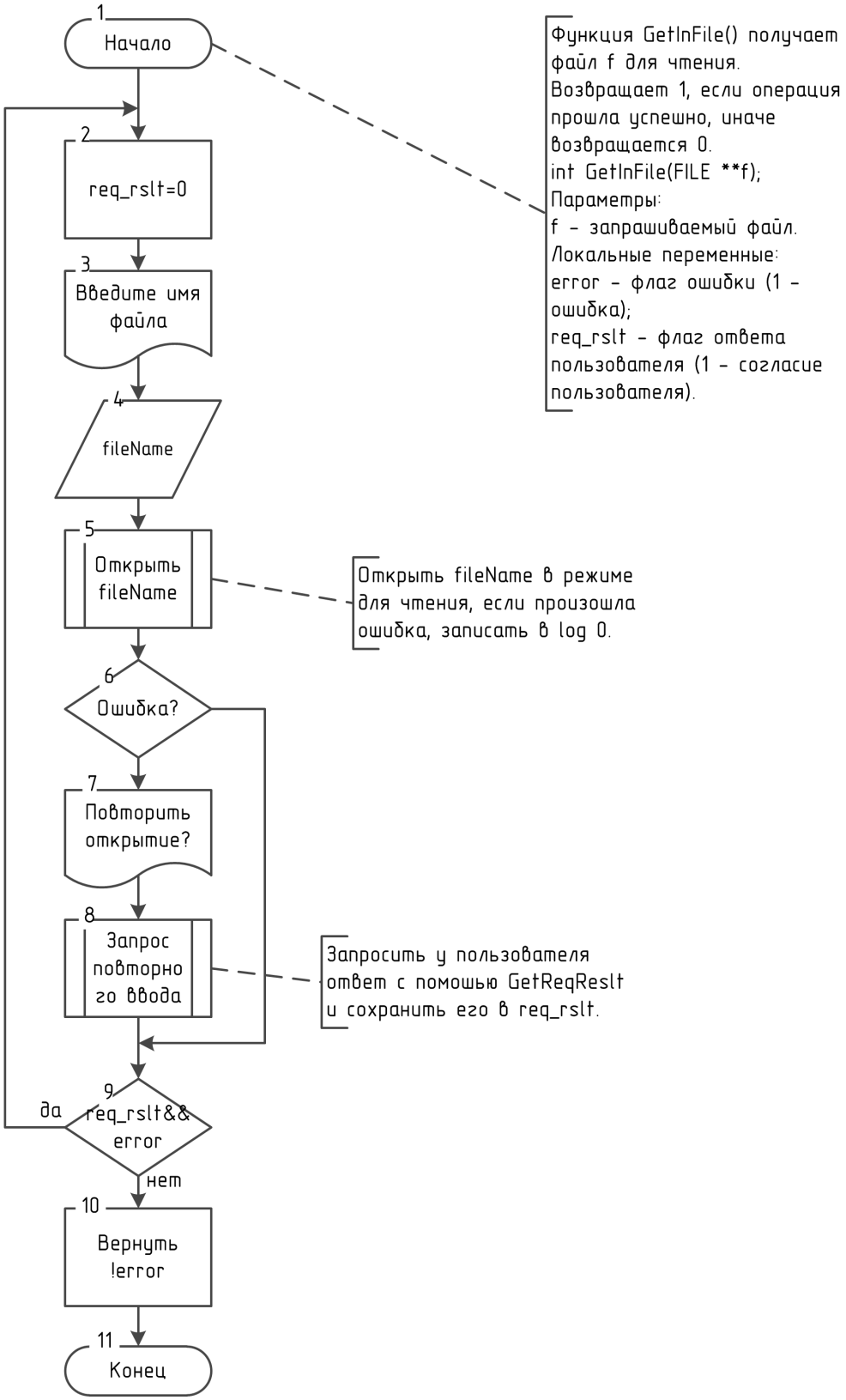


Рисунок 5 - Схема алгоритма получения файла-источника

На рисунках 7, 8-9 и 10 представлены схемы алгоритмов заполнения матрицы различными способами: на рисунке 7 — с клавиатуры, рисунках 8 и 9 — из файла, и на рисунке 10 — случайными числами. Естественно, при таком

Рисунок 1.10 - Схема алгоритма получения файла-результата

количестве различных способов алгоритм выбора лучше поместить в отдельную подпрограмму; его схему можно увидеть на рисунке 11.

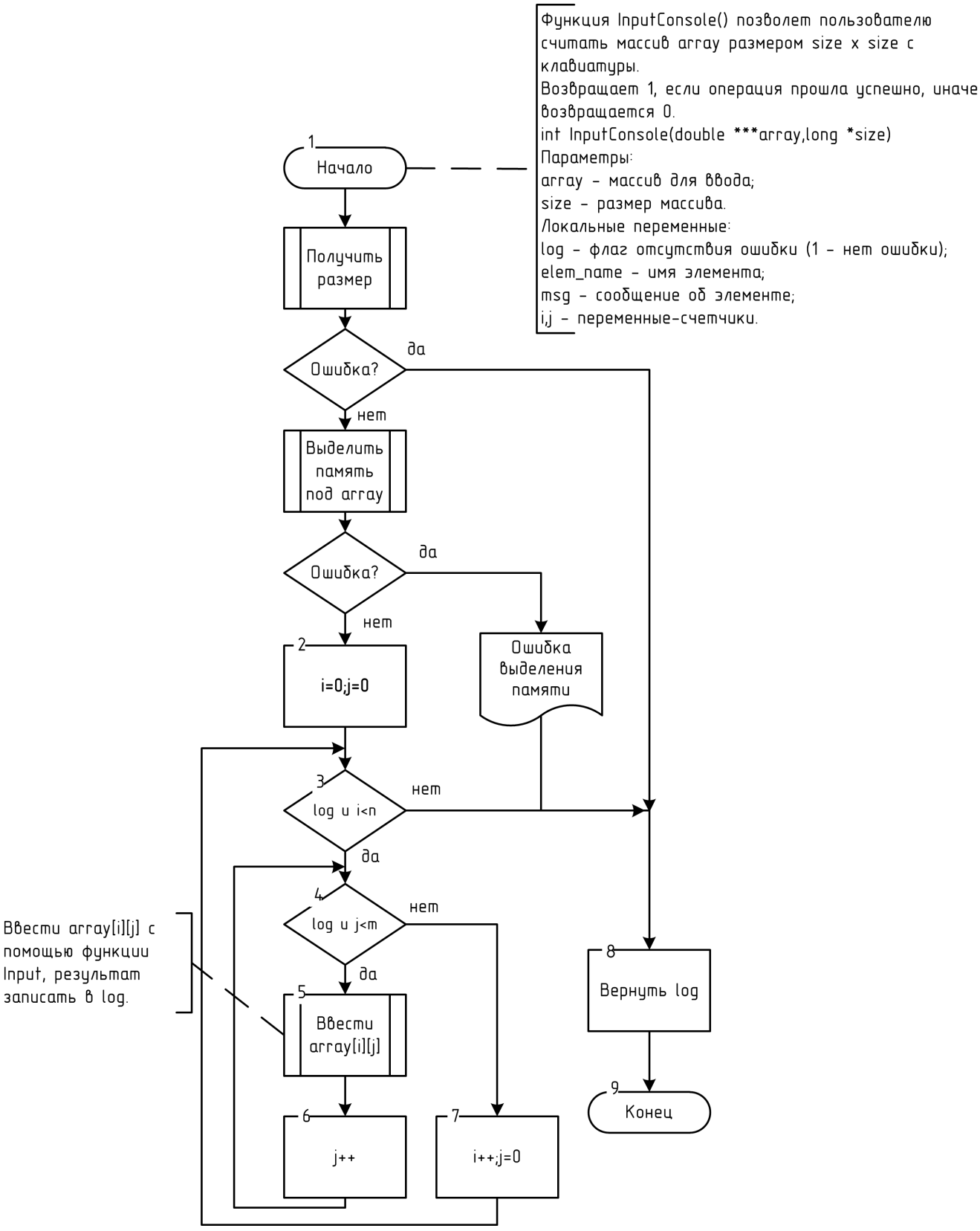


Рисунок 7 - Схема алгоритма заполнения матрицы с клавиатуры

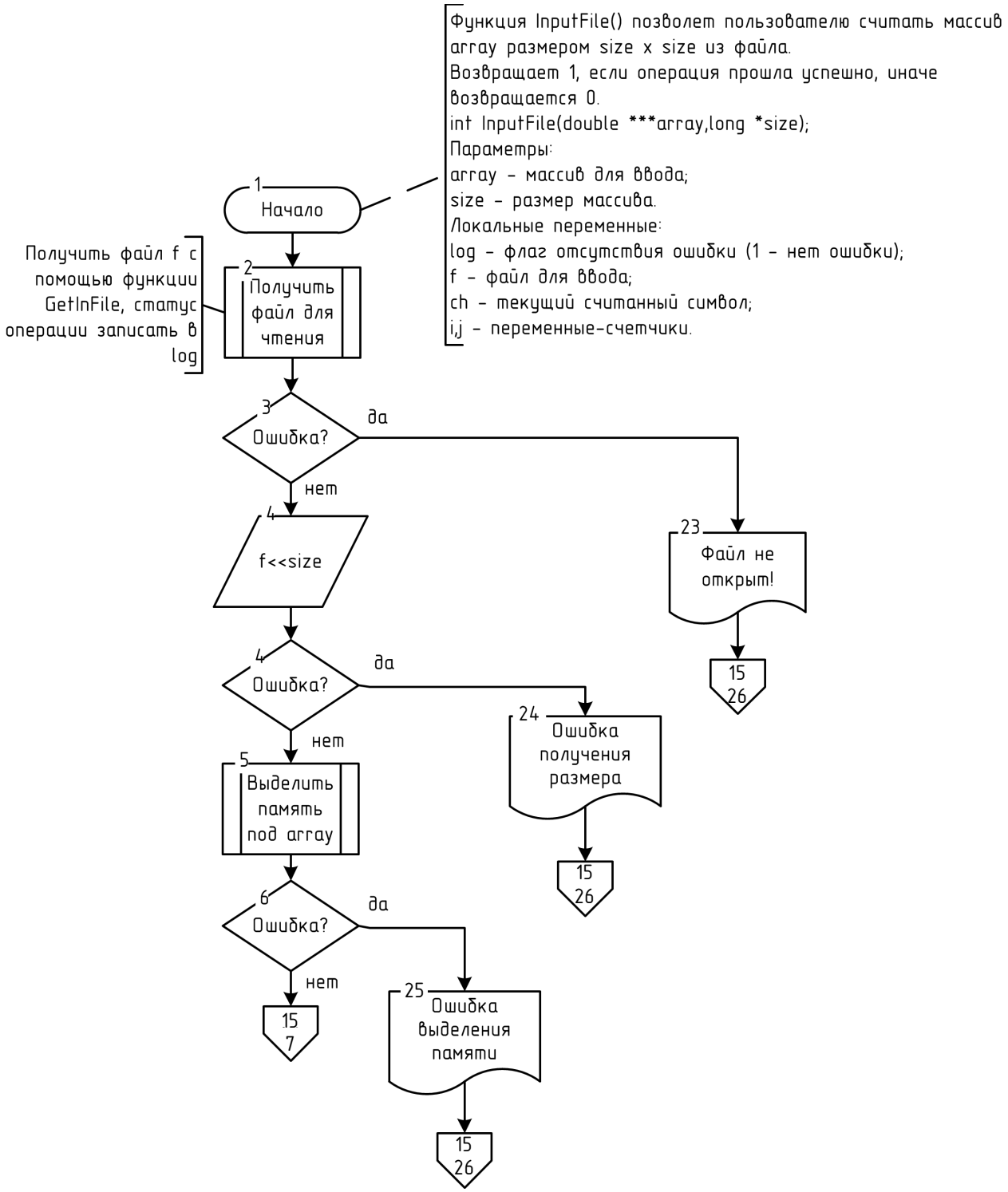


Рисунок 8 - Схема алгоритма заполнения матрицы из файла (начало)

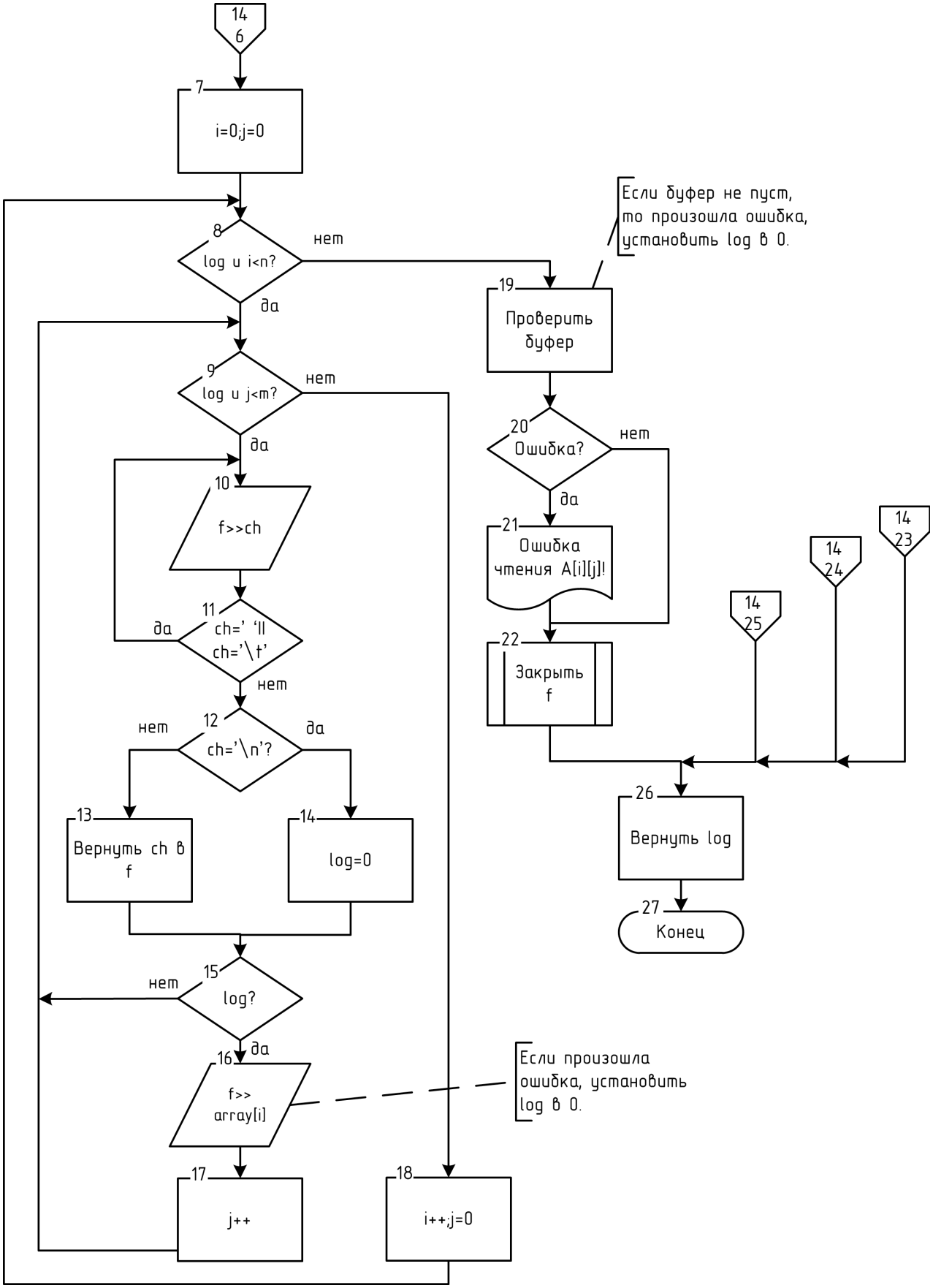


Рисунок 9 - Схема алгоритма заполнения матрицы из файла (продолжение)

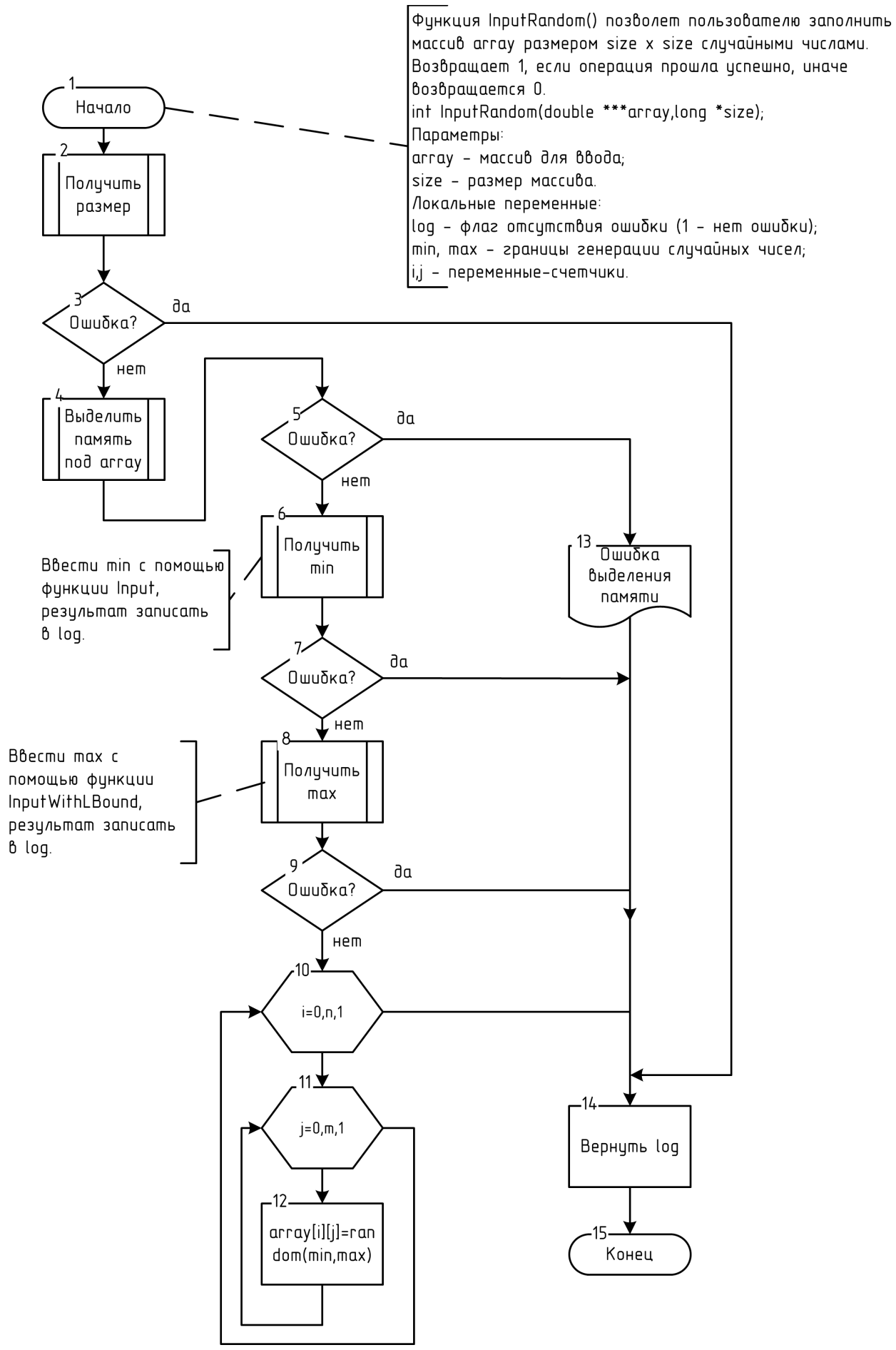


Рисунок 10 - Схема алгоритма заполнения матрицы случайными числами

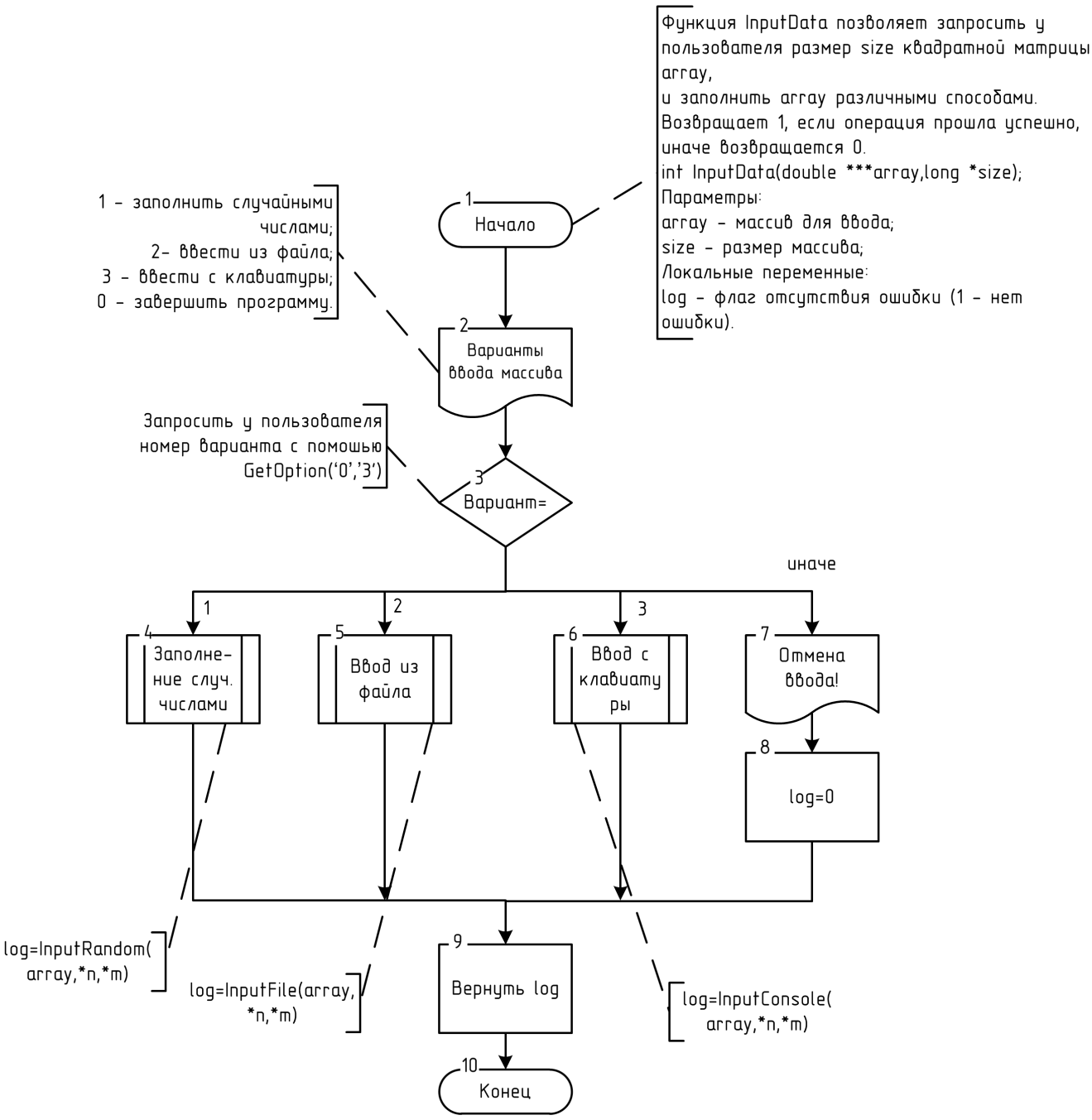


Рисунок 11 - Схема алгоритма заполнения матрицы различными способами

Для печати на экран или в файл по выбору пользователя также предусмотрен отдельный алгоритм, схема которого представлена на рисунке 12.

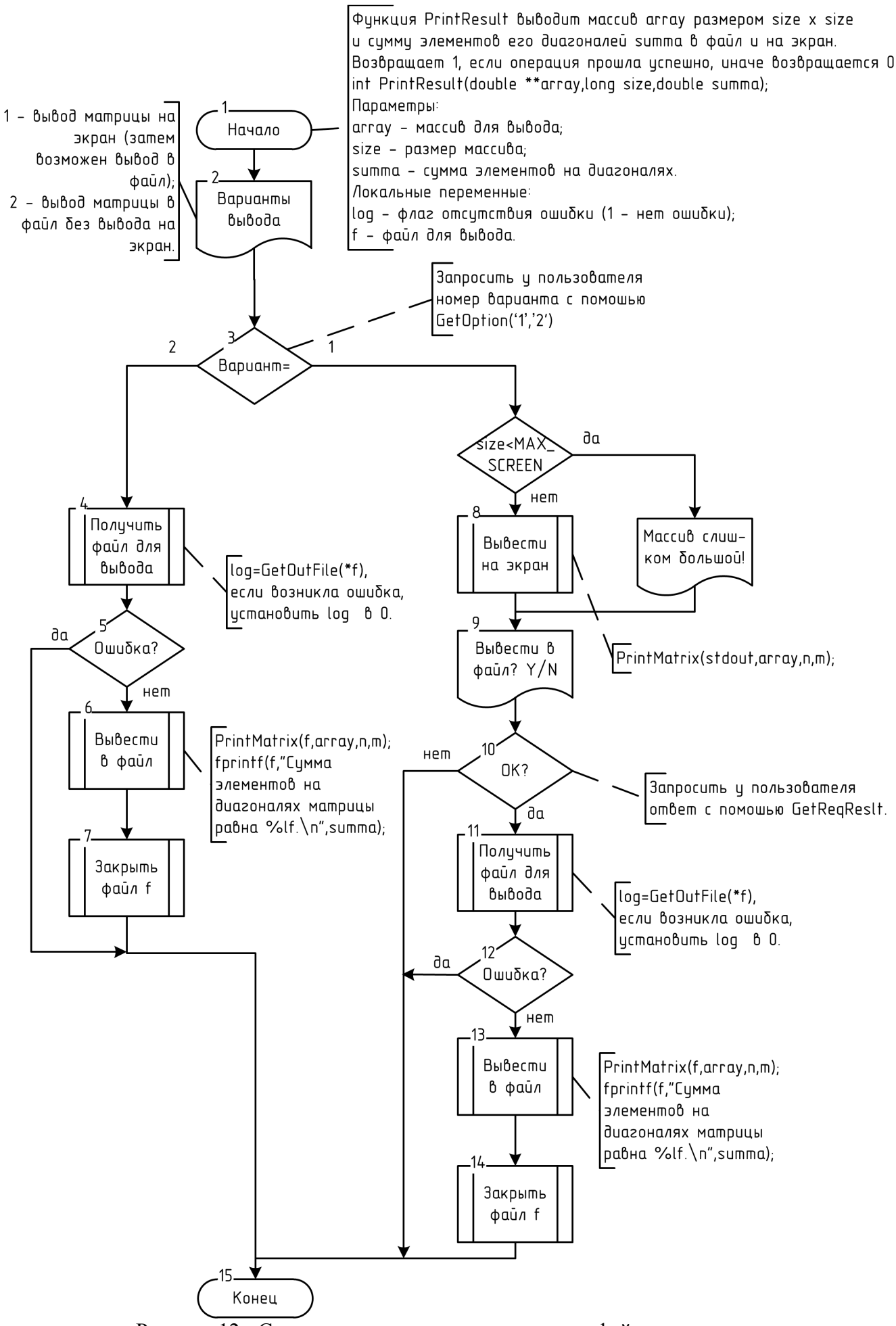


Рисунок 12 - Схема алгоритма вывода матрицы в файл или на экран

Проверка данных также производится отдельным алгоритмом, и его схема — на рисунке 13.

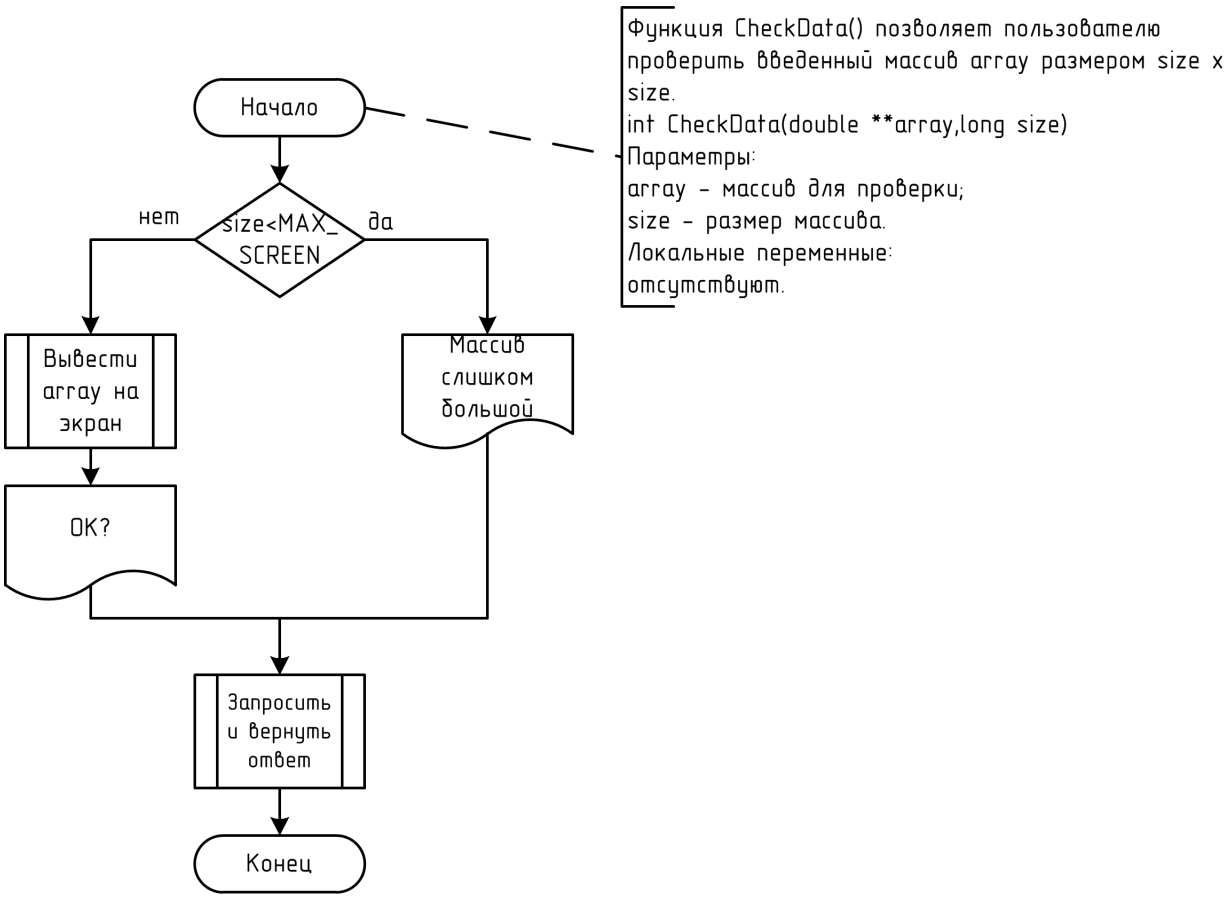


Рисунок 13 - Схема алгоритма проверки матрицы

На рисунке 14 представлена схема вывода матрицы в файл.

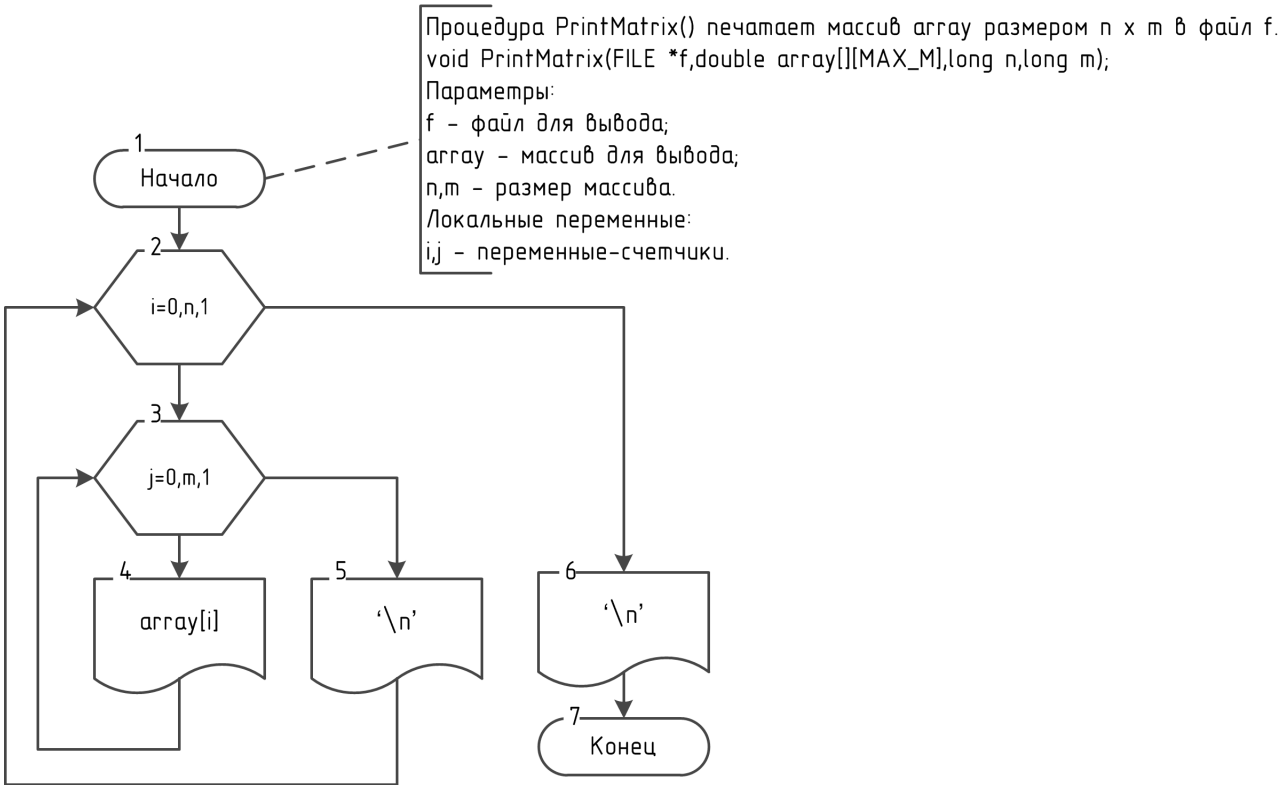


Рисунок 14 - Схема алгоритма вывода двумерного массива в файл

Схему алгоритма получения ответа от пользователя на поставленный вопрос, можно увидеть на рисунке 15.

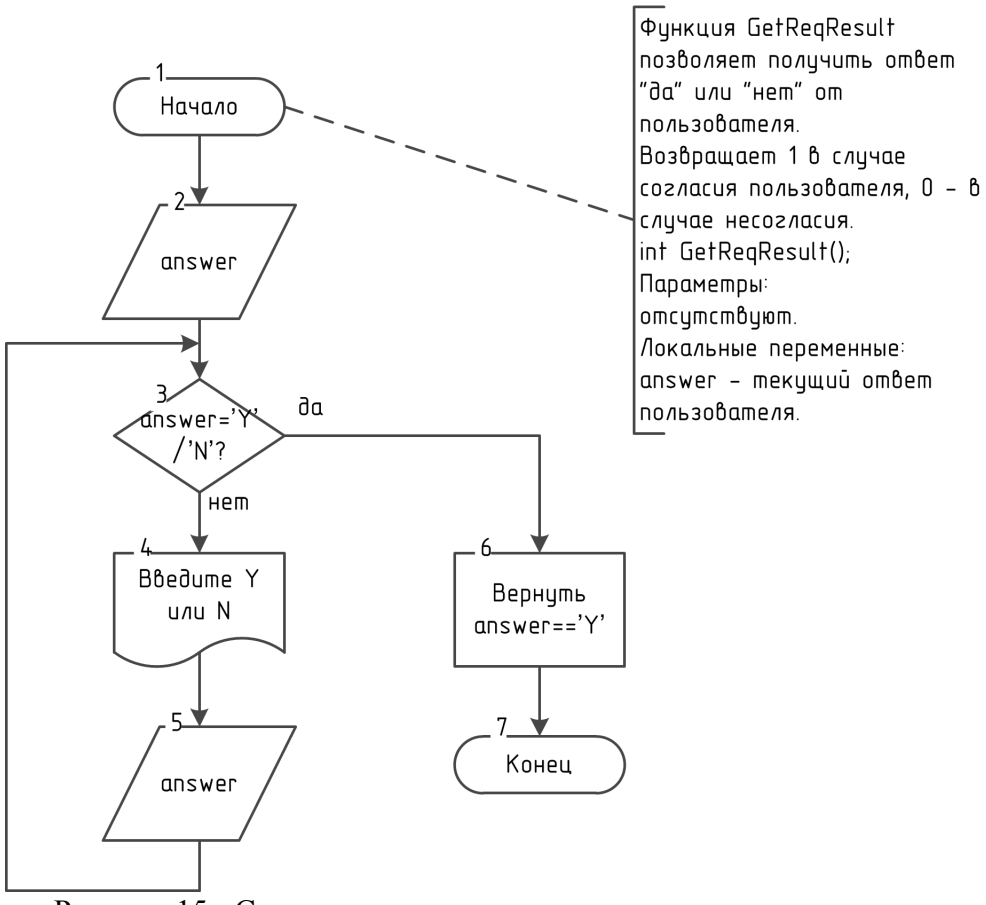


Рисунок 15 - Схема алгоритма получения ответа от пользователя

На рисунке 16 представлена схема алгоритма предоставления пользователю различных вариантов ответа.

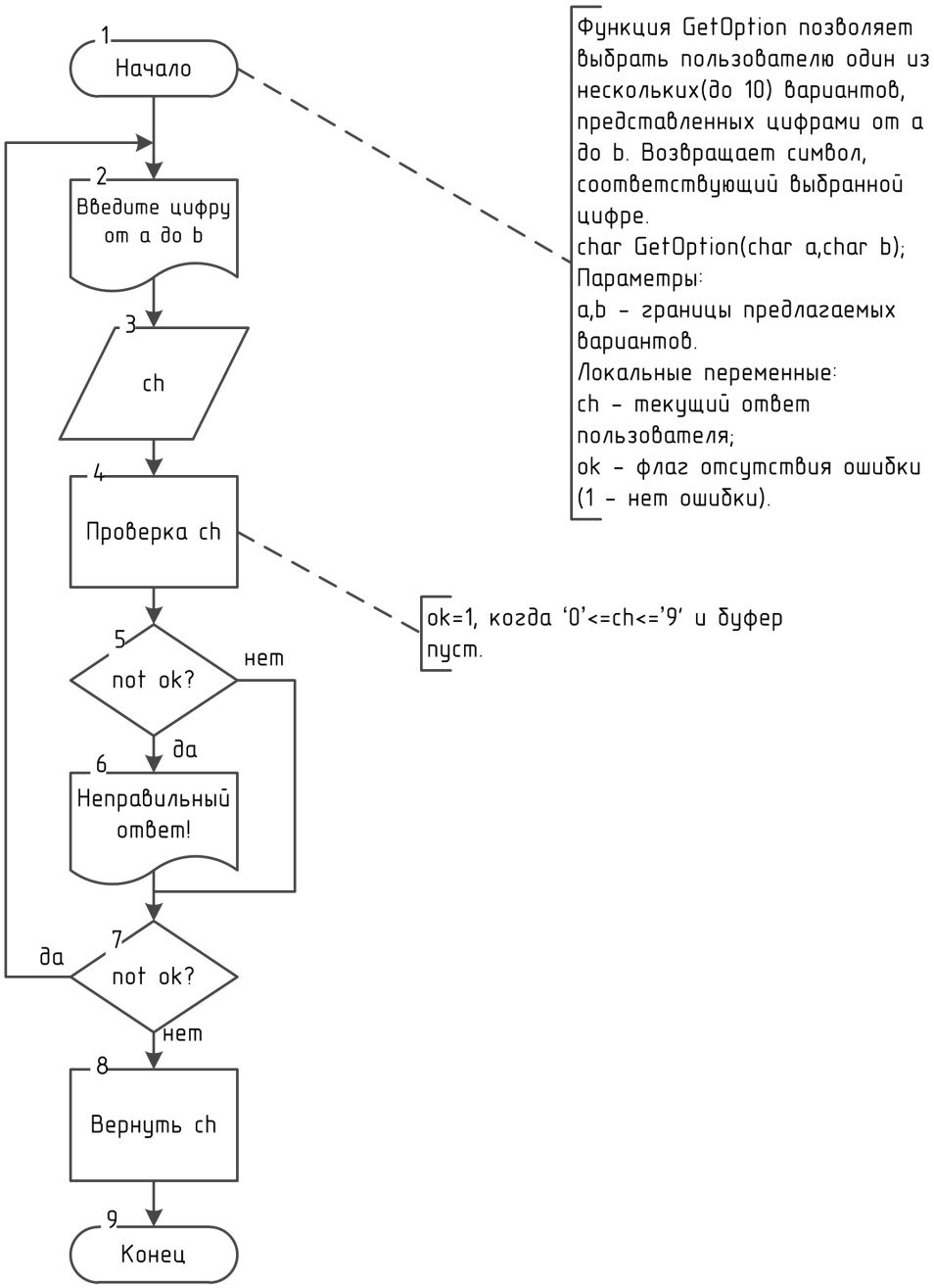


Рисунок 16 - Схема алгоритма предоставления пользователю различных вариантов ответа

2. Инструкция пользователю

Данная программа находит сумму диагональных элементов квадратной матрицы.

Для работы передайте программе размер массива — натуральное число из сообщенного программой диапазона. После этого можно заполнить матрицу случайными числами, из файла или вручную с клавиатуры. При вводе матрицы из файла следует учитывать, что количество строк файла должно соответствовать указанному, количество элементов в строке также должно быть равно указанному значению. Перед элементами матрицы — в первой строке файла — находится число, значение которого есть размер матрицы. Элементы разделяются пробелами или\и табуляциями. После ввода матрицы предлагается возможность её проверки.

После подсчета суммы диагональных элементов можно вывести в файл или на экран по вашему желанию. При выборе файла следует учитывать потерю содержащийся в нем информации — файл будет перезаписан.

3. Инструкция программисту

При создании программы расчета суммы диагональных элементов квадратной матрицы были предприняты следующие действия.

Объявлены константы MAX_N – максимальный размер массива, и MAX_SCREEN_ELEM — максимальное число элементов, умещающееся на экране.

Подключены заголовочные файлы <ctype.h> - для функции toupper() и <stdio.h> для функций ввода-вывода.

Объявлены следующие структуры данных, представленные в таблице 1:

Таблица 1 - Структуры данных, используемые в в основной части программы сортировки главной диагонали

имя	**тип**	**предназначение**
size	long	размер массива;
array	double**	обрабатываемый массив;
summa	double	сумма элементов на диагоналях;

Также программа была разбита на следующие подпрограммы:

1. Функция SumDiag рассчитывает сумму диагональных элементов квадратной матрицы array размером size.

Возвращает искомую сумму.

double SumDiag(double **array,long size);

Параметры функции представлены в таблице 2:

Таблица 2 - Параметры функции суммирования диагональных элементов

имя	**тип**	**предназначение**
array	double**	массив для суммирования диагональных элементов;
size	long	размер массива.

Локальные переменные функции представлены в таблице 3:

Таблица 3 - Локальные переменные функции суммирования диагональных элементов

имя	**тип**	**предназначение**
s	double	сумма диагональных элементов.
i	long	переменная – счётчик.

2. Функция GetInFile() получает файл f для чтения.

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

int GetInFile(FILE **f);

Параметры функции представлены в таблице 4:

Таблица 4 - Параметры функции получения файла для чтения

имя	тип	предназначение
f	FILE *	запрашиваемый файл.

Локальные переменные функции представлены в таблице 5:

Таблица 5 - Локальные переменные функции получения файла для чтения

имя	тип	предназначение
error	int	флаг ошибки (1 - ошибка);
req_rslt	int	флаг ответа пользователя (1 - согласие пользователя).

3. Функция GetOutFile() получает файл f для записи.

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

int GetOutFile(FILE **f);

Параметры функции представлены в таблице 6:

Таблица 6 - Параметры функции получения файла для записи

имя	тип	предназначение
f	FILE *	запрашиваемый файл.

Локальные переменные функции представлены в таблице 7:

Таблица 7- Локальные переменные функции получения файла для записи

имя	тип	предназначение
error	int	флаг ошибки (1 - ошибка);
req_rslt	int	флаг ответа пользователя (1 - согласие пользователя).

4. Функция PrintResult выводит массив array размером size x size и сумму элементов его диагоналей summa в файл и на экран.

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

int PrintResult(double **array,long size,double summa);

Параметры функции представлены в таблице 8:

Таблица 8 - Параметры функции вывода матрицы в файл или на экран

имя	тип	предназначение
array	double**	массив для вывода;
size	long	размер массива.

summa	double	сумма элементов на диагоналях;

Локальные переменные функции представлены в таблице 9:

Таблица 9 - Локальные переменные функции вывода матрицы в файл или на экран

имя	тип	предназначение
log	int	флаг отсутствия ошибки (1 - нет ошибки);
f	FILE *	файл для вывода;

5. Процедура PrintMatrix() печатает массив array размером size x size в файл f.

void PrintMatrix(FILE *f,double **array,long size);

Параметры функции представлены в таблице 10:

Таблица 10 - Параметры функции вывода матрицы в файл

имя	тип	предназначение
f	FILE *	файл для вывода;
array	double*	массив для вывода;
size	long	размер массива.

Локальные переменные функции представлены в таблице 11:

Таблица 11 - Локальные переменные функции вывода матрицы в файл

имя	тип	предназначение
i ,j	long	переменные-счетчики.

6. Функция Input() получает от пользователя некоторое вещественное число param.

При этом она в процессе запроса выводит поясняющее сообщение message, и приглашение ко вводу в виде имени параметра – name.

```
int Input(const char message[],
          const char name[],
          double *param
              );
```

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

Параметры функции представлены в таблице 12:

Таблица 12 - Параметры функции получения вещественного числа

имя	тип	предназначение
message	char*	сообщение о параметре;
name	char*	имя параметра;
param	double*	запрашиваемый параметр.

Локальные переменные функции представлены в таблице 13:

Таблица 13 - Локальные переменные функции получения вещественного числа

имя	тип	предназначение
ok	int	флаг отсутствия ошибки (1 - нет ошибки);
req_rslt	int	флаг ответа пользователя (1 - согласие пользователя).

7. Функция InputWithLBound() получает от пользователя некоторое вещественное число param,минимальное значение которого ограничено значением l_bound.

При этом она в процессе запроса выводит поясняющее сообщение message, и приглашение ко вводу в виде имени параметра - name.

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

function InputWithLBound(const message:string;const name:string; var param:real;const l_bound:real):boolean

Параметры представлены в таблице 14:

Таблица 14 - Параметры функции получения вещественного числа с левой границей

имя	тип	предназначение
message	char *	сообщение о параметре
name	char *	имя параметра
param	double *	запрашиваемый параметр
l_bound	double	нижняя граница параметра.

Локальные переменные представлены в таблице 3.15:

Таблица 15 - Локальные переменные функции получения вещественного числа с левой границей

имя	тип	предназначение
ok	int	флаг отсутствия ошибки (1 - нет ошибки);
req_rslt	int	флаг ответа пользователя (1 - согласие пользователя).

8. Функция InputIndex() получает от пользователя индекс массива - некоторое длинное целое число index из промежутка от 0 до верхней границы h_bound.

При этом она в процессе запроса выводит поясняющее сообщение message, и приглашение ко вводу в виде имени параметра – name.

int InputIndex(const char message[],const char name[],long *index,long h_bound);

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

Параметры функции представлены в таблице 16:

Таблица 16 - Параметры функции получения индекса элемента массива

имя	тип	предназначение
message	char*	сообщение о параметре;
name	char*	имя параметра;
index	long*	запрашиваемый индекс;
h_bound	long*	верхняя граница индекса.

Локальные переменные функции представлены в таблице 17:

Таблица 17 - Локальные переменные функции получения индекса элемента массива

имя	тип	предназначение
ok	int	флаг отсутствия ошибки (1 - нет ошибки);
req_rslt	int	флаг ответа пользователя (1 - согласие пользователя).

9. Функция InputConsole() позволет пользователю считать массив array размером size x size с клавиатуры.

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

int InputConsole(double ***array,long *size);

Параметры функции представлены в таблице 18:

Таблица 18 - Параметры функции заполнения матрицы с клавиатуры

имя	тип	предназначение
array	double***	массив для ввода;
size	long	размер массива.

Локальные переменные функции представлены в таблице 19:

Таблица 19 - Локальные переменные функции заполнения матрицы с клавиатуры

имя	тип	предназначение
log	int	флаг отсутствия ошибки (1 - нет ошибки);
elem_name	char[20]	имя элемента;
msg	char[150]	сообщение об элементе;
i,j	long	переменные-счетчики.

10. Функция InputFile() позволет пользователю считать массив array размером size x size из файла.

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

int InputFile(double ***array,long *size);

Параметры функции представлены в таблице 20:

Таблица 20 - Параметры функции заполнения матрицы из файла

имя	тип	предназначение
array	double***	массив для ввода;
size	long	размер массива.

Локальные переменные функции представлены в таблице 21:

Таблица 21 - Локальные переменные функции заполнения матрицы из файла

имя	тип	предназначение
log	int	флаг отсутствия ошибки (1 - нет ошибки);
f	FILE *	файл для ввода;
ch	char	текущий считанный символ;
i ,j	long	переменные-счетчики.

11. Функция InputRandom() позволет пользователю заполнить массив array случайными числами.

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

int InputRandom(double ***array,long *size);

Параметры функции представлены в таблице 22:

Таблица 22 - Параметры функции заполнения матрицы случайными числами

имя	тип	предназначение
array	double***	массив для ввода;
size	long	размер массива.

Локальные переменные функции представлены в таблице 23:

Таблица 23 - Локальные переменные функции заполнения матрицы случайными числами

имя	тип	предназначение
log	int	флаг отсутствия ошибки (1 - нет ошибки);
req_rslt	int	флаг ответа пользователя (1 - согласие пользователя);
min, max	double	границы генерации случайных чисел;
i ,j	long	переменные-счетчики.

12. Функция InputData позволяет запросить у пользователя размер size массива array, и заполнить array различными способами.

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

int InputData(double ***array,long *size);

Параметры функции представлены в таблице 24:

Таблица 24 - Параметры функции ввода данных программы

имя	тип	предназначение
array	double***	массив для ввода;
size	long*	размер массива.

Локальные переменные функции представлены в таблице 25:

Таблица 25 - Локальные переменные функции ввода данных программы

имя	тип	предназначение
log	int	флаг отсутствия ошибки (1 - нет ошибки);
msg	char[255]	сообщение об параметре.

13. Функция clearline используется для очистки строки файла.

Заголовок функции:

int clearline(FILE *f);

Функция считывает до конца файла или строки файла f символы в цикле while и возвращает их количество.

Параметры функции представлены в таблице 26, локальные переменные — в таблице 27.

Таблица 26 - Параметры функции получения сочетания

имя	тип	предназначение
f	FILE*	Файл, в котором очищается строка.

Таблица 27 - Локальные переменные функции получения сочетания

имя	тип	предназначение
count	long	Счетчик считанных символов.

14. Функция GetReqResult используется для получения ответов пользователя на запросы программы.

Заголовок функции:

int GetReqResult();

Функция запрашивает у пользователя символы 'y','Y','n' или 'N' до тех пор, пока он не введет какой-либо из них. Соответственно возвращается либо 1, если символ 'Y', либо 0, если 'N'.

Локальные переменные функции представлены в таблице 28.

Таблица 28 - Локальные переменные функции получения сочетания

имя	тип	предназначение
answer	char	Ответ пользователя.

15. Функция GetOption позволяет выбирать пользователю из нескольких (до 10) вариантов ответа.

Заголовок функции:

int GetOption(char a,char b);

Функция запрашивает у пользователя символы из промежутка от a до b до тех пор, пока он не введет какой-либо из них. Соответственно возвращается значение введенного символа.

Параметры функции представлены в таблице 29, локальные переменные — в таблице 30.

Таблица 3.29 - Параметры функции получения сочетания

имя	тип	предназначение
a,b	char	Различные варианты представлены цифрами от a до b.

Таблица 3.30 - Локальные переменные функции получения сочетания

имя	тип	предназначение
ch	char	Ответ пользователя.
ok	int	Флаг — равен 1, если ответ пользователя допустим, иначе равен 0.

16. Функция CheckData() позволяет пользователю проверить введенный массив array размером size x size.

int CheckData(double **array,long size);

Параметры представлены в таблице 31:

Таблица 31 - Параметры функции проверки данных

имя	тип	предназначение
array	double **	массив для проверки;
size	long	размер массива.

Локальные переменные:

отсутствуют.

17. Функция AllocArray() выделяет динамическую память под массив array размером size x size.

Возвращает 1, если операция прошла успешно, иначе возвращается 0.

int AllocArray(double ***array,long size);

Параметры представлены в таблице 32:

Таблица 32 - Параметры функции выделения памяти под массив

имя	**тип**	**предназначение**
array	double ***	массив для выделения памяти;
size	long	размер массива.

Локальные переменные представлены в таблице 33:

Таблица 33 - Локальные переменные функции выделения памяти под массив

имя	**тип**	**предназначение**
log	int	флаг отсутствия ошибки (1 – нет ошибки);
i	long	переменная–счетчик;

18. Функция-процедура FreeArray() освобождает выделенную ранее память под массив array размером size x size.

void FreeArray(double ***array,long size);

Параметры представлены в таблице 34:

Таблица 34 - Параметры функции освобождения памяти под массив

имя	**тип**	**предназначение**
array	double ***	массив для освобождения памяти;
size	long	размер массива.

Локальные переменные представлены в таблице 35:

Таблица 35 - Локальные переменные функции освобождения памяти под массив

имя	**тип**	**предназначение**
i	long	переменная–счетчик;

4. Текст программы

Ниже представлен текст программы, написанной на языке Borland C 3.1, которая подсчитывает сумму диагональных элементов квадратной матрицы.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include <string.h>
#define MAX_SCREEN_ELEM 15
#define MAX_SIZE 5L
/*
Функция SumDiag рассчитывает сумму диагональных элементов квадоатной матрицы array
размером size. Возвращает искомую сумму.
Параметры:
array - массив для суммирования диагональных элементов;
size - размер массива.
Локальный переменные:
s - сумма диагональных элементов.
i - переменная-счнтчик.
*/
double SumDiag(double **array,long size);

/*
Функция clearline очищает строку в файле f. Возвращает количество
непробельных символов в считанной строке.
Параметры:
f - Файл для очистки строки.
Локальные переменные:
count - количество непробельных символов;
ch - текущий считанный символ.
*/
int clearline(FILE *f);

/*
Функция GetOption позволяет выбрать пользователю один из нескольких(до 10)
вариантов,
представленных цифрами от a до b. Возвращает символ, соответствующий выбранной
цифре.
Параметры:
a,b - границы предлагаемых вариантов.
Локальные переменные:
ch - текущий ответ пользователя;
ok - флаг отсутствия ошибки (1 - нет ошибки).
```

```
*/
char GetOption(char a,char b);

/*
Функция GetReqResult позволяет получить ответ "да" или "нет" от пользователя.
Возвращает 1 в случае согласия пользователя, 0 - в случае несогласия.
Параметры:
отсутствуют.
Локальные переменные:
answer - текущий ответ пользователя.
*/
int GetReqResult();

/*
Функция fexists() проверяет существование файла с именем fname.
Возвращает 1, если такой файл существует, или 0, если нет.
Параметры:
fname - имя файла для проверки.
Локальные переменные:
f - временный файл для проверки.
*/
int fexists(char  *fname);

/*
Функция GetInFile() получает файл f для чтения.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
f - запрашиваемый файл.
Локальные переменные:
error - флаг ошибки (1 - ошибка);
req_rslt - флаг ответа пользователя (1 - согласие пользователя).
*/
int GetInFile(FILE **f);

/*
Функция GetOutFile() получает файл f для записи.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
f - запрашиваемый файл.
Локальные переменные:
error - флаг ошибки (1 - ошибка);
req_rslt - флаг ответа пользователя (1 - согласие пользователя).
*/
int GetOutFile(FILE * * f);
```

```
/*
Основная часть программы подсчета суммы элементов на диагоналях матрицы.
Переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
req_rslt - флаг ответа пользователя (1 - согласие пользователя);
array - массив для подсчета суммы элементов на диагоналях;
summa - сумма элементов на диагоналях;
size - размер массива.
*/
int main();

/*
Функция CheckData() позволяет пользователю проверить введенный массив array
размером size x size.
Параметры:
array - массив для проверки;
size - размер массива.
Локальные переменные:
отсутствуют.
*/
int CheckData(double **array,long size);

/*
Функция InputFile() позволет пользователю считать массив array размером size x
size из файла.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
array - массив для ввода;
size - размер массива.
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
f - файл для ввода;
ch - текущий считанный символ;
i,j - переменные-счетчики.
*/
int InputFile(double ***array,long *size);

/*
Функция InputConsole() позволет пользователю считать массив array размером size x
size с клавиатуры.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
array - массив для ввода;
```

```
size - размер массива.
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
elem_name - имя элемента;
msg - сообщение об элементе;
i,j - переменные-счетчики.
*/
int InputConsole(double ***array,long *size);

/*
Функция InputRandom() позволет пользователю заполнить массив array размером size x
size случайными числами.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
array - массив для ввода;
size - размер массива.
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
min, max - границы генерации случайных чисел;
i,j - переменные-счетчики.
*/
int InputRandom(double ***array,long *size);

/*
Функция InputData позволяет запросить у пользователя размер size квадратной
матрицы array,
и заполнить array различными способами.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
array - массив для ввода;
size - размер массива;
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки).
*/
int InputData(double ***array,long *size);

/*
Функция PrintResult выводит массив array размером size x size
и сумму элементов его диагоналей summa в файл и на экран.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
array - массив для вывода;
size - размер массива;
summa - сумма элементов на диагоналях.
```

```
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
f - файл для вывода.
*/
int PrintResult(double **array,long size,double summa);

/*
Функция AllocArray() выделяет динамическую память под массив array
размером size x size.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
array - массив для выделения памяти;
size - размер массива.
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
i - переменная-счетчик;
*/
int AllocArray(double ***array,long size);
/*
Функция-процедура FreeArray() освобождает выделенную ранее память под массив array
размером size x size.
Параметры:
array - массив для освобождения памяти;
size - размер массива.
Локальные переменные:
i - переменная-счетчик;
*/
void FreeArray(double ***array,long size);

/*
Процедура PrintMatrix() печатает массив array размером size x size в файл f.
Параметры:
f - файл для вывода;
array - массив для вывода;
size - размер массива.
Локальные переменные:
i,j - переменные-счетчики.
*/
void PrintMatrix(FILE *f,double **array,long size);

/*
Функция Input() получает от пользователя некоторое вещественное число param.
При этом она в процессе запроса выводит поясняющее сообщение message,
и приглашение ко вводу в виде имени параметра - name.
```

```
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
message - сообщение о параметре;
name - имя параметра;
param - запрашиваемый параметр.
Локальные переменные:
ok - флаг отсутствия ошибки (1 - нет ошибки);
req_rslt - флаг ответа пользователя (1 - согласие пользователя).
*/
int Input(const char message[], //paramName - имя запрашиваемого параметра;
              const char name[], //paramCond - дополнительная информация о
параметре;
              double *param
                    );

/*
Функция InputIndex() получает от пользователя индекс массива -
некоторое длинное целое число index из промежутка от 0 до верхней границы h_bound.
При этом она в процессе запроса выводит поясняющее сообщение message,
и приглашение ко вводу в виде имени параметра - name.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
message - сообщение о параметре;
name - имя параметра;
index - запрашиваемый индекс;
h_bound - верхняя граница индекса.
Локальные переменные:
ok - флаг отсутствия ошибки (1 - нет ошибки);
req_rslt - флаг ответа пользователя (1 - согласие пользователя).
*/
int InputIndex(const char message[],const char name[],long *index,long h_bound);

/*
Функция InputWithLBound() получает от пользователя некоторое вещественное число
param,
минимальное значение которого ограничено значением l_bound.
При этом она в процессе запроса выводит поясняющее сообщение message,
и приглашение ко вводу в виде имени параметра - name.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
message - сообщение о параметре;
name - имя параметра;
param - запрашиваемый параметр;
h_bound - верхняя граница параметра.
```

```
Локальные переменные:
ok - флаг отсутствия ошибки (1 - нет ошибки);
req_rslt - флаг ответа пользователя (1 - согласие пользователя).
*/
int InputWithLBound(const char message[],const char name[],double *param,double
l_bound);

double SumDiag(double **array,long size){
  long i; double s=0;
    for (i=0;i<size;i++){
        s+=array[i][i]+array[i][size-1-i];
    }
    if (size%2) s-=array[size/2][size/2];
    return s;
}

int clearline(FILE *f){
  int count=0;char ch;
      while (!feof(f)&&((ch=getc(f))!='\n'))
        if(ch!=' '&&ch!='\t')
            count++;
      return count;
}

char GetOption(char a,char b){
  char ch; int ok;
    do{
        printf("Введите цифру от %c до %c.\n",a,b);
            ch=getchar();
            ok=(a<=ch)&&(ch<=b)&&!clearline(stdin);
            if (!ok)
            printf("Неправильный ответ!\n");
    }while (!ok);
      return ch;
}

int GetReqResult(void);

int GetReqResult(){
  char answer;
    answer=getchar();
    clearline(stdin);
    while (toupper(answer)!='Y'
           &&toupper(answer)!='N'){
```

```
        printf("Неправильный ответ! Допустимо:\n"\
                                "Y - да; N - нет.\n");
        answer=getchar();
        clearline(stdin);
    }

    return toupper(answer)=='Y';
}
int fexists(char  *fname){
  FILE *f;
    f=fopen(fname,"r");
    if (f!=NULL)
        fclose(f);
    return f!=NULL;
}
int GetInFile(FILE **f){
  int error,req_rslt;
  char fileName[255]="\0";
    do{
        req_rslt=0;
        printf("Введите имя файла-источника.\n");
        //считать строку (gets() deprecated!)
        printf("%s: ","Файл");scanf("%255[^\n]",fileName);
        clearline(stdin);
        *f=fopen(fileName,"r");
        error=*f==NULL;
        if (error) {
            printf("Неправильное имя файла! \n");
            printf("Хотите повторить ввод? (Y/N)\n");
            req_rslt=GetReqResult();
        };
    } while (req_rslt&&error);
    return !error;
  };
////////////////////////////////////
int GetOutFile(FILE * * f){
  char fileName [255]="\0";
  int error=0,req_rslt;
    do{
        req_rslt=0;
        printf("Введите имя файла-результата.\n");
        //считать строку (gets() deprecated!)
        printf("%s: ","Файл");scanf("%255[^\n]",fileName);
        clearline(stdin);
```

```
        if (fexists(fileName)){
            error=1;
            printf("ВНИМАНИЕ! Указанное имя файла занято!\n");
            printf("ПЕРЕЗАПИСАТЬ ФАЙЛ? (Y/N)\n");
            error=!GetReqResult();
        }
        if (!error){
            *f=fopen(fileName,"w");
            error=*f==NULL;
        }
        if (error) {
            printf("Неправильное имя файла! \n");
            printf( "Хотите повторить ввод? (Y/N)\n");
            req_rslt=GetReqResult();
        };
    } while (req_rslt&&error);
    return !error;
  };
int main(){
  int log,req_rslt;
  double **array,summa; long size;
    printf("Программа находит сумму элементов диагоналей\n"\
                    "матрицы размера mxm.\n");

    do {
        log=InputData(&array,&size);
        if (log){
            log=CheckData(array,size);
            if (!log){
                FreeArray(&array,size);
                printf( "Введенные данные некорректны или ввод отменён!\n");
                printf( "Хотите повторить ввод? (Y/N)\n");
                req_rslt=GetReqResult();
        }

        }
    } while (!log&&req_rslt);
    if (log){
        summa=SumDiag(array,size);
        log=PrintResult(array,size,summa);
        FreeArray(&array,size);
    } ;
    if (log) printf("Работа успешно выполнена!\n");else printf("Работа программы
прервана!\n");
```

```
    printf("Нажмите <Enter>...\n");
    clearline(stdin);
    return 0;
}
int PrintResult(double **array,long size,double summa){
  int log=1; FILE *f;
    printf("Сумма элементов на диагоналях матрицы равна %lf.\n",summa);
    printf("1 - вывод матрицы на экран (затем возможен вывод матрицы и суммы в
файл);\n"\
                                          "2 - вывод матрицы и суммы в файл без
вывода на экран.\n");
    switch(GetOption('1','2')){
        case '1':
            if(size>MAX_SCREEN_ELEM){
                printf( "Массив %ldx%ld слишком большой для вывода на
экран!\n",size,size);
                log=0;
            } else
                PrintMatrix(stdout,array,size);
            printf("Хотите вывести матрицу в файл? Y/N\n");
            if (GetReqResult()){
                if ((log=GetOutFile(&f))){
                    PrintMatrix(f,array,size);fclose(f);
                    fprintf(f,"Сумма элементов на диагоналях матрицы равна
%lf.\n",summa);
                }
        }
        break;
        case '2':
            if ((log=GetOutFile(&f))){
                PrintMatrix(f,array,size);
                fprintf(f,"Сумма элементов на диагоналях матрицы равна
%lf.\n",summa);
            }
        break;
    }
    return log;
}
void PrintMatrix(FILE *f,double **array,long size){
  long i,j;
    for (i=0;(i<=size-1);i++){
        for (j=0;(j<=size-1);j++)
            fprintf(f,"%8.4lf ",array[i][j]);
        fprintf(f,"\n");
```

```
    }
}

int AllocArray(double ***array,long size){
  int log; long i;
    *array=NULL;
    *array=(double**)malloc(size*sizeof(double*));
    log=*array!=NULL;
    if(log) {
        for (i=0;i<=size-1;i++)
            (*array)[i]=NULL;
        for (i=0;(i<=size-1)&&log;i++){
            (*array)[i]=(double*)malloc(size*sizeof(double));
            log=**array!=NULL;
        }
    }
    if (!log)
        FreeArray(array,size);
    return log;
}

void FreeArray(double ***array,long size){
  long i;
    if (*array!=NULL){
        for (i=0;(i<=size-1)&&(*array)[i]!=NULL;i++){
            free((*array)[i]);
            (*array)[i]=NULL;
        };
        free(*array);
        *array=NULL;
    };
}
int InputWithLBound(const char message[],const char name[],double *param,double
l_bound){
  int log,req_rslt;
    do{
        req_rslt=0;
        printf("%s",message);
        printf("%s: ",name);
        log=scanf("%lf",param);
        log=!clearline(stdin)&&log;
        if (!log){  //введено не вещественное число?
            printf("Введено не число!\n"
                   "Повторить ввод? Y/N\n");
```

```
            req_rslt=GetReqResult();
        }
        if (log&&(*param<l_bound)){
            log=0;
            printf("Ошибка! %s меньше %lf!\n"
                   "Повторить ввод(Y/N)?\n",name,l_bound);
            req_rslt=GetReqResult();
        }
    } while(!log&&req_rslt);
    return log;
}

int InputIndex(const char message[],const char name[],long *index,long h_bound){
  int log,req_rslt;
    do{
        req_rslt=0;
        printf("%s",message);
        printf("%s: ",name);
        log=scanf("%ld",index);
        log=!clearline(stdin)&&log;
        if (!log){  //введено не вещественное число?
            printf("Введено не число!\n"
                   "Повторить ввод? Y/N\n");
            req_rslt=GetReqResult();
        }
        if (log&&(*index<1||*index>h_bound)){
            log=0;
            printf("Ошибка! %s не принадлежит [1..%ld]!\n"
                   "Повторить ввод(Y/N)?\n",name,h_bound);
            req_rslt=GetReqResult();
        }
    } while(!log&&req_rslt);
    return log;
}
int InputConsole(double ***array,long *size){
  int log=1;
  long i,j; char msg[255],name[50];
    sprintf(msg,"Введите  m - размер матрицы. 1<=m<=%ld.\n",MAX_SIZE);
    log=InputIndex(msg,"m",size,MAX_SIZE);
    if (log) {
        log=AllocArray(array,*size);
        if (log){
            for (i=0;(i<=*size-1)&&log;i++)
                for (j=0;(j<=*size-1)&&log;j++){
```

```
                    sprintf(msg,"Введите элемент A[%ld,%ld].\n",i,j);
                    sprintf(name,"A[%ld,%ld]: ",i,j);
                    log=Input(msg,name,(*array)[i]+j);
                }
        if (!log) FreeArray(array,*size);
        } else
             printf("Ошибка выделения памяти!\n");
    };
    return log;
}

int InputFile(double ***array,long *size){
  int log; FILE *f; long i,j; char ch;
    log=/*1;f=stdin;*/GetInFile(&f);
    if (log){
        //log=InputFileLong(f,size)&&(size>0);
        log=fscanf(f,"%ld",size)&&!clearline(f);
        if (!log){
            printf("Размер массива не указан в файле!\n");

        } else {
            log=AllocArray(array,*size);
            if (log){
                for (i=0;(i<=*size-1)&&log;i++){
                    for (j=0;(j<=*size-2)&&log;j++){
                        //log=InputFile(f,(*array)[i]+j);
                        while((ch=getc(f))==' '||ch=='\t');
                        if (ch!='\n')
                            ungetc(ch,f);
                        else
                            log=0;
                        if (log){
                            log=fscanf(f,"%lf",(*array)[i]+j);
                        }
                    }
                    if (log){
                        while((ch=getc(f))==' '||ch=='\t');
                        if (ch!='\n')
                            ungetc(ch,f);
                        else
                            log=0;
                        if (log)
                            log=fscanf(f,"%lf",(*array)[i]+j)&&!clearline(f);j++;
                    }
```

```
                };
                if (!log){
                    //i--;j--;
                    printf("Ошибка при чтении из файла элемента %ld,%ld.\n",i,j);

                }
                if (f!=stdin)
                    fclose(f);
                if (!log) FreeArray(array,*size);
            } else
                 printf("Ошибка выделения памяти!\n");
        }
    } else printf("Неправильное имя файла! \n");

    return log;
}
int InputRandom(double ***array,long *size){
  int log=1; double min,max; long i,j; char msg[150];
    sprintf(msg,"Введите  m - размер матрицы. 1<=m<=%ld.\n",MAX_SIZE);
    log=InputIndex(msg,"m",size,MAX_SIZE);
    if (log){
        log=AllocArray(array,*size);
    if (log)
        log=Input("Введите min - минимальное из случайных чисел.\n","min",&min);
    if (log)
        log=InputWithLBound("Введите max - максимальное из случайных
чисел.\n","max",&max,min);


        if (log){
            for (i=0;i<=*size-1;i++)
                for (j=0;j<=*size-1;j++)
                    (*array)[i][j]=(double)((double)rand()/RAND_MAX *(max-min) +
min);

        } else printf("Ошибка выделения памяти!\n");
    }
    return log;
}
int InputData(double ***array,long *size){
  int log=1;
    printf("Заполните матрицу:\n"\
                                        "1 - заполнить случайными числами;\n"\
                                        "2 - ввести из файла;\n"\
```

```
                                        "3 - ввести с клавиатуры;\n"\
                                        "0 - завершить программу.\n");
    switch(GetOption('0','3')){
        case '1':
            log=InputRandom(array,size);
        break;
        case '2':
            log=InputFile(array,size);
        break;
        case '3':
            log=InputConsole(array,size);
        break;
        default:
            printf("Ввод отменён!\n");
            log=0;
    }
    return log;
}
int Input(const char message[], //paramName - имя запрашиваемого параметра;
              const char name[], //paramCond - дополнительная информация о
параметре;
              double *param
                    ){
  int log,req_rslt;
    do {
        printf("%s",message);
        printf("%s: ",name);
        log=scanf("%lf",param);
        log=!clearline(stdin)&&log;
        if (!log){  //введено не вещественное число?
            printf( "Введено неправильное значение!\n");
            printf("Хотите повторить ввод? (Y/N)\n");
            req_rslt=GetReqResult();
        }
    } while (!log&&req_rslt); //пока пользователь не отказался или число
некорректное
    return log;

}


int CheckData(double **array,long size){
    printf("Будет выполнен расчет суммы элементов диагоналях матрицы %ldx%ld
:\n",size,size);
```

```
    if (size<=MAX_SCREEN_ELEM){
        PrintMatrix(stdout,array,size);
        printf("Продолжить?(Y/N)\n");
    } else {
        printf("Массив %ldx%ld слишком большой для вывода на экран!\n",size,size);
        printf("Вы уверены в правильности введенных данных?(Y/N)\n");
    }

    return GetReqResult();
}
```

5. Тестовый пример

На рисунке 18 представлен пример работы программы для массива заполняемого из файла input.txt. Содержимое файла представлено на рисунке 17.

4

1 2 3 4

3 4 2 1

2 1 3 4

3 4 1 2

Рисунок 17 — Содержимое входного файла

```
Программа находит сумму элементов диагоналей
матрицы размера mxm.
Заполните матрицу:
1 - заполнить случайными числами;
2 - ввести из файла;
3 - ввести с клавиатуры;
0 - завершить программу.
Введите цифру от 0 до 3.
2
Введите имя файла-источника.
Файл: input.txt
Будет выполнен расчет суммы элементов диагоналях матрицы 4x4 :
  1.0000   2.0000   3.0000   4.0000
  3.0000   4.0000   2.0000   1.0000
  2.0000   1.0000   3.0000   4.0000
  3.0000   4.0000   1.0000   2.0000
Продолжить?(Y/N)
y
Сумма элементов на диагоналях матрицы равна 20.000000.
1 - вывод матрицы на экран (затем возможен вывод матрицы и суммы в файл);
2 - вывод матрицы и суммы в файл без вывода на экран.
Введите цифру от 1 до 2.
1
  1.0000   2.0000   3.0000   4.0000
  3.0000   4.0000   2.0000   1.0000
  2.0000   1.0000   3.0000   4.0000
  3.0000   4.0000   1.0000   2.0000
Хотите вывести матрицу в файл? Y/N
n
Работа успешно выполнена!
Нажмите <Enter>...
```

Рисунок 18 — Пример работы программы при считывании данных из файла

На рисунке 19 представлен пример работы программы для массива размером 4, заполняемого случайными числами (конкретные значения можно увидеть на рисунках 19 и 20). Результат после работы выводится в файл output.txt, его содержимое представлено на рисунке 20.

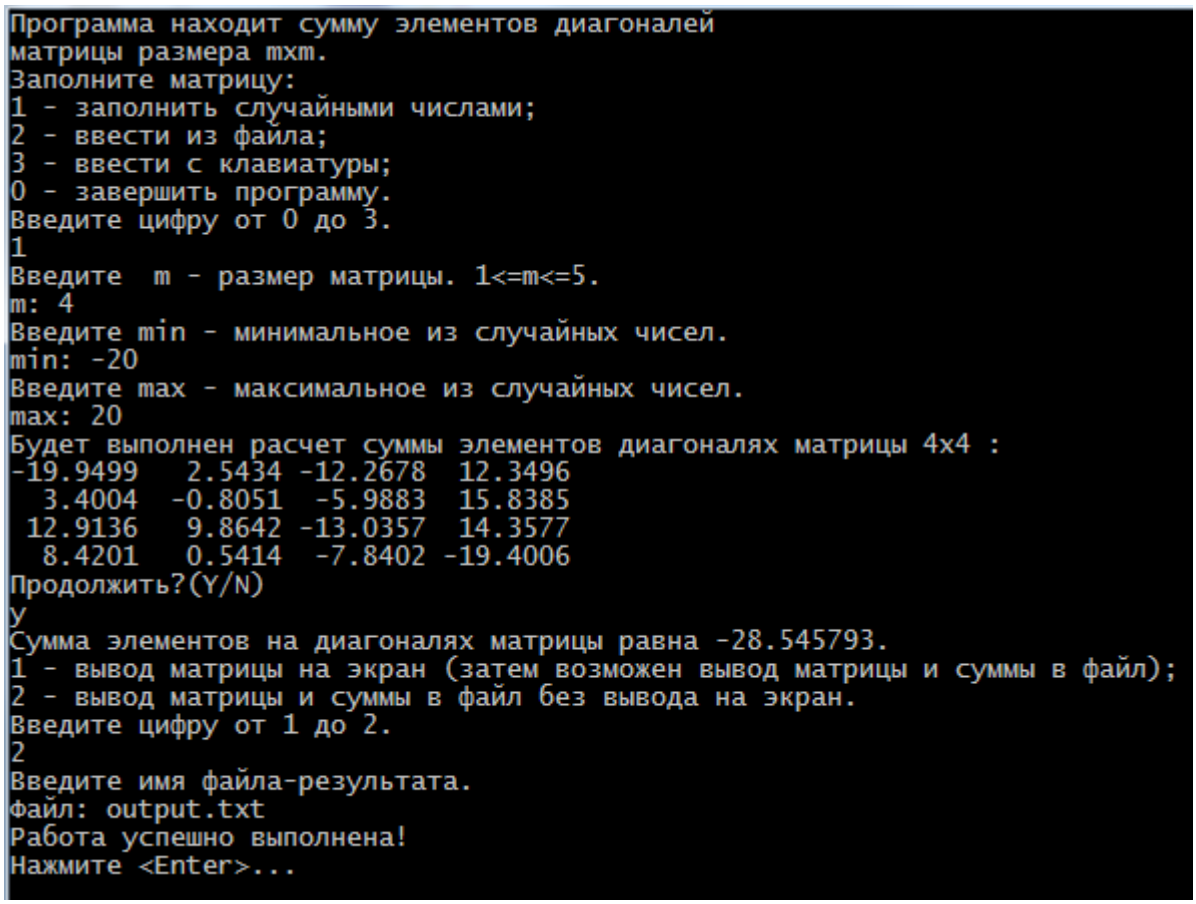


Рисунок 19 — Пример работы программы при заполнении случайными числами и выводе данных в файла

-19.9499 2.5434 -12.2678 12.3496

3.4004 -0.8051 -5.9883 15.8385

12.9136 9.8642 -13.0357 14.3577

8.4201 0.5414 -7.8402 -19.4006

Сумма элементов на диагоналях матрицы равна -28.545793.

Рисунок 20 — Содержимое выходного файла

Вывод

В ходе выполнения данной лабораторной работы я научился использовать динамическую память и массивы динамической памяти при написании программ на языке Си. Динамическая память позволяет быстро обрабатывать большие объёмы данных, что позволяет с успехом решать научные, инженерные, экономические и многие другие задачи в тех областях, гле требуется быстрая и качественная обработка больших объёмов данных, однако требует от программиста повышенного внимания и аккуратности. К сожалению, в 16-разрядной реализации Borland C 3.1 размер выделяемого сегмента динамической памяти ограничен 64 килобайтами, что создает довольно неприятные ограничения на размер входных данных.