

Министерство образования и науки РФ
Государственное образовательное учреждение
высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

ПРОГРАММИРОВАНИЕ ТИПОВЫХ АЛГОРИТМИЧЕСКИХ СТРУКТУР

Курсовая работа по курсу
«Программирование на языках высокого уровня»

Вариант № 4

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

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

Тула 2011

Аннотация

Данная работа является завершающей при изучении курса "Программирование на языках высокого уровня". В ней демонстрируются навыки, приобретенные за все время изучения данной дисциплины.

Работа содержит в себе 4 больших раздела, посвященные решению задач различного класса из различных областей на языках высокого уровня, которые причисляются к языкам т. н. структурно- и процедурно-ориентированного программирования — Borland Pascal и С.

Структура работы состоит из оглавления, содержания, разделов решения задач, выводов по работе и списка использованной литературы.

В основной части освещается решение следующих четырех задач:

1. нахождение интеграла на интервале $[a,b]$ для функции, заданной графически, по итерационной формуле левых прямоугольников;
2. вычисление таблицы значений функции заданной в виде разложения в ряд;
3. сортировка главной диагонали матрицы;
4. составление алфавитного частотного словаря.

К каждой задаче приводится её описание, разъясняется метод решения, проводится разработка используемых структур данных и алгоритмов, приводится текст программы, инструкции пользователю и программисту. В конце каждого из 4-х разделов приводятся результаты тестового запуска разработанной программы с обсуждением ожидаемых и полученных данных, аналитическим решением и оценки корректности программы. К некоторым задачам, имеющим математический характер, приводится также и математическая формулировка и описание.

Итого, работа состоит из 161 страницы, на которых размещено 116 таблиц, 57 рисунков, и в общей сумме больше 20000 слов.

Содержание

Содержание.....	2
Введение.....	4
1. Вычисление определенного интеграла для функции, заданной графически.....	10
1.1. Постановка задачи.....	10
1.2. Математическая формулировка задачи.....	10
1.3. Метод левых прямоугольников.....	11
1.4. Разработка структур данных.....	15
1.5. Разработка структуры алгоритма решения задачи	15
1.8. Текст программы.....	26
1.6. Инструкция пользователю.....	33
1.7. Инструкция программисту.....	34
1.9. Тестовый пример.....	41
2. Вычисление таблицы значений функции, заданной в виде разложения в ряд..	43
2.1. Постановка задачи.....	43
2.2 Математическая формулировка задачи.....	43
2.3. Численный метод решения задачи.....	43
2.4. Разработка структур данных.....	44
2.5. Разработка структуры алгоритма решения задачи.....	45
2.6. Текст программы.....	55
2.7. Инструкция программисту.....	64
2.8. Инструкция пользователю.....	72
2.9. Тестовый пример.....	74
3. Вычисление средних арифметических значений положительных элементов каждой строки матрицы.....	78
3.1. Постановка задачи.....	78
3.2. Математическая формулировка задачи.....	78

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.	Белым А.А.			
Проф.	Сулимова В.В.			
Н. контр.				
Утв.				

Вариант №4

Пояснительная записка к
контрольно-курсовой работе по курсу
«Программирование на ЯВУ» по теме
«Программирование типовых
алгоритмических структур»

Лит. Лист Листов
ТулГУ гр. 220601

3.3. Метод Хоара: быстрая сортировка.....	78
3.4. Разработка структур данных.....	82
3.5. Разработка структуры алгоритма решения задачи.....	82
3.6. Текст программы.....	98
3.7. Инструкция программисту.....	110
3.8. Инструкция пользователю.....	121
3.9. Тестовый пример.....	122
4. Задача составления алфавитного частотного словаря.....	125
4.1. Постановка задачи.....	125
4.2. Математическая формулировка задачи.....	125
4.3. Метод поиска по бинарному(двоичному) дереву.....	125
4.4. Разработка структур данных.....	133
4.5. Разработка структуры алгоритма решения задачи	134
4.6. Текст программы.....	142
4.7. Инструкция программисту.....	150
4.8. Инструкция пользователю.....	157
4.9. Тестовый пример.....	158
Заключение.....	160
Список используемых источников.....	161

Изм.	Лист	№ докум.	Подп.	Дата

Введение

Войти в XXI век образованным человеком можно, только хорошо владея информационными технологиями. Ведь деятельность людей все в большей степени зависит от их информированности, способности эффективно использовать информацию. Для свободной ориентации в информационных потоках современный специалист любого профиля должен уметь получать, обрабатывать и использовать информацию с помощью компьютеров, телекоммуникаций и других средств связи. Конечно, для этих целей уже создано огромное множество замечательных программных сред и систем, но по полной использовать громадные ресурсы компьютерной техники может лишь тот, кто сам умеет создавать собственные программы.

В данной работе программирование рассматривается с точки зрения методологий структурного и процедурного программирования. В работе на структурно- и процедурно ориентированных языках Паскаль и Си решаются 4 различные задачи.

Структурное программирование — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Предложена в 70-х годах XX века Э. Дейкстрой, разработана и дополнена Н. Виртом.

В соответствии с данной методологией

Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:

- последовательное выполнение — однократное выполнение операций в том порядке, в котором они записаны в тексте программы;
- ветвление — однократное выполнение одной из двух или более операций, в зависимости от выполнения некоторого заданного условия;
- цикл — многократное выполнение одной и той же операции до тех пор, пока выполняется некоторое заданное условие (условие продолжения цикла).

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.

Повторяющиеся фрагменты программы (либо не повторяющиеся, но представляющие собой логически целостные вычислительные блоки) могут оформляться в виде т. н. подпрограмм (процедур или функций). В этом случае в тексте основной программы, вместо помещённого в подпрограмму фрагмента, вставляется инструкция вызова подпрограммы. При выполнении такой инструкции выполняется вызванная подпрограмма, после чего исполнение программы продолжается с инструкции, следующей за командой вызова подпрограммы.

Разработка программы ведётся пошагово, методом «сверху вниз».

Сначала пишется текст основной программы, в котором, вместо каждого связного логического фрагмента текста, вставляется вызов подпрограммы, которая будет выполнять этот фрагмент. Вместо настоящих, работающих подпрограмм, в программу вставляются «заглушки», которые ничего не делают. Полученная программа проверяется и отлаживается. После того, как программист убедится, что подпрограммы вызываются в правильной последовательности (то есть общая структура программы верна), подпрограммы-заглушки последовательно заменяются на реально работающие, причём разработка каждой подпрограммы ведётся тем же методом, что и основной программы. Разработка заканчивается тогда, когда не останется ни одной «затычки», которая не была бы удалена. Такая последовательность гарантирует, что на каждом этапе разработки программист одновременно имеет дело с обозримым и понятным ему множеством фрагментов, и может быть уверен, что общая структура всех более высоких уровней программы верна. При сопровождении и внесении изменений в программу выясняется, в какие именно процедуры нужно внести изменения, и они вносятся, не затрагивая части программы, непосредственно не связанные с ними. Это позволяет гарантировать, что при внесении изменений и исправлении ошибок не выйдет из строя какая-то

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

часть программы, находящаяся в данный момент вне зоны внимания программиста.

Теорема о структурном программировании:

Любую схему алгоритма можно представить в виде композиции вложенных блоков begin и end, условных операторов if, then, else, циклов с предусловием (while) и может быть дополнительных логических переменных (флагов). Теорема говорит нам о том, как можно избежать использование оператора перехода goto.

Процедурное (императивное) программирование является отражением архитектуры традиционных ЭВМ, которая была предложена фон Нейманом в 1940-х годах. Теоретической моделью процедурного программирования служит алгоритмическая система под названием Машина Тьюринга.

Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения программиста имеются программа и память, причем первая последовательно обновляет содержимое последней.

Процедурный язык программирования предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Особенность таких языков программирования состоит в том, что задачи разбиваются на шаги и решаются шаг за шагом. Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов.

Как уже говорилось, процедурные языки, выбранные для написания задач данной работы — Turbo Pascal и Turbo C.

Как известно, Turbo Pascal является компилируемым языком, что обеспечивает высокую скорость выполнения программы. Но это не единственное его достоинство. Несмотря на появление новых технологий Pascal, во многом

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

задуманный как язык для обучения, и на сегодняшний день остается одним из самых удобных средств для изучения программирования. Это определяет его популярность среди широкой аудитории начинающих программистов: школьников и студентов. В синтаксисе языка, несмотря на его лаконичность и простоту, существуют самые различные структуры данных, что позволяет применять его для решения самых разнообразных задач. Имеется также доступ и к низкоуровневым компонентам оборудования. Также следует отметить большую заслугу фирмы Borland, которая помимо огромного вклада в развитие данного языка, разработала IDE Borland Pascal, содержащую удобный текстовый редактор, функциональный отладчик и превосходную справочную систему.

Си — это универсальный язык программирования с компактным способом записи выражений, современными механизмами управления структурами данных и богатым набором операторов. Си не является ни языком "очень высокого уровня", ни "большим" языком, не рассчитан он и на какую-то конкретную область применения. Однако благодаря широким возможностям и универсальности для решения многих задач он удобнее и эффективнее, чем предположительно более мощные языки.

Первоначально Си был создан Деннисом Ритчи как инструмент написания операционной системы UNIX для машины PDP-11 и реализован в рамках этой операционной системы. И операционная система, и Си- компилятор, и, по существу, все прикладные программы системы UNIX написаны на Си. Фирменные Си-компиляторы существуют и на нескольких машинах других типов, среди которых IBM/370, Honeywell 6000 и Interdata 8/32. Си не привязан к конкретной аппаратуре или системе, однако на нем легко писать программы, которые без каких-либо изменений переносятся на другие машины, где осуществляется его поддержка.

Опыт показал, что Си - удобный, выразительный и гибкий язык, пригодный для программирования широкого класса задач. Его легко выучить, и чем больше

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

работаешь с Си, тем он становится удобнее. Мы надеемся, что эта книга поможет вам хорошо его освоить

Как уже было сказано, в ходе данной работы на языке Паскаль были разработаны 4 программы, решающие математические задачи путем использования численных методов:

- 1) нахождение интеграла на интервале $[a,b]$ для функции, заданной графически, по итерационной формуле левых прямоугольников;
- 2) вычисление таблицы значений функции заданной в виде разложения в ряд;
- 3) сортировка главной диагонали матрицы;
- 4) составление алфавитного частотного словаря.

Для каждой задачи приводится её условие и математическая формулировка. Далее описывается и поясняется используемый при решении задачи численный метод. В следующем разделе приводится описание разработки и обоснование переменных и структур данных, которые вводятся исходя из численного метода и служебных нужд (организация ввода-вывода, проверка корректности значений и т.д.). Далее приводится блок-схема алгоритма решения указанной задачи. После следуют инструкции по применению для программиста и пользователя. Завершает описание программы тестовый пример, в котором демонстрируются результаты работы программы для некоторых тестовых данных и оценка правильности данных результатов.

Хотелось бы кратко сказать о самих методах решения каждой задачи.

Для решения первой задачи используется метод разбиения на прямоугольники по итерационной формуле левых прямоугольников. Вычисления надо закончить при выполнении условия $|I_n - I_{2n}| < \varepsilon$, где $\varepsilon > 0$ – достаточно малое значение, задаваемое пользователем (точность вычислений).

Во второй задаче применен метод последовательного вычисления значений членов функции, разложенной в ряд, при помощи рекуррентного соотношения. Решение задачи заключается в нахождении этого соотношения и применения его при вычислениях.

Изм.	Лист	№ докум.	Подп.	Дата

В третьей задаче необходимо отсортировать главную диагональ матрицы, причем матрица заполняется различными способами — с клавиатуры, случайными числами и из файла. После сортировки матрица может выводится как в файл, так и на экран по желанию пользователя. Сортировка ведется с помощью процедуры быстрой сортировки К. Хоара, которая использует организованный вручную стек в динамической памяти.

В четвертой задаче из файла с литературным произведением составляется алфавитный частотный словарь. Эта работа ведется с помощью двоичного дерева, находящегося в типизированном файле.

Всю работу завершает раздел «Заключение» и приводится список использованной литературы.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

1. Вычисление определенного интеграла для функции, заданной графически

1.1. Постановка задачи

Составить программу на языке Turbo Pascal 7.0 вычисления значения определенного интеграла с точностью ϵ на интервале $[a, b]$ для функции, заданной на рисунке 1.1.

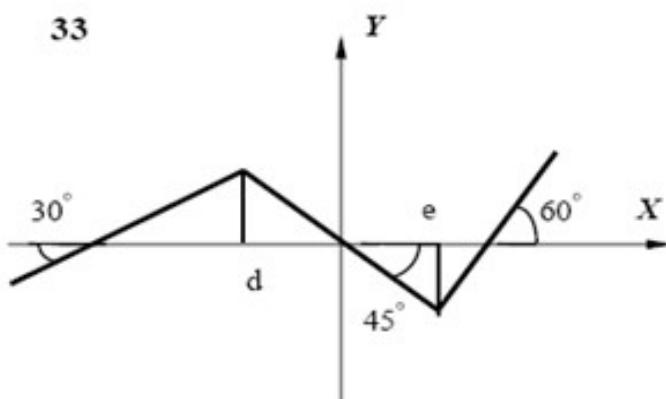


Рисунок 1.1 – График анализируемой функции

Численные значения всех величин, участвующих в вычислениях, считать параметрами программы, и определить их путём ввода.

1.2. Математическая формулировка задачи

Интеграл на интервале $[a, b]$ для функции $f(x)$, согласно определению, есть $F(b) - F(a)$, где $F(x)$ – первообразная функции $f(x)$. С другой стороны интеграл – это площадь криволинейной трапеции, ограниченной функцией, осью ОХ и прямыми $x = a$ и $x = b$. Поэтому для произвольной заданной графически функции значение интеграла можно найти как площадь данной трапеции.

Получим также аналитическое задание функции.

Согласно графику функции, она может задаваться кусочно с помощью трёх линейных функций, соседние из которых имеют точки пересечения. Можно заметить, что центральная линия задается уравнением $y = -x$; соответственно, значения функции в точках пересечения линий будут равны $-d$ и $-c$.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Наклон самой левой линии к оси ОХ равен 60° , значит, прямая параллельная ей и проходящая через начало координат, задается уравнением $y=\frac{\sqrt{3}}{3}x$. Сама же линия сдвинута от нее вверх и влево на $-d$, значит, она задается уравнением $y=\frac{\sqrt{3}}{3}(x+(-d))+(-d)=\frac{\sqrt{3}}{3}(x-d)-d$. Проводя аналогичные рассуждения для правой части функции, получаем, что она является сдвинутой вправо-вниз функцией $y=\sqrt{3}$, и соответственно, задается уравнением $y=\sqrt{3}(x-c)-c$.

Итак, запишем уравнение кусочной функции:

$$y = \begin{cases} \frac{\sqrt{3}}{3}(x-d)-d, & x \leq d, d < 0; \\ -x, & d < x \leq c, c > 0 \\ \sqrt{3}(x-c)-c, & c < x, c > 0. \end{cases}$$

1.3. Метод левых прямоугольников

Значение интеграла вычисляется приближённо с помощью метода левых прямоугольников. Сущность этого метода заключается в следующем: разобьем отрезок $[a;b]$ на n равных отрезков точками $a = x_0 < x_1 < x_2 < \dots < x_{n-1} < x_n = b$ и на каждом из полученных отрезков построим прямоугольник, одной стороной которого будет отрезок $[x_i, x_{i+1}]$, а другой – отрезок, длина которого равна $f(x_i)$. Если увеличивать число отрезков $[x_i, x_{i+1}]$, т.е. отрезок $[a;b]$ разбивать на большее число равных отрезков, то сумма их площадей всё с большей точностью будет совпадать с площадью криволинейной трапеции.

Значит, точность вычисления площади криволинейной трапеции определяется величиной числа n . Площадь каждого прямоугольника можно вычислить так. Одна из сторон прямоугольника, построенного на отрезке $[x_i, x_{i+1}]$

равна $h = \frac{(b-a)}{n}$, а вторая – $f(x_i)$. Поэтому площадь «левого» прямоугольника

равняется $s = h \cdot f(x_i) = \frac{(b-a)}{n} \cdot f(x_i)$. Тогда площадь криволинейной трапеции равна сумме площадей всех прямоугольников:

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

$$S = \frac{(b-a)}{n} \cdot f(x_0) + \frac{(b-a)}{n} \cdot f(x_1) + \dots + \frac{(b-a)}{n} \cdot f(x_{n-1}) = \\ = \frac{(b-a)}{n} \cdot [f(x_0) + f(x_1) + \dots + f(x_{n-1})] .$$

Таким образом, мы нашли интеграл функции $f(x)$ на отрезке $[a;b]$ при числе разбиений отрезка n . Очевидно, что чем больше значение n , тем больше точность вычисления значения интеграла; однако, при этом время, затрачиваемое на решение задачи, также прямо пропорционально числу n . Отсюда возникает вопрос: как правильно выбрать значение числа n , чтобы за наименьшее время найти значение искомого интеграла с предельно допустимой точностью?

Пусть мы знаем, что при разбиении отрезка интегрирования на n частей интеграл функции на данном интервале равен I_1 . Тогда мы можем также найти значение интеграла при числе разбиений отрезка, равном $2n$, и это значение будет равняться некоторому числу I_2 . Очевидно, что оценкой точности вычислений будет являться величина абсолютной погрешности $|I_1 - I_2|$.

Но в таком случае, именно величина абсолютной погрешности и есть тот критерий, который показывает, следует ли дальше продолжать вычисления или найденное значение интеграла уже удовлетворяет предельно допустимому уровню погрешности в поставленной задаче.

Также следует учесть, что границы интервала интегрирования могут быть подобраны таким образом, что ещё вначале вычислений, при большом шаге изменения x , программа завершится раньше достижения точности. Например, возьмём функцию $(|x|-1)^2$, на интервале $[-2,2]$. Тогда при первой итерации значением интеграла становится площадь прямоугольника с шириной $(|-2|-1)^2=1$ и длиной $2-(-2)=4$, т. е. 4. На следующей итерации шаг делится пополам (получается 2), абсцисса точка деления $-2+2=0$. фигура делится на 2 прямоугольника, ширина первого прямоугольника получается равной $(|-2|-1)^2=1$ и длина $0-(-2)=2$, его площадь — 2. У второго прямоугольника ширина $((|0|-1)^2=1$ и длина $2-0=2$, его площадь тоже 2. Суммируем площадь этих прямоугольников и получаем 4. Итак, нет разницы между текущим и предыдущим значением интеграла, и с точки зрения программы можно завершать работу. Но очевидно,

Изм.	Лист	№ докум.	Подп.	Дата

что площадь криволинейной трапеции будет меньше, так как $(|x|-1)^2$ на интервале $[-2,2]$ есть сдвинутый вправо и отраженный относительно оси ОY кусочек параболы $[-1;1]$, т. е. отрезок функции получается вписанным в прямоугольник 2×4 (который получается на первой итерации) и делит его на две части, из которых нижняя и является значением интеграла. Итак программа работает неправильно. Для решения этой проблемы можно задать некоторое минимальное количество разбиений отрезка интегрирования, и запретить программе завершать вычисления, пока текущее количество разбиений отрезка меньше заданного минимального.

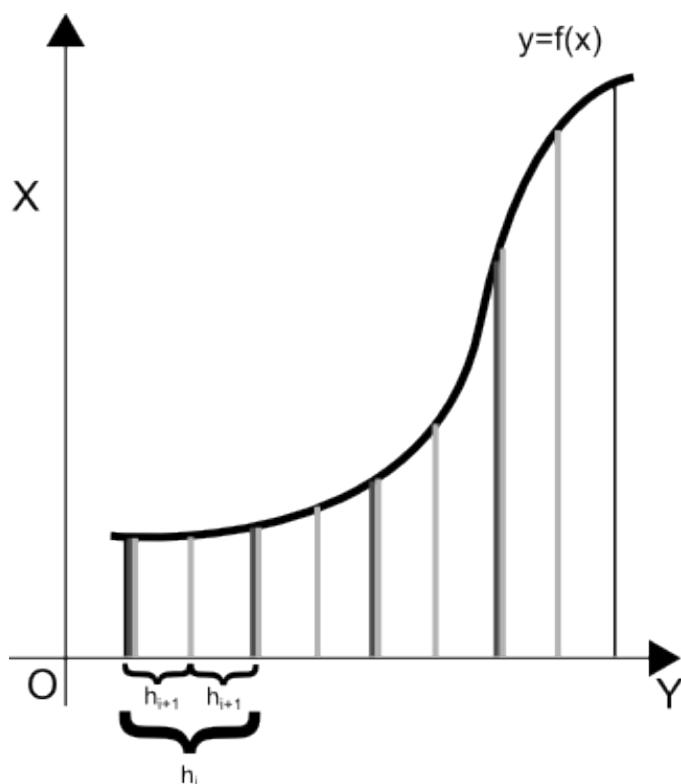
Исходя из того, что на каждой итерации значение шага уменьшается вдвое, можно заключить, что при расчёте текущего значения интеграла каждое второе значение функции будет просчитано дважды, т. к. $x_{i,0}=a$; $x_{i,1}=a+h_i$; $x_{i,2}=x+2h_i=x_{i-1,1}+$

$+h_{i-1}$ (т.к. $h_i = \frac{h_{i-1}}{2}$). Поэтому, начиная со второй итерации, целесообразно

начинать со значения $x=a+h_i$, где $h_i = \frac{h_{i-1}}{2}$, но брать $h_1=h_0$. Тогда текущее значение интеграла можно вычислить с использованием предыдущего результата:

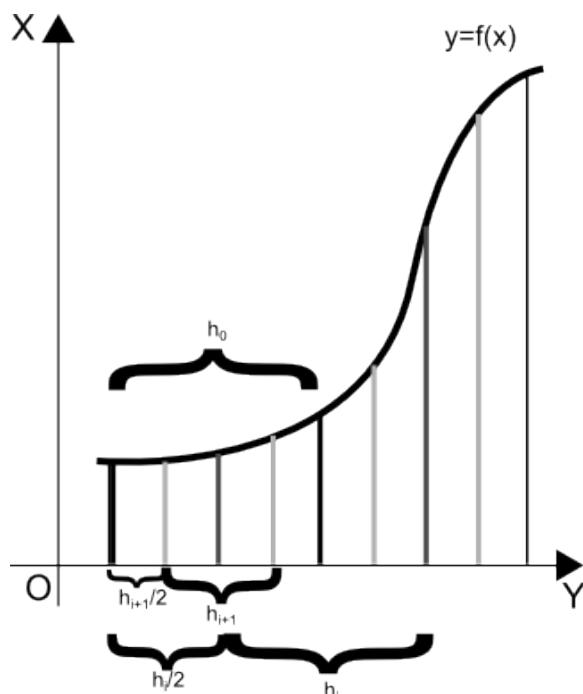
$I_n = \frac{I_{n-1}}{2} + \frac{h_n}{2} \cdot [f(x_0) + f(x_1) + \dots + f(x_{n-1})]$. Наглядно идею можно понять, рассмотрев рисунки 1.3 и 1.4.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------



- - расчет значений на этапе i (шаг h_i)
- - расчет значений на этапе $i+1$ ($h_{i+1}=h_i/2$)

Рисунок 1.2 — Вычисление значений функции по стандартному методу прямоугольников



- - расчет значений при инициализации ($i=0$, шаг h_0)
- - расчет значений на этапе $i>0$ (шаг h_i , плюс начальный сдвиг $h_i/2$)
- - расчет значений на этапе $i+1$ ($h_{i+1}=h_i/2$ (при $i=0$ $h_{i+1}=h_0$),
плюс начальный сдвиг $h_{i+1}/2$)

Рисунок 1.3 - Вычисление значений функции по оптимизированному методу прямоугольников

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

1.4. Разработка структур данных

Согласно численному методу, необходимо ввести следующие переменные:

d — параметры подынтегральной функции.

a — нижняя граница интервала интегрирования;

b — соответственно, верхняя граница интервала интегрирования;

eps – точность вычислений.

Все эти величины должны вводится пользователем. Далее описываются не вводимые пользователем переменные:

h - текущий шаг изменения аргумента анализируемой функции

reslt0 — значение интеграла на предыдущей итерации;

reslt - значение интеграла на текущей итерации;

x - аргумент функции на текущем отрезке разбиения;

nmin - минимальное количество разбиений отрезка интегрирования.

Далее описаны переменные, используемые для служебных нужд:

ok - указывает на отсутствие ошибок при вводе пользователем данных или желания пользователя прервать программу.

error – указывает на ошибку при значении true.

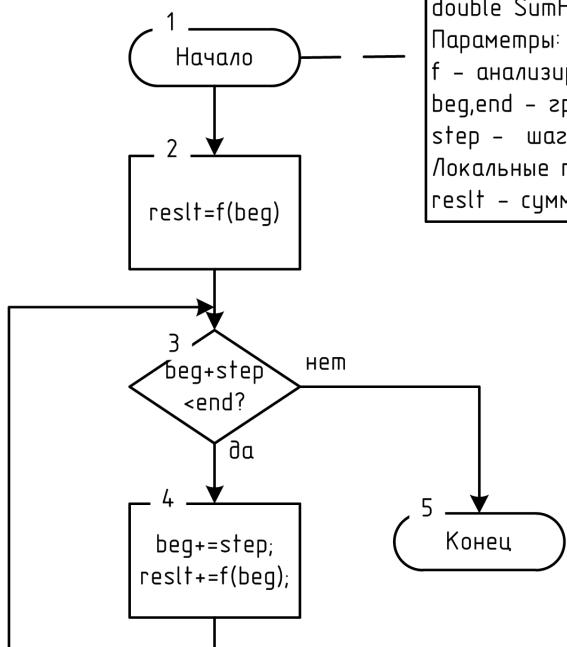
req_rslt – выражает согласие или несогласие пользователя по различным вопросам.

ch - содержит ответ пользователя на запрос программы о завершении, повторном вводе данных и завершении работы.

1.5. Разработка структуры алгоритма решения задачи

На рисунке 1.5 представлена схема алгоритма получения интеграла с заданной точностью, а на рисунке 1.4 - схема алгоритма суммирования значений функции, которую использует алгоритм получения интеграла.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------



Функция SumFunc() рассчитывает значения функции f , для каждого x из диапазона от beg до end с шагом step , и возвращает сумму вычисленных значений.

`double SumFunc(double (*f)(double x), double beg,double end,double step)`

Параметры:

f - анализируемая функция;

beg, end - границы изменения аргумента функции f ;

step - шаг изменения аргумента функции f .

Локальные переменные:

reslt - сумма вычисленных значений функции f .

Рисунок 1.4 - Схема алгоритма суммирования значений некоторой функции

Изм.	Лист	№ докум.	Подп.	Дата

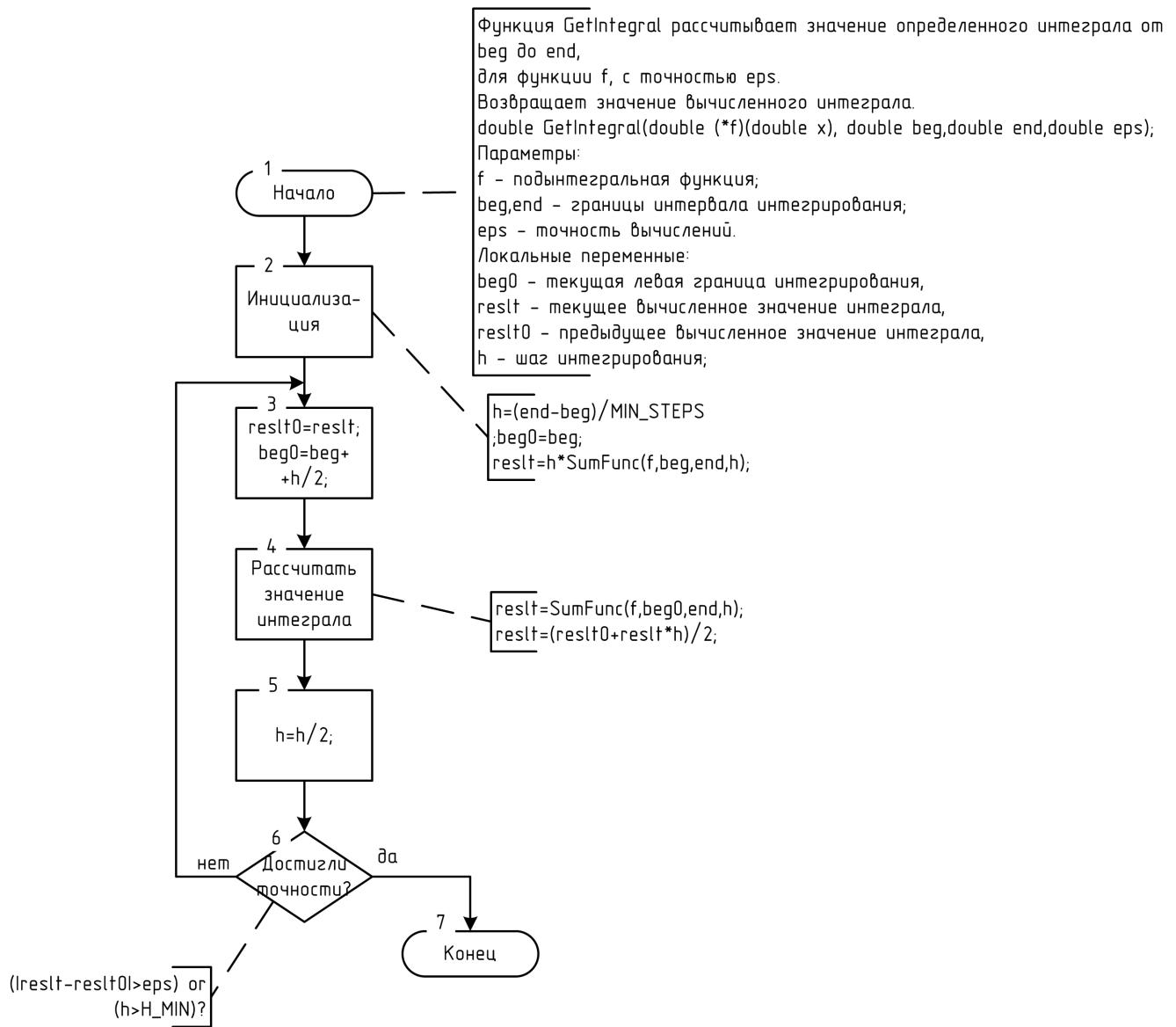


Рисунок 1.5 - Схема алгоритма расчета определенного интеграла с заданной точностью

На рисунках 1.6, 1.7, 1.8 представлены схемы алгоритмов получения некоторого параметра расчета в нескольких различных вариациях — если значение параметра ограничено минимальным значением, максимальным значением и неограниченного никаким значением.

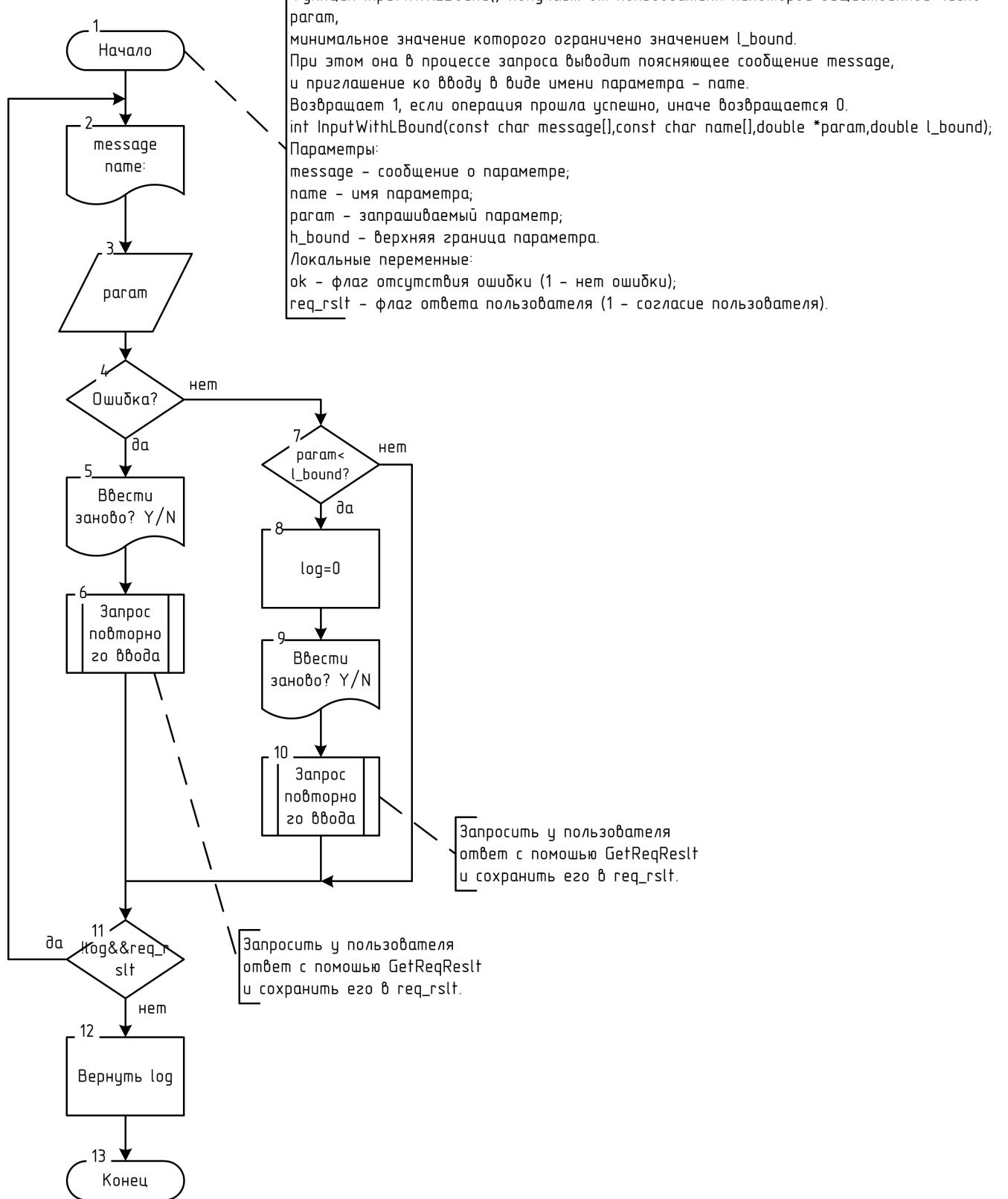


Рисунок 1.6 - Схема алгоритма ввода вещественного числа, ограниченного минимальным значением

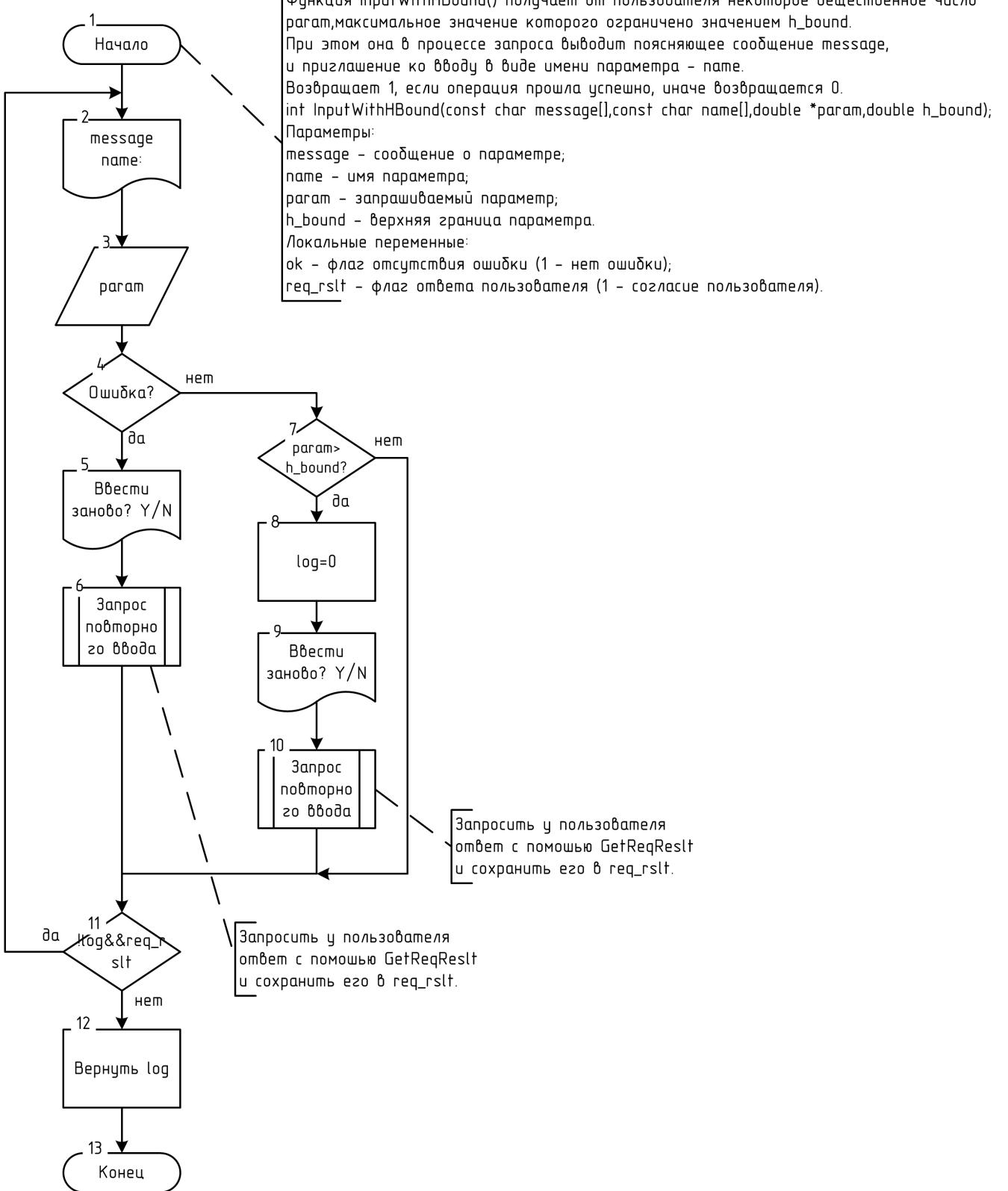


Рисунок 1.7 - Схема алгоритма ввода вещественного числа, ограниченного максимальным значением

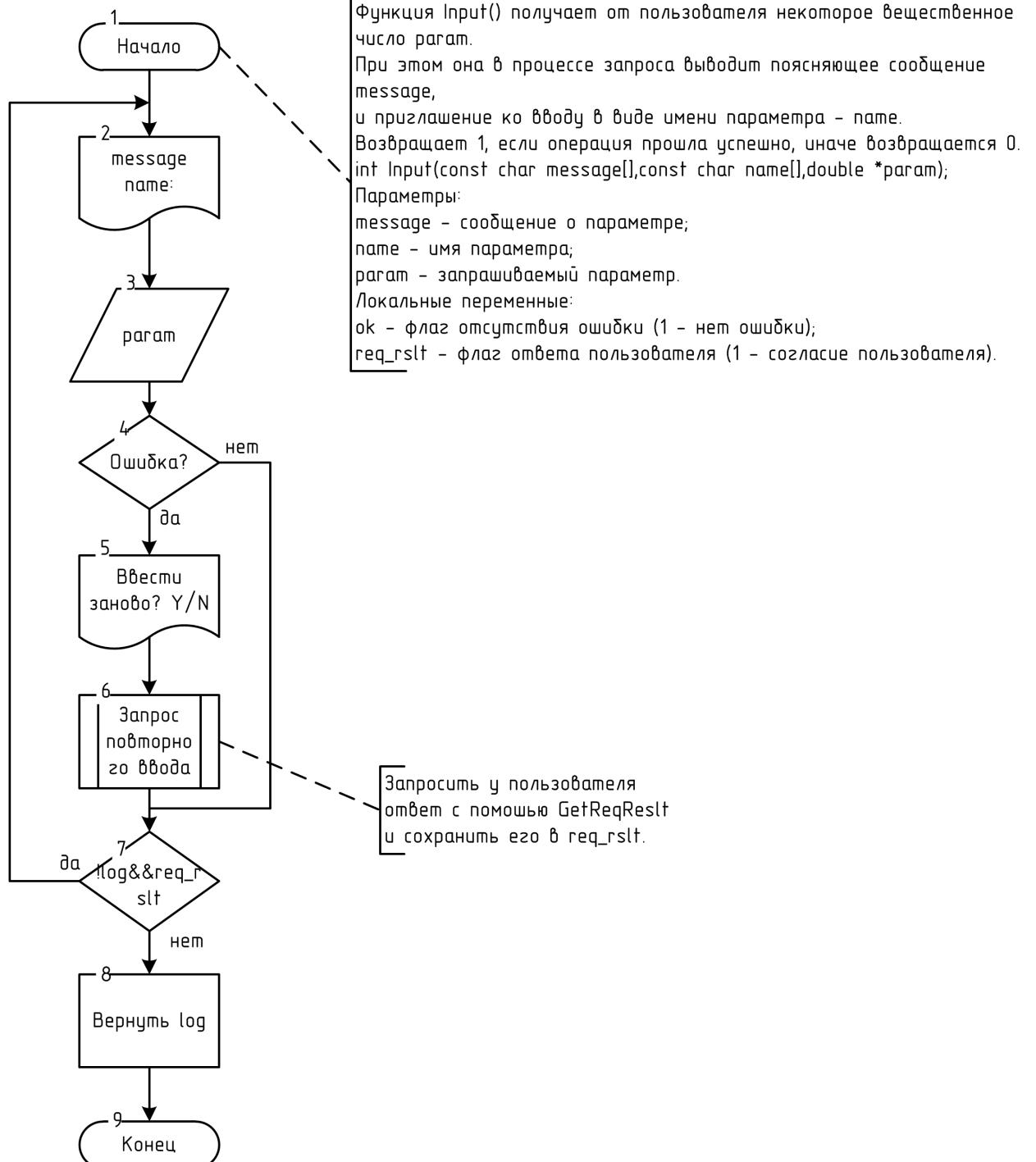
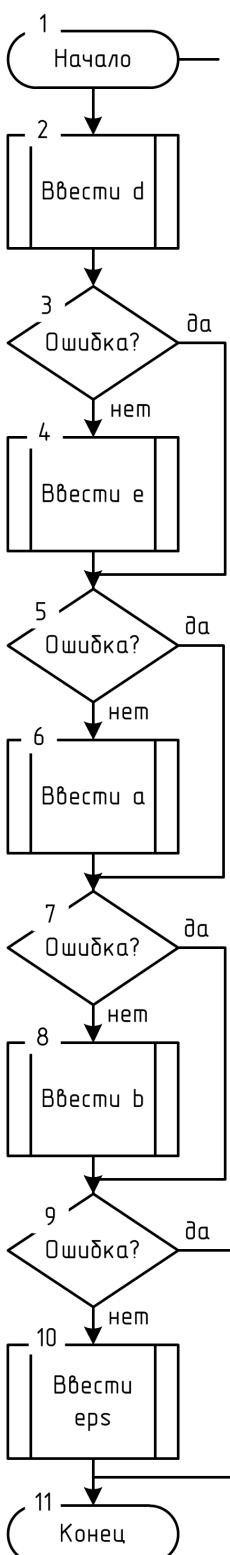


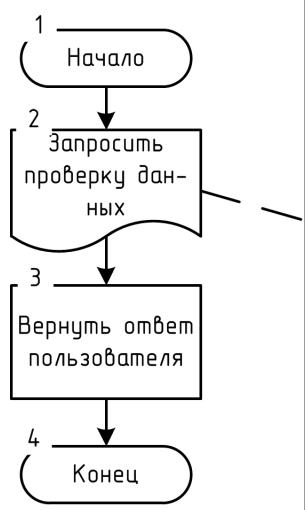
Рисунок 1.8 - Схема алгоритма ввода вещественного числа

На рисунке 1.9 представлена схема алгоритма ввода данных для программы получения интеграла с заданной точностью, а на рисунке 1.10 — проверки введенных данных.



Функция InputData позволяет запросить у пользователя параметры функции d и e ,
 границы интегрирования a и b , и точность вычислений eps .
 Возвращаем 1, если операция прошла успешно, иначе возвращается 0.
`int InputData(double *d,double *e,double *a, double *b,double *eps);`
 Параметры:
 d,e – параметры функции;
 a,b – границы интервала интегрирования;
 eps – точность вычислений.
 Локальные переменные:
 log – флаг отсутствия ошибки (1 – нет ошибки).

Рисунок 1.9 - Схема алгоритма ввода параметров расчета интеграла



Функция CheckData() позволяет пользователю проверить введенные данные: параметры функции d и e , границы интеграла интегрирования a и b , и точность вычислений eps . Возвращает 1, если пользователь подтвердил правильность данных, иначе возвращается 0.

```

int CheckData(
    const double d,
    const double e,
    const double a,
    const double b,
    const double eps
);

```

Параметры:
 d, e - параметры функции;
 a, b - границы интервала интегрирования;
 eps - точность вычислений.
Локальные переменные: отсутствуют.

Рисунок 1.10 - Схема алгоритма проверки параметров расчета интеграла

На рисунке 1.11 представлена общая схема алгоритма расчета интеграла - получения данных, их проверки, расчета интеграла и вывода найденного значения на экран.

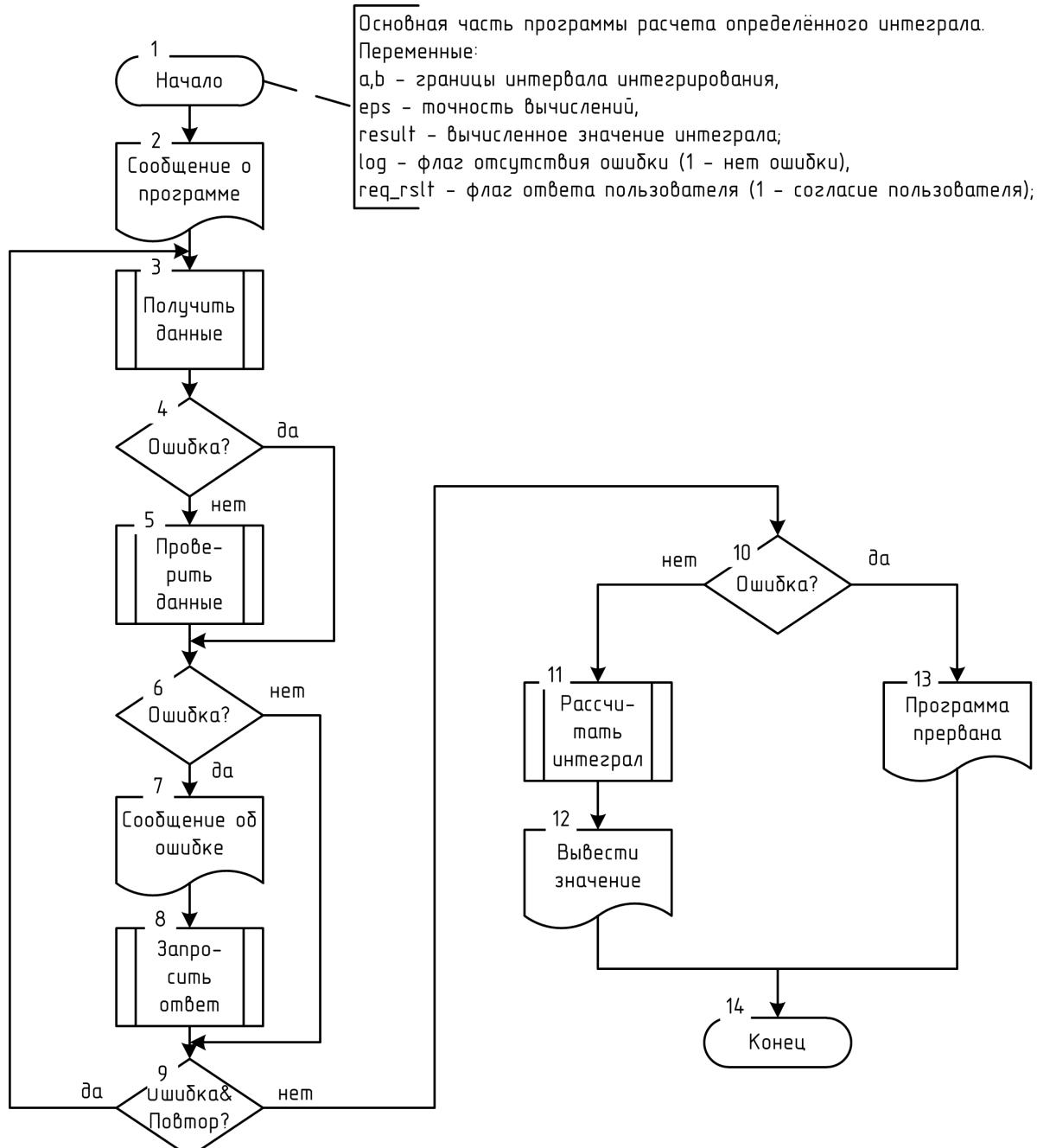


Рисунок 1.11 - Схема обобщенного алгоритма расчета интеграла

Схему алгоритма получения ответа от пользователя на поставленный вопрос, можно увидеть на рисунке 1.12.

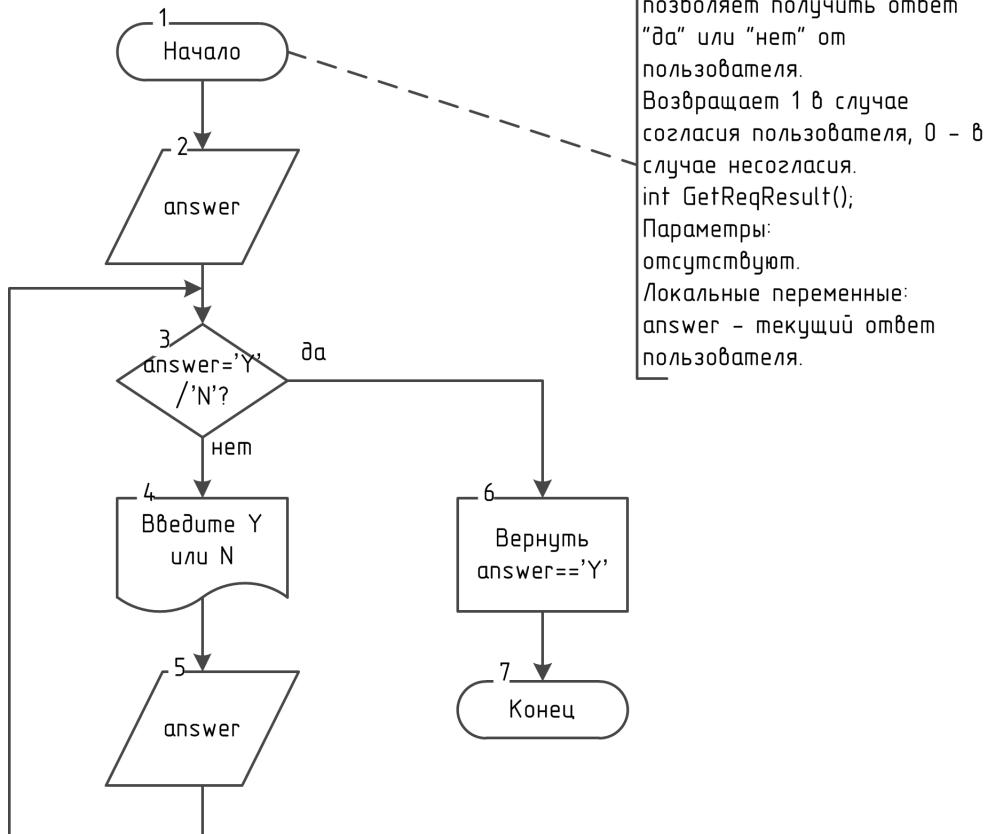


Рисунок 1.12 - Схема алгоритма получения ответа от пользователя

На рисунке 1.13 представлена схема алгоритма предоставления пользователю различных вариантов ответа.

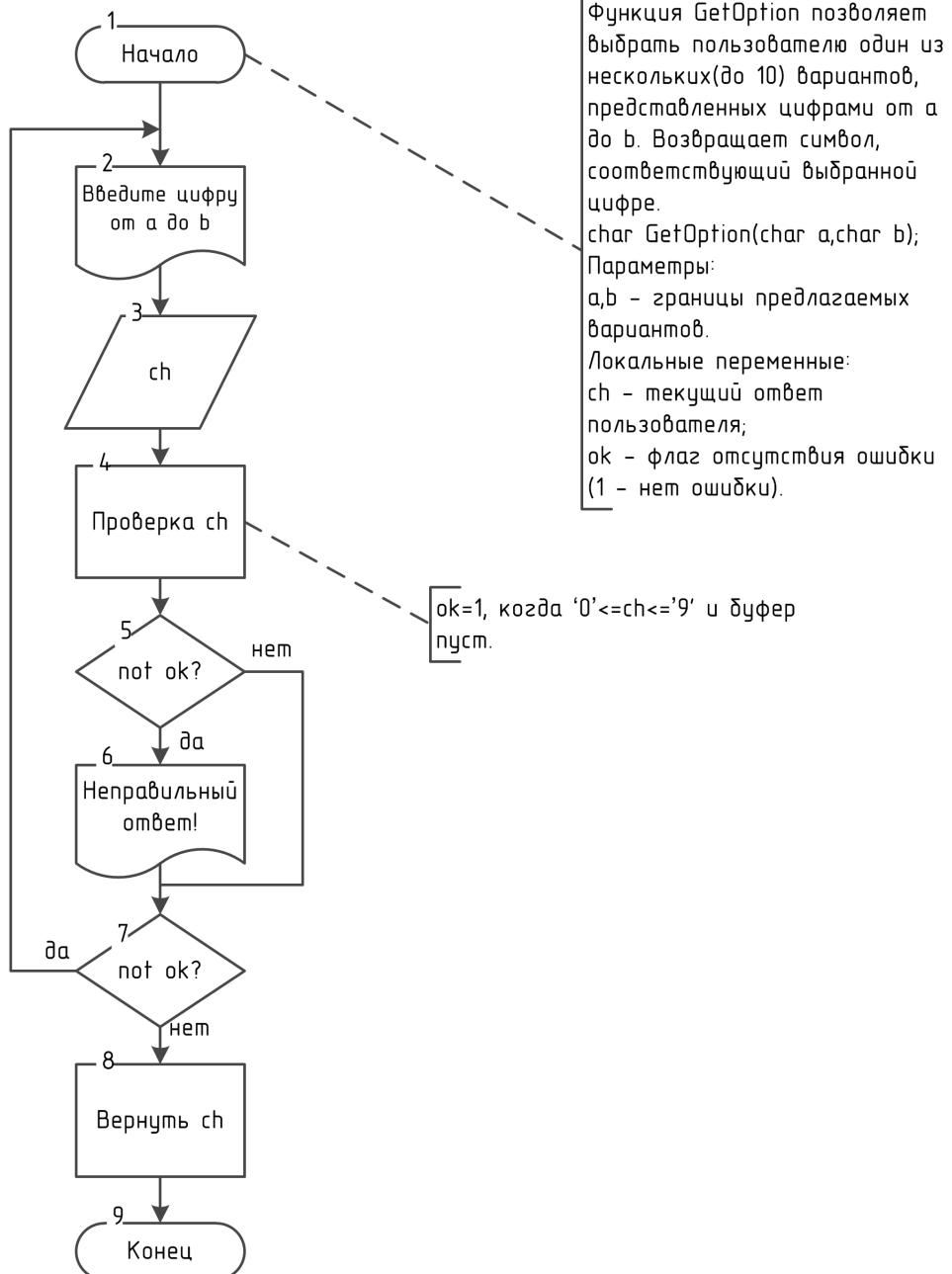


Рисунок 1.13- Схема алгоритма предоставления пользователю различных вариантов ответа

1.8. Текст программы

В этом разделе приведен исходный текст программы на языке Borland C 3.1, реализующей вычисление интеграла.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <math.h>
#include <ctype.h>
#define FIRST_STEPS_COUNT 16
#define H_MIN 0.1
#define EPS_MIN 0.00001
double d,e;
double f(double x){
    if (x<=d)
        return sqrt(3)*(x-d)/3-d;
    else if (x<=e)
        return -x;
    else
        return (x-e)*sqrt(3)-e;
}

/*****************/
/*
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 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 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
```

Изм.	Лист	№ докум.	Подп.	Дата

```

    );

/*
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);

/*
Функция InputWithHBound() получает от пользователя некоторое вещественное число
param,
максимальное значение которого ограничено значением 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;
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 InputWithHBound(const char message[], const char name[], double *param, double
h_bound);

/*
Основная часть программы расчета определенного интеграла.
Переменные:
a,b - границы интервала интегрирования,
eps - точность вычислений,
result - вычисленное значение интеграла;
log - флаг отсутствия ошибки (1 - нет ошибки),
req_rslt - флаг ответа пользователя (1 - согласие пользователя);
*/
int main();

/*
Функция InputData позволяет запросить у пользователя параметры функции d и e,
границы интеграла интегрирования a и b, и точность вычислений eps.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
d,e - параметры функции;
a,b - границы интервала интегрирования;
eps - точность вычислений.
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки).
*/
int InputData(
    double *d,
    double *e,

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

double *a, //a,b - границы отрезка поиска корня;
double *b,
double *eps //eps - точность вычислений;
);

/*
Функция CheckData() позволяет пользователю проверить введенные данные:
параметры функции d и e, границы интеграла интегрирования a и b,
и точность вычислений eps.
Возвращает 1, если пользователь подтвердил правильность данных, иначе возвращается
0.
Параметры:
d,e - параметры функции;
a,b - границы интервала интегрирования;
eps - точность вычислений.
Локальные переменные:
отсутствуют.
*/
int CheckData(
    const double d,
    const double e,
    const double a, //a,b - границы отрезка поиска корня;
    const double b,
    const double eps //eps - точность вычислений;
);

/*
Функция GetIntegral рассчитывает значение определенного интеграла от beg до end,
для функции f, с точностью eps.
Возвращает значение вычисленного интеграла.
Параметры:
f - подынтегральная функция;
beg,end - границы интервала интегрирования;
eps - точность вычислений.
Локальные переменные:
beg0 - текущая левая граница интегрирования,
reslt - текущее вычисленное значение интеграла,
reslt0 - предыдущее вычисленное значение интеграла,
h - шаг интегрирования;
*/
double GetIntegral(double (*f) (double x),
                    double beg,
                    double end,
                    double eps
                    );

/*
Функция SumFunc() рассчитывает значения функции f,
для каждого x из диапазона от beg до end с шагом step,
и возвращает сумму вычисленных значений.
Параметры:
f - анализируемая функция;
beg,end - границы изменения аргумента функции f;
step - шаг изменения аргумента функции f.
Локальные переменные:
reslt - сумма вычисленных значений функции f.
*/
double SumFunc(double (*f) (double x),
                double beg,
                double end,
                double step
                );
*******/


```

Изм.	Лист	№ докум.	Подп.	Дата

```

int clearline(FILE *f) {
    int count=0;
    while (!feof(f)&&(getc(f) !='\n')) count++;
    return count;
}

int main() {
    double a,b,eps,result;
    int log,req_rslt;
    printf("Программа выполняет расчет определенного интеграла\n\
            "на интервале от a до b с точностью eps для f(x)...\n\
            "           / (\sqrt{3}/3)*(x+d)/3+d , x<=d, d<0 (1);\n\
            "f(x) = || -x, d<x<=e, d<0, e>0 (2);\n\
            "           \\\n            \sqrt{3}*(x-e)-e , e<x, e>0 (3).\n");
    printf("\n\
            Y ^\n\
            |\n\
            |      @ (3)\n\
            xx | .\n\
            * | o| c @ X\n\
-----*---x--o--x-.---->\n\
    *      d |o | @\n\
**      | o|.n\
(1)      | xx \n\
          | (2)\n\
          |\n\
          |\n");
    printf("*** - ЛИНИЯ (1) - наклон к оси ОХ - 30°.\n\
            "ooo - ЛИНИЯ (2) - наклон к оси ОХ - -45°.\n\
            "@@ - ЛИНИЯ (3) - наклон к оси ОХ - 60°.\n\
            "x - точки пересечения графиков функций (c и d);.\n\
            "           задаются с клавиатуры.\n");

    do {
        log=InputData(&d,&e,&a,&b,&eps);
        if (log)
            log=CheckData(d,e,a,b,eps);
        if (!log){
            printf("Введенные данные некорректны!\n");
            printf("Хотите повторить ввод? Y/N\n");
            req_rslt=GetReqResult();
        }
    } while (!log&&req_rslt);
    if (log){
        result=GetIntegral(f,a,b,eps);
        printf("Интеграл на промежутке [%1.4lf,%1.4lf] равен %1.8lf+-\n\
%1.8lf.\n",a,b,result,eps);
    } else printf("Работа программы прервана!\n");
    printf("Нажмите <Enter>...\n");
    clearline(stdin);
    return 0;
}
int Input(const char message[], //paramName - имя запрашиваемого параметра;
            const char name[], //paramCond - дополнительная информация о
параметре;
            double *param
            ){
    int log,req_rslt;
    do {
        req_rslt=0;
        printf("%s",message);
        printf("%s: ",name);
        log=scanf("%lf",param);
    }
}
```

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подп.</i>	<i>Дата</i>

```

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 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 InputWithHBound(const char message[],const char name[],double *param,double
h_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>h_bound)) {
            log=0;
            printf("Ошибка! %s больше %lf!\n"
                   "Повторить ввод(Y/N)?\n",name,h_bound);
            req_rslt=GetReqResult();
        }
    } while(!log&&req_rslt);
    return log;
}
int InputData(
    double *d,
    double *e,
    double *a, //a,b - границы отрезка поиска корня;
    double *b,
    double *eps //eps - точность вычислений;
)
int log=1; char msg[100];

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

log=InputWithHBound("Введите d - точка пересечения линий (1) и (2) -  

вещественное число. d<0.\n","d",d,0);
if (log)
    log=InputWithLBound("Введите e - точка пересечения линий (2) и (3) -  

вещественное число. e>0.\n","e",e,0);
if (log)
    log=Input("Введите a - ниж. граница интегрирования - вещественное  

число.\n","a",a);
if (log)
    sprintf(msg,"Введите b - верх. граница интегрирования -вещественное число.  

b>%lf.\n",*a);
    log=InputWithLBound(msg,"b",b,*a);
}
if (log)
    log=InputWithLBound("Введите eps - точность вычислений - вещественное  

число. eps>0.\n","eps",eps, EPS_MIN);
    return log;
}

double SumFunc(double (*f) (double x),
                double beg,
                double end,
                double step
               ) {
double reslt;
reslt=(*f)(beg);
while (beg+step<end) {
    beg+=step;
    reslt+=(*f)(beg);

}
return reslt;
}
double GetIntegral(double (*f) (double x),
                    double beg,
                    double end,
                    double eps
                   ) {
double beg0,reslt,reslt0,h;
h=(end-beg)/FIRST_STEPS_COUNT;beg0=beg;
reslt=h*SumFunc(f,beg,end,h); //инициализация
do {
    reslt0=reslt; beg0=beg+h/2;
    /*начинаем со второго прямоугольника - площадь первого (удвоенную) уже
     вычислили на предыдущей итерации, шаг пока не изменяется - чтобы не
     обрабатывать прямоугольники с нечётными номерами*/
    reslt=SumFunc(f,beg0,end,h);
    reslt=(reslt0+reslt*h)/2; //так как удвоенная площадь "нечетных"
                                // прямоугольников вычислена ранее,
                                // делим её пополам: шаг пока еще удвоенный -
                                // делим его пополам и считаем площадь
"четных"
                                // прямоугольников; суммируем - получаем
площадь
                                // фигуры
    h=h/2;
    putchar('.');
} while ((fabs(reslt-reslt0)>eps) || (h>H_MIN));
printf("OK!\n");
return reslt;
}
int GetReqResult(){
char answer;
answer=getchar();
clearline(stdin);
while (toupper(answer) !='Y'

```

Изм.	Лист	№ докум.	Подп.	Дата

```

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

return toupper(answer)=='Y';
}

int CheckData(
    const double d,
    const double e,
    const double a, //a,b - границы отрезка поиска корня;
    const double b,
    const double eps //eps - точность вычислений;
) {
printf("Будет выполнен расчет определенного интеграла\n"\
       "на интервале от %1.4lf до %1.4lf с точностью %1.8lf.\n"\
       "Параметры функции: d=%1.4lf; e=%1.4lf.\n"\
       "Продолжить? (Y/N)\n",a,b,eps,d,e);
return GetReqResult();
}

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

1.6. Инструкция пользователю

Программа вычисляет значение интеграла на интервале [a, b] с точностью eps для функции (1.1). При вводе параметров необходимо соблюдать следующие условия: параметр d — вещественное отрицательное число; e — вещественное положительное число; границы интегрирования a, b — вещественные числа, причём $b > a$; точность eps — вещественное положительное число. При неверном вводе ввод можно повторить, ответив 'Y'. Можно прервать программу сочетанием клавиш Control+C. После ввода данных программа предложит их проверить, чтобы начать вычисления, нажмите 'y' и Enter. В случае неуспешного ввода или признания данных некорректными программа предложит повторить ввода данных.

Если данные были введены корректно, то после окончания расчетов программа выведет искомое значение интеграла.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

1.7. Инструкция программисту

При создании программы расчета определённого интеграла были предприняты следующие действия.

Объявлены константы FIRST_STEPS_COUNT и H_MIN — наименьшое количество интервалов разбиения и минимальный шаг интегрирования, EPS_MIN — минимальная точность вычислений.

В главной части объявлены переменные, описание которых приводится в таблице 1.1:

Таблица 1.1 - Переменные программы расчета определённого интеграла

имя	тип	предназначение
a,b	double	границы интервала интегрирования,
d,e	double	параметры подынтегральной функции,
eps	double	точность вычислений,
result	double	вычисленное значение интеграла;
log	int	флаг отсутствия ошибки (1 – нет ошибки),
req_rslt	int	флаг ответа пользователя (1 – согласие пользователя);

Описаны следующие подпрограммы:

1. Подынтегральная функция f.

double f(double x);

Использует глобальные переменные d и e. Возвращает значение функции, рассмотренной в разделе 1.2.

2. 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 1.2, локальные переменные — в таблице 1.3.

Таблица 1.2 - Параметры функции очистки буфера

имя	тип	предназначение

f	FILE*	Файл, в котором очищается строка.
---	-------	-----------------------------------

Таблица 1.3 - Локальные переменные функции очистки буфера

имя	тип	предназначение
count	long	Счетчик считанных символов.

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

Заголовок функции:

```
int GetReqResult();
```

Функция запрашивает у пользователя символы 'у','Y','н' 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 1.4.

Таблица 1.4 - Локальные переменные функции получения ответа

имя	тип	предназначение
answer	char	Ответ пользователя.

4. 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 1.5, локальные переменные — в таблице 1.6.

Таблица 1.5 - Параметры функции получения варианта

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

Таблица 1.6 - Локальные переменные функции получения варианта

имя	тип	предназначение
ch	char	Ответ пользователя.

ok	int	Флаг — равен 1, если ответ пользователя допустим, иначе равен 0.
----	-----	--

5. 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.

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

Параметры функции представлены в таблице 1.7:

Таблица 1.7 - 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 1.8:

Таблица 1.8 - 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)

6. 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.

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

Параметры функции представлены в таблице 1.9:

Таблица 1.9 - Параметры функции получения вещественного числа с нижней границей

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

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

Локальные переменные функции представлены в таблице 1.10:

Таблица 1.10 - Локальные переменные функции получения вещественного числа с нижней границей

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

7. Функция InputWithHBound() получает от пользователя некоторое вещественное число param, максимальное значение которого ограничено значением 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.

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

Параметры функции представлены в таблице 1.11:

Таблица 1.11 - Параметры функции получения вещественного числа с верхней границей

имя	тип	предназначение
message	char *	сообщение о параметре
name	char *	имя параметра
param	double *	запрашиваемый параметр
h_bound	double	верхняя граница параметра

Локальные переменные функции представлены в таблице 1.12:

Таблица 1.12 - Локальные переменные функции получения вещественного числа с верхней границей

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

8. Функция InputData позволяет запросить у пользователя параметры функции d и e, границы интеграла интегрирования a и b, и точность вычислений eps.

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

```

int InputData(
    double *d,
    double *e,
    double *a,
    double *b,
    double *eps
);

```

Параметры функции представлены в таблице 1.13:

Таблица 1.13 - Параметры функции получения данных

имя	тип	предназначение
d,e	double*	параметры функции
a,b	double*	границы интервала интегрирования
eps	double*	точность вычислений

Локальные переменные функции представлены в таблице 1.14:

Таблица 1.14 - Локальные переменные функции получения данных

имя	тип	предназначение
log	int	флаг отсутствия ошибки (1 - нет ошибки)

9. Функция CheckData() позволяет пользователю проверить введенные данные: параметры функции d и e, границы интеграла интегрирования a и b, и точность вычислений eps.

Возвращает 1, если пользователь подтвердил правильность данных, иначе возвращается 0.

```

int CheckData(
    const double d,
    const double e,
    const double a, //a,b - границы отрезка поиска корня;
    const double b,
    const double eps //eps - точность вычислений;
);

```

Параметры функции представлены в таблице 1.15:

Таблица 1.15 - Параметры функции проверки данных

имя	тип	предназначение
d,e	double	параметры функции

a,b	double	границы интервала интегрирования
eps	double	точность вычислений

Локальные переменные:

отсутствуют.

10. Функция GetIntegral рассчитывает значение определенного интеграла от beg до end, для функции f, с точностью eps. Возвращает значение вычисленного интеграла.

```
double GetIntegral(double (*f)(double x),
                  double beg,
                  double end,
                  double eps
);
```

Параметры функции представлены в таблице 1.16:

Таблица 1.16 - Параметры функции получения определенного интеграла

имя	тип	предназначение
f	(*f)(double x)	подынтегральная функция
beg,end	double	границы интервала интегрирования
eps	double	точность вычислений

Локальные переменные функции представлены в таблице 1.17:

Таблица 1.17 - Локальные переменные функции получения определенного интеграла

имя	тип	предназначение
beg0	double	текущая левая граница интегрирования
reslt	double	текущее вычисленное значение интеграла
reslt0	double	предыдущее вычисленное значение интеграла
h	double	шаг интегрирования

11. Функция SumFunc() рассчитывает значения функции f, для каждого x из диапазона от beg до end с шагом step, и возвращает сумму вычисленных значений.

```
double SumFunc(double (*f)(double x),
               double beg,
               double end,
               double step
);
```

Параметры функции представлены в таблице 1.18:

Изм.	Лист	№ докум.	Подп.	Дата

Таблица 1.18 - Параметры функции получения суммы значений функции

имя	тип	предназначение
f	(*f)(double x)	подынтегральная функция
beg,end	double	границы интервала интегрирования
step	double	шаг изменения аргумента функции f

Локальные переменные функции представлены в таблице 1.19:

Таблица 1.19 - Локальные переменные функции получения суммы значений функции

имя	тип	предназначение
reslt	double	сумма вычисленных значений функции f

1.9. Тестовый пример

Ниже на рисунке 1.4 показан пример работы программы вычисления определённого интеграла для параметров функции $d=-4$, $c=3$, границами интегрирования $[-8, 7]$, с точностью eps .

```
Программа выполняет расчет определенного интеграла
на интервале от a до b с точностью eps для f(x)...
    / ( $\sqrt{3}/3$ ) $^x$ (x+d)/3+d , x<=d, d<0 (1);
f(x) =| | -x, d<x<=e, d<0, e>0 (2);
    \  $\sqrt{3}^x$ (x-e)-e , e<x, e>0 (3).
Y ^

        @ (3)
        xx | c   .      x
-----*-----x---o-x----->
        * | o | @
        d   | o | .
-----*-----x---o-x----->
(1)           xx
                (2)

*** - ЛИНИЯ (1) - наклон к оси ОХ - 30° .
ooo - ЛИНИЯ (2) - наклон к оси ОХ - -45° .
@. @ - ЛИНИЯ (3) - наклон к оси ОХ - 60° .
x - точки пересечения графиков функций (c и d);
      задаются с клавиатуры.
Введите d - точка пересечения линий (1) и (2) - вещественное число. d<0.
d: -4
Введите e - точка пересечения линий (2) и (3) - вещественное число. e>0.
e: 3
Введите a - ниж. граница интегрирования - вещественное число.
a: -8
Введите b - верх. граница интегрирования -вещественное число. b>-8.000000.
b: 7
Введите eps - точность вычислений - вещественное число. eps>0.
eps: 0.0001
Будет выполнен расчет определенного интеграла
на интервале от -8.0000 до 7.0000 с точностью 0.00010000.
Параметры функции: d=-4.0000; e=3.0000.
Продолжить?(Y/N)
Y
.....OK!
Интеграл на промежутке [-8.0000,7.0000] равен 16.73754029+-0.00010000.
Нажмите <Enter>...
```

Рисунок 1.14 — Результат работы программы вычисления интеграла
Рассчитаем интеграл вручную.

Проинтегрируем наши функции.

$$Y = \begin{cases} \int_{-8}^{-4} \left(\frac{\sqrt{3}}{3}(x+4) + 4 \right) dx \\ \int_3^3 -x dx; \\ \int_3^7 \sqrt{3}(x-3) - 3. \end{cases} = \begin{cases} \int_{-8}^{-4} \left(\frac{\sqrt{3}}{6}x^2 + \frac{4}{3}(\sqrt{3}+3)x \right)' dx \\ \int_{-4}^3 \left(\frac{-x^2}{2} \right)' dx; \\ \int_3^7 \left(\frac{\sqrt{3}}{2}x^2 - 3(\sqrt{3}+1)x \right)' dx. \end{cases}$$

6

Итак, рассчитаем значение первого интеграла:

$$4,61880 - 25,23760430 - (18,47250 - 50,4752086) = 11,381189.$$

Рассчитаем значение второго интеграла:

$$-4,5 + 8 = 3,5$$

Рассчитаем значение третьего интеграла:

$$(42,43524 - 57,373067) - (7,79422863 - 24,588457) = 1,85640137$$

Суммируем и получаем $11,381189 + 3,5 + 1,85640137 = 16,73759037$.

Программа работает верно, требуемая точность достигнута.

Изм.	Лист	№ докум.	Подп.	Дата

2. Вычисление таблицы значений функции, заданной в виде разложения в ряд

2.1. Постановка задачи

Разработать алгоритм и составить программу вычисления таблицы значений функции, заданной в виде разложения в ряд.

$$f(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + \frac{x^{2k}}{2k!} + \dots \quad (2.1)$$

Значение функции вычислять с точностью $\varepsilon > 0$, т.е. вычисление суммы членов ряда необходимо прекратить, когда абсолютная величина очередного члена ряда разложения окажется меньше ε : $|a_k| < \varepsilon$.

При составлении программы необходимо по возможности воспользоваться операторами организации циклов **WHILE**, **REPEAT**, **FOR**.

Границы интервала вычислений функций **a** и **b**, величина шага изменения аргумента **h** и точность вычисления функции **ε** задаются при вводе. На печать выводятся номер по порядку, значение аргумента, соответствующие ему, значение функции и номер члена ряда, на котором закончилось вычисление значение функции, как показано на таблице:

№	X	f(x)	№ чл.р.
1			
2			
3			
...			

2.2 Математическая формулировка задачи

Рассчитать значения функции $f(x)$ для каждого x_i , где $x_i = x_{i-1} + h$; $i = 1, 2, \dots, n$; где n — номер последнего x , h — некоторая константа; причём $x_1 = a$ и $x_n \leq b$, где a и b — некоторые константы.

2.3. Численный метод решения задачи

При вычислении очередного члена целесообразно воспользоваться рекурентным выражением:

$$a_{k+1} = c_k a_k; \quad k = 0, 1, 2, 3, \dots, \quad (2.2)$$

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

где a_k - некоторый k -ый член ряда; a_{k-1} — предыдущий, $k-1$ -ый член ряда; c_k - коэффициент, определяемый номером k .

Найдем это выражение. Т. к. $a_k = \frac{x^{2k}}{2k!}$, то $a_{k+1} = \frac{x^{2k+1}}{(2k+1)!} =$

$\frac{x^{2k} \cdot x^2}{2k! \cdot (2k+1) \cdot (2k+2)}$, $c_k = a_{k+1} / a_k$, получим формулу для вычисления c_k :

$$c_k = \frac{x^2}{(2k+1) \cdot (2k+2)} \quad (2.3)$$

Возьмём $a_0=1$ (т. е. $k=0$) и $x_1=a$, и будем вычислять каждое следующее a_{k+1} , умножая a_k на c_k пока $|a_{k+1}| < \varepsilon$. Так как x^2 из формулы (2.3) получается постоянной величиной, есть смысл ввести дополнительную переменную для запоминания этого значения. Кроме того, есть смысл увеличивать k перед вычислением следующего значения, ведь тогда формула (2.3) преобразуется к виду $\frac{x^2}{(2k-1) \cdot (2k)}$, что несколько приятнее на вид. При выполнении условия $|a_{k+1}| < \varepsilon$ выведем необходимые данные и перейдём к следующему x из промежутка $[a,b]$. Повторим всю процедуру с начала до тех пор, пока текущий x не выйдет из указанного промежутка. Для удобства написания можно заранее подсчитать количество вычисляемых значений (и различных x , соответственно), чтобы вычислять значения функции, перебирая аргумент с помощью цикла for.

2.4. Разработка структур данных

Согласно численному методу, необходимо ввести следующие переменные:

a, b - нижняя и верхняя границы изменения аргумента функции;

h - текущий шаг изменения аргумента анализируемой функции;

eps - точность вычисления определённого интеграла.

Все эти величины должны вводится пользователем. Далее описываются не вводимые пользователем переменные:

table – структура, состоящая из числа и массива, число Count обзначает размер массива, а массив lines символизирует строки таблицы. Каждая "строка" тоже является записью и состоит из следующих компонент:

x - текущий аргумент функции;

Изм.	Лист	№ докум.	Подп.	Дата

у - значение функции для текущего аргумента;

last – номер последнего вычисленного члена ряда.

Опишем также некоторые дополнительные переменные, которые помогут при расчете значения ряда:

k - порядковый номер вычисляемого элемента последовательности;

ук - вычисляемый элемент последовательности ;

sqrх - квадрат текущего аргумента .

Далее описаны переменные, используемые для служебных нужд:

ok,log - указывает на отсутствие ошибок при вводе пользователем данных или желания пользователя прервать программу.

error – указывает на ошибку при значении true.

req_rslt – выражает согласие или несогласие пользователя по различным вопросам.

i – переменная-счетчик для доступа к строкам.

fin, fout – файлы для ввода и вывода информации.

buf - буферная переменная, в которой содержатся введённые пользователем значения переменных в формате текстовой строки.

code - сюда записывается значение кода ошибки при извлечении данных из буфера , если ошибки нет, то code равно 0.

ch - содержит ответ пользователя на запрос программы о завершении, повторном вводе данных и завершении работы.

2.5. Разработка структуры алгоритма решения задачи

На рисунке 2.1 представлена схема алгоритма расчета всей таблицы значений, а на рисунке 2.2 — лишь одного значения ряда.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

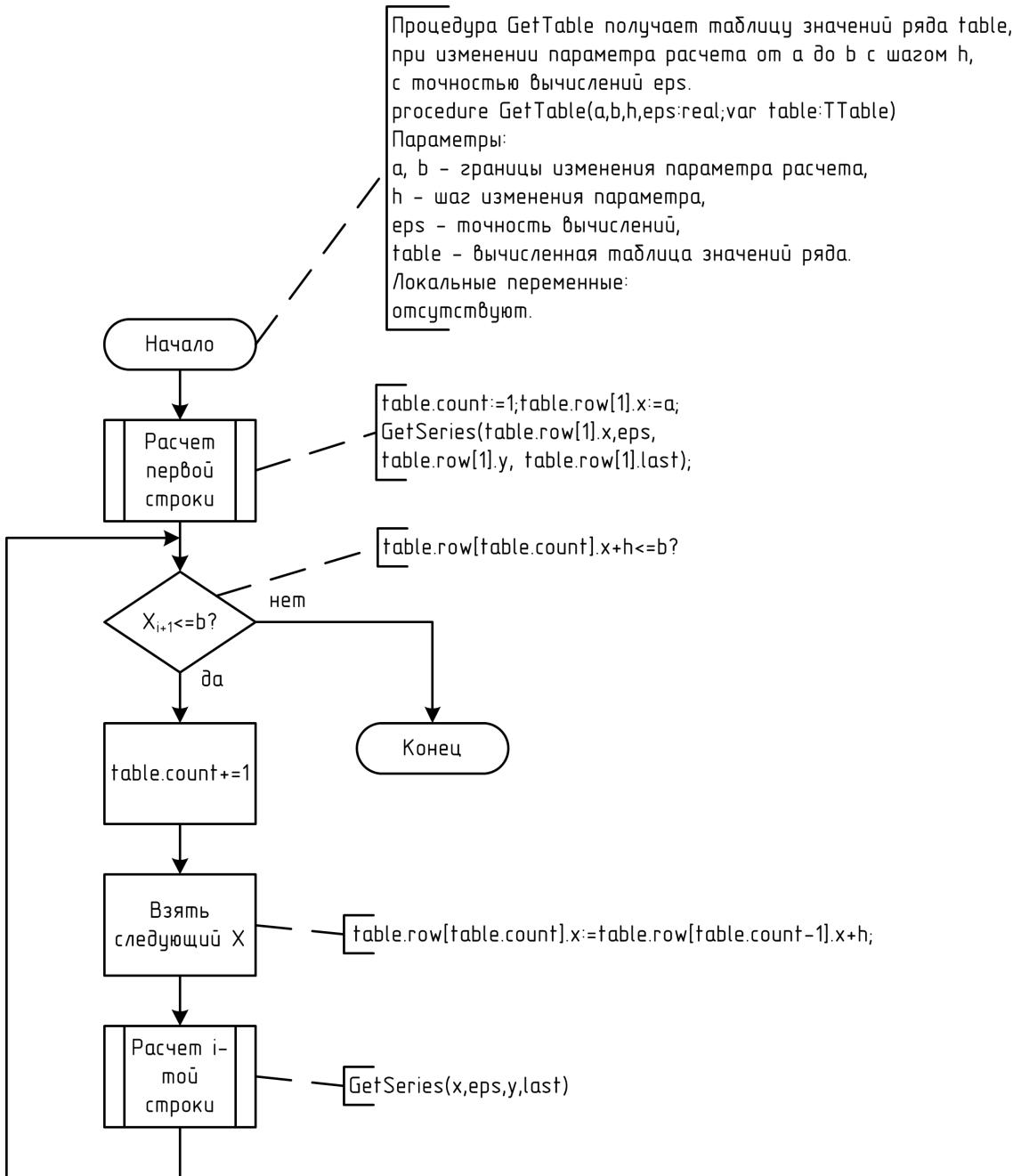


Рисунок 2.1 - Схема алгоритма расчета таблицы значений

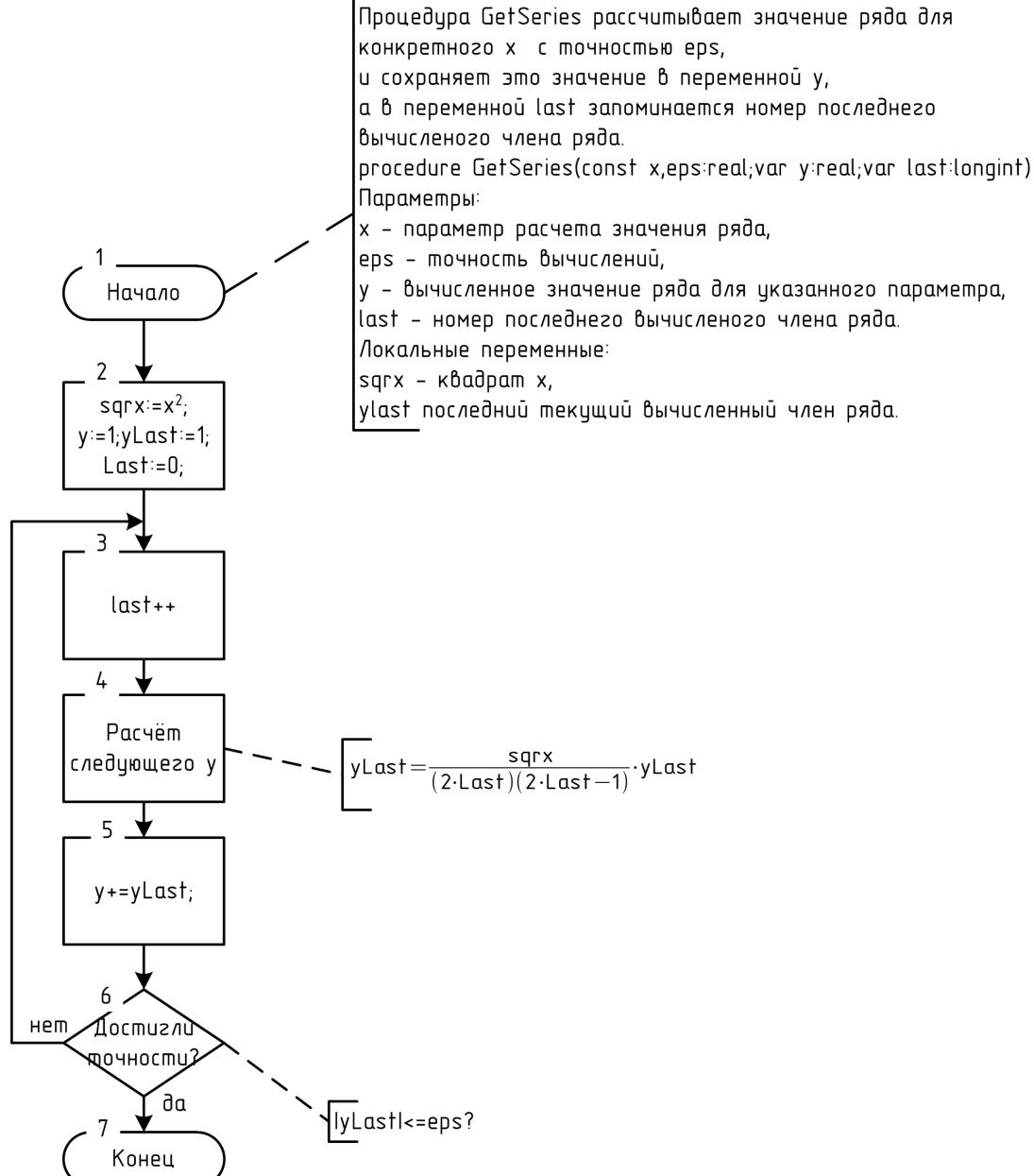
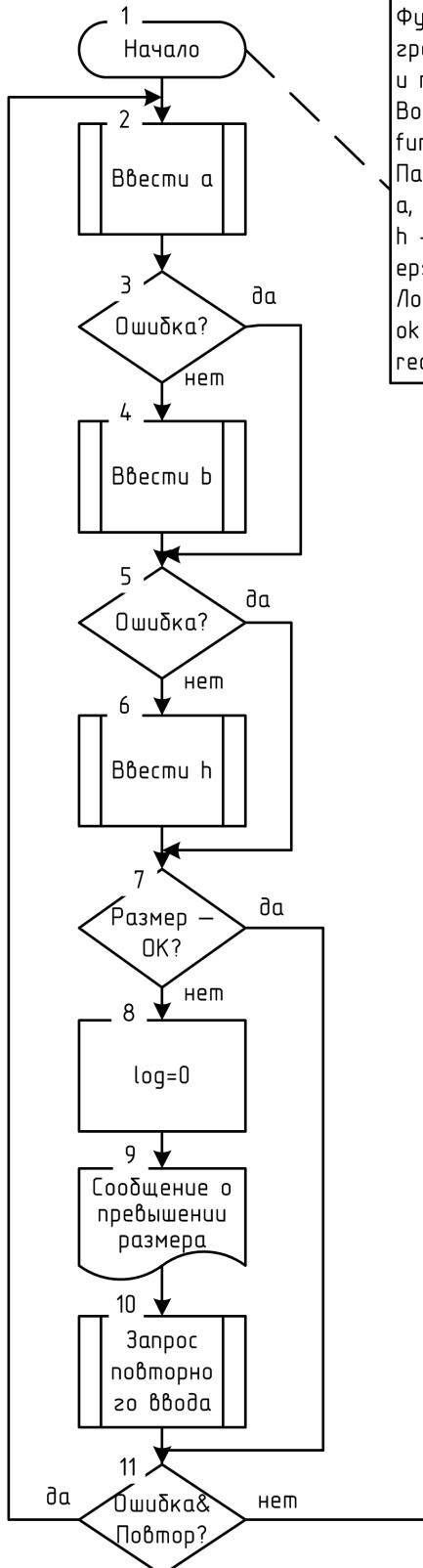


Рисунок 2.2 - Схема алгоритма рассчета значения ряда

Схемы алгоритмов получения некоторого параметра расчета были уже рассмотрены в разделе 1.4. На рисунке 2.3 же представлена схема алгоритма ввода всех параметров расчета с клавиатуры. Также на рисунке 2.4 представлена и схема проверки введенных ранее данных.



Функция InputConsole() позволяет пользователю считать с клавиатуры границы изменения параметра расчета *a* и *b*, шаг изменения параметра *h*, и точность вычислений *eps*.
 Возвращает true, если операция прошла успешно, иначе возвращается false.
 function InputConsole(var a,b,h,eps:real):boolean
 Параметры:
a, b – границы изменения параметра расчета,
h – шаг изменения параметра,
eps – точность вычислений.
 Локальные переменные:
ok – флаг отсутствия ошибки (true – нет ошибки);
req_rslt – флаг ответа пользователя (true – согласие пользователя).

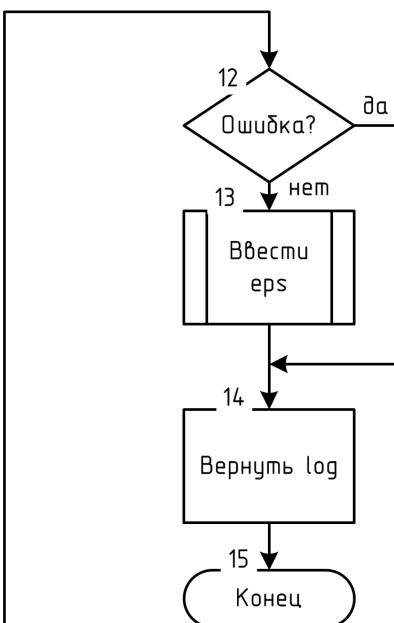
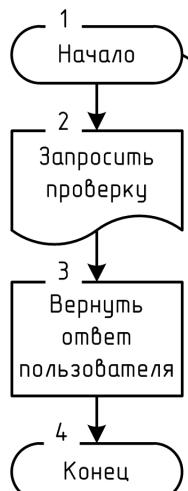


Рисунок 2.3 - Схема алгоритма ввода параметров с клавиатуры



Функция CheckData() позволяет пользователю проверить введенные данные: границы параметра расчета a и b , шаг изменения параметра h , и точность вычислений eps . Возвращает true, если пользователь подтвердил правильность данных, иначе возвращается false.

```
function CheckData(const a,b,h,eps:real):boolean;
```

Параметры:

- a, b - границы изменения параметра расчета,
- h - шаг изменения параметра,
- eps - точность вычислений.

Локальные переменные: отсутствуют.

Рисунок 2.4 - Схема алгоритма проверки данных

В общем же схема алгоритма ввода данных, их проверки, расчета таблицы и вывода её на экран представлена на рисунке 2.5.

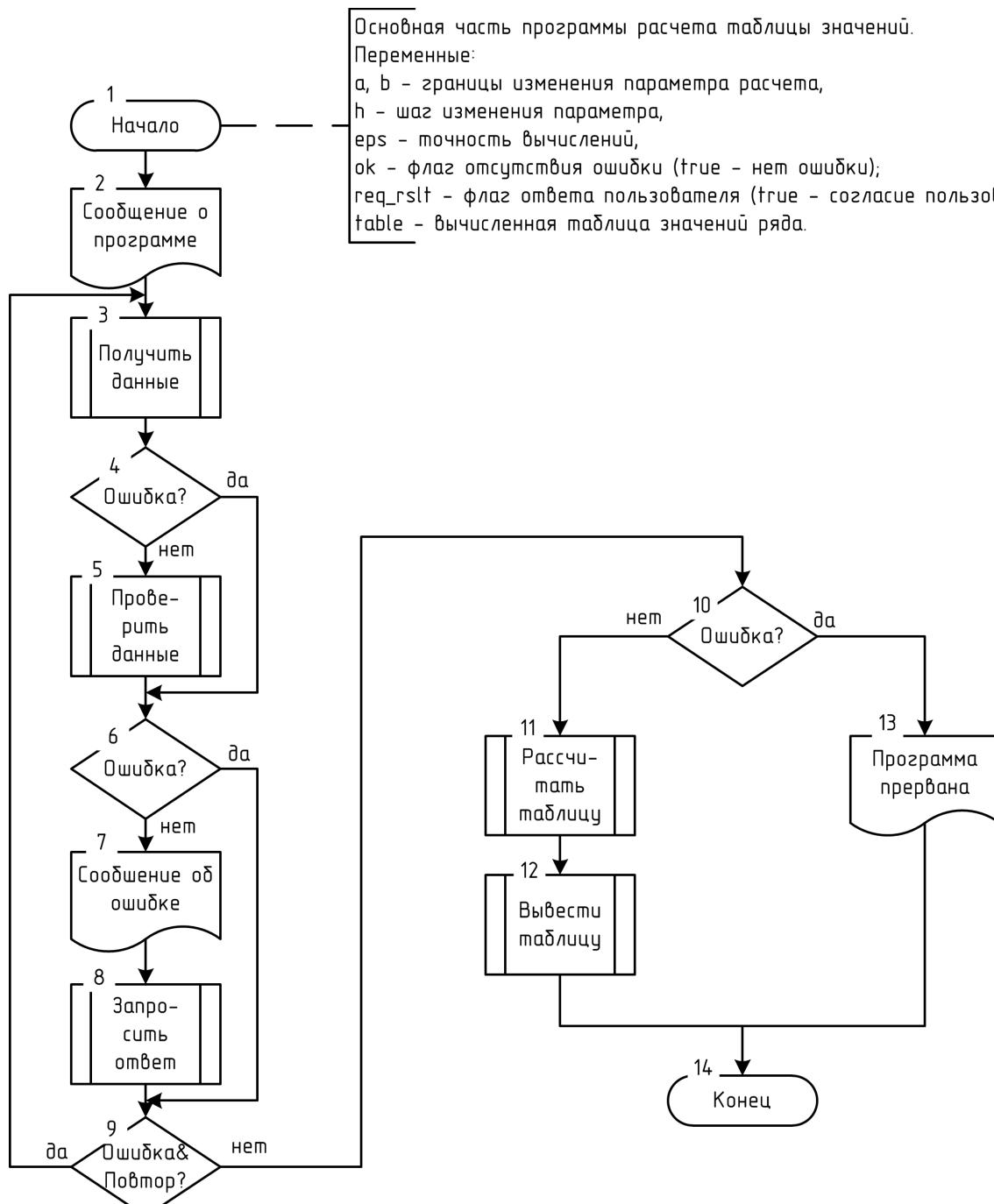


Рисунок 2.5 — Схема обобщенного алгоритма ввода параметров, расчета таблицы и её вывода

Так как таблица может выводится и на экран, и в файл, лучше вынести эту операцию в отдельную процедуру, схема которой представлена на рисунке 2.6.

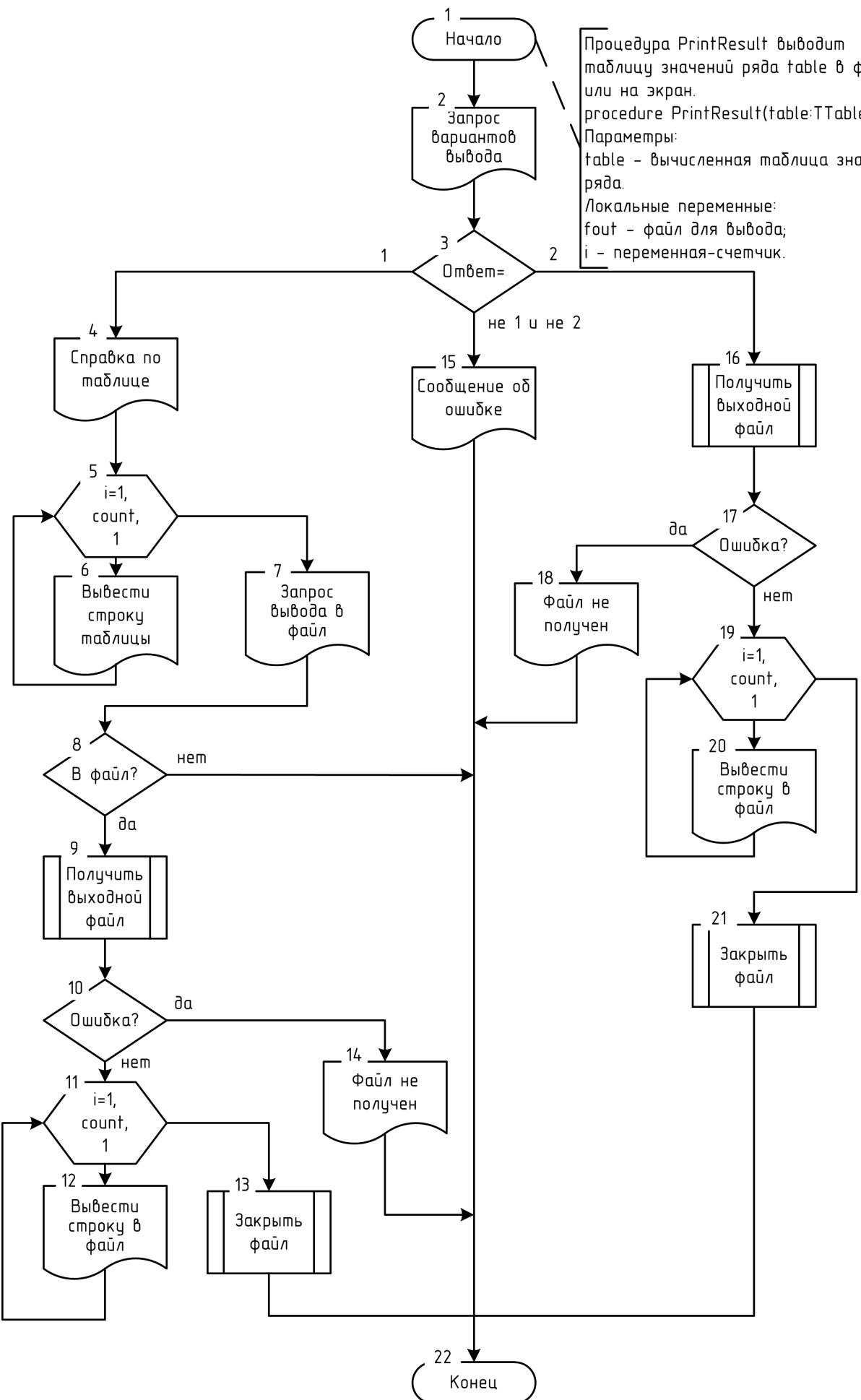


Рисунок 2.6 - Схема алгоритма вывода таблицы значений

Раз уж мы выводим таблицу в файл, неплохо было бы и предусмотреть и ввод параметров из файлов (а не только с клавиатуры), поэтому нужно выделить функцию выбора способа заполнения (представлена на рисунке 2.7) и ввода параметров из файла (можно увидеть на рисунке 2.8).

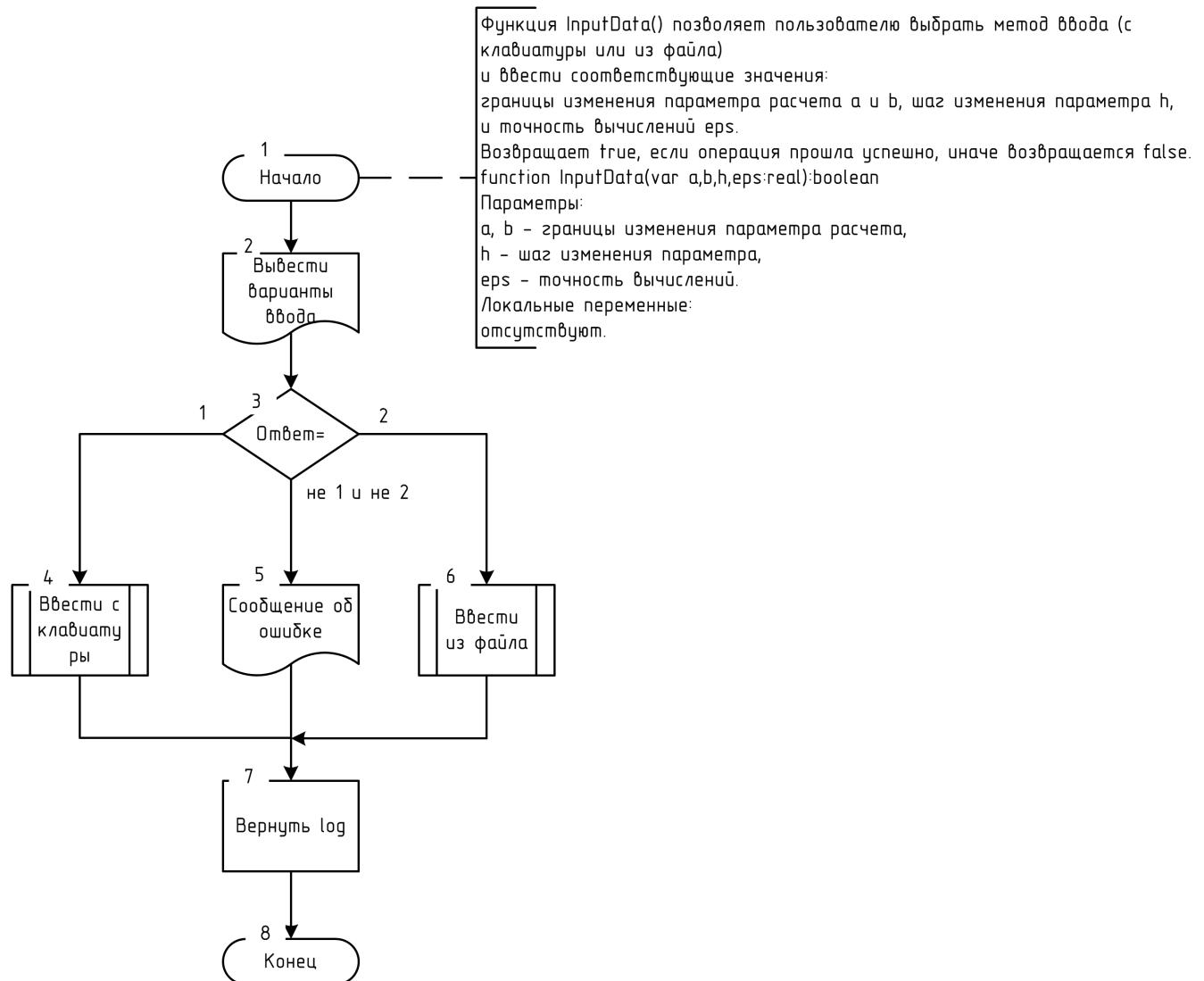


Рисунок 2.7 - Схема алгоритма получения данных различными способами

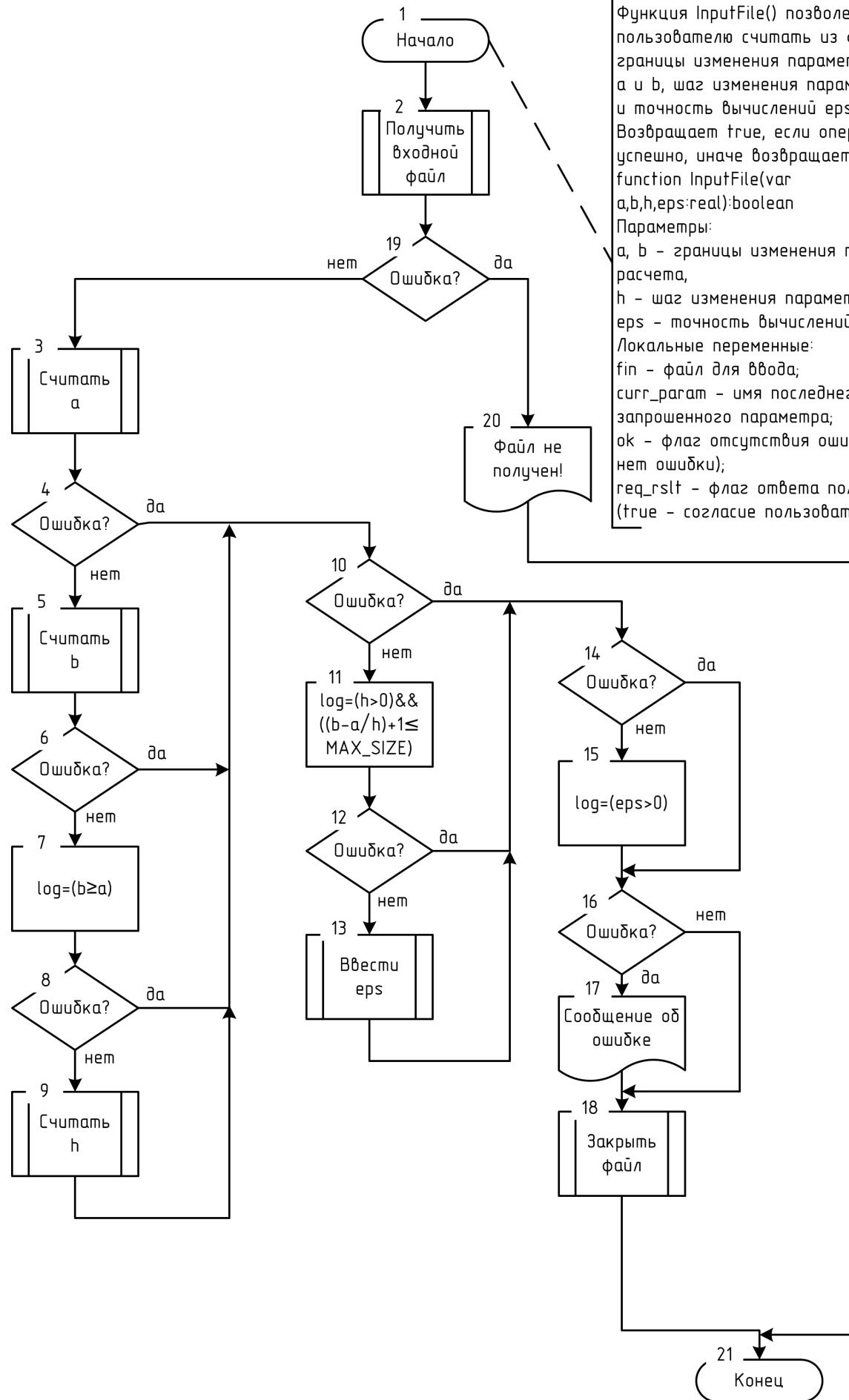


Рисунок 2.8 - Схема алгоритма считывания данных из файла

Исключительно из-за особенностей языка Паскаль получение числа из файла и проверки корректности его получения тоже неплохо вынести в отдельную функцию, которая бы возвращала, успешно или нет заверчилась эта операция получения значения. Схема подобного алгоритма представлена на рисунке 2.9.

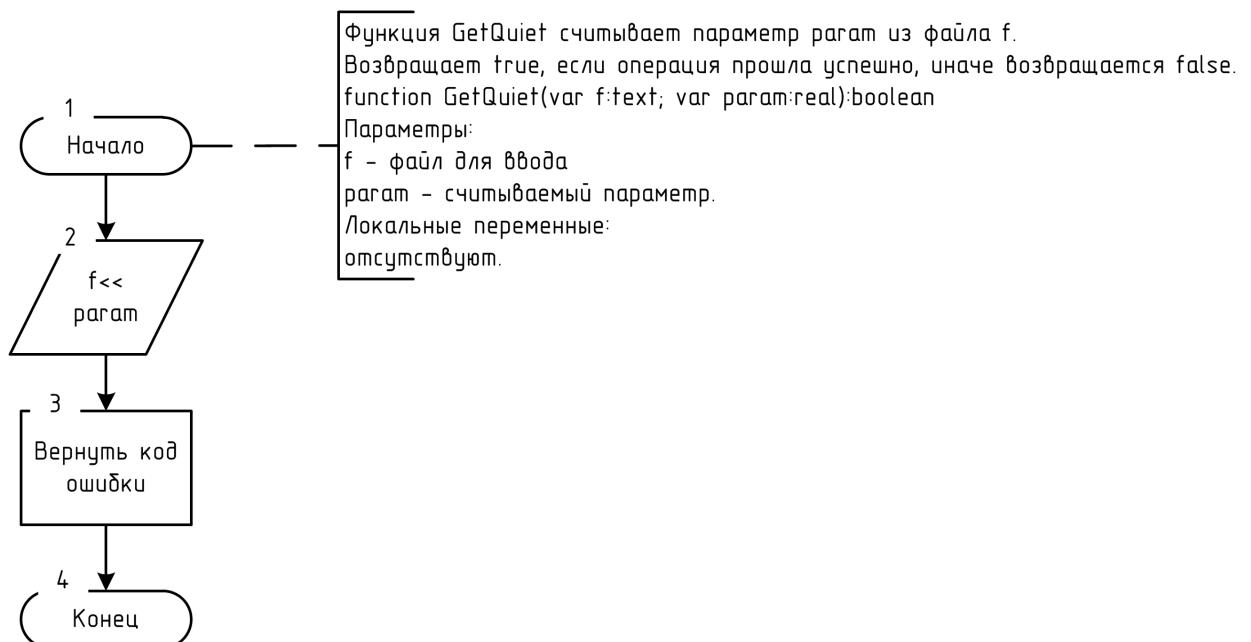


Рисунок 2.9 - Схема алгоритма считывания числа из файла

2.6. Текст программы

Ниже представлен текст программы вычисления таблицы значений функции (2.1). Программа написана на языке программирования Turbo Pascal 7.0.

```
program Func;
const MAX_COUNT=25;
type TTable=record
    count:word;
    line:array [1..MAX_COUNT] of record
        x,y:real; last:longint
    end;
end;
const h_min=1e-4;
eps_min=1e-4;

(* ****)
(*
Функция GetReqResult позволяет получить ответ "да" или "нет" от пользователя.
Возвращает true в случае согласия пользователя, false - в случае несогласия.
Параметры:
отсутствуют.
Локальные переменные:
answer - текущий ответ пользователя.
*)
function GetReqReslt:boolean; forward;

(*
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 (true - нет ошибки).
*)
function GetOption(a,b:char):char; forward;

(*
Функция GetInFile() получает файл f для чтения.
Возвращает true, если операция прошла успешно, иначе возвращается false.
Параметры:
f - запрашиваемый файл.
Локальные переменные:
error - флаг ошибки (true - ошибка);
req_rslt - флаг ответа пользователя (true - согласие пользователя).
*)
function GetInFile(var f:text):boolean; forward;

(*
Функция GetOutFile() получает файл f для записи.
Возвращает true, если операция прошла успешно, иначе возвращается false.
Параметры:
f - запрашиваемый файл.
Локальные переменные:
error - флаг ошибки (true - ошибка);
req_rslt - флаг ответа пользователя (true - согласие пользователя).
*)
function GetOutFile(var f:text):boolean; forward;
```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

(*
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 true, если операция прошла успешно, иначе возвращается false.
Параметры:
message - сообщение о параметре;
name - имя параметра;
param - запрашиваемый параметр.
Локальные переменные:
buf - временный строковый буфер;
code - код ошибки получения значения из буфера;
ok - флаг отсутствия ошибки (true - нет ошибки);
req_rslt - флаг ответа пользователя (true - согласие пользователя).
*)
function Input(const message:string; const name:string; var param:real):boolean;
forward;

(*
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 true, если операция прошла успешно, иначе возвращается false.
Параметры:
message - сообщение о параметре;
name - имя параметра;
index - запрашиваемый индекс;
h_bound - верхняя граница индекса.
Локальные переменные:
buf - временный строковый буфер;
code - код ошибки получения значения из буфера;
ok - флаг отсутствия ошибки (true - нет ошибки);
req_rslt - флаг ответа пользователя (true - согласие пользователя).
*)
function InputWithLBound(const message:string; const name:string; var param:real; const l_bound:real):boolean; forward;

(*
Функция InputWithHBound() получает от пользователя некоторое вещественное число param,
максимальное значение которого ограничено значением h_bound.
При этом она в процессе запроса выводит поясняющее сообщение message,
и приглашение ко вводу в виде имени параметра - name.
Возвращает true, если операция прошла успешно, иначе возвращается false.
Параметры:
message - сообщение о параметре;
name - имя параметра;
index - запрашиваемый индекс;
h_bound - верхняя граница индекса.
Локальные переменные:
buf - временный строковый буфер;
code - код ошибки получения значения из буфера;
ok - флаг отсутствия ошибки (true - нет ошибки);
req_rslt - флаг ответа пользователя (true - согласие пользователя).
*)
function InputWithHBound(const message:string; const name:string; var param:real; const h_bound:real):boolean; forward;

(*
Процедура GetSeries рассчитывает значение ряда для конкретного x с точностью eps,
и сохраняет это значение в переменной y,
а в переменной last запоминается номер последнего вычисленного члена ряда.
Параметры:

```

Изм.	Лист	№ докум.	Подп.

x - параметр расчета значения ряда,
 eps - точность вычислений,
 y - вычисленное значение ряда для указанного параметра,
 last - номер последнего вычисленного члена ряда.
 Локальные переменные:
 sqrx - квадрат x,
 ylast последний текущий вычисленный член ряда.
 *)

```

procedure GetSeries(const x,eps:real;var y:real;var last:longint); forward;
  
```

(*
 Функция CheckData() позволяет пользователю проверить введенные данные:
 границы параметра расчета a и b, шаг изменения параметра h,
 и точность вычислений eps.
 Возвращает true, если пользователь подтвердил правильность данных, иначе
 возвращается false.
 Параметры:
 a, b - границы изменения параметра расчета,
 h - шаг изменения параметра,
 eps - точность вычислений.
 Локальные переменные:
 отсутствуют.
 *)

```

function CheckData(const a,b,h,eps:real):boolean; forward;
  
```

(*
 Функция GetQuiet считывает параметр param из файла f.
 Возвращает true, если операция прошла успешно, иначе возвращается false.
 Параметры:
 f - файл для ввода
 param - считываемый параметр.
 Локальные переменные:
 отсутствуют.
 *)

```

function GetQuiet(var f:text; var param:real):boolean; forward;
  
```

(*
 Функция InputConsole() позволяет пользователю считать с клавиатуры
 границы изменения параметра расчета a и b, шаг изменения параметра h,
 и точность вычислений eps.
 Возвращает true, если операция прошла успешно, иначе возвращается false.
 Параметры:
 a, b - границы изменения параметра расчета,
 h - шаг изменения параметра,
 eps - точность вычислений.
 Локальные переменные:
 ok - флаг отсутствия ошибки (true - нет ошибки);
 req_rslt - флаг ответа пользователя (true - согласие пользователя).
 *)

```

function InputConsole(var a,b,h,eps:real):boolean; forward;
  
```

(*
 Функция InputFile() позволяет пользователю считать из файла
 границы изменения параметра расчета a и b, шаг изменения параметра h,
 и точность вычислений eps.
 Возвращает true, если операция прошла успешно, иначе возвращается false.
 Параметры:
 a, b - границы изменения параметра расчета,
 h - шаг изменения параметра,
 eps - точность вычислений.
 Локальные переменные:
 fin - файл для ввода;
 curr_param - имя последнего запрошенного параметра;
 ok - флаг отсутствия ошибки (true - нет ошибки);
 req_rslt - флаг ответа пользователя (true - согласие пользователя).

Изм.	Лист	№ докум.	Подп.

```

*)
function InputFile(var a,b,h,eps:real):boolean; forward;

(*
Функция InputData() позволяет пользователю выбрать метод ввода (с клавиатуры или
из файла)
и ввести соответствующие значения:
границы изменения параметра расчета a и b, шаг изменения параметра h,
и точность вычислений eps.
Возвращает true, если операция прошла успешно, иначе возвращается false.
Параметры:
a, b - границы изменения параметра расчета,
h - шаг изменения параметра,
eps - точность вычислений.
Локальные переменные:
отсутствуют.
*)
function InputData(var a,b,h,eps:real):boolean; forward;

(*
Процедура GetTable получает таблицу значений ряда table,
при изменении параметра расчета от a до b с шагом h,
с точностью вычислений eps.
Параметры:
a, b - границы изменения параметра расчета,
h - шаг изменения параметра,
eps - точность вычислений,
table - вычисленная таблица значений ряда.
Локальные переменные:
отсутствуют.
*)
procedure GetTable(a,b,h,eps:real;var table:TTable); forward;

(*
Процедура PrintResult выводит таблицу значений ряда table в файл или на экран.
Параметры:
table - вычисленная таблица значений ряда.
Локальные переменные:
fout - файл для вывода;
i - переменная-счетчик.
*)
procedure PrintResult(table:TTable); forward;
(******)
function GetReqReslt:boolean;
var answer:char;
begin
    ReadLn(answer);
    while (UpCase(answer)<>'Y')
        and(UpCase(answer)<>'N') do begin
        WriteLn('Неправильный ответ! Допустимо:',#13#10,'Y - да; N - нет.');
        ReadLn(answer);
    end;
    GetReqReslt:=UpCase(answer)='Y';
end;

function GetOption(a,b:char):char;
var ch:char; ok,req_rslt:boolean;
begin
    repeat
        WriteLn('Введите цифру от ',a,', до ',b,'.');
        Readln(ch);
        ok:=(a<=ch)and(ch<=b);
        if not ok then
            WriteLn('Неправильный ответ!');
    until((a<=ch)and(ch<=b));

```

```

GetOption:=ch;
end;

function FileExists(var f:text):boolean;
var exist:boolean;
begin
  {$I-}
    ReSet(f);
  {$I+}
  exist:=ioresult=0;
  if exist then close(f);
  FileExists:=exist;
end;
function GetInFile(var f:text):boolean;
var error,req_rslt:boolean;
  fileName:string;
begin
  error:=false; req_rslt:=false;
  repeat
    WriteLn('Введите имя файла-источника.');
    Write('Файл:'); ReadLn(fileName);
    Assign(f,fileName);
    {$I-}
      Reset(f);
    {$I+}
    error:=(ioresult<>0)or(fileName='');
    if error then begin
      WriteLn('Неверное имя файла! Повторить ввод? <Y>/<N>');
      req_rslt:=GetReqReslt;
    end;
  until not error or not req_rslt;
  GetInFile:=not error;
end;

function GetOutFile(var f:text):boolean;
var error,req_rslt:boolean;
  fileName:string;
begin
  error:=false; req_rslt:=false;
  repeat
    WriteLn('Введите имя файла-результата.');
    Write('Файл:'); ReadLn(fileName);
    Assign(f,fileName);
    if FileExists(f) then begin
      error:=true;
      WriteLn('Указанный файл существует! Перезаписать? (Y/N)');
      error:=not GetReqReslt;
    end;
    if not error then begin
      {$I-}
        ReWrite(f);
      {$I+}
      error:=(ioresult<>0)or(fileName '');
    end;
    if error then begin
      WriteLn('Ошибка при создании файла! Повторить ввод? <Y>/<N>');
      req_rslt:=GetReqReslt;
    end;
  until not error or not req_rslt;
  GetOutFile:=not error;
end;

procedure GetSeries(const x,eps:real;var y:real;var last:longint);
var sqrx,ylast:real;
begin

```

Изм.	Лист	№ докум.	Подп.	Дата

```

sqrX:=sqr(x); {запомним x^2}
y:=1; ylast:=1; last:=0; {инициализация}
repeat
    inc(last);
    ylast:=ylast*sqrX/(2*last)/(2*last-1); {очередной элемент
последовательности}
    y:=y+ylast;
until abs(ylast)<eps;
end;

function CheckData(const a,b,h,eps:real):boolean;
begin
    WriteLn('Вычисление значений функции');
    WriteLn('на интервале от ',a:1:4,' до ',b:1:4,' с шагом ',h:1:4,'. Точность -
',eps:1:4);
    WriteLn('Данные корректны? (Y/N)');
    CheckData:=GetReqReslt;
end;

function Input(const message:string; const name:string; var param:real):boolean;
var buf :string; ok,req_rslt:boolean; code:integer;
begin
    ok:=true; req_rslt:=false;
repeat
    ok:=True; {пока всё хорошо}
    WriteLn(message);
    Write(name,': ');
    readln(buf);
    val(buf,param,code);
    if code<>0 then begin
        ok:=false; {Ошибка}
        WriteLn('Некорректное значение! Повторить ввод?(Y/N)!');
        req_rslt:=GetReqReslt;
    end;
until ok or not req_rslt;
Input:=ok;
end;
function InputWithLBound(const message:string; const name:string; var
param:real;const l_bound:real):boolean;
var buf :string; ok,req_rslt:boolean; code:integer;
begin
    ok:=true; req_rslt:=false;
repeat
    ok:=True; {пока всё хорошо}
    WriteLn(message);
    Write(name,': ');
    readln(buf);
    val(buf,param,code);
    if code<>0 then begin
        ok:=false; {Ошибка}
        WriteLn('Некорректное значение! Повторить ввод?(Y/N)!');
        req_rslt:=GetReqReslt;
    end else
        if (param<l_bound) then begin
            ok:=false;
            WriteLn(name,'<',l_bound,'! Повторить ввод?(Y/N)');
            req_rslt:=GetReqReslt;
        end;
until ok or not req_rslt;
InputWithLBound:=ok;
end;
function InputWithHBound(const message:string; const name:string; var
param:real;const h_bound:real):boolean;
var buf :string; ok,req_rslt:boolean; code:integer;
begin

```

Изм.	Лист	№ докум.	Подп.	Дата

```

ok:=true; req_rslt:=false;
repeat
    ok:=True; {пока всё хорошо}
    WriteLn(message);
    Write(name, ': ');
    readln(buf);
    val(buf,param,code);
    if code<>0 then begin
        ok:=false; {Ошибка}
        WriteLn('Некорректное значение! Повторить ввод? (Y/N) ! ');
        req_rslt:=GetReqReslt;
    end else
        if (param>h_bound) then begin
            ok:=false;
            WriteLn(name, '>', h_bound, '! Повторить ввод? (Y/N) ');
            req_rslt:=GetReqReslt;
        end;
    until ok or not req_rslt;
    InputWithHBound:=ok;
end;

function GetQuiet(var f:text; var param:real):boolean;
begin
    {$I-}
    read(f,param);
    {$I+}

    GetQuiet:=(ioresult=0);
end;

function InputConsole(var a,b,h,eps:real):boolean;
var ok,req_rslt:boolean;
begin

    ok:=true;req_rslt:=false;
    repeat
        ok:=true;req_rslt:=false;
        ok:=Input('Введите нижнюю границу расчета (a) - вещественное число', 'a', a);
        if ok then
            ok:=InputWithLBound('Введите верхнюю границу расчета (b) - вещественное
число. b>=a.', 'b', b, a);
        if ok then
            ok:=InputWithLBound('Введите шаг изменения x (h) - вещественное число.
h>0.', 'h', h, h_min);
        if ok and (trunc(b-a)/h+1>MAX_COUNT) then begin
            ok:=false;
            WriteLn('Таблица слишком большая! Повторить ввод параметров?
Y/N');
            req_rslt:=GetReqReslt;
        end;
    until ok or not req_rslt;
    if ok then
        ok:=InputWithLBound('Введите точность вычислений (eps) - вещественное
число. eps>0.', 'eps', eps, eps_min);
    InputConsole:=ok;
end;
function InputFile(var a,b,h,eps:real):boolean;
var ok,req_rslt:boolean; fin:text; curr_param:string;
begin
    ok:=true; req_rslt:=false;
    ok:=GetInFile(fin);
    if ok then begin
        ok:=GetQuiet(fin,a); curr_param:='a';
        if ok then
            ok:=GetQuiet(fin,b); curr_param:='b';
    end;
end;

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

    if ok then
        ok:=b>=a;
    if ok then
        ok:=GetQuiet(fin,h); curr_param:='h';
    if ok then
        ok:=h>=0;
    if ok then
        ok:=GetQuiet(fin,eps); curr_param:='eps';
    if ok then
        ok:=eps>0;
    if not ok then
        WriteLn('Ошибка при чтении параметра ',curr_param,'!');
        Close(fin);
    end else
        WriteLn('Ошибка при открытии файла!');
    InputFile:=ok;
end;

function InputData(var a,b,h,eps:real):boolean;
begin
    WriteLn('Желаете считать параметры из файла?');
    WriteLn('1 - из файла, 2 - с клавиатуры, 0 - завершить программу');
    case GetOption('0','2') of
        '1':InputData:=InputFile(a,b,h,eps);
        '2':InputData:=InputConsole(a,b,h,eps);
    else begin
        writeln('Отказ от ввода!'); InputData:=false;
    end
    end;
end;

procedure GetTable(a,b,h,eps:real;var table:TTable);
begin
    table.count:=1;table.line[1].x:=a;
    with table.line[1] do
        GetSeries(x,eps,y,last);

    while table.line[table.count].x+h<=b do begin
        inc(table.count);
        table.line[table.count].x:=table.line[table.count-1].x+h; {следующий
шаг}
        with table.line[table.count] do
            GetSeries(x,eps,y,last);
    end;
end;

procedure PrintResult(table:TTable);
var fout:text; i:word;
begin
    WriteLn('Желаете просмотреть таблицу на экране?');
    WriteLn('1 - вывести на экран (в файл можно будет записать после
просмотра), ');
    WriteLn('2 - вывести в файл без вывода на экран, ');

    case GetOption('1','2') of
        '1':begin
            WriteLn('Справка:');
            WriteLn('n - порядковый номер вычисления');
            WriteLn('x - текущий аргумент функции f(x)');
            WriteLn('f(x) - значение функции');
            WriteLn('Nx - номер последнего вычисленного элемента ряда');
            writeln('n':4,'x':16,'f(x)':16,'Nx':4);
        end;
    end;
end;

```

```

        for i:=1 to table.count do
            with table.line[i] do
                writeln(i:4,x:16:4,y:16:4,last:4);
        WriteLn;
        WriteLn('Желаете выводить результаты в файл? Y/N');
        if GetReqReslt then begin
            if GetOutFile(fout) then begin
                writeln(fout,'n':4,'x':16,'f(x)':16,'Nx':4);
                for i:=1 to table.count do
                    with table.line[i] do
                        writeln(fout,i:4,x:16:4,y:16:4,last:4);
                close(fout);
            end else WriteLN('Файл не был открыт...');

            end;
        end;
        '2':begin
            if GetOutFile(fout) then begin
                writeln(fout,'n':4,'x':16,'f(x)':16,'Nx':4);
                for i:=1 to table.count do
                    with table.line[i] do
                        writeln(fout,i:4,x:16:4,y:16:4,last:4);
                close(fout);
            end else WriteLN('Файл не был открыт...');

            end;
        end;
    end;

var a,b,h,eps:real;
    ok,req_rslt:boolean; table:TTable;
BEGIN
    WriteLn('Программа вычисляет значения функции f(x) с точностью eps');
    WriteLn('где x изменяется от a до b с шагом h');
    WriteLn('Функция f(x):');
    WriteLn('      x^2      x^4          x^2k');
    writeln('f(x)=1 + --- + --- + ... + -----');
    writeln('      2!      4!          (2k)!');
    {начало ввода}
    repeat
        ok:=InputData(a,b,h,eps);
        if ok then
            ok:=CheckData(a,b,h,eps);
        if not ok then begin
            WriteLn('Введённые данные некорректны! Повторить ввод? (Y/N)');
            req_rslt:=GetReqReslt;
        end;
    until ok or not req_rslt;

    {конец ввода}
    {начало обработки}
    if ok then begin
        GetTable(a,b,h,eps,table);
        PrintResult(table);
    end else
        WriteLn('Программа завершена...');

    {конец обработки}
    WriteLn('Нажмите <Enter>...');

    ReadLn;
END.

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

2.7. Инструкция программисту

При создании программы вычисления таблицы значений были предприняты следующие действия.

Объявлена константа MAX_COUNT — максимальное количество строк в таблице значений; EPS_MIN и H_MIN — минимальная точность вычислений и минимальный шаг.

Объявлен тип-запись TTableLine — строка таблицы значений, описание его полей приведено в таблице 2.1.

Таблица 2.1 - Описание полей типа "строка таблицы"

имя	тип	предназначение
x,y	real;	аргумент и значения ряда
last	longint	номер последнего вычисленного члена ряда

Объявлен тип-запись TTable — таблица значений, описание его полей приведено в таблице 2.2

Таблица 2.2 - Описание полей типа "таблица значений"

имя	тип	предназначение
count	word;	количество строк в таблице
line	array [1..MAX_COUNT] of TTableLine	массив строк таблицы

В главной части программы описаны структуры данных, которые представлены в таблице 2.3.

Таблица 2.3 - Переменные программы вычисления таблицы значений

имя	тип	предназначение
a, b	real	границы изменения параметра расчета,
h	real	шаг изменения параметра,
eps	real	точность вычислений,
ok	boolean	флаг отсутствия ошибки (true – нет ошибки);
req_rslt	boolean	флаг ответа пользователя (true – согласие пользователя),
table	TTable	вычисленная таблица значений ряда.

Программа разбита на следующие подпрограммы:

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

Заголовок функции:

```
function GetReqReslt:boolean;
```

Функция запрашивает у пользователя символы 'у','Y','н' или 'N' до тех пор, пока он не введет какой-либо из них. Соответственно возвращается либо true, если символ 'Y', либо false, если 'N'.

Локальные переменные функции представлены в таблице 2.4.

Таблица 2.4 - Локальные переменные функции получения ответа

имя	тип	предназначение
answer	char	Ответ пользователя.

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

Заголовок функции:

```
function GetOption(a,b:char):char;
```

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 2.5, локальные переменные — в таблице 2.6.

Таблица 2.5 - Параметры функции получения варианта

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

Таблица 2.6 - Локальные переменные функции получения варианта

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

3. 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.

```
function Input(const message:string;const name:string; var param:real):boolean;
```

Параметры представлены в таблице 2.7:

Таблица 2.7 - Параметры функции вещественного числа

имя	тип	предназначение
message	string	сообщение о параметре
name	string	имя параметра
param	real	запрашиваемый параметр

Локальные переменные представлены в таблице 2.8:

Таблица 2.8 - Локальные переменные функции получения вещественного числа

имя	тип	предназначение
ok	boolean	флаг отсутствия ошибки (1 - нет ошибки);
req_rslt	boolean	флаг ответа пользователя (1 - согласие пользователя).
buf	string	временный строковый буфер;
code	integer	код ошибки получения значения из буфера;

4. 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 2.9:

Таблица 2.9 - Параметры функции получения вещественного числа с левой границей

имя	тип	предназначение
message	string	сообщение о параметре
name	string	имя параметра
param	real	запрашиваемый параметр
l_bound	real	нижняя граница параметра.

Локальные переменные представлены в таблице 2.10:

Таблица 2.10 - Локальные переменные функции получения вещественного числа с левой границей

имя	тип	предназначение
ok	boolean	флаг отсутствия ошибки (1 - нет ошибки);
req_rslt	boolean	флаг ответа пользователя (1 - согласие пользователя).
buf	string	временный строковый буфер;
code	integer	код ошибки получения значения из буфера;

5. Функция InputWithHBound() получает от пользователя некоторое вещественное число param, максимальное значение которого ограничено значением 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.

```
function InputWithHBound(const message:string;const name:string; var param:real;const h_bound:real):boolean;
```

Параметры представлены в таблице 2.11:

Таблица 2.11 - Параметры функции получения вещественного числа с правой границей

имя	тип	предназначение
message	string	сообщение о параметре
name	string	имя параметра
param	real	запрашиваемый параметр
h_bound	real	верхняя граница параметра

Локальные переменные представлены в таблице 2.12:

Таблица 2.12 - Локальные переменные функции получения вещественного числа с правой границей

имя	тип	предназначение
ok	boolean	флаг отсутствия ошибки (1 - нет ошибки);
req_rslt	boolean	флаг ответа пользователя (1 - согласие пользователя).
buf	string	временный строковый буфер;
code	integer	код ошибки получения значения из буфера;

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

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

```
function GetInFile(var f:text):boolean;
```

Параметры функции представлены в таблице 2.13:

Таблица 2.13 - Параметры функции получения файла для чтения

имя	тип	предназначение
f	text	запрашиваемый файл.

Локальные переменные функции представлены в таблице 2.14:

Таблица 2.14 - Локальные переменные функции получения файла для чтения

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

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

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

function GetOutFile(var f:text):boolean

Параметры функции представлены в таблице 2.15:

Таблица 2.15 - Параметры функции получения файла для записи

имя	тип	предназначение
f	text	запрашиваемый файл.

Локальные переменные функции представлены в таблице 2.5:

Таблица 2.16 - Локальные переменные функции получения файла для записи

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

8. Процедура GetSeries рассчитывает значение ряда для конкретного x с точностью eps, и сохраняет это значение в переменной y, а в переменной last запоминается номер последнего вычисленного члена ряда.

procedure GetSeries(const x,eps:real;var y:real;var last:longint);

Параметры представлены в таблице 2.17:

Таблица 2.17 - Параметры функции получения значения ряда

имя	тип	предназначение
x	real	параметр расчета значения ряда,
eps		точность вычислений,
y	real	вычисленное значение ряда для указанного параметра,
last	longint	номер последнего вычисленного члена ряда.

Локальные переменные представлены в таблице 2.18:

Таблица 2.18 - Локальные переменные функции получения значения ряда

Изм.	Лист	№ докум.	Подп.	Дата

имя	тип	предназначение
sqrх	real	квадрат x,
ylast	real	последний текущий вычисленный член ряда.

9. Функция CheckData() позволяет пользователю проверить введенные данные: границы параметра расчета a и b, шаг изменения параметра h, и точность вычислений eps.

Возвращает true, если пользователь подтвердил правильность данных, иначе возвращается false.

```
function CheckData(const a,b,h,eps:real):boolean;
```

Параметры представлены в таблице 2.19:

Таблица 2.19 - Параметры функции проверки данных

имя	тип	предназначение
a, b	real	границы изменения параметра расчета,
h	real	шаг изменения параметра,
eps	real	точность вычислений.

Локальные переменные:

отсутствуют.

10. Функция GetQuiet считывает параметр param из файла f.

Возвращает true, если операция прошла успешно, иначе возвращается false.

```
function GetQuiet(var f:text; var param:real):boolean;
```

Параметры представлены в таблице 2.20:

Таблица 2.20 - Параметры функции проверки данных

имя	тип	предназначение
f	text	файл для ввода
param	real	считываемый параметр.

Локальные переменные:

отсутствуют.

11. Функция InputConsole() позволяет пользователю считать с клавиатуры границы изменения параметра расчета a и b, шаг изменения параметра h, и точность вычислений eps.

Возвращает true, если операция прошла успешно, иначе возвращается false.

```
function InputConsole(var a,b,h,eps:real):boolean;
```

Параметры представлены в таблице 2.21:

Таблица 2.21 - Параметры функции ввода с клавиатуры

имя	тип	предназначение
a, b	real	границы изменения параметра расчета,
h	real	шаг изменения параметра,
eps	real	точность вычислений.

Локальные переменные представлены в таблице 2.22:

Таблица 2.22 - Локальные переменные функции ввода с клавиатуры

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

12. Функция InputFile() позволяет пользователю считать из файла границы изменения параметра расчета a и b, шаг изменения параметра h, и точность вычислений eps.

Возвращает true, если операция прошла успешно, иначе возвращается false.

```
function InputFile(var a,b,h,eps:real):boolean;
```

Параметры представлены в таблице 2.23:

Таблица 2.23 - Параметры функции ввода из файла

имя	тип	предназначение
a, b	real	границы изменения параметра расчета,
h	real	шаг изменения параметра,
eps	real	точность вычислений.

Локальные переменные представлены в таблице 2.24:

Таблица 2.24 - Локальные переменные функции ввода из файла

имя	тип	предназначение
fin	text	файл для ввода;

curr_param	string	имя последнего запрошенного параметра;
ok	boolean	флаг отсутствия ошибки (true - нет ошибки);
req_rslt	boolean	флаг ответа пользователя (true - согласие пользователя).

13. Функция InputData() позволяет пользователю выбрать метод ввода (с клавиатуры или из файла) и ввести соответствующие значения: границы изменения параметра расчета a и b, шаг изменения параметра h, и точность вычислений eps.

Возвращает true, если операция прошла успешно, иначе возвращается false.

```
function InputData(var a,b,h,eps:real):boolean;
```

Параметры представлены в таблице 2.25:

Таблица 2.25 - Параметры функции получения данных

имя	тип	предназначение
a, b	real	границы изменения параметра расчета,
h	real	шаг изменения параметра,
eps	real	точность вычислений.

Локальные переменные:

отсутствуют.

14. Процедура GetTable получает таблицу значений ряда table, при изменении параметра расчета от a до b с шагом h, с точностью вычислений eps.

```
procedure GetTable(a,b,h,eps:real;var table:TTable);
```

Параметры представлены в таблице 2.26:

Таблица 2.26 - Параметры функции получения таблицы значений

имя	тип	предназначение
a, b	real	границы изменения параметра расчета,
h	real	шаг изменения параметра,
eps	real	точность вычислений,
table	TTable	вычисленная таблица значений ряда.

Локальные переменные:

отсутствуют.

15. Процедура PrintResult выводит таблицу значений ряда table в файл или на экран.

```
procedure PrintResult(table:TTable);
```

Параметры представлены в таблице 2.27:

Таблица 2.27 - Параметры функции вывода таблицы значений

имя	тип	предназначение
table	TTable	вычисленная таблица значений ряда

Локальные переменные представлены в таблице 2.28:

Таблица 2.28 - Локальные переменные функции вывода таблицы значений

имя	тип	предназначение
fout	text	файл для вывода;
i	word	переменная-счетчик.

2.8. Инструкция пользователю

Программа вычисляет с заданной точностью таблицу значений функции (2.1) от аргумента, изменяющегося с заданным шагом на заданном интервале. Можно вводить данные с клавиатуры и из файла. При вводе с клавиатуры необходимо ввести границы изменения аргумента — сначала нижнюю, затем верхнюю, причем верхняя должна быть больше или равна нижней. Затем программе передается шаг изменения аргумента, который больше или равен 0.0001, и точность вычисления значений, которая также больше 0.0001. Все параметры вещественные числа. Отказаться от ввода и завершить программу можно сочетанием клавиш Control-C. При вводе из файла значения в файле должны быть разделены пробелами или табуляциями. Если верхняя граница равна нижней или шаг равен 0, то будет вычислено лишь одно значение функции от аргумента, равного нижней границе интервала его изменения. Иначе будет создана таблица значений для аргументов из указанного промежутка, изменяющихся с указанным шагом. Таблицу можно выводить в файл, а можно сначала проверить её на экране, а только затем выводить в файл. Для вывода в файл нужно передать программе имя файла. Если такой файл существует,

программа запросит подтверждение перезаписи. Информация, содержащаяся в это время в файле, после перезаписи будет уничтожена.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Вариант №4

Лист

73

2.9. Тестовый пример

Ниже на рисунках 2.10, 2.11 и 2.12 показан пример работы программы вычисления таблицы значений функции (2.1) для $x=2$ и точностью 0.01.

На рисунке 2.10 можно увидеть приветственное сообщение.

```
Программа вычисляет значения функции f(x) с точностью eps
где x изменяется от a до b с шагом h
Функция f(x):
      x^2      x^4      x^2k
f(x)=1 + --- + --- + ... + -----
      2!      4!      (2k)!

Желаете считать параметры из файла?
1 - из файла, 2 - с клавиатуры, 0 - завершить программу
Ведите цифру от 0 до 2.
2

Вычисление значений функции
на интервале от -5.0000 до 5.0000 с шагом 0.5000. Точность - 0.0001
Данные корректны? (Y/N)
Y
Желаете просмотреть таблицу на экране?
1 - вывести на экран (в файл можно будет записать после просмотра),
2 - вывести в файл без вывода на экран,
Ведите цифру от 1 до 2.
2

Ведите имя файла-результата.
Файл:table.txt
Нажмите <Enter>...
```

Рисунок 2.10 — Приветственное сообщение программы расчета таблицы значений

```
Введите нижнюю границу расчета (a) - вещественное число
a: 2
Введите верхнюю границу расчета (b) - вещественное число. b>=2.0000.
b: 2
Введите шаг изменения x (h) - вещественное число. h>=0.0010.
h: 1
Введите точность вычислений (eps) - вещественное число. eps>=0.0010.
eps: 0.01
Вычисление значений функции
на интервале от 2.0000 до 2.0000 с шагом 1.0000. Точность - 0.0100
Данные корректны? (Y/N)
Y
Желаете просмотреть таблицу на экране?
1 - вывести на экран (в файл можно будет записать после просмотра),
2 - вывести в файл без вывода на экран,
Ведите цифру от 1 до 2.
```

Рисунок 2.11 — Ввод данных в программу расчета таблицы значений

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Справка :
 n – порядковый номер вычисления
 x – текущий аргумент функции $f(x)$
 $f(x)$ – значение функции
 Nx – номер последнего вычисленного элемента ряда

n	x	$f(x)$	Nx
1	2.0000	3.7619	4

 Желаете выводить результаты в файл? Y/N
 n
 Нажмите <Enter>...

Рисунок 2.12 — Результат работы программы для $x=2$ и точностью 0.01
Проверим этот результат, рассчитав значение функции вручную:

- Первый элемент последовательности $a_1 = \frac{4}{2} = 2$, $2 > 0.01$, $f(x) = 3$, значит, вычисляем следующий элемент ряда...
- $a_2 = \frac{16}{2 \cdot 3 \cdot 4} = 0.666667$, $f(x) = 3.666667$, $0.666667 > 0.01$, продолжаем вычисления...
- $a_3 = \frac{64}{24 \cdot 5 \cdot 6} = 0.088889$, $f(x) = 3.755556$, $0.088889 > 0.01$, продолжаем вычисления...
- $a_4 = \frac{256}{720 \cdot 7 \cdot 8} = 0.00634$, $f(x) = 3.761896$, $0.00634 < 0.01$, текущий элемент меньше заданной точности, значит, вычисление закончено.

Итак, искомое значение было вычислено машиной с точностью до второго знака после запятой, что даже несколько более точно, чем требовалось (достаточно было до первого). Программа работает правильно.

Более полно возможности программы раскрываются при работе со значениями аргумента из некоторого заданного промежутка. Программа позволяет создавать таблицы значений так, как показано на рисунке 2.13.

```

Программа вычисляет значения функции f(x) с точностью eps
где x изменяется от a до b с шагом h
Функция f(x):

$$f(x)=1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \dots + \frac{x^{2k}}{(2k)!}$$

Желаете считать параметры из файла?
1 - из файла, 2 - с клавиатуры, 0 - завершить программу
Введите цифру от 0 до 2.
2
Ведите нижнюю границу расчета (a) - вещественное число
a: -5
Ведите верхнюю границу расчета(b) - вещественное число. b>=-5.0000.
b: 5
Ведите шаг изменения x (h) - вещественное число. h>=0.0001.
h: 0.5
Ведите точность вычислений (eps) - вещественное число. eps>=0.0001.
eps: 0.0001_

```

Рисунок 2.11 — Результат запуска программы для x из промежутка [-5;5] с шагом 0.5 и точностью 0.0001

Ниже приводится содержание table.txt (границы строк и столбцов выделены для удобства чтения и в исходном файле отсутствуют).

n	x	f(x)	Nx
1	-5.0000	74.2099	10
2	-4.5000	45.0141	9
3	-4.0000	27.3082	9
4	-3.5000	16.5728	8
5	-3.0000	10.0677	7
6	-2.5000	6.1323	7
7	-2.0000	3.7622	6
8	-1.5000	2.3524	5
9	-1.0000	1.5431	4
10	-0.5000	1.1276	3
11	0.0000	1.0000	1
12	0.5000	1.1276	3
13	1.0000	1.5431	4
14	1.5000	2.3524	5
15	2.0000	3.7622	6
16	2.5000	6.1323	7
17	3.0000	10.0677	7
18	3.5000	16.5728	8

19	4.0000	27.3082	9
20	4.5000	45.0141	9
21	5.0000	74.2099	10

3. Вычисление средних арифметических значений положительных элементов каждой строки матрицы

3.1. Постановка задачи

Упорядочить по возрастанию элементы главной диагонали матрицы $A(L,L)$, $L \leq 75$.

3.2. Математическая формулировка задачи

Математическая формулировка задачи затруднительна.

3.3. Метод Хоара: быстрая сортировка

В методе быстрой сортировки фиксируется какой-либо ключ (базовый), относительно которого все элементы с большим весом перемещаются вправо, а с меньшим — влево. При этом весь список элементов делится относительно базового ключа на две части. Для каждой части процесс повторяется. Поясним метод на примере.

На рис. 3.1 представлены этапы быстрой сортировки Хоара. В первой строке указана исходная последовательность.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Номер шага	$i \rightarrow$								Примечание
	40	11	83	57	32	21	75	64	
1	40							64	Исходный список
2	40						75		$k_0 < k_j$
3	40				21	40			Обмен; $k_0 > k_j$
4		11			40				$k_i < k_0$
5			83	40		40	83		Обмен; $k_i > k_0$
6			40	32		40			Обмен; $k_0 > k_j$
7			57	40		57			Обмен; $k_i > k_0$
	21	11	32	40	57	83	75	64	Полученный список

Рисунок 3.1 — Пример работы метода Хоара

Примем первый элемент последовательности за базовый ключ, выделим его квадратом и обозначим $k_0 = 40$. Установим два указателя i и j , из которых i начинает отсчет слева ($i = 1$), а j — справа ($j = n$).

Сравниваем базовый ключ k_0 и текущий ключ k_j . Если $k_0 \leq k_j$, то устанавливаем $j = j - 1$ и проводим следующее сравнение k_0 и k_j . Продолжаем уменьшать j до тех пор, пока не достигнем условия $k_0 > k_j$. После этого меняем местами ключи k_0 и k_j (шаг 3 на рис. 5.11).

Теперь начинаем изменять индекс $i = i + 1$ и сравнивать элементы k_i и k_0 . Продолжаем увеличение i до тех пор, пока не получим условие $k_i > k_0$, после чего следует обмен k_i и k_0 (см. шаг 5). Снова возвращаемся к индексу j , уменьшаем его. Чередуя уменьшение j и увеличение i , продолжаем этот процесс с обоих концов к середине до тех пор, пока не получим $i = j$ (см. шаг 7).

Указанная процедура сортировки применяется независимо к левой и правой частям. Сложность метода Хоара: $O(n \log_2 n)$.

Анализ быстрой сортировки. Чтобы изучить эффективность быстрой сортировки, нужно сначала исследовать поведение процесса разделения. После

выбора разделяющего значения x просматривается весь массив. Поэтому выполняется в точности n сравнений. Число обменов может быть определено с помощью следующего вероятностного рассуждения.

Если положение разделяющего значения фиксировано и соответствующее значение индекса равно i , то среднее число операций обмена равно числу элементов в левой части сегмента, а именно i , умноженному на вероятность того, что элемент попал на свое место посредством обмена. Обмен произошел, если элемент принадлежал правой части; вероятность этого равна $(n-i)/n$.

Если нам сильно везет и в качестве границы всегда удается выбрать медиану, то каждый процесс разделения разбивает массив пополам, и число необходимых для сортировки проходов равно $\log(n)$. Тогда полное число сравнений равно $n * \log(n)$, а полное число обменов - $n * \log(n)/6$.

Разумеется, нельзя ожидать, что с выбором медианы всегда будет так везти, ведь вероятность этого всего лишь $1/n$. Но удивительно то, что средняя эффективность алгоритма Quicksort хуже оптимального случая только на множитель $2 * \ln(2)$, если разделяющее значение выбирается в массиве случайно.

Однако и у алгоритма Quicksort есть свои подводные камни. Прежде всего при малых n его производительность не более чем удовлетворительна, как и для всех эффективных методов. Но его преимущество над другими эффективными методами заключается в легкости подключения какого-нибудь простого метода для обработки коротких сегментов. Это особенно важно для рекурсивной версии алгоритма.

Однако еще остается проблема наихудшего случая. Как поведет себя Quicksort тогда? Увы, ответ неутешителен, и здесь выявляется главная слабость этого алгоритма. Например, рассмотрим неудачный случай, когда каждый раз в качестве разделяющего значения x выбирается наибольшее значение в разделяемом сегменте. Тогда каждый шаг разбивает сегмент из n элементов на левую часть из $n-1$ элементов и правую часть из единственного элемента. Как следствие нужно сделать n разделений вместо $\log(n)$, и поведение в худшем случае оказывается порядка n^2 .

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Очевидно, что ключевым шагом здесь является выбор разделяющего значения x . В приведенном варианте алгоритма на эту роль выбирается средний элемент. Но с равным успехом можно выбрать первый или последний элемент. В этих случаях наихудший вариант поведения будет иметь место для изначально упорядоченного массива; то есть алгоритм Quicksort явно «не любит» легкие задачки и предпочитает беспорядочные наборы значений. При выборе среднего элемента это странное свойство алгоритма Quicksort не так очевидно, так как изначально упорядоченный массив оказывается наилучшим случаем. На самом деле если выбирается средний элемент, то и производительность в среднем оказывается немного лучшей. Хоор предложил выбирать x случайным образом или брать медиану небольшой выборки из, скажем, трех ключей. Такая предосторожность вряд ли ухудшит среднюю производительность алгоритма, но она сильно улучшает его поведение в наихудшем случае. Во всяком случае, ясно, что сортировка с помощью алгоритма Quicksort немного похожа на тотализатор, и пользователь должен четко понимать, какой проигрыш он может себе позволить, если удача от него отвернется.

Отсюда можно извлечь важный урок для программиста. Каковы последствия поведения алгоритма Quicksort в наихудшем случае, указанном выше? Мы уже знаем, что в такой ситуации каждое разделение дает правый сегмент, состоящий из единственного элемента, и запрос на сортировку этого сегмента сохраняется на стеке для выполнения в будущем. Следовательно, максимальное число таких запросов и, следовательно, необходимый размер стека равны n . Конечно, это совершенно неприемлемо. (Заметим, что дело обстоит еще хуже в рекурсивной версии, так как вычислительная система, допускающая рекурсивные вызовы процедур, должна автоматически сохранять значения локальных переменных и параметров всех активаций процедур, и для этого будет использоваться скрытый стек.) Выход здесь в том, чтобы сохранять на стеке запрос на обработку более длинной части, а к обработке короткой части приступать немедленно. Тогда размер стека M можно ограничить величиной $\log(n)$.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

3.4. Разработка структур данных

Совершено очевидно, что программе понадобится структура для хранения матрицы:

array — двумерный массив, который используется для хранения квадратной матрицы.

size — размер матрицы;

Также нам необходим будет доступ к элементам матрицы. Для доступа к элементам матрицы будем использовать переменные-счетчики *i* и *j*.

Файлы для ввода и вывода информации будем называть *fin* и *fout* соответственно.

Для оценки правильности выполнения программы введем различные флаги — пусть *ok* и *log* свидетельствуют о правильности работы при значениях 1, *error* символизирует ошибку при значении 1, а *req_rslt* хранит ответ пользователя на различные запросы (на которые предполагается ответ или да, или нет).

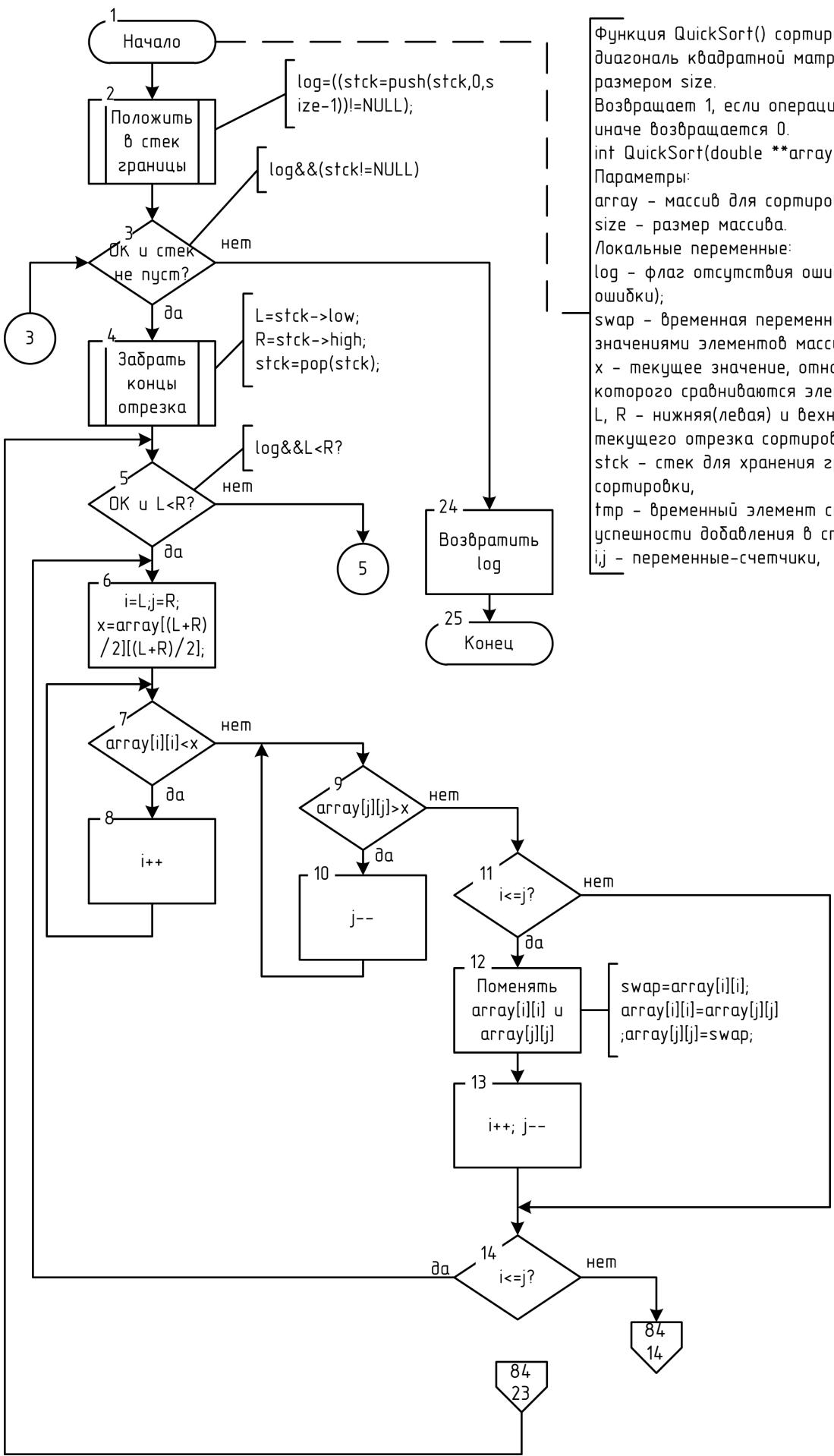
Так как стеком в процедуре сортировки будет ручное управление, нужен список, в котором будут храниться данные. Пусть элемент этого списка состоит из полей *high* и *low* — верхней и нижней границы отрезка сортировки, и указателя на предыдущий элемент стека *prev*.

Границы текущего отрезка будем обозначать *L* и *R*, верхняя и нижняя границы соответственно.

3.5. Разработка структуры алгоритма решения задачи

На рисунках 3.2 и 3.3 представлена схема алгоритма сортировки главной диагонали матрицы по методу Хоора.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------



Функция QuickSort() сортирует главную диагональ квадратной матрицы array размером size.
Возвращаем 1, если операция прошла успешно, иначе возвращается 0.
int QuickSort(double **array, long size)
Параметры:
array - массив для сортировки;
size - размер массива.
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
swap - временная переменная для обмена значениями элементов массива;
x - текущее значение, относительно которого сравниваются элементы;
L, R - нижняя(левая) и верхняя(правая) границы текущего отрезка сортировки;
stck - стек для хранения границ отрезков сортировки;
tmp - временный элемент стека для проверки успешности добавления в стек;
ij - переменные-счетчики,

Рисунок 3.2 - Схема алгоритма сортировки главной диагонали матрицы (начало)

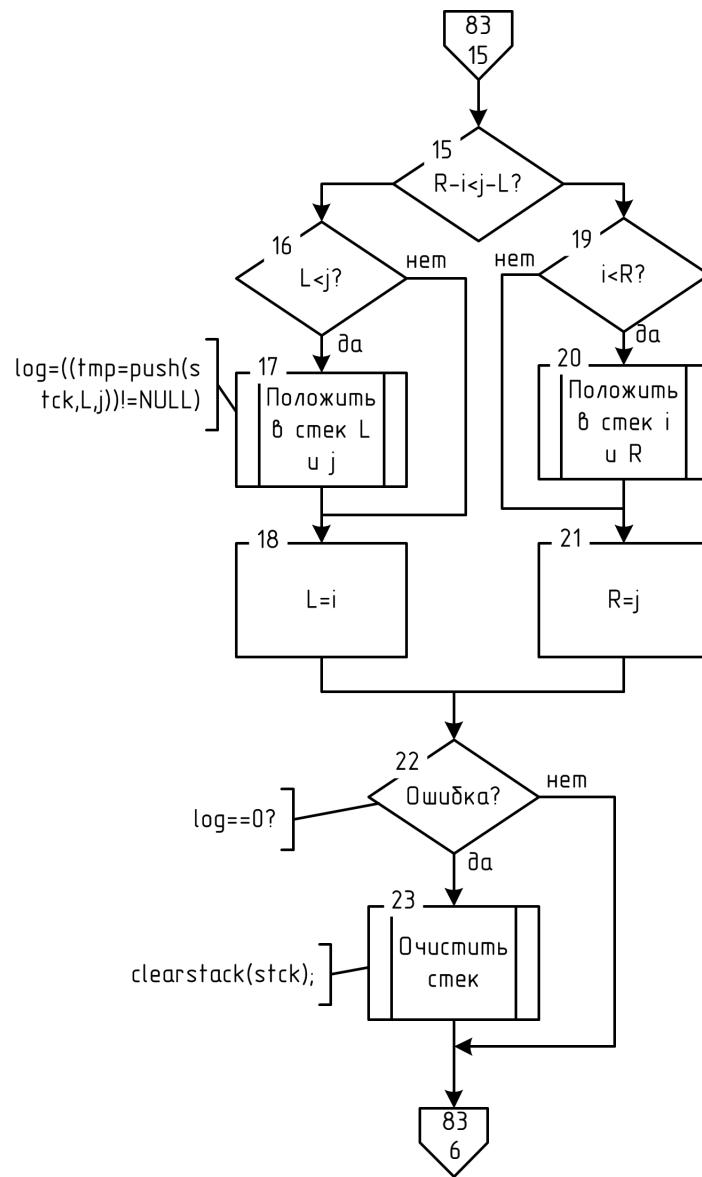


Рисунок 3.3 - Схема алгоритма сортировки главной диагонали матрицы (продолжение)

Схему общего алгоритма (ввод данных в матрицу её проверка, сортировка диагонали и вывод) сортировки главной диагонали можно увидеть на рисунке 3.4.

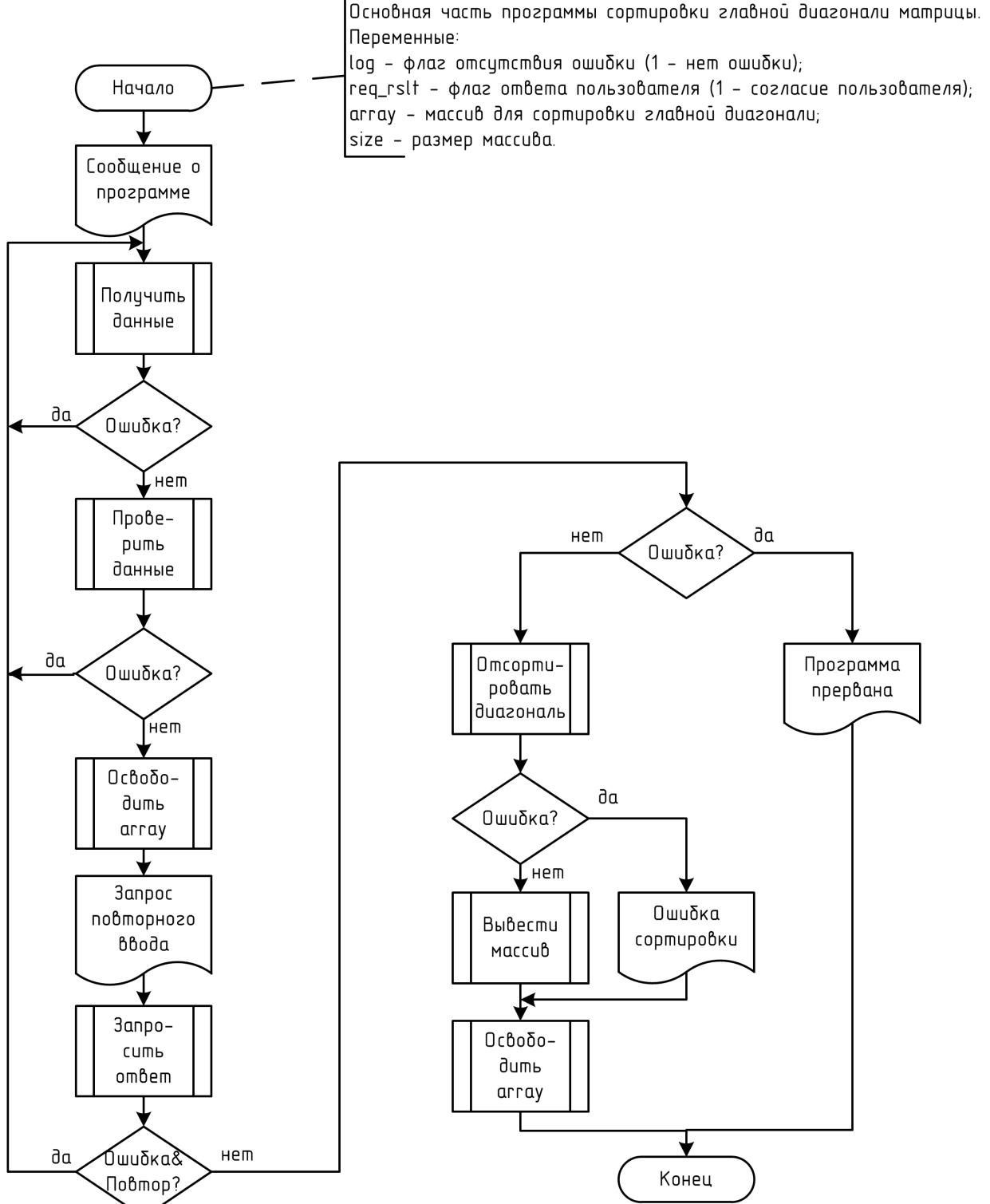


Рисунок 3.4 - Схема обобщенного алгоритма заполнения, сортировки и вывода матрицы

Так как память под матрицу выделяется динамически, то соответственно нужны процедуры выделения памяти и освобождения памяти. Блок-схемы данных алгоритмов представлены на рисунках 3.5 и 3.6.

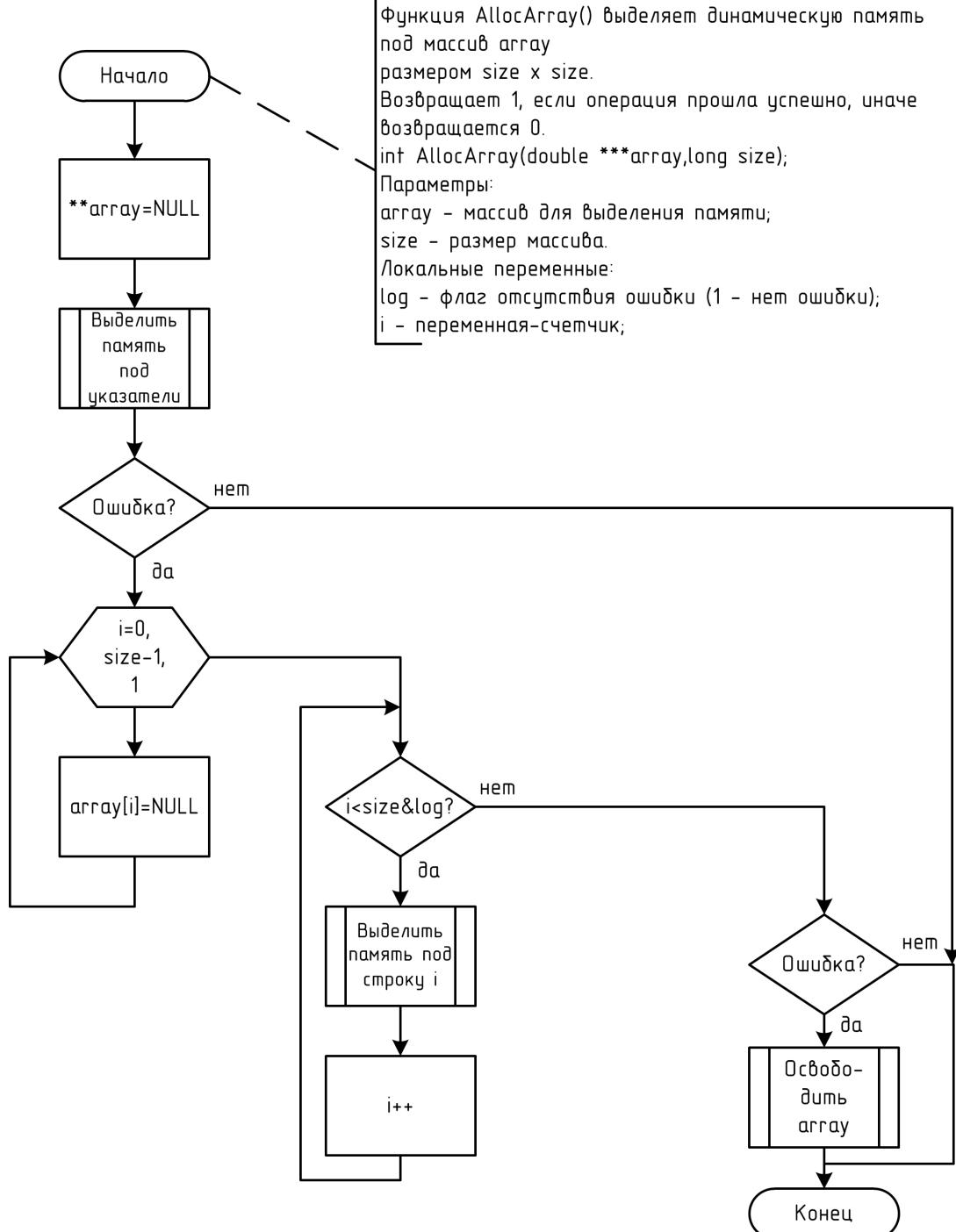
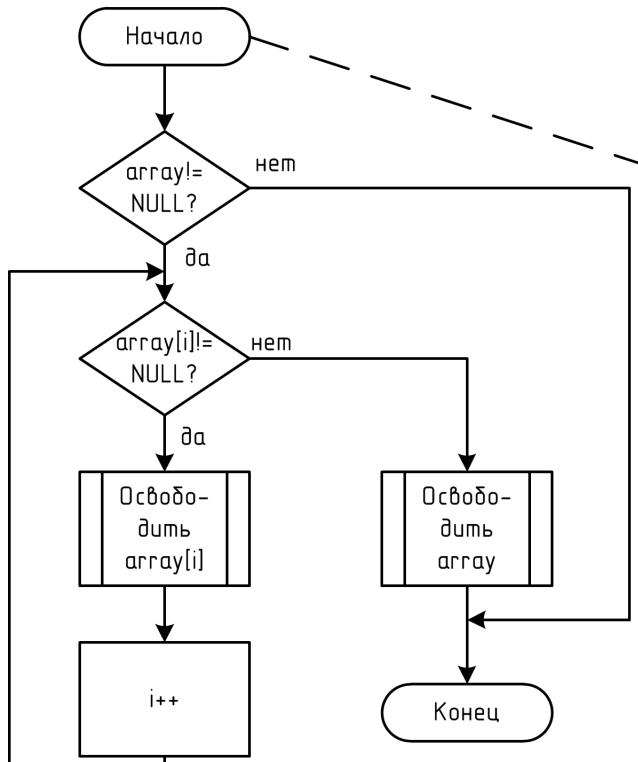


Рисунок 3.5 - Схема алгоритма выделения памяти под матрицу



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

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

Параметры:
`array` — массив для освобождения памяти;
`size` — размер массива.

Локальные переменные:
`i` — переменная-счетчик;

Рисунок 3.6 - Схема алгоритма освобождения памяти под матрицу

На рисунках 3.7, 3.8-3.9 и 3.10 представлены схемы алгоритмов заполнения матрицы различными способами: на рисунке 3.7 — с клавиатуры, рисунках 3.8 и 3.9 — из файла, и на рисунке 3.10 — случайными числами. Естественно, при таком количестве различных способов алгоритм выбора лучше поместить в отдельную подпрограмму; его схему можно увидеть на рисунке 3.11.

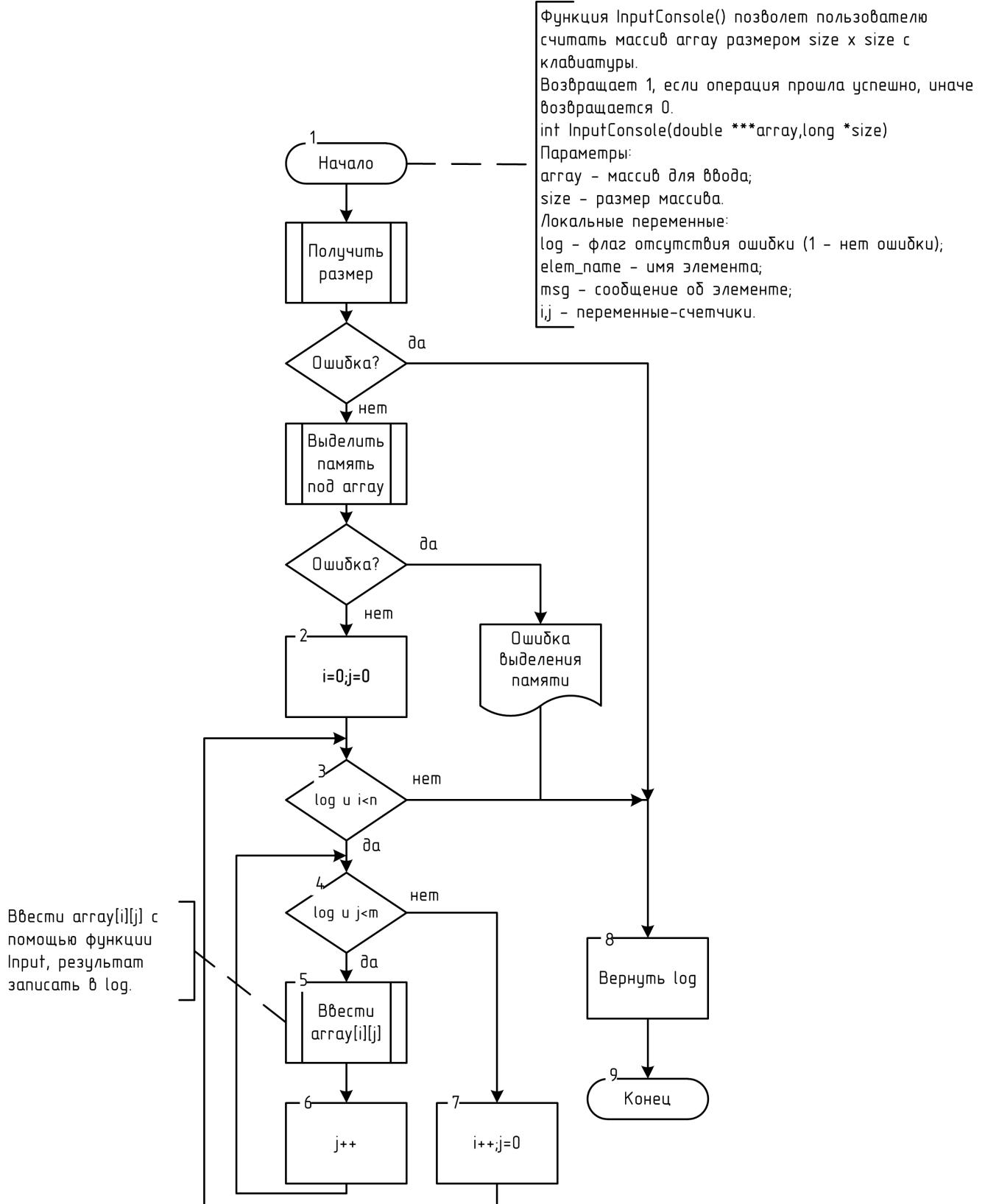
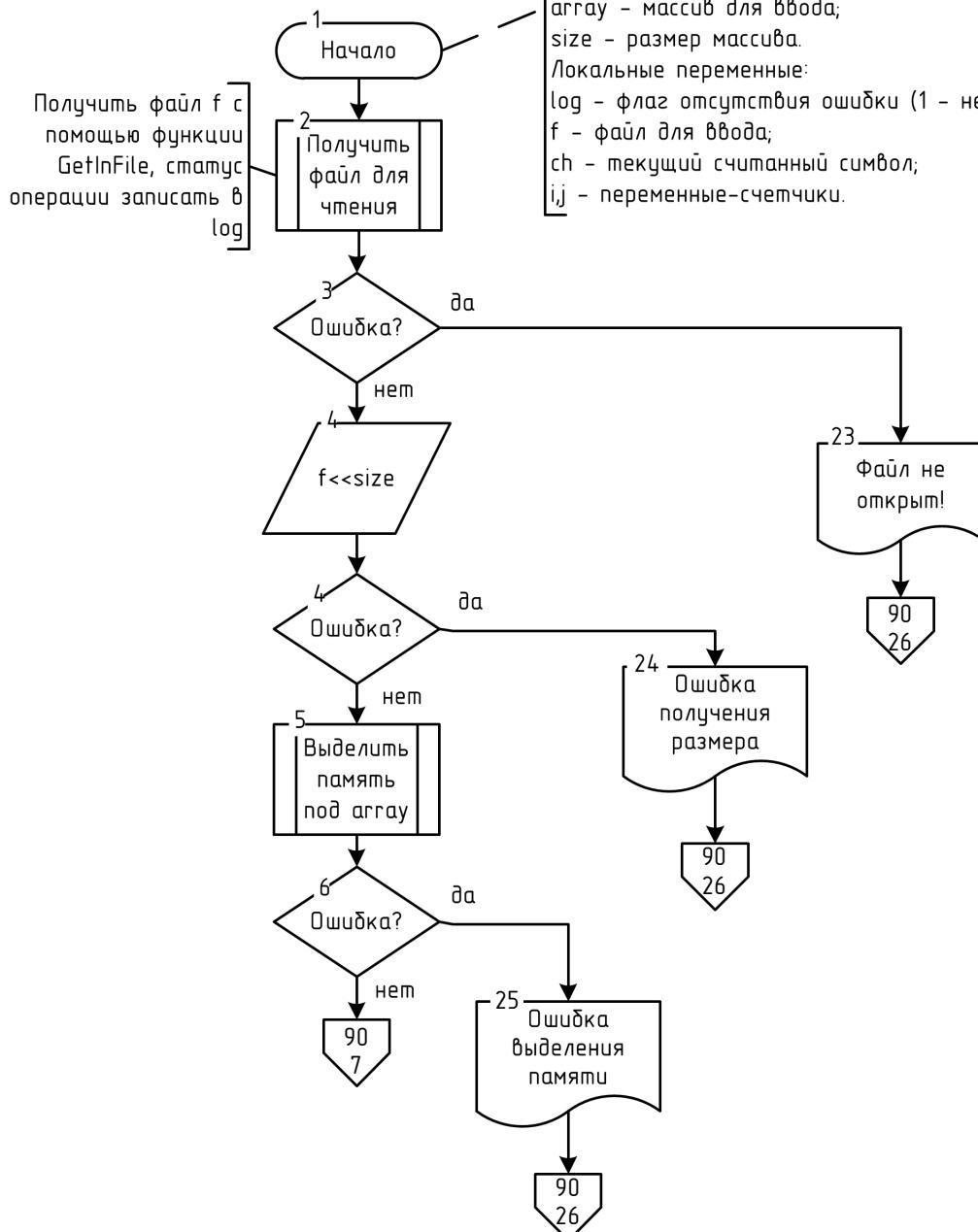


Рисунок 3.7 - Схема алгоритма заполнения матрицы с клавиатуры

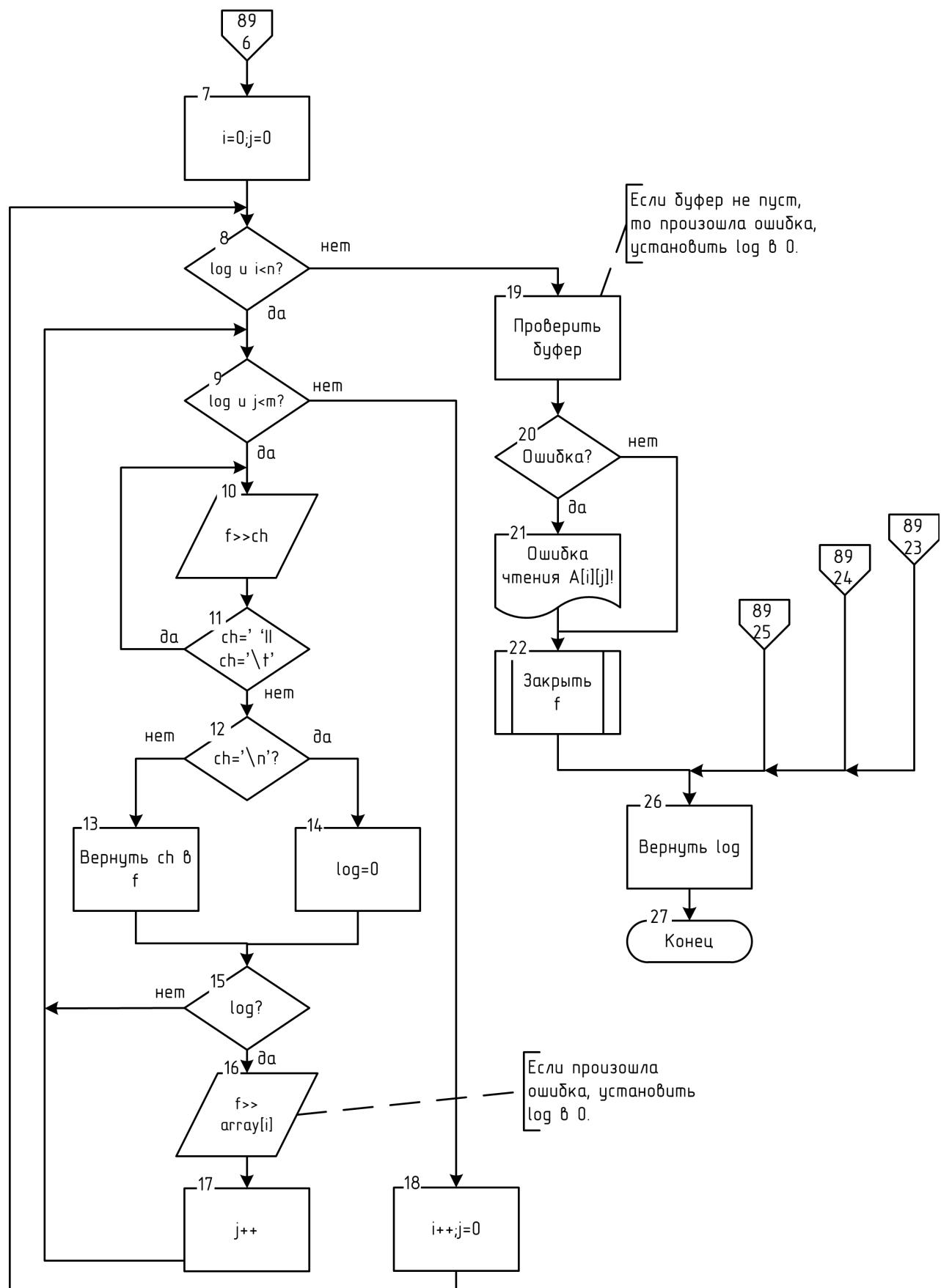


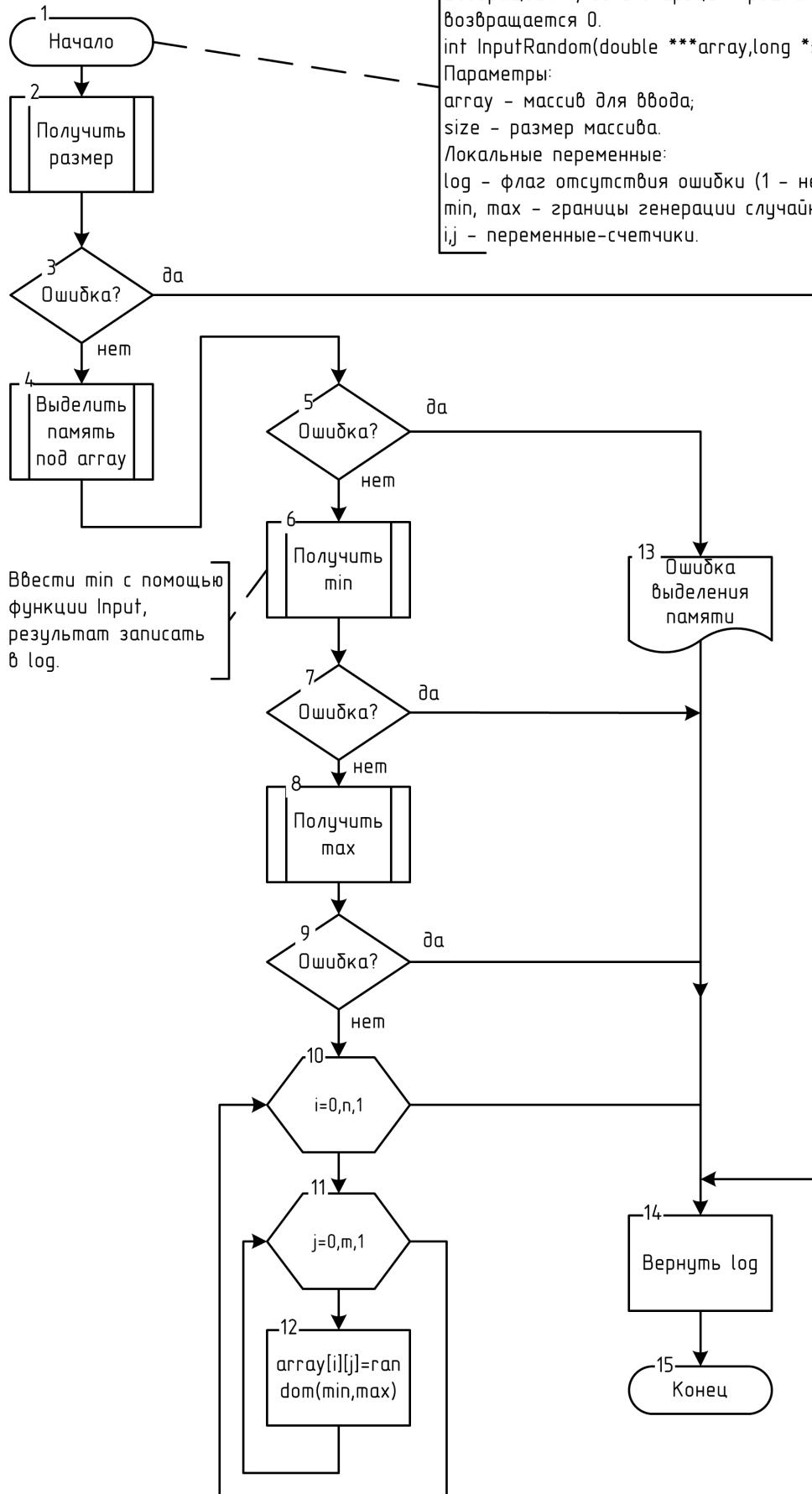
Функция InputFile() позволяет пользователю считать массив aggray размером size x size из файла.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.

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

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;
ij - переменные-счетчики.

Рисунок 3.8 - Схема алгоритма заполнения матрицы из файла (начало)





Функция `InputRandom()` позволяет пользователю заполнить массив `array` размером `size x size` случайными числами. Возвращает 1, если операция прошла успешно, иначе возвращается 0.

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

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.

Рисунок 3.10 - Схема алгоритма заполнения матрицы случайными числами

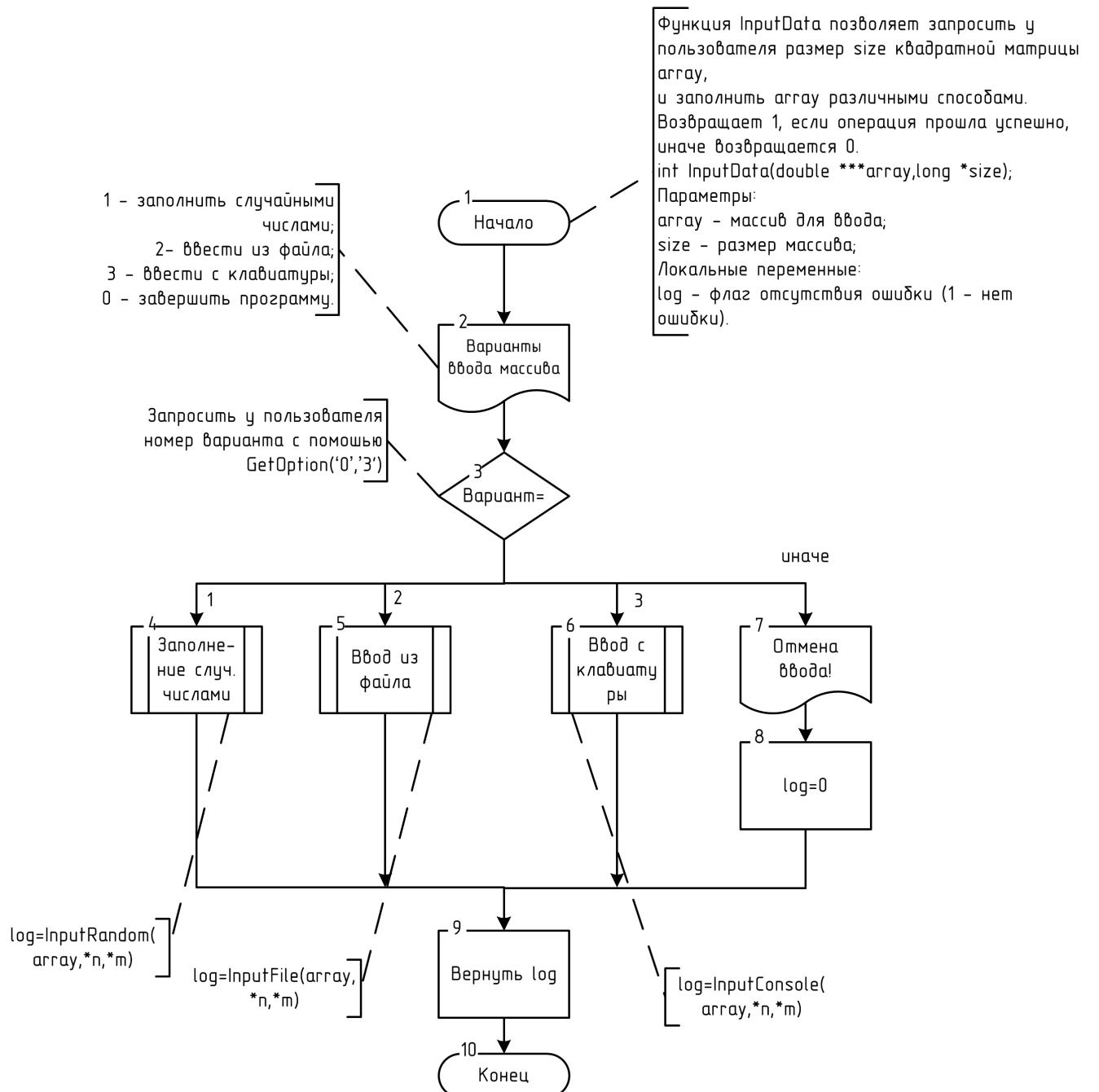
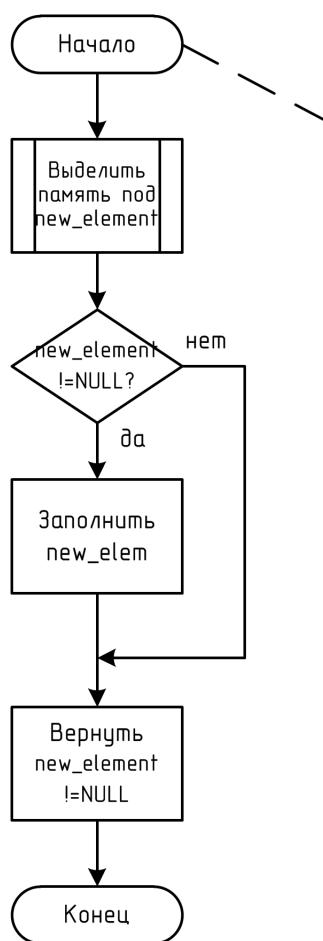


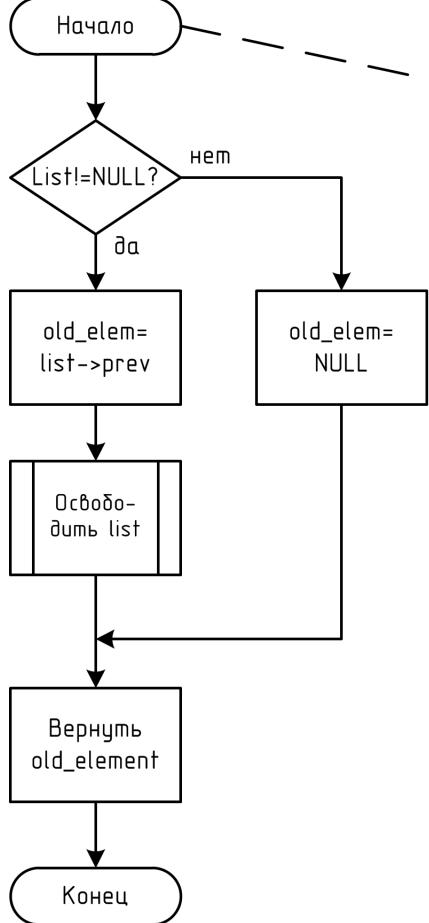
Рисунок 3.11 - Схема алгоритма заполнения матрицы различными способами

Допустим, мы не удовлетворены ни опасностью "падения" рекурсивной процедуры сортировки из-за переполнения стека, ни статическим стеком в виде массива. Наиболее интересным тогда кажется вариант создать стек с помощью связного списка. Естественно, для работы с такой структурой данных необходимы отдельные процедуры добавления в стек, удаления из стека, и очистки стека, и они соответственно представлены на рисунках 3.12, 3.13 и 3.14.



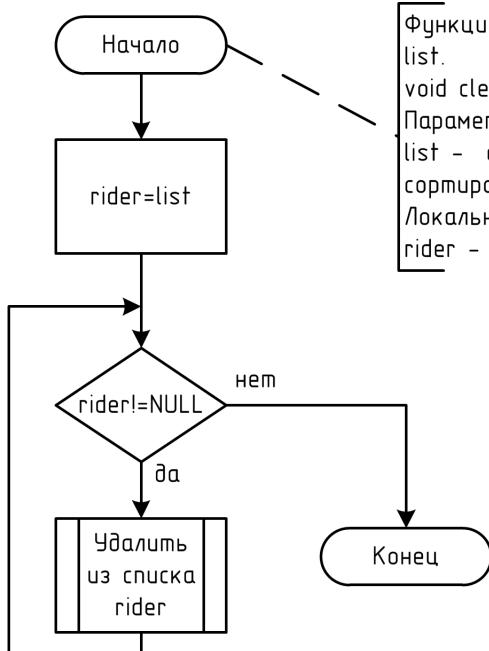
Функция `push` добавляет границы `low` и `high` отрезка сортировки в стек `list`. Возвращает новый указатель на верхний(последний) элемент стека. Если память под новый элемент не была получена, возвращается `NULL`.
`STACK* push(STACK *list, long low, long high);`
Параметры:
`high,low` – верхняя и нижняя границы текущего отрезка сортировки;
`list` – стек для хранения границ отрезков сортировки.
Локальные переменные:
`new_elem` – создаваемый новый элемент стека.

Рисунок 3.12 - Схема алгоритма вставки элемента в стек



Функция `pop` удаляет последний элемент стека `list`.
 Возвращает новый указатель на предыдущий элемент стека.
 Если удаленный элемент был первым элементом стека, возвращается `NULL`.
`STACK* pop(STACK *list);`
 Параметры:
`list` – стек для хранения границ отрезков сортировки.
 Локальные переменные:
`old_elem` – предыдущий перед удаляемым элементом стека.

Рисунок 3.13 - Схема алгоритма удаления элемента из стека



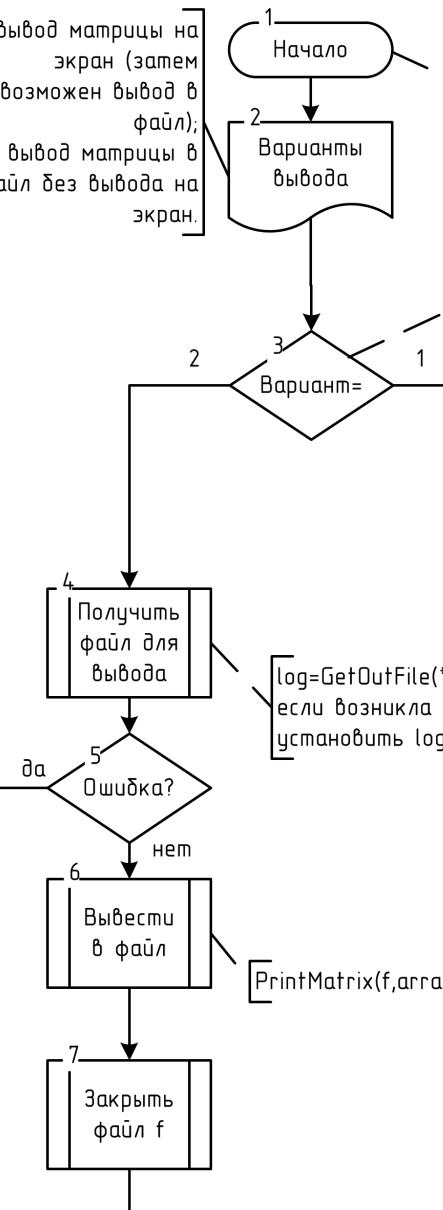
Функция `clearstack` полностью очищает стек `list`.
`void clearstack(STACK *list);`
 Параметры:
`list` – стек для хранения границ отрезков сортировки.
 Локальные переменные:
`rider` – текущий удаляемый элемент стека.

Рисунок 3.14 - Схема алгоритма очистки стека

Для печати на экран или в файл по выбору пользователя также предусмотрен отдельный алгоритм, схема которого представлена на рисунке 3.15.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

1 - вывод матрицы на экран (затем возможен вывод в файл);
2 - вывод матрицы в файл без вывода на экран.



Функция PrintResult выводит отсортированный массив array размером size x size в файл и на экран.

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

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

Параметры:

array - массив для вывода;

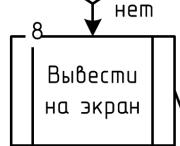
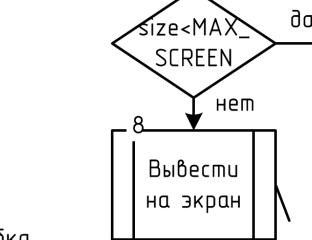
size - размер массива.

Локальные переменные:

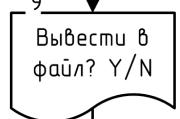
log - флаг отсутствия ошибки (1 - нет ошибки);

f - файл для вывода.

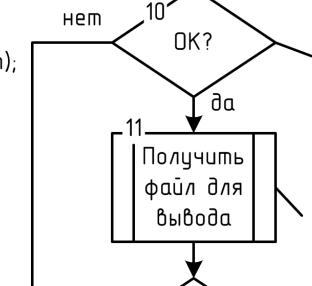
Запросить у пользователя номер варианта с помощью `GetOption('1','2')`



Массив слишком большой!



`PrintMatrix(stdout, array, n, m);`



Запросить у пользователя ответ с помощью `GetReqReslt`.

`log=GetOutFile(*f),`
если возникла ошибка,
установить log в 0.

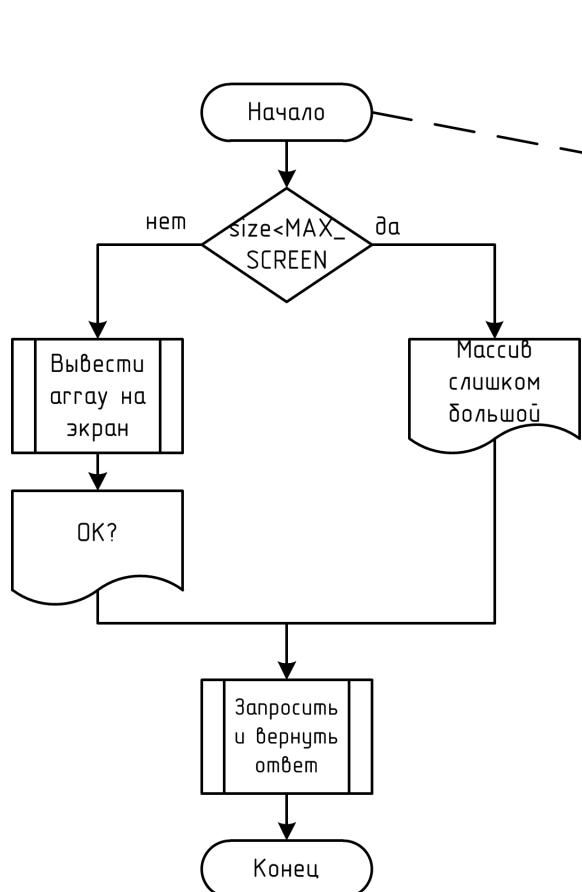


`PrintMatrix(f, array, n, m);`



Рисунок 3.15 - Схема алгоритма вывода матрицы в файл или на экран

Проверка данных также производится отдельным алгоритмом, и его схема — на рисунке 3.16.



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

Рисунок 3.16 - Схема алгоритма проверки матрицы

На рисунке 3.17 представлена схема вывода матрицы в файл.

Изм.	Лист	№ докум.	Подп.	Дата

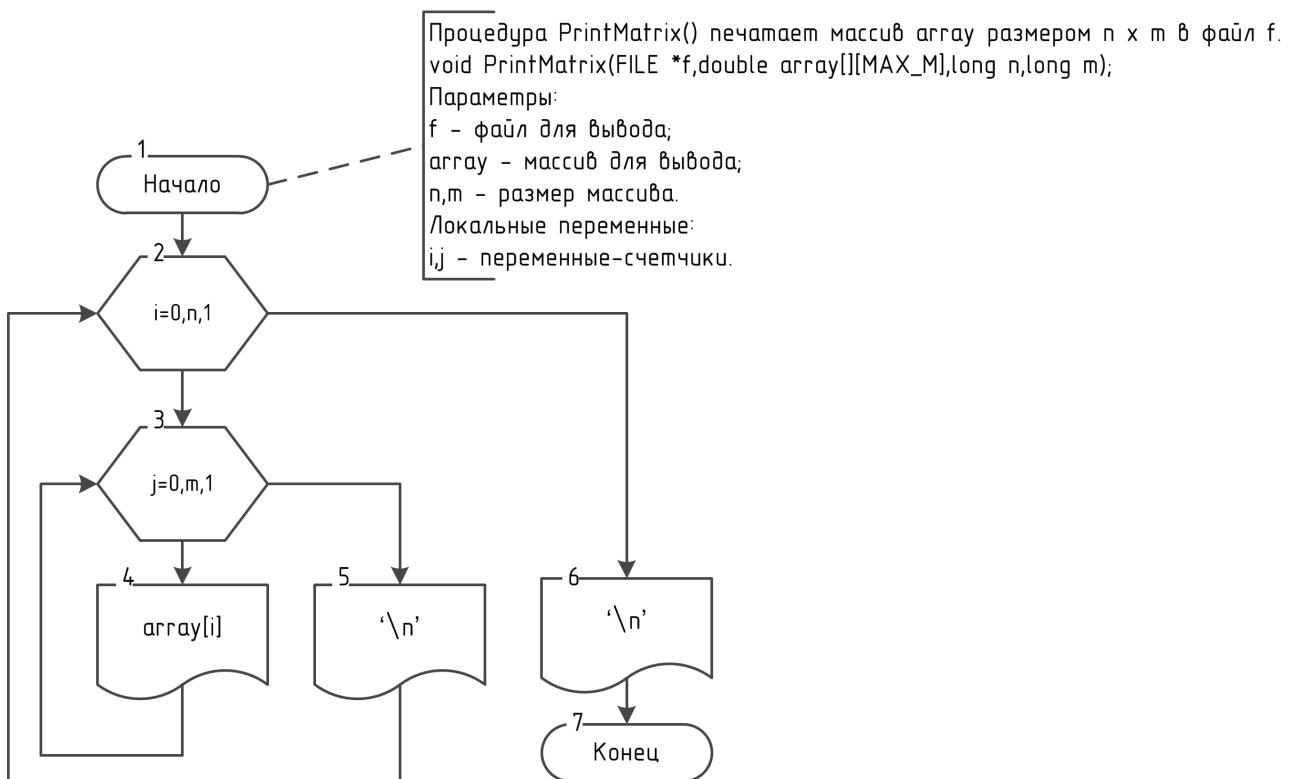


Рисунок 3.17 - Схема алгоритма вывода двумерного массива в файл

3.6. Текст программы

Ниже приведен исходный текст программы на языке Borland C 3.1, сортирующей главную диагональ 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
//связанный список-стек для функции сортировки
typedef struct stack {
    long high,low; //верхняя и нижняя границы текущего отрезка сортировки
    struct stack *prev; //предыдущий элемент стека
} STACK;

/*
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);
```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

/*
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 сортировки главной диагонали матрицы.
Переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
req_rslt - флаг ответа пользователя (1 - согласие пользователя);
array - массив для сортировки главной диагонали;
size - размер массива.
*/
int main();

/*
Функция push добавляет границы low и high отрезка сортировки в стек list.
Возвращает новый указатель на верхний(последний) элемент стека.
Если память под новый элемент не была получена, возвращается NULL.
Параметры:
high,low - вехняя и нижняя границы текущего отрезка сортировки;
list - стек для хранения границ отрезков сортировки.
Локальные переменные:
new_elem - создаваемый новый элемент стека.
*/
STACK* push(STACK *list, long low, long high);

/*
Функция pop удаляет последний элемент стека list.
Возвращает новый указатель на предыдущий элемент стека.
Если удаленный элемент был первым элементом стека, возвращается NULL.
Параметры:
list - стек для хранения границ отрезков сортировки.
Локальные переменные:
old_elem - предыдущий перед удаляемым элемент стека.
*/
STACK* pop(STACK *list);

/*
Функция clearstack полностью очищает стек list.
Параметры:
list - стек для хранения границ отрезков сортировки.
Локальные переменные:
rider - текущий удаляемый элемент стека.
*/
void clearstack(STACK *list);

/*

```

Изм.	Лист	№ докум.	Подп.	Дата

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

Параметры:

array - массив для проверки;
size - размер массива.

Локальные переменные:
отсутствуют.

```
/*
Функция InputFile() позволяет 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() позволяет 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() позволяет 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).
*/

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

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

/*
Функция PrintResult выводит отсортированный массив array размером size x size в
файл и на экран.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
array - массив для вывода;
size - размер массива.
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
f - файл для вывода.
*/
int PrintResult(double **array, long size);

/*
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);

/*
Функция QuickSort() сортирует главную диагональ квадратной матрицы array
размером size.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
arr - массив для сортировки;
size - размер массива.
Локальные переменные:
log - флаг отсутствия ошибки (1 - нет ошибки);
swap - временная переменная для обмена значениями элементов массива,
x - текущее значение, относительно которого сравниваются элементы;
L, R - нижняя(левая) и вехняя(правая) границы текущего отрезка сортировки;
stck - стек для хранения границ отрезков сортировки;
tmp - временный элемент стека для проверки успешности добавления в стек;
i, j - переменные-счетчики,
*/
int QuickSort(double **arr, 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.
*/

```

Изм.	Лист	№ докум.	Подп.	Дата

```

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);

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

```

Изм.	Лист	№ докум.	Подп.	Дата

```

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)) {

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

        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; long size;
    printf("Программа сортирует по возрастанию элементы главной диагонали\n\
            "матрицы размера mxm.\n");

    do {
        log=InputData(&array,&size);
        if (log)
            log=CheckData(array,size);
        if (!log) {
            printf( "Введенные данные некорректны или ввод отменён!\n");
            printf( "Хотите повторить ввод? (Y/N)\n");
            req_rslt=GetReqResult();
        }
    } while (!log&&req_rslt);
    if (log){
        log=QuickSort(array,size);
        if (log){
            log=PrintResult(array,size);
        } else
            printf("Массив слишком большой!\n");
        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){
    int log=1; FILE *f;
    printf("Главная диагональ матрицы размера %ldx%ld
отсортирована!...\n",size,size);
    printf("1 - вывод матрицы на экран (затем возможен вывод в файл);\n\
            "2 - вывод матрицы 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))){

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

        PrintMatrix(f, array, size); fclose(f);
    }
}
break;
case '2':
    if ((log=GetOutFile(&f)))
        PrintMatrix(f, array, size);
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");
}
}

STACK* push(STACK *list, long low, long high) {
STACK *new_elem;
new_elem=(STACK*)malloc(sizeof(STACK));
// new_elem=malloc(sizeof(STACK));
if (new_elem!=NULL) {
    new_elem->low=low;
    new_elem->high=high;
    new_elem->prev=list;
}
return new_elem;
}
STACK* pop(STACK *list) {
STACK *old_elem;
if (list!=NULL) {
    old_elem=list->prev;
    free(list);
} else old_elem=NULL;
return old_elem;
};
void clearstack(STACK *list) {
STACK *rider;
rider=list;
while(rider!=NULL) {
    rider=pop(rider);
}
};

int QuickSort(double **array, long size) {
int log;

double swap,x;
STACK *stck=NULL,*tmp=NULL;
long i,j, L, R;

log=((stck=push(stck,0,size-1))!=NULL);
while (log&&(stck!=NULL)) {
    L=stck->low; R=stck->high;
    stck=pop(stck);
    while (log&&L<R) {
        i=L; j=R; x=array[(L+R)/2][(L+R)/2];
        do{
            while (array[i][i]<x) i++;
            while (x<array[j][j]) j--;

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

        if (i<=j) {
            swap=array[i][i];array[i][i]=array[j][j];array[j][j]=swap;
            i++;j--;
        }
    }while(i<=j);
    if (R-i<j-L) {
        if (L<j) {
            log=((tmp=push(stck,L,j))!=NULL);
            if (log)
                stck=tmp;
            else
                clearstack(stck);
        }
        L=i;
    } else {
        if (i<R) {
            log=((tmp=push(stck,i,R))!=NULL);
            if (log)
                stck=tmp;
            else
                clearstack(stck);
        }
        R=j;
    }
}
return log;
}
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;

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

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);
}

```

Изм.	Лист	№ докум.	Подп.	Дата

```

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=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){
        log=AllocArray(array,*size);

        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");
    }
}

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

    }
    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 выполнена сортировка главной диагонали матрицы %ldx%ld\n",
: \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();
}

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

3.7. Инструкция программисту

При создании программы сортировки главной диагонали матрицы были предприняты следующие действия.

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

Объявлен тип-структура `stack`(с псевдонимом `STACK`), описание полей которого приводится в таблице 3.1.

Таблица 3.1 - Описание полей структуры "стек"

имя	тип	предназначение
high,low	long	верхняя и нижняя границы текущего отрезка сортировки
prev	struct stack *	предыдущий элемент стека

Объявлены следующие структуры данных, представленные в таблице 3.2:

Таблица 3.2 - Структуры данных, используемые в основной части программы сортировки главной диагонали

имя	тип	предназначение
size	long	размер массива;
array	double**	обрабатываемый массив;
K	double	число для сравнения элементов массива;
und,eq,ov	long	кол-во элементов меньших, равных и больших K;
i	long	переменная-счетчик.

Также программа была разбита на следующие подпрограммы:

1. 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 3.3:

Таблица 3.3 - Параметры функции получения файла для чтения

имя	тип	предназначение
f	FILE *	запрашиваемый файл.

Локальные переменные функции представлены в таблице 3.4:

Таблица 3.4 - 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).

2. 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 3.5:

Таблица 3.5 - Параметры функции получения файла для записи

имя	тип	предназначение
f	FILE *	запрашиваемый файл.

Локальные переменные функции представлены в таблице 3.6:

Таблица 3.6 - 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).

3. Функция PrintResult выводит массив array размером size x size в файл и на экран.

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

Параметры функции представлены в таблице 3.7:

Таблица 3.7 - Параметры функции вывода матрицы в файл или на экран

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

Локальные переменные функции представлены в таблице 3.8:

Таблица 3.8 - 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;

4. 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 3.9:

Таблица 3.9 - 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 3.10:

Таблица 3.10 - Локальные переменные функции вывода матрицы в файл

имя	тип	предназначение
i, j	long	переменные-счетчики.

5. 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 3.11:

Таблица 3.11 - 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 3.12:

Таблица 3.13 - 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).

12. 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 3.14:

Таблица 3.14 - 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 3.15 - 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).

6. 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 3.16:

Таблица 3.16 - 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 3.17:

Таблица 3.17 - 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).

7. Функция InputConsole() позволяет 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 3.18:

Таблица 3.18 - Параметры функции заполнения матрицы с клавиатуры

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

Локальные переменные функции представлены в таблице 3.19:

Таблица 3.19 - 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.

8. Функция InputFile() позволяет 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 3.20:

Таблица 3.20 - Параметры функции заполнения матрицы из файла

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

Локальные переменные функции представлены в таблице 3.21:

Таблица 3.21 - 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.

9. Функция InputRandom() позволяет 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 3.22:

Таблица 2.22 - Параметры функции заполнения матрицы случайными числами

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

Локальные переменные функции представлены в таблице 3.23:

Таблица 3.23 - 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.

10. 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 3.24:

Таблица 3.24 - Параметры функции ввода данных программы

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

Локальные переменные функции представлены в таблице 3.25:

Таблица 3.25 - Локальные переменные функции ввода данных программы

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

11.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 3.26, локальные переменные — в таблице 3.27.

Таблица 3.26 - Параметры функции получения сочетания

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

Таблица 3.27 - Локальные переменные функции получения сочетания

имя	тип	предназначение
count	long	Счетчик считанных символов.

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

Заголовок функции:

```
int GetReqResult();
```

Функция запрашивает у пользователя символы 'у','Y','н' 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 3.28.

Таблица 3.28 - Локальные переменные функции получения сочетания

имя	тип	предназначение
answer	char	Ответ пользователя.

13. 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 3.29, локальные переменные — в таблице 3.30.

Таблица 3.29 - Параметры функции получения сочетания

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

Функция push добавляет границы low и high отрезка сортировки в стек list.

Возвращает новый указатель на верхний(последний) элемент стека.

Если память под новый элемент не была получена, возвращается NULL.

STACK* push(STACK *list,long low, long high);

Параметры представлены в таблице 3.31:

Таблица 3.31 - Параметры функции добавления в стек

имя	тип	предназначение
high,low	long	вехняя и нижняя границы текущего отрезка сортировки;
list	STACK*	стек для хранения границ отрезков сортировки.

Локальные переменные представлены в таблице 3.32:

Таблица 3.32 - Локальные переменные функции добавления в стек

имя	тип	предназначение
new_elem	STACK*	создаваемый новый элемент стека.

Функция pop удаляет последний элемент стека list.

Возвращает новый указатель на предыдущий элемент стека.

Если удаленный элемент был первым элементом стека, возвращается NULL.

STACK* pop(STACK *list);

Параметры представлены в таблице 3.33:

Таблица 3.33 - Параметры функции удаления из стека

имя	тип	предназначение
list	STACK*	стек для хранения границ отрезков сортировки.

Локальные переменные представлены в таблице 3.34:

Таблица 3.34 - Локальные переменные функции удаления из стека

имя	тип	предназначение
old_elem	STACK*	предыдущий перед удалаемым элемент стека.

Функция clearstack полностью очищает стек list.

```
void clearstack(STACK *list);
```

Параметры представлены в таблице 3.35:

Таблица 3.35 - Параметры функции очистки стека

имя	тип	предназначение
list	STACK*	стек для хранения границ отрезков сортировки.

Локальные переменные представлены в таблице 3.36:

Таблица 3.36 - Локальные переменные функции очистки стека

имя	тип	предназначение
rider	STACK*	текущий удаляемый элемент стека.

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

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

Параметры представлены в таблице 3.37:

Таблица 3.37 - Параметры функции проверки данных

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

Локальные переменные:

отсутствуют.

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 3.38:

Таблица 3.38 - Параметры функции выделения памяти под массив

имя	тип	предназначение
array	double ***	массив для выделения памяти;

size	long	размер массива.
------	------	-----------------

Локальные переменные представлены в таблице 3.39:

Таблица 3.39 - Локальные переменные функции выделения памяти под массив

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

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

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

Параметры представлены в таблице 3.40:

Таблица 3.40 - Параметры функции освобождения памяти под массив

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

Локальные переменные представлены в таблице 3.41:

Таблица 3.41 - Локальные переменные функции освобождения памяти под массив

имя	тип	предназначение
i	long	переменная–счетчик;

Функция QuickSort() сортирует главную диагональ квадратной матрицы array размером size.

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

int QuickSort(double **arr, long size);

Параметры представлены в таблице 3.42:

Таблица 3.42 - Параметры функции сортировки главной диагонали

имя	тип	предназначение
arr	double **	массив для сортировки;
size	long	размер массива.

Локальные переменные представлены в таблице 3.43:

Таблица 3.43 - Локальные переменные функции сортировки главной диагонали

имя	тип	предназначение
log	int	флаг отсутствия ошибки (1 – нет ошибки);

swap	double	временная переменная для обмена значениями элементов массива,
x	double	текущее значение, относительно которого сравниваются элементы;
L, R	long	нижняя(левая) и вехняя(правая) границы текущего отрезка сортировки;
stck	STACK*	стек для хранения границ отрезков сортировки,
tmp	STACK*	временный элемент стека для проверки успешности добавления в стек;
i,j	long	переменные–счетчики,

3.8. Инструкция пользователю

Данная программа сортирует по неубыванию главную диагональ 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. Элементы разделяются пробелами или табуляциями.

После умножения указанного столбца на указанное число матрицу можно вывести в файл или на экран по вашему желанию. При выборе файла следует учитывать потерю содержащейся в нем информации — файл будет перезаписан.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

3.9. Тестовый пример

Ниже на рисунке 3.18 показан пример работы программы сортировки главной диагонали квадратной матрицы для матрицы, сгенерированной случайными числами.

Как можно заметить, программа вывела правильные результаты;

```
Программа сортирует по возрастанию элементы главной диагонали
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.
1
Введите m - размер матрицы. 1<=m<=5.
m: 5
Введите min - минимальное из случайных чисел.
min: -20
Введите max - максимальное из случайных чисел.
max: 20
Будет выполнена сортировка главной диагонали матрицы 5x5 :
-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  -16.3439  -5.4219  -14.1075 -13.3641
 19.5410  -2.1723  -15.2367 -19.8132 -19.6435
Продолжить?(Y/N)
y
Главная диагональ матрицы размера 5x5 отсортирована!...
1 - вывод матрицы на экран (затем возможен вывод в файл);
2 - вывод матрицы в файл без вывода на экран.
Введите цифру от 1 до 2.
1
-19.9499  2.5434 -12.2678  12.3496  3.4004
-0.8051  -19.6435  15.8385  12.9136  9.8642
-13.0357  14.3577 -14.1075   0.5414  -7.8402
-19.4006  -16.3439  -5.4219  -5.9883 -13.3641
 19.5410  -2.1723  -15.2367 -19.8132   8.4201
Хотите вывести матрицу в файл? Y/N
y
Введите имя файла-результата.
файл: matrix.txt
Работа успешно выполнена!
Нажмите <Enter>...
```

Рисунок 3.18 - Пример работы программы сортировки главной диагонали матрицы

Рассмотрим содержимое файла matrix.txt:

```
-19.9499  2.5434 -12.2678  12.3496  3.4004
-0.8051  -19.6435  15.8385  12.9136  9.8642
-13.0357  14.3577 -14.1075   0.5414  -7.8402
-19.4006  -16.3439  -5.4219  -5.9883 -13.3641
 19.5410  -2.1723  -15.2367 -19.8132   8.4201
```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Как можно заметить, программа вывела правильные результаты:

В неотсортированном виде диагональ имела вид -19.9499 -5.9883 8.4201
-14.1075 -19.6435, а после сортировки приняла вид -19.9499 -19.6435 -14.1075
-5.9883 8.4201. Программа работает корректно.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Вариант №4

Лист

123

Изм.	Лист	№ докум.	Подп.	Дата

Вариант №4

Лист

124

4. Задача составления алфавитного частотного словаря

4.1. Постановка задачи

Литературное произведение на русском языке записано в текстовом файле INPUT.TXT (размер файла до 1 Мб). Получить и записать в файл OUTPUT.TXT частотный словарь этого произведения, то есть алфавитный перечень слов (словоформ), встречающихся в тексте с указанием того, сколько раз входит в текст данное слово (словоформа). Словом считается последовательность букв, не содержащая пробелов и знаков препинания. Слова в исходном файле не переносятся.

4.2. Математическая формулировка задачи

Математическая формулировка задачи затруднительна.

4.3. Метод поиска по бинарному(двоичному) дереву

Двойчное дерево — древовидная структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками.

Для практических целей обычно используют два подвида бинарных деревьев — двоичное дерево поиска и двоичная куча.

Существует следующее рекурсивное определение двоичного дерева:

```
<дерево> ::= ( <данные> <дерево> <дерево> ) | nil .
```

То есть двоичное дерево либо является пустым, либо состоит из данных и двух поддеревьев (каждое из которых может быть пустым). Очевидным, но важным для понимания фактом является то, что каждое поддерево в свою очередь тоже является деревом. Если у некоторого узла оба поддерева пустые, то он называется листовым узлом (листовой вершиной).

Например, показанное справа на рис. 1 дерево, согласно этой грамматике можно было бы записать так:

(m

Изм.	Лист	№ докум.	Подп.	Дата

Вариант №4

Лист

125

```

(e
  (c
    (a nil nil)
    nil
  )
  (g
    nil
    (k nil nil)
  )
)
(s
  (p (o nil nil) (s nil nil) )
  (y nil nil)
)
)

```

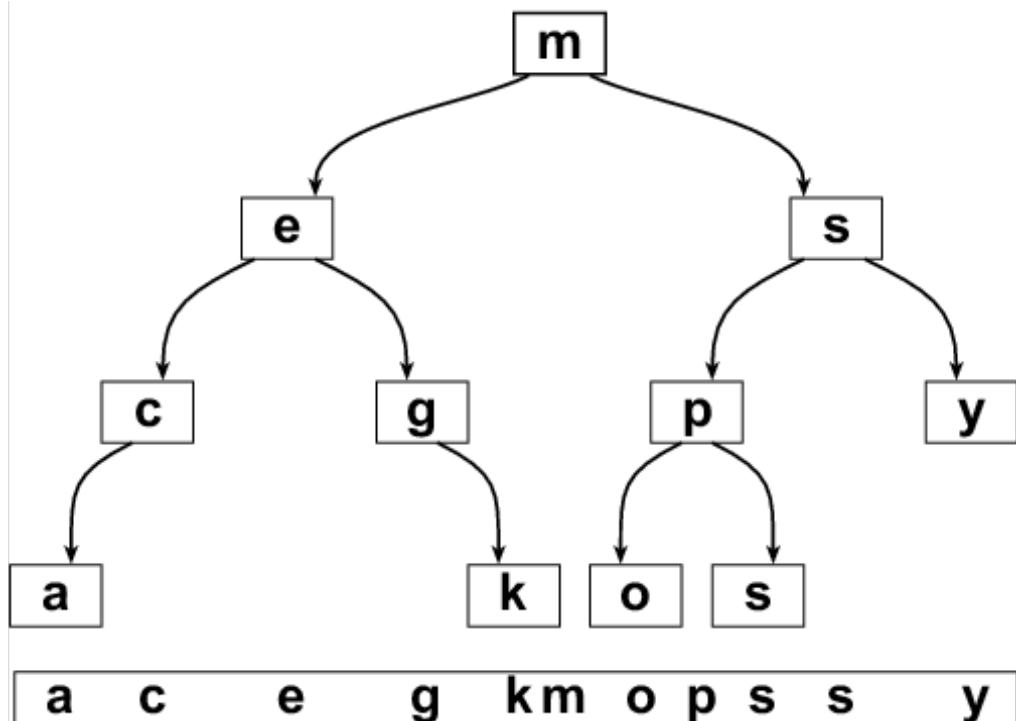


Рис. 4.1 - Двоичное дерево поиска, в котором ключами являются латинские символы упорядоченные по алфавиту

Каждый узел в дереве задаёт *поддерево*, корнем которого он является. У вершины $n=(\text{data}, \text{left}, \text{right})$ есть два ребёнка (левый и правый) left и right и, соответственно, два поддерева (левое и правое) с корнями left и right .

Двоичное дерево поиска (англ. *binary search tree*, BST) — это двоичное дерево, для которого выполняются следующие дополнительные условия (*свойства дерева поиска*):

- Оба поддерева — левое и правое, являются двоичными деревьями поиска.

- У всех узлов левого поддерева произвольного узла X значения ключей данных *меньше*, нежели значение ключа данных узла X.

- У всех узлов правого поддерева произвольного узла X значения ключей данных *больше*, нежели значение ключа данных узла X.

Очевидно, данные в каждом узле должны обладать ключами на которых определена операция сравнения *меньше*.

Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных. Однако, это касается реализации, а не природы двоичного дерева поиска.

Для целей реализации двоичное дерево поиска можно определить так:

- Двоичное дерево состоит из узлов (вершин) — записей вида (data, left, right), где data — некоторые данные привязанные к узлу, left и right — ссылки на узлы, являющиеся детьми данного узла - левый и правый сыновья соответственно. Для оптимизации алгоритмов конкретные реализации предполагают также, определения в каждом узле кроме корневого поля parent - ссылки на родительский элемент.

- Данные (data) обладают ключом (key) на котором определена операция сравнения "меньше". В конкретных реализациях это может быть пара (key, value) - (ключ и значение), или ссылка на такую пару, или простое определение операции сравнения на необходимой структуре данных или ссылке на неё.

- Для любого узла X выполняются свойства дерева поиска: $\text{key}[\text{left}[X]] < \text{key}[X] \leq \text{key}[\text{right}[X]]$, т. е. ключи данных родительского узла больше ключей данных левого сына и нестрого меньше ключей данных правого.

Двоичное дерево поиска не следует путать с двоичной кучей, построенной по другим правилам.

Основным преимуществом двоичного дерева поиска перед другими структурами данных является возможная высокая эффективность реализации основанных на нём алгоритмов поиска и сортировки.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Двоичное дерево поиска применяется для построения более абстрактных структур, таких как множества, мульти множества, ассоциативные массивы.

Базовый интерфейс двоичного дерева поиска состоит из трех операций:

- FIND(K) — поиск узла, в котором хранится пара (key, value) с key = K.
- INSERT(K,V) — добавление в дерево пары (key, value) = (K, V).
- REMOVE(K) — удаление узла, в котором хранится пара (key, value) с key = K.

По сути, двоичное дерево поиска — это структура данных, способная хранить таблицу пар (key, value) и поддерживающая три операции: FIND, INSERT, REMOVE.

Кроме того, интерфейс двоичного дерева включает ещё три дополнительных операции обхода узлов дерева: INFIX_TRAVERSE, PREFIX_TRAVERSE и POSTFIX_TRAVERSE. Первая из них позволяет обойти узлы дерева в порядке неубывания ключей.

1. Поиск элемента (FIND)

Дано: дерево Т и ключ К.

Задача: проверить, есть ли узел с ключом К в дереве Т, и если да, то вернуть ссылку на этот узел.

Алгоритм:

1. Если дерево пусто, сообщить, что узел не найден, и остановиться.
2. Иначе сравнить К со значением ключа корневого узла X.
3. Если K=X, выдать ссылку на этот узел и остановиться.
4. Если K>X, рекурсивно искать ключ K в правом поддереве T.
5. Если K<X, рекурсивно искать ключ K в левом поддереве T.

2. Добавление элемента (INSERT)

Дано: дерево Т и пара (K,V).

Изм.	Лист	№ докум.	Подп.	Дата

Задача: добавить пару (K, V) в дерево T.

Алгоритм:

1. Если дерево пусто, заменить его на дерево с одним корневым узлом ((K,V), null, null) и остановиться.
2. Иначе сравнить K с ключом корневого узла X.
3. Если K>=X, рекурсивно добавить (K,V) в правое поддерево T.
4. Если K<X, рекурсивно добавить (K,V) в левое поддерево T.

3. Удаление узла (REMOVE)

Дано: дерево T с корнем n и ключом K.

Задача: удалить из дерева T узел с ключом K (если такой есть).

Алгоритм:

1. Если дерево T пусто, остановиться;
2. Иначе сравнить K с ключом X корневого узла n.
3. Если K>X, рекурсивно удалить K из правого поддерева T;
4. Если K<X, рекурсивно удалить K из левого поддерева T;
5. Если K=X, то необходимо рассмотреть три случая.
6. Если обоих детей нет, то удаляем текущий узел и обнуляем ссылку на него у родительского узла;
7. Если одного из детей нет, то значения полей второго ребёнка m ставим вместо соответствующих значений корневого узла, затирая его старые значения, и освобождаем память, занимаемую узлом m;
8. Если оба ребёнка присутствуют, то
 - а) найдём узел m, являющийся самым левым узлом правого поддерева с корневым узлом Right(n);
 - б) скопируем данные (кроме ссылок на дочерние элементы) из m в n;

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

с) рекурсивно удалим узел m .

4. Обход дерева (TRAVERSE)

Есть три операции обхода узлов дерева, отличающиеся порядком обхода узлов.

Первая операция — INFIX_TRAVERSE — позволяет обойти все узлы дерева в порядке возрастания ключей и применить к каждому узлу заданную пользователем функцию обратного вызова f . Эта функция обычно работает только с парой (K, V) , хранящейся в узле. Операция INFIX_TRAVERSE реализуется рекурсивным образом: сначала она запускает себя для левого поддерева, потом запускает данную функцию для корня, потом запускает себя для правого поддерева.

INFIX_TRAVERSE (f) — обойти всё дерево, следуя порядку (левое поддерево, вершина, правое поддерево).

PREFIX_TRAVERSE (f) — обойти всё дерево, следуя порядку (вершина, левое поддерево, правое поддерево).

POSTFIX_TRAVERSE (f) — обойти всё дерево, следуя порядку (левое поддерево, правое поддерево, вершина).

INFIX_TRAVERSE:

Дано: дерево T и функция f

Задача: применить f ко всем узлам дерева T в порядке возрастания ключей

Алгоритм:

1. Если дерево пусто, остановиться.

2. Иначе:

- a) Рекурсивно обойти левое поддерево T .
- b) Применить функцию f к корневому узлу.
- c) Рекурсивно обойти правое поддерево T .

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

После ознакомления с понятием и свойствами дерева, рассмотрим решаемую нами задачу. Для формирования алфавитного частотного словаря нам необходимо запоминать слово и количество его вхождений в данный текст. Очевидно, что в различных текстах имеется огромное и совершенно различное количество различных слов, с точки зрения алфавитного порядка расположенных произвольно. Поэтому идея использования массива для хранения вышеуказанных данных отпадает.

Неопределенное и изменяющееся количество слов в тексте приводит нас к идеи использования динамических структур данных. Использование упорядоченного списка кажется гораздо более удачной идеей, чем статический массив. К сожалению, связные списки при больших размерах становится тяжело обрабатывать. Возможно, придется проходить весь список — от первого до последнего элемента, чтобы вставить в него значение. Желая оптимизировать вставку значения в список, мы приходим к идеи использования в качестве базовой структуры данных дерева.

Очевидно, что число различных ключей, которые можно представить на некоторой произвольной глубине дерева, растет экспоненциально с глубиной дерева. Соответственно, чем больше различных ключей в исходном тексте, тем большим становится наш выигрыш по сравнению со связным списком.

Однако дерево имеет некоторый недостаток по сравнению со списком. Дерево в некотором смысле является обобщением списка, что отражается в его более рекурсивной структуре. В частности, непросто будет уйти от рекурсивного стиля процедуры печати ключей дерева. При этом всегда существует вероятность (достаточно малая), что ключи будут встречаться в исходном тексте так, что дерево выродится в связный список. Тогда при этом мы потеряем весь выигрыш производительности на этапе вставки ключа в дерево, а взамен приобретём возможность переполнения стека в рекурсивной процедуре вывода. Однако, как было сказано, вероятность такого исхода достаточно мала, поэтому использование дерева является хорошим выбором при решении данной задачи.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Итак, взвесив плюсы и минусы использования дерева, опишем метод решения задачи формирования алфавитного частотного словаря. Пусть ключом дерева будут слова из данного текста. Также каждый узел дерева должен зранить количество этого слова в тексте. Создадим подобное двоичное дерево, из данных, содержащихся в данном файле, вставляя ключи согласно лексикографическому порядку и при совпадении увеличивая счетчик слова в имеющемся узле. Этими действиями мы одновременно и упорядочили слова по алфавиту, и подсчитали количество каждого слова в тексте. Осталось обойти и распечатать дерево от самого левого листа к самого правому (процедурой INFIX_TRAVERSE), чтобы получить запрошенный алфавитный частотный словарь.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

4.4. Разработка структур данных

Так как исходный текст может быть размером до 1 МБ, то оперативной памяти может нехватить при создании и обработки дерева. Поэтому само дерево будет находиться во временно создаваемом типизированном файле Tree.

Разумеется, нужны файлы для ввода и вывода слов в виде текста, для этого служат файлы fin и fout.

Для оценки правильности выполнения программы введем различные флаги — пусть ok и log свидетельствуют о правильности работы при значениях 1, error символизирует ошибку при значении 1, а req_rslt хранит ответ пользователя на различные запросы (на которые предполагается ответ или да, или нет).

Сам элемент дерева будет обозначаться указателем на тип Node, который будет состоять из следующих частей: значение строкового типа key – словоформа, count – количество вхождений словоформы в текст и номера left и right – номер ячеек файла с потомками данного узла.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

4.5. Разработка структуры алгоритма решения задачи

На рисунке 4.2 можно увидеть схему алгоритма получения файлов, создания и вывода алфавитного частотного словаря.

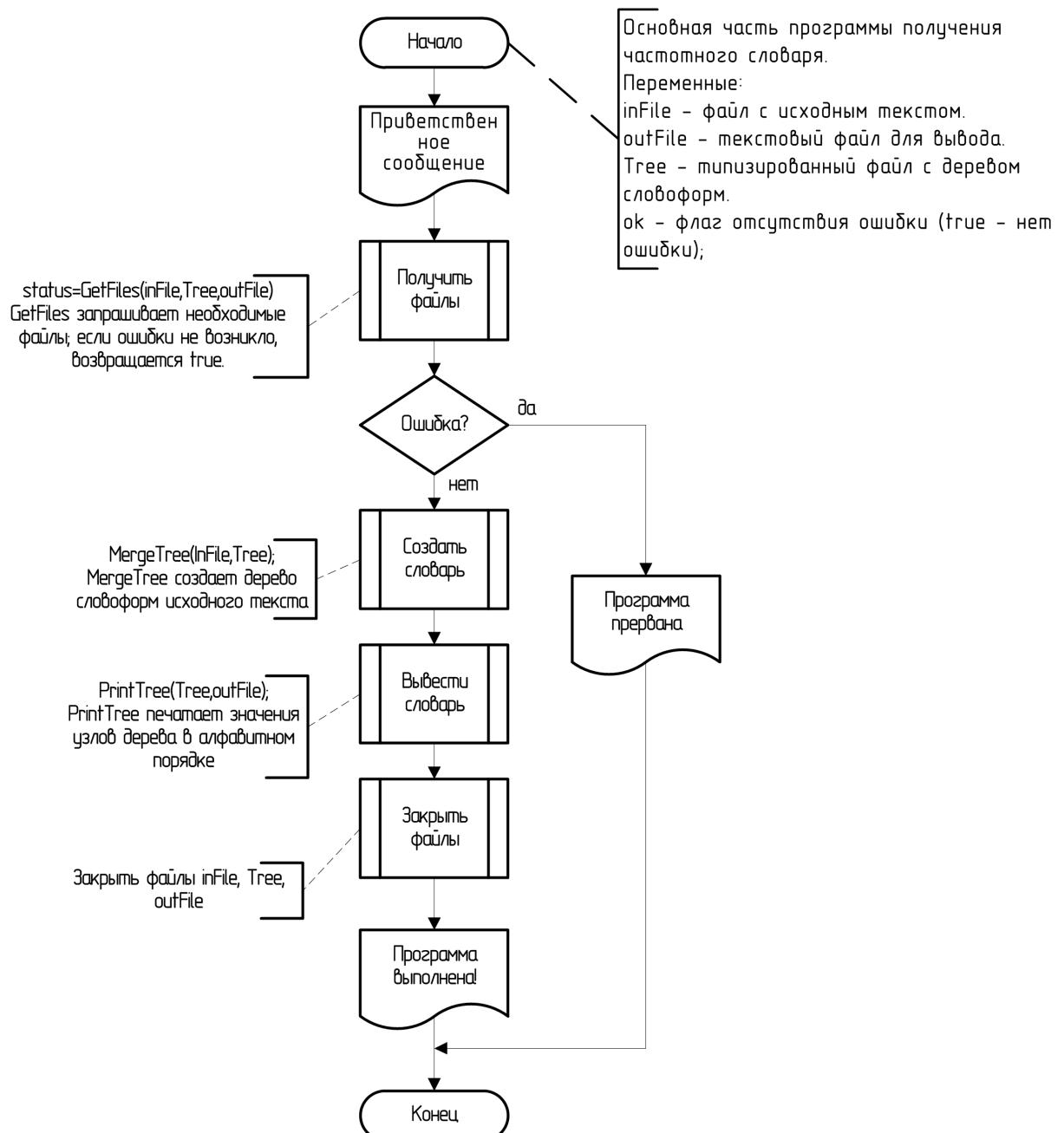


Рисунок 4.2 - Схема обобщенного алгоритма создания дерева-частотного словаря

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

На рисунке 4.2 изображена схема алгоритма дерева-основы частотного словаря.

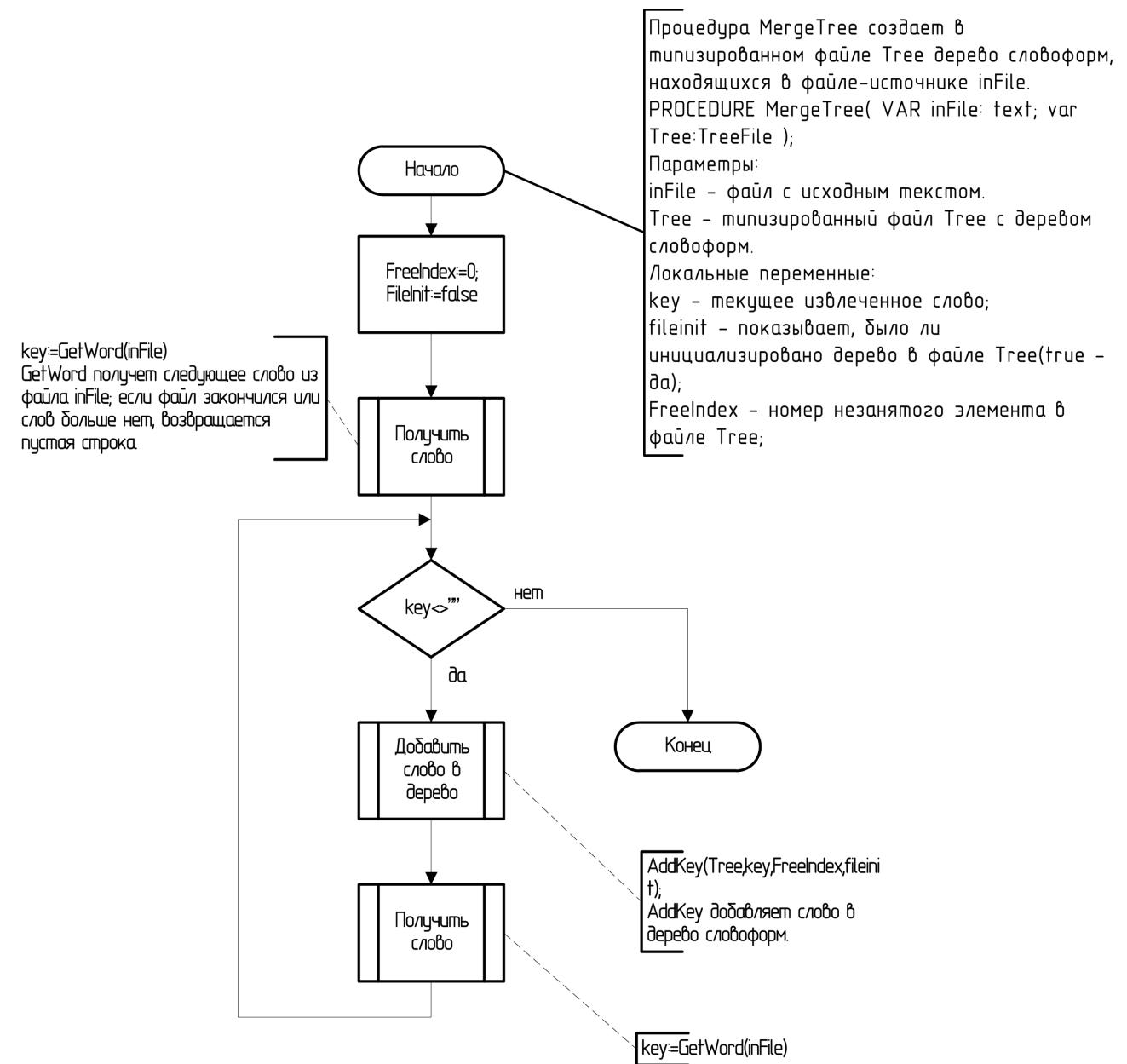


Рисунок 4.3 - Схема алгоритма создания дерева-словаря из файла

Схема алгоритма добавления ключа в дерево представлена на рисунках 4.4 и 4.5.

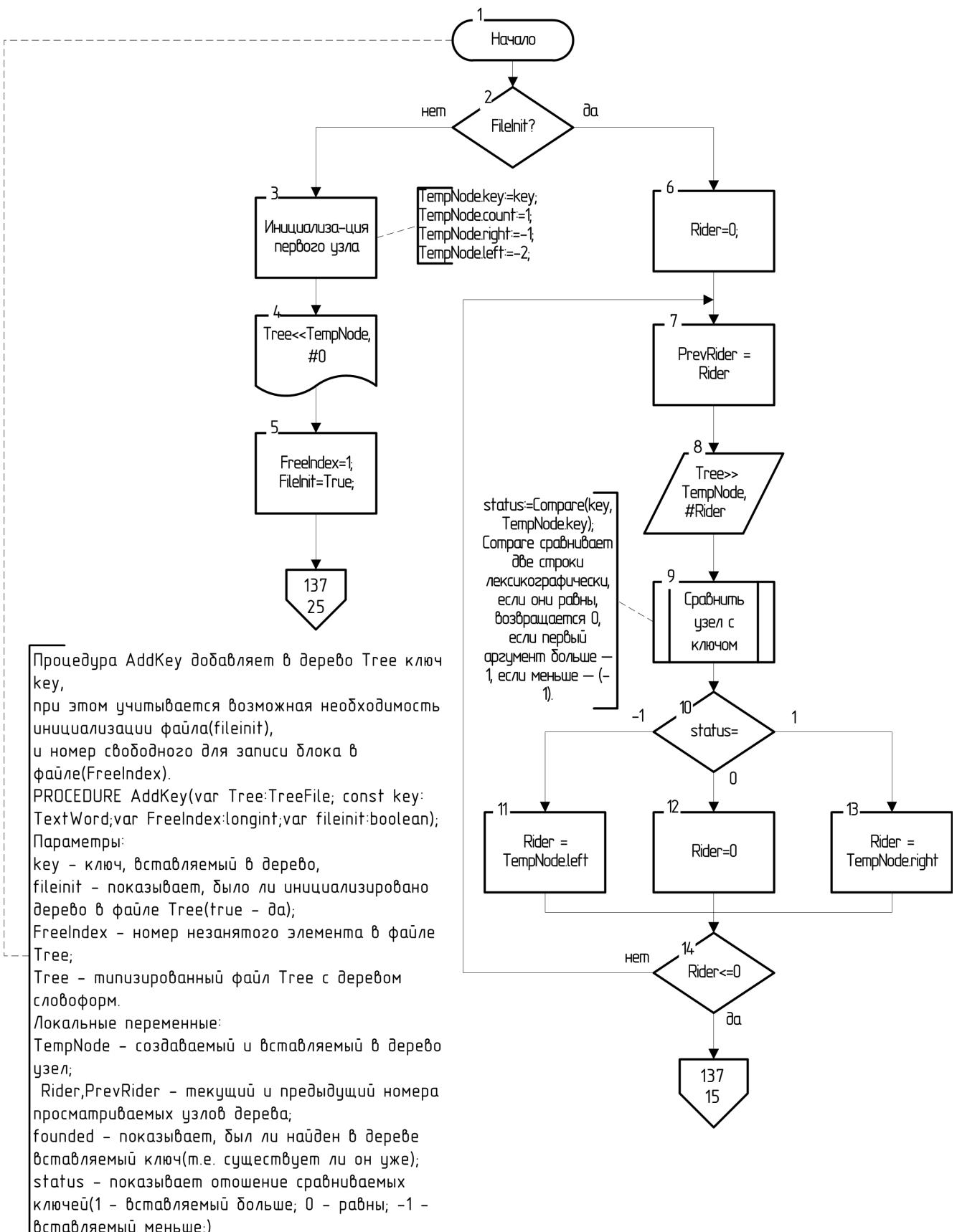


Рисунок 4.4 - Схема алгоритма вставки слова в дерево (начало)

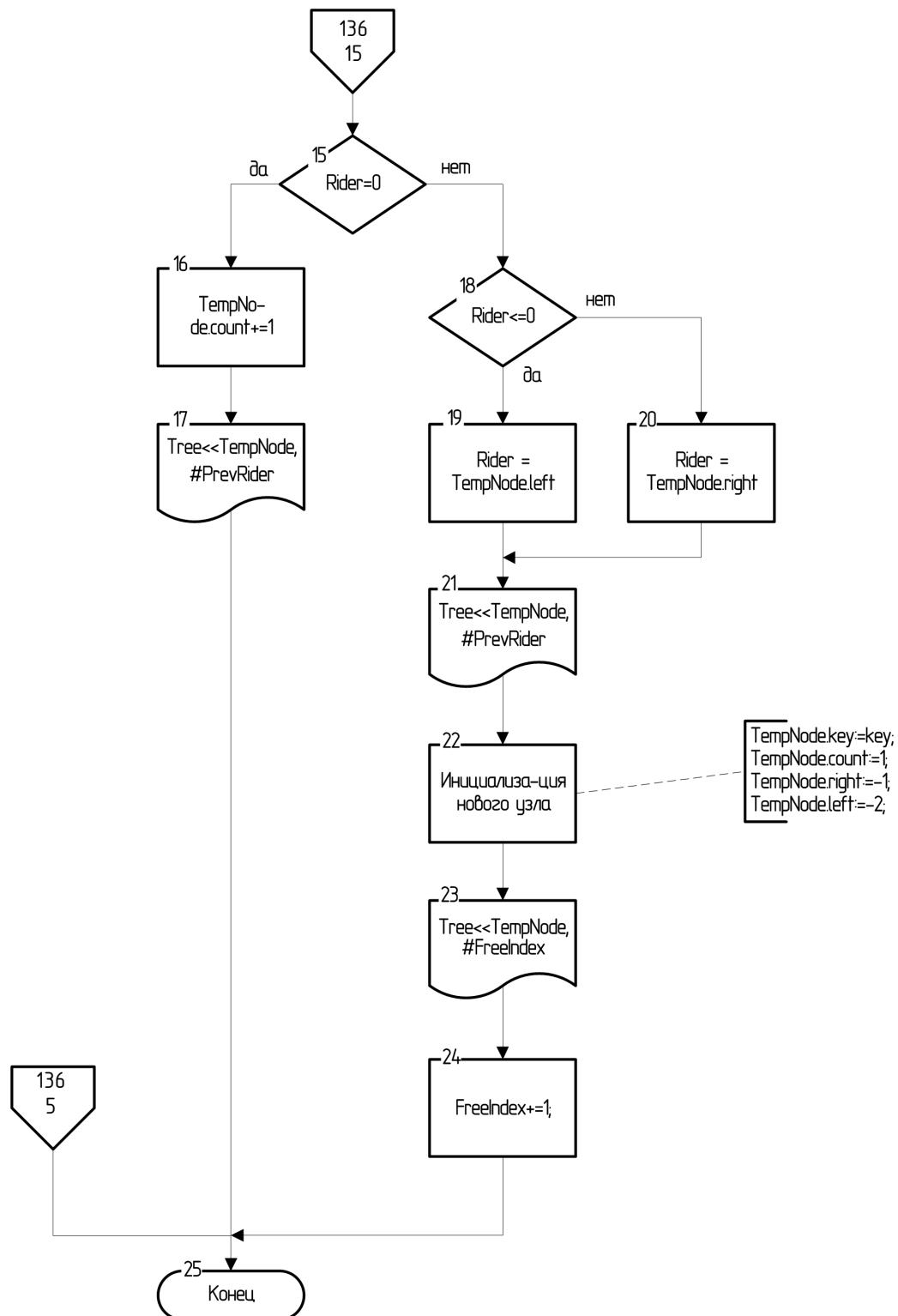


Рисунок 4.5 - Схема алгоритма вставки слова в дерево (продолжение)

Схему получения слова из файла можно увидеть на рисунке 4.6.

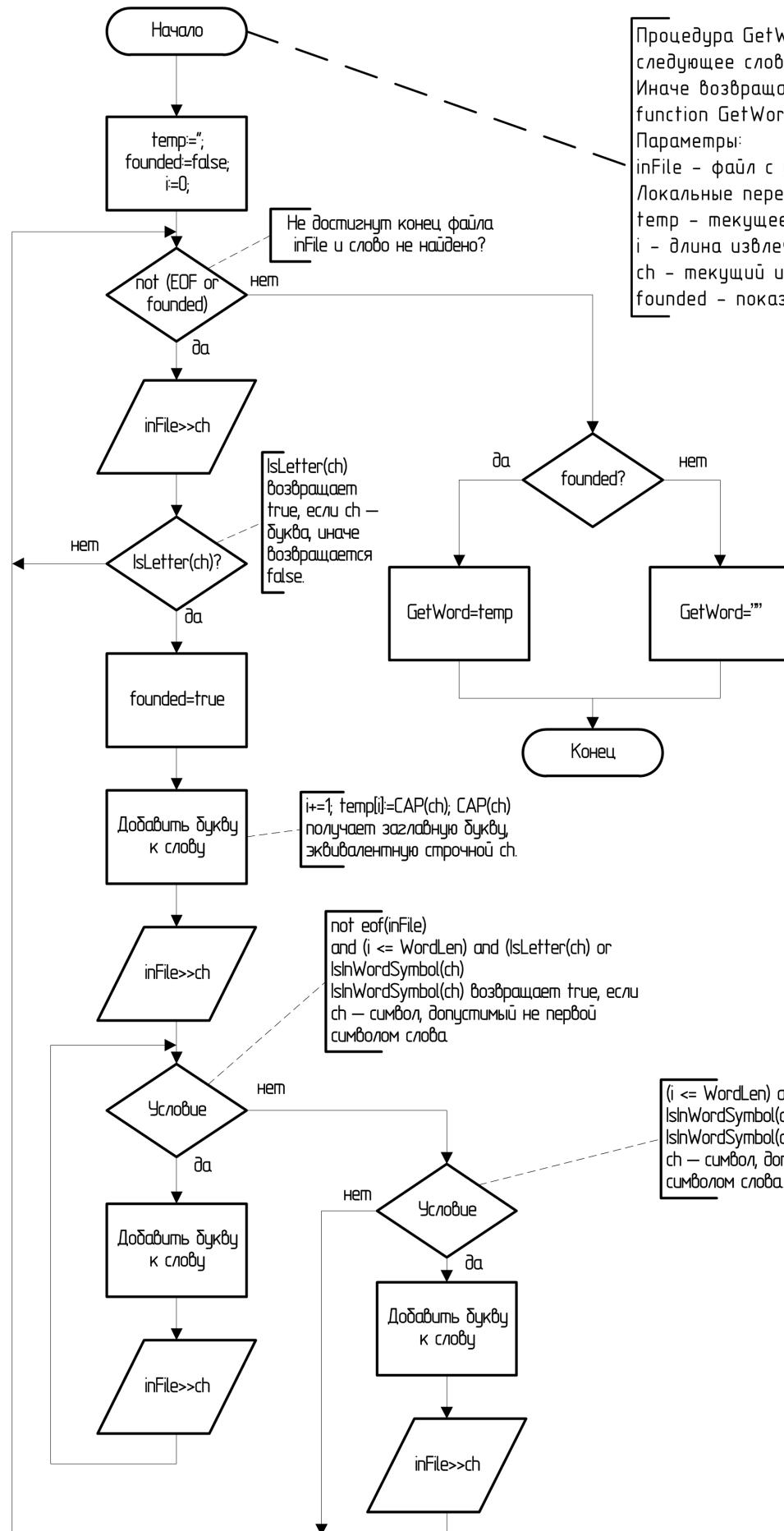


Рисунок 4.6 - Схема алгоритма получения слова из файла

На рисунке 4.7 можно увидеть блок-схему печати дерева в файл.

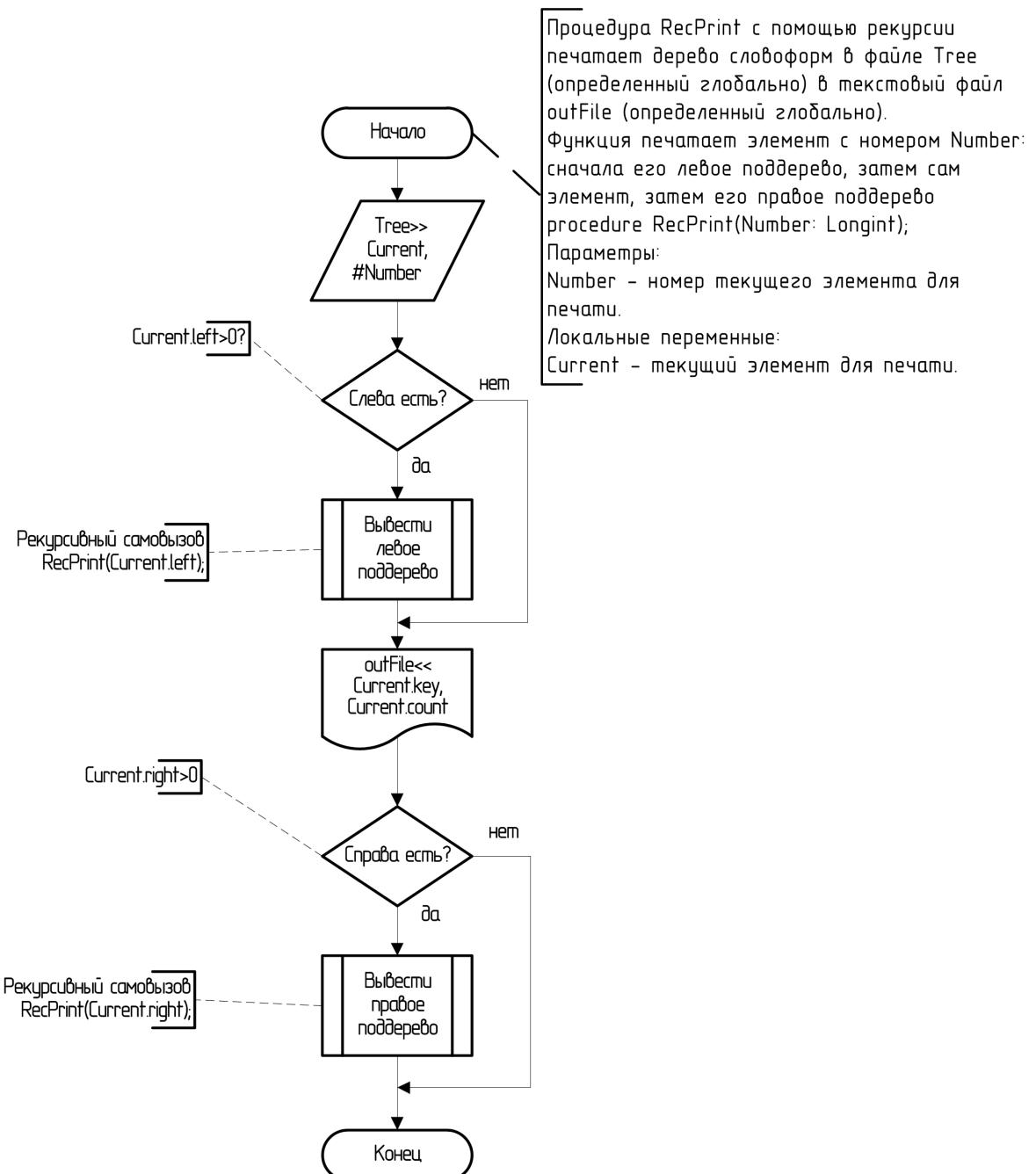


Рисунок 4.7 - Схема алгоритма печати дерева словоформ

Схема алгоритма лексикографического сравнения строк представлена на рисунке 4.8.

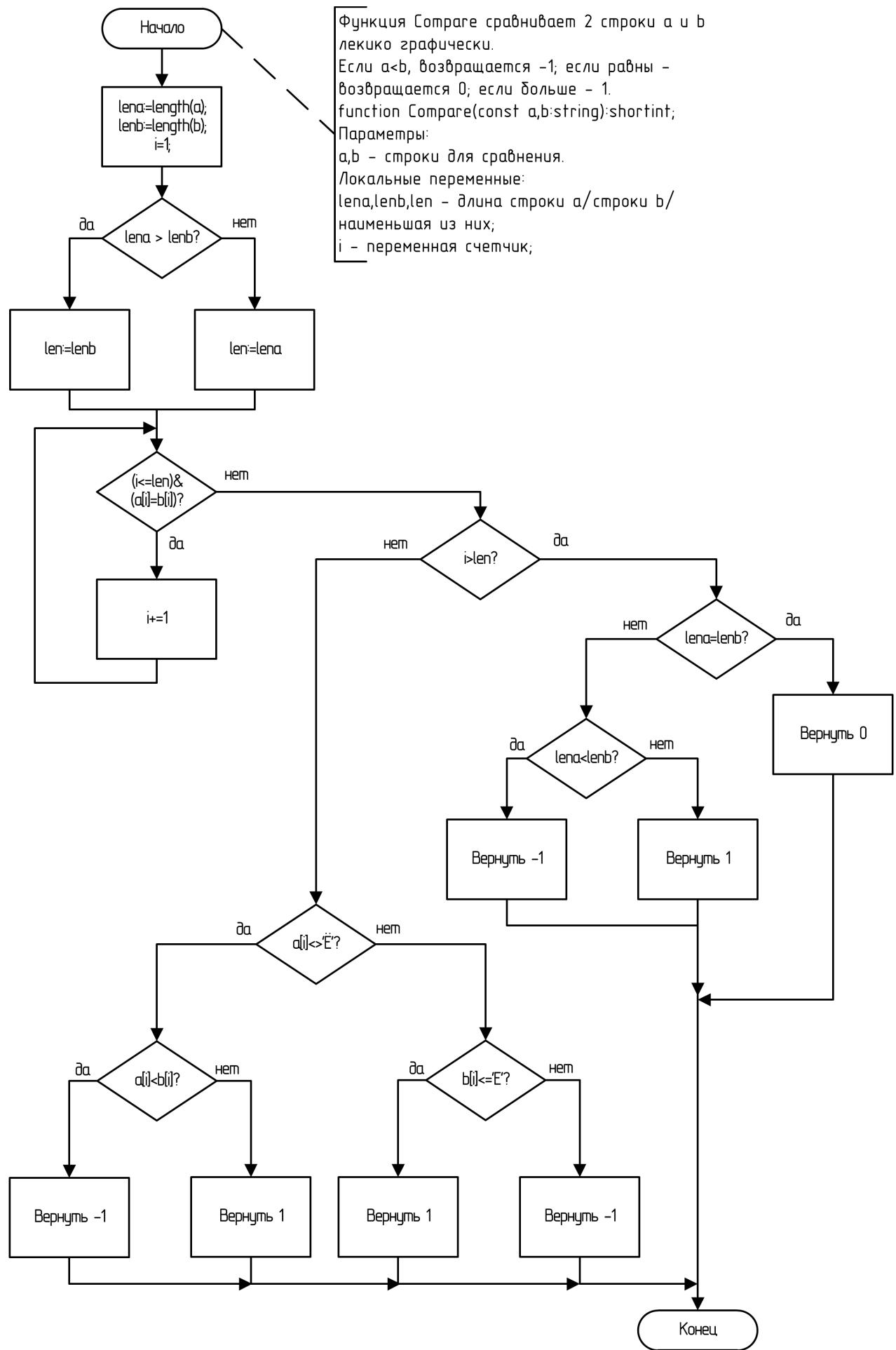


Рисунок 4.8 - Схема алгоритма лексикографического сравнения 2-х строк

На рисунке 4.9. рассмотрена схема алгоритма получения заглавной буквы из строчной для кодировки CP866.

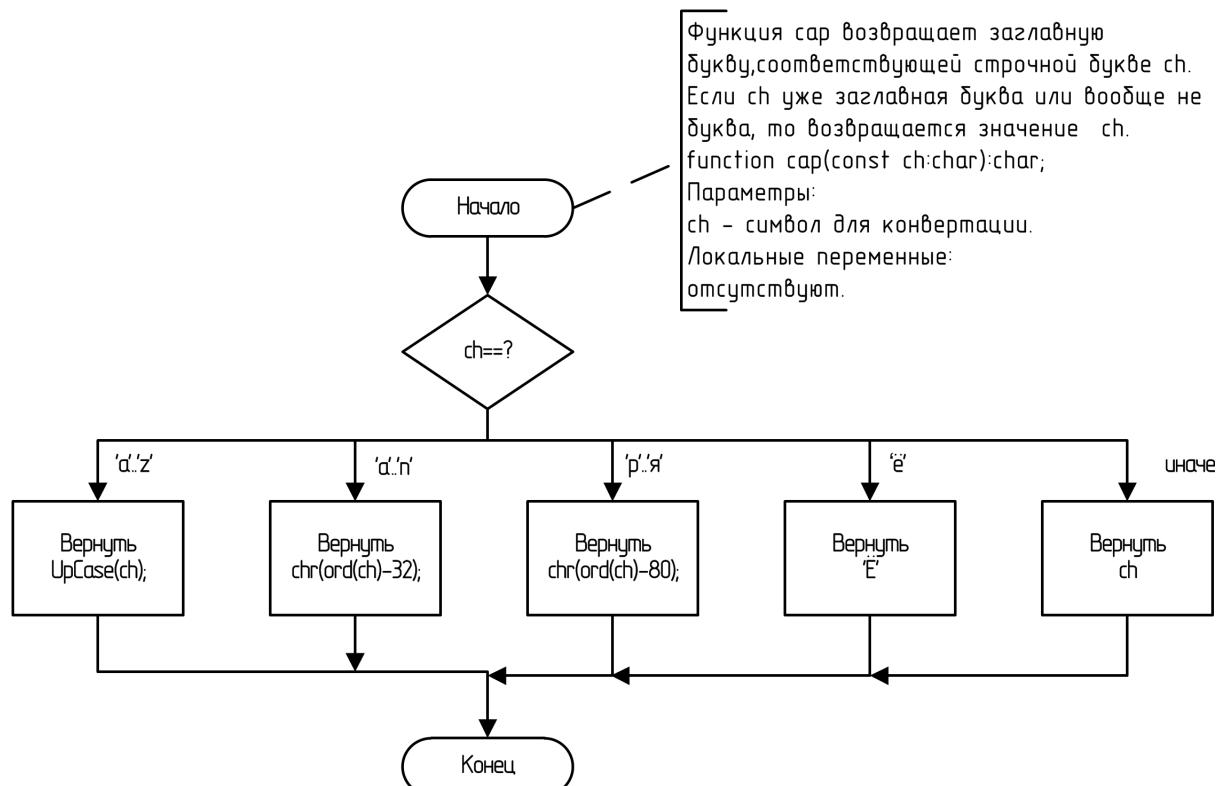


Рисунок 4.9 — Схема алгоритма получения заглавной буквы

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

4.6. Текст программы

Ниже представлен текст программы для составления алфавитного частотного словаря, написанной на языке Borland Pascal 7.0.

```
Program FreqDict;
CONST WordLen = 31;
TYPE TextWord = string[WordLen];
Node= RECORD
    key: TextWord;
    count: longint;
    left, right: LongInt
END;
TreeFile=file of Node;
(*++++++ПРОЦЕДУРЫ И ФУНКЦИИ+++++*)

(*
Функция GetReqResult позволяет получить ответ "да" или "нет" от
пользователя.
Возвращает true в случае согласия пользователя, false - в случае несогласия.
Параметры:
отсутствуют.
Локальные переменные:
answer - текущий ответ пользователя.
*)
function GetReqReslt:boolean;
var answer:char;
begin
    ReadLn(answer);
    while (UpCase(answer)<>'Y')
        and(UpCase(answer)<>'N') do begin
        WriteLn('Неправильный ответ! Допустимо:',#13#10,'Y - да; N -
нет.');
        ReadLn(answer);
    end;
    GetReqReslt:=UpCase(answer)='Y';
end;
(*
Функция IsLetter проверяет, является ли символ ch буквой.
Если да, возвращает true, иначе - false.
Параметры:
ch - символ для проверки.
Локальные переменные:
отсутствуют.
*)
function IsLetter(ch:char):boolean;
begin
    if ('A'<=ch) and (ch<='Z') or
       ('a'<=ch) and (ch<='z') or
       ('А'<=ch) and (ch<='п') or
       ('P'<=ch) and (ch<='ё')
      then IsLetter:=true
      else IsLetter:=false;
end;

(*
Функция IsInWordSymbol проверяет, является ли символ ch допустимым для
использования внутри слова.
Если да, возвращает true, иначе - false.
Параметры:
ch - символ для проверки.
Локальные переменные:
```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

    отсутствуют.
    *)
function IsInWordSymbol(ch:char) :boolean;
begin
    if ('0'<=ch) and (ch<='9')
        or (ch='''') or (ch='-') or (ch='_')
            then IsInWordSymbol:=true
            else IsInWordSymbol:=false;
end;

    /*
Функция сар возвращает заглавную букву, соответствующей строчной букве ch.
Если ch уже заглавная буква или вообще не буква, то возвращается значение
ch.

Параметры:
ch - символ для конвертации.

Локальные переменные:
отсутствуют.
*)

function cap(const ch:char):char;
begin
    case ch of
        'a'..'z':cap:=UpCase(ch);
        'а'..'п':cap:=chr(ord(ch)-32);
        'р'..'я':cap:=chr(ord(ch)-80);
        'ё':cap:='Ё';
        else cap:=ch;
    end;
end;

/*
Функция Compare сравнивает 2 строки a и b лексикографически.
Если a<b, возвращается -1; если равны - возвращается 0;
если больше - 1.

Параметры:
a,b - строки для сравнения.

Локальные переменные:
lena,lenb,len - длина строки a/строки b/наименьшая из них;
i - переменная счетчик;
*)
function Compare(const a,b:string):shortint;
var lena,lenb,len:byte;
    i:byte;
begin
    lena:=length(a); lenb:=length(b);
    if lena > lenb then len:=lenb else len:=lena;
    i:=1;
    while (i<=len) and (a[i]=b[i]) do begin
        inc(i);
    end;

    if i>len then begin
        if lena=lenb then
            Compare:=0
        else if lena<lenb then
            Compare:=-1
        else Compare:=1;
    end else begin
        if a[i] <>'Ё' then
            if a[i]<b[i] then
                Compare:=-1
            else Compare:=1
        else
            if b[i]<='Ё' then
                Compare:=1

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

    else
        Compare:=-1;
    end;

    (*
Процедура PrintTree печатает дерево словоформ в файле Tree в текстовый файл
outFile.
Является "оберткой" к функции рекурсивной печати RecPrint.
Параметры:
outFile - текстовый файл для вывода.
Tree - типизированный файл Tree с деревом словоформ.
Локальные переменные:
отсутствуют.
*)
PROCEDURE PrintTree(var Tree:TreeFile; var outFile:Text) ;

    (*
Процедура RecPrint с помощью рекурсии печатает дерево словоформ
в файле Tree (определенный глобально) в текстовый файл outFile
(определенный глобально).
Функция печатает элемент с номером Number:
сначала его левое поддерево, затем сам элемент, затем его правое
поддерево/
Параметры:
Number - номер текущего элемента для печати.
Локальные переменные:
Current - текущий элемент для печати.
*)
procedure RecPrint(Number: Longint);
var Current:Node;
BEGIN
    seek(Tree,Number); Read(Tree,Current);

    IF Current.left>0 Then begin
        RecPrint(Current.left);
    end;
    writeln(outFile, Current.key, ' ', Current.count);
    IF Current.right>0 Then begin
        RecPrint(Current.right);
    end;
END {RecPrint};

begin
    recprint(0);
end {PrintTree};

    (*
Процедура GetWord получает из файла inFile следующее слово, если оно есть.
Иначе возвращается пустая строка.
Параметры:
inFile - файл с исходным текстом.
Локальные переменные:
temp - текущее извлеченное слово;
i - длина извлеченного слова;
ch - текущий извлеченный символ;
founded - показывает, было ли найдено слово.
*)
function GetWord(var inFile:text):TextWord;
var temp:TextWord; founded:boolean; ch:char; i:integer;
begin
    temp:='';
    founded:=false; i:=0;
    while not (eof(inFile) or founded) do begin

```

Изм.	Лист	№ докум.	Подп.	Дата

```

Read(inFile, ch);
IF IsLetter(ch) THEN BEGIN
    founded:=true; inc(i); temp[i] := cap(ch); Read(inFile, ch);

    while not eof(inFile) and (i <= WordLen)
        and (IsLetter(ch) or IsInWordSymbol(ch)) do begin
            inc(i); temp[i] := cap(ch);
            Read(inFile, ch);

        end;
        if IsLetter(ch) or IsInWordSymbol(ch) then begin
            inc(i); temp[i] := cap(ch);
            end;
        temp[0] := chr(i); (* конец цепочки литер *)
    END;

    end;
    if founded then
        GetWord:=temp
    else
        GetWord:='';

end;

(*
Процедура MergeTree создает в типизированном файле Tree дерево словоформ,
находящихся в файле-источнике inFile.

Параметры:
inFile - файл с исходным текстом.
Tree - типизированный файл Tree с деревом словоформ.
Локальные переменные:
key - текущее извлеченное слово;
fileinit - показывает, было ли инициализировано дерево в файле Tree(true -
да);
FreeIndex - номер незанятого элемента в файле Tree;
*)
PROCEDURE MergeTree( VAR inFile: text; var Tree:TreeFile );

(
Процедура AddKey добавляет в дерево Tree ключ key,
при этом учитывается возможная необходимость инициализации
файла(fileinit),
и номер свободного для записи блока в файле(FreeIndex).
Параметры:
key - ключ, вставляемый в дерево,
fileinit - показывает, было ли инициализировано дерево в файле
Tree(true - да);
FreeIndex - номер незанятого элемента в файле Tree;
Tree - типизированный файл Tree с деревом словоформ.
Локальные переменные:
TempNode - создаваемый и вставляемый в дерево узел;
Rider,PrevRider - текущий и предыдущий номера просматриваемых узлов
дерева;
founded - показывает, был ли найден в дереве вставляемый ключ(т.е.
существует ли он уже);
status - показывает отошение сравниваемых ключей
(1 - вставляемый больше;
0 - равны;
-1 - вставляемый меньше;)

*)
PROCEDURE AddKey(var Tree:TreeFile; const key: TextWord;var
FreeIndex:longint;var fileinit:boolean);
var TempNode:Node;
Rider,PrevRider:longint; founded:boolean; status:shortint;
BEGIN

```

Изм.	Лист	№ докум.	Подп.	Дата

```

{**Root node position**}
if FileInit then begin {**File initialized**}
    {**Read root node**}
    Rider:=0;
    repeat
        PrevRider:=Rider;
        seek(Tree,Rider);Read(Tree,tempNode);
        status:=Compare(key,tempNode.key);
        Case status of
            1:Rider:=tempNode.right;
            -1:Rider:=tempNode.left;
            0:begin
                Rider:=0;
                end;
            end;
        until Rider<=0;
        if Rider=0 then begin
            inc(TempNode.Count);
            seek(Tree,PrevRider);write(Tree,tempNode);
        end else begin
            {**Prepare and write parent node**}

            Case Rider of
                -1:TempNode.right:=FreeIndex;
                -2:TempNode.left:=FreeIndex;
            End;
            seek(Tree,PrevRider);write(Tree,tempNode);
            {**Prepare and write new child node**}
            TempNode.key:=key; TempNode.count:=1;
            TempNode.right:=-1; TempNode.left:=-2;
            seek(Tree,FreeIndex);write(Tree,tempNode);
            inc(FreeIndex);
        end;
    end else begin
        {**Prepare and write root node**}

        TempNode.key:=key; TempNode.count:=1;
        TempNode.right:=-1; TempNode.left:=-2;
        seek(Tree,0);write(Tree,tempNode);
        FreeIndex:=1; fileinit:=true;
    end;
END {AddKey};

VAR
    key: TextWord; fileinit:boolean; FreeIndex:longint;
BEGIN
    FreeIndex:=0;fileinit:=false;
    key:=GetWord(inFile);
    WHILE (key<>'') DO begin
        AddKey(Tree,key,FreeIndex,fileinit);
        key:=GetWord(inFile);
    END;
END {MergeTree};

function TreeFileExists(var f:TreeFile):boolean;
begin
    {$I-}
    Reset(f);
    {$I+}
    if iore result=0 then begin

        TreeFileExists:=true;
        Close(f);
    end else TreeFileExists:=false

```

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

```

end {FileExist};

function FileExists(var f:text):boolean;
var exist:boolean;
begin
    {$I-}
        ReSet(f);
    {$I+}
    exist:=ioresult=0;
    if exist then close(f);
    FileExists:=exist;
end;
(*
Функция GetFiles получает файл и исходным текстом inFile,
временный файл для дерева Tree,
и файл для вывода outFile.
Возвращает true, если операция прошла успешно, иначе возвращается false.
Параметры:
inFile - файл с исходным текстом.
outFile - текстовый файл для вывода.
Tree - типизированный файл с деревом словоформ.
Локальные переменные:
ok - флаг отсутствия ошибки (true - нет ошибки).
*)

function GetFiles(var inFile:text; var Tree:TreeFile; var
outFile:text):boolean;
(*
Функция GetInFile() получает файл f для чтения.
Возвращает true, если операция прошла успешно, иначе возвращается
false.
Параметры:
f - запрашиваемый файл.
Локальные переменные:
error - флаг ошибки (true - ошибка);
req_rslt - флаг ответа пользователя (true - согласие пользователя).
*)
function GetInFile(var f:text):boolean;
var error,req_rslt:boolean;
    fileName:string;
begin
    error:=false; req_rslt:=false;
    repeat
        WriteLn('Введите имя файла с текстом для анализа.');
        Write('Файл:'); ReadLn(fileName);
        Assign(f,fileName);
        {$I-}
            Reset(f);
        {$I+}
        error:=(ioresult<>0)or(fileName='');
        if error then begin
            WriteLn('Неправильное имя файла! Повторить ввод?
<Y>/<N>');
            req_rslt:=GetReqReslt;
        end;
        until not error or not req_rslt;
        GetInFile:=not error;
end;

(*
Функция GetTmpFile() получает временный типизированный файл f.
Возвращает true, если операция прошла успешно, иначе возвращается
false.
Параметры:
f - запрашиваемый файл.
Локальные переменные:

```

Изм.	Лист	№ докум.	Подп.	Дата

```

        error - флаг ошибки (true - ошибка);
        req_rslt - флаг ответа пользователя (true - согласие пользователя).
        *)
    function GetTmpFile(var f:TreeFile):boolean;
const DefaultName='$temp.tmp';
var ch:char; error,req_rslt:boolean;
    fileName:string;
begin
    FileName:=DefaultName;
    repeat
        assign(f,fileName);
        If TreeFileExists(f) then begin
            WriteLn('Введите имя временного файла');
            Write('Файл:');ReadLn(fileName);
        end else begin
            {$I-}
            ReWrite(f);
            {$I+}
            error:=(ioresult<>0)or(fileName='');
            if error then begin
                WriteLn('Ошибка при создании файла! Повторить ввод?
<Y>/<N>');
                req_rslt:=GetReqReslt;
            end;
        end;
        until not req_rslt or not error;
    GetTmpFile:= not error;
end {GetTmpFile};

(*
Функция GetOutFile() получает файл f для записи.
Возвращает true, если операция прошла успешно, иначе возвращается
false.

Параметры:
f - запрашиваемый файл.

Локальные переменные:
error - флаг ошибки (true - ошибка);
req_rslt - флаг ответа пользователя (true - согласие пользователя).
*)
    function GetOutFile(var f:text):boolean;
var error,req_rslt:boolean;
    fileName:string;
begin
    error:=false; req_rslt:=false;
    repeat
        WriteLn('Введите имя файла для частотного словаря.');
        Write('Файл:');ReadLn(fileName);
        Assign(f,fileName);
        if FileExists(f) then begin
            error:=true;
            WriteLn('Указанный файл существует! Перезаписать?
(Y/N)');
            error:=not GetReqReslt;
        end;
        if not error then begin
            {$I-}
            ReWrite(f);
            {$I+}
            error:=(ioresult<>0)or(fileName='');
        end;
        if error then begin
            WriteLn('Ошибка при создании файла! Повторить ввод?
<Y>/<N>');
            req_rslt:=GetReqReslt;
        end;

```

Изм.	Лист	№ докум.	Подп.	Дата

```

        until not error or not req_rslt;
        GetOutFile:=not error;
    end;

var ok:boolean;
begin
    ok:=GetInFile(inFile);
    if ok then
        ok:=GetTmpFile(Tree);
    if ok then
        ok:=GetOutFile(outFile);
    GetFiles:=ok;
end;
(*-----ПРОЦЕДУРЫ И ФУНКЦИИ-----*)
VAR inFile, outFile: Text;
    Tree: TreeFile; ok:boolean;i:byte;
BEGIN
    WriteLn('Программа составляет алфавитный частотный словарь');
    WriteLn('из файла, записанного в текстовом файле,');
    WriteLn('и выводит словарь в выходной текстовый файл');
    ok:=GetFiles(inFile, Tree, outFile);
    if ok then begin
        writeln('Обработка входного текста... ');
        writeln('Пожалуйста, подождите... ');
        MergeTree(InFile,Tree);
        WriteLn;
        writeln('Запись результатов... ');
        writeln('Пожалуйста, подождите... ');
        PrintTree(Tree,outFile);
        Close(inFile); close(outFile);
        close(Tree); Erase(Tree);
        WriteLn;
        WriteLn('Все операции завершены успешно!');
    end else
        WriteLn('Работа программы прервана... ');
        WriteLn('Нажмите <Enter>...'); Readln;
END.

```

Изм.	Лист	№ докум.	Подп.	Дата

4.7. Инструкция программисту

Для создания программы составления алфавитного частотного словаря были предприняты следующие действия.

Объявлена константа WordLen – максимальная длина слова.

Объявлен тип TextWord — слово из текста - как string[WordLen];

Объявлен тип-запись Node – узел дерева. Описание его полей представлено в таблице 4.1.

Таблица 4.1 - Описание полей структуры "узел дерева"

имя	тип	предназначение
key	TextWord	словоформа - ключ
count	longint	количество слова в тексте
left, right	LongInt	номера левого и правого узлов-потомков в файле дерева.

В главной части программы объявлены структуры данных, описание которых приводится в таблице 4.2.

Таблица 4.2 - Описание переменных программы составления алфавитного частотного словаря

имя	тип	предназначение
inFile	text	файл с исходным текстом.
outFile	text	текстовый файл для вывода.
Tree	Treefile	типовизированный файл Tree с деревом словоформ.
ok	boolean	флаг отсутствия ошибки (true – нет ошибки).

Также программа была разбита на следующие подпрограммы.

1. Функция IsLetter проверяет, является ли символ ch буквой.

Если да, возвращает true, иначе – false.

function IsLetter(ch:char):boolean;

Параметры функции представлены в таблице 4.3:

Таблица 4.3 - Параметры функции получения варианта

имя	тип	предназначение
ch	char	символ для проверки.

Локальные переменные:

отсутствуют.

2. Функция IsInWordSymbol проверяет, является ли символ ch допустимым для использования внутри слова.

Если да, возвращает true, иначе - false.

function IsInWordSymbol(ch:char):boolean;

Параметры функции представлены в таблице 4.4:

Таблица 4.4 - Параметры функции получения варианта

имя	тип	предназначение
ch	char	символ для проверки.

Локальные переменные:

отсутствуют.

3. Функция cap возвращает заглавную букву, соответствующей строчной букве ch.

Если ch уже заглавная буква или вообще не буква, то возвращается значение ch.

function cap(const ch:char):char;

Параметры функции представлены в таблице 4.5:

Таблица 4.5 - Параметры функции получения варианта

имя	тип	предназначение
ch	char	символ для конвертации.

Локальные переменные:

отсутствуют.

4. Функция Compare сравнивает 2 строки a и b лексикографически.

Если a<b, возвращается -1; если равны - возвращается 0; если больше - 1.

function Compare(const a,b:string):shortint;

Параметры функции представлены в таблице 4.6:

Таблица 4.6 - Параметры функции получения варианта

имя	тип	предназначение
a,b	string	строки для сравнения.

Локальные переменные функции представлены в таблице 4.7:

Таблица 4.7 - Локальные переменные функции получения файла для записи

имя	тип	предназначение

lena,lenb,len	byte	длина строки a/строки b/наименьшая из них
i	byte	переменная счетчик;

5. Процедура PrintTree печатает дерево словоформ в файле Tree в текстовый файл outFile.

Является "оберткой" к функции рекурсивной печати RecPrint.

PROCEDURE PrintTree(var Tree:TreeFile; var outFile:Text);

Параметры функции представлены в таблице 4.8:

Таблица 4.8 - Параметры функции получения варианта

имя	тип	предназначение
outFile	text	текстовый файл для вывода.
Tree	TreeFile	типовизированный файл Tree с деревом словоформ.

Локальные переменные:

отсутствуют.

6. Процедура RecPrint с помощью рекурсии печатает дерево словоформ в файле Tree (определенный глобально) в текстовый файл outFile (определенный глобально).

Функция печатает элемент с номером Number: сначала его левое поддерево, затем сам элемент, затем его правое поддерево.

procedure RecPrint(Number: Longint);

Параметры функции представлены в таблице 4.9:

Таблица 4.9 - Параметры функции получения варианта

имя	тип	предназначение
Number	LongInt	номер текущего элемента для печати.

Локальные переменные функции представлены в таблице 4.10:

Таблица 4.10 - Локальные переменные функции получения файла для записи

имя	тип	предназначение
Current	Node	текущий элемент для печати.

7. Процедура GetWord получает из файла inFile следующее слово, если оно есть.

Иначе возвращается пустая строка.

```
function GetWord(var inFile:text):TextWord;
```

Параметры функции представлены в таблице 4.11:

Таблица 4.11 - Параметры функции получения варианта

имя	тип	предназначение
inFile	text	файл с исходным текстом.

Локальные переменные функции представлены в таблице 4.12:

Таблица 4.12 - Локальные переменные функции получения файла для записи

имя	тип	предназначение
temp	TextWord	текущее извлеченное слово;
i	byte	длина извлеченного слова;
ch	char	текущий извлеченный символ;
founded	boolean	показывает, было ли найдено слово.

8. Процедура MergeTree создает в типизированном файле Tree дерево словоформ, находящихся в файле-источнике inFile.

```
PROCEDURE MergeTree( VAR inFile: text; var Tree:TreeFile );
```

Параметры функции представлены в таблице 4.13:

Таблица 4.13 - Параметры функции получения варианта

имя	тип	предназначение
inFile	text	файл с исходным текстом.
Tree	TreeFile	типовизированный файл Tree с деревом словоформ.

Локальные переменные функции представлены в таблице 4.14:

Таблица 4.14 - Локальные переменные функции получения файла для записи

имя	тип	предназначение
key	TextWord	текущее извлеченное слово;
fileinit	boolean	показывает, было ли инициализировано дерево в файле Tree(true – да);
FreeIndex	longint	номер незанятого элемента в файле Tree;

9. Процедура AddKey добавляет в дерево Tree ключ key, при этом учитывается возможная необходимость инициализации файла(fileinit), и номер свободного для записи блока в файле(FreeIndex).

```
PROCEDURE AddKey(var Tree:TreeFile; const key: TextWord; var FreeIndex:longint; var fileinit:boolean);
```

Параметры функции представлены в таблице 4.15:

Изм.	Лист	№ докум.	Подп.	Дата

Таблица 4.15 - Параметры функции получения варианта

имя	тип	предназначение
key	TextWord	ключ, вставляемый в дерево,
fileinit	boolean	показывает, было ли инициализировано дерево в файле Tree(true – да);
FreeIndex	Longint	номер незанятого элемента в файле Tree;
Tree	Treefile	типовизированный файл Tree с деревом словоформ.

Локальные переменные функции представлены в таблице 4.16:

Таблица 4.16 - Локальные переменные функции получения файла для записи

имя	тип	предназначение
TempNode	Node	создаваемый и вставляемый в дерево узел;
Rider, PrevRider	longint	текущий и предыдущий номера просматриваемых узлов дерева;
founded	boolean	показывает, был ли найден в дереве вставляемый ключ(т.е. существует ли он уже);
status	shortint	показывает отошение сравниваемых ключей (1 – вставляемый больше; 0 – равны; 1 – вставляемый меньше;)

10. Функция GetFiles получает файл и исходным текстом inFile, временный файл для дерева Tree, и файл для вывода outFile.

Возвращает true, если операция прошла успешно, иначе возвращается false.

function GetFiles(var inFile:text; var Tree:TreeFile; var outFile:text):boolean;

Параметры функции представлены в таблице 4.17:

Таблица 4.17 - Параметры функции получения варианта

имя	тип	предназначение
inFile	text	файл с исходным текстом.
outFile	text	текстовый файл для вывода.
Tree	TreeFile	типовизированный файл с деревом словоформ.

Локальные переменные функции представлены в таблице 4.18:

Таблица 4.18 - Локальные переменные функции получения файла для записи

имя	тип	предназначение
ok	boolean	флаг отсутствия ошибки (true – нет ошибки).

11. Функция GetTmpFile() получает временный типизированный файл f.
Возвращает true, если операция прошла успешно, иначе возвращается false.

function GetTmpFile(var f:TreeFile):boolean;

Параметры функции представлены в таблице 4.19:

Таблица 4.19 - Параметры функции получения типизированного файла

имя	тип	предназначение
f	TreeFile	запрашиваемый файл.

Локальные переменные функции представлены в таблице 4.20:

Таблица 4.20 - Локальные переменные функции получения типизированного файла

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

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

Возвращает true, если операция прошла успешно, иначе возвращается false.

function GetInFile(var f:text):boolean;

Параметры функции представлены в таблице 4.21:

Таблица 4.21 - Параметры функции получения файла для чтения

имя	тип	предназначение
f	text	запрашиваемый файл.

Локальные переменные функции представлены в таблице 4.22:

Таблица 4.22 - Локальные переменные функции получения файла для чтения

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

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

Возвращает true, если операция прошла успешно, иначе возвращается false.

function GetOutFile(var f:text):boolean

Параметры функции представлены в таблице 4.23:

Таблица 4.23 - Параметры функции получения файла для записи

имя	тип	предназначение

f	text	запрашиваемый файл.
---	------	---------------------

Локальные переменные функции представлены в таблице 4.24:

Таблица 4.24 - Локальные переменные функции получения файла для записи

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

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

Заголовок функции:

```
function GetReqReslt:boolean;
```

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

Локальные переменные функции представлены в таблице 4.25.

Таблица 4.25 - Локальные переменные функции получения ответа

имя	тип	предназначение
answer	char	Ответ пользователя.

4.8. Инструкция пользователю

Данная программа создает частотный словарь некоторого литературного текста, то есть алфавитный перечень слов (словоформ), встречающихся в тексте с указанием того, сколько раз входит в текст данное слово (словоформа).

Для работы программы надо указать имя файла с исходным текстом в кодировке CP866, и имя файла для вывода. При существовании файла для вывода будет выдан запрос перезаписи. При перезаписи информация в файле будет стерта. Также может потребоваться ввести имя временного файла.

После окончания работы программа выведет в файл для вывода требуемый словарь.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

4.9. Тестовый пример

Составим алфавитный частотный словарь для некоторого отрывка из произведения Л.Н. Толстого "Война и мир":

"Между тем все это молодое поколение: Борис -- офицер, сын княгини Анны Михайловны, Николай -- студент, старший сын графа, Соня -- пятнадцатилетняя племянница графа, и маленький Петруша -- меньшой сын, все разместились в гостиной и, видимо, старались удержать в границах приличия оживление и веселость, которыми еще дышала каждая их черта. Видно было, что там, в задних комнатах, откуда они все так стремительно прибежали, у них были разговоры веселее, чем здесь о городских сплетнях, погоде и comtesse Apraksine. Изредка они взглядывали друг на друга и едва удерживались от смеха."

Пример работы программы составления алфавитного частотного словаря представлен на рисунке 4.10.

```
Программа составляет алфавитный частотный словарь
из файла, записанного в текстовом файле,
и выводит словарь в выходной текстовый файл
Введите имя файла с текстом для анализа.
Файл:voinap.txt
Введите имя файла для частотного словаря.
Файл:dict.txt
Обработка входного текста...
Пожалуйста, подождите...

Запись результатов...
Пожалуйста, подождите...

Все операции завершены успешно!
Нажмите <Enter>...
```

Рисунок 4.10 - Пример работы программы составления алфавитного частотного словаря

Рассмотрим ниже содержимое файла dict.txt (дано в 4 колонки для экономии места).

Изм.	Лист	№ докум.	Подп.	Дата

APRAKSINE 1	ДЫШАЛА 1	НИХ 1	СПЛЕТНЯХ 1
COMTESSE 1	ЕДВА 1	О 1	СТАРАЛИСЬ 1
АННЫ 1	ЕЩЕ 1	ОЖИВЛЕНИЕ 1	СТАРШИЙ 1
БОРИС 1	ЗАДНИХ 1	ОНИ 2	СТРЕМИТЕЛЬНО 1
БЫЛИ 1	ЗДЕСЬ 1	ОТ 1	СТУДЕНТ 1
БЫЛО 1	И 5	ОТКУДА 1	СЫН 3
В 3	ИЗРЕДКА 1	ОФИЦЕР 1	ТАК 1
ВЕСЕЛЕЕ 1	ИХ 1	ПЕТРУША 1	ТАМ 1
ВЕСЕЛОСТЬ 1	КАЖДАЯ 1	ПЛЕМЯННИЦА 1	ТЕМ 1
ВЗГЛЯДЫВАЛИ 1	КНЯГИНИ 1	ПОГОДЕ 1	У 1
ВИДИМО 1	КОМНАТАХ 1	ПОКОЛЕНИЕ 1	УДЕРЖАТЬ 1
ВИДНО 1	КОТОРЫМИ 1	ПРИБЕЖАЛИ 1	УДЕРЖИВАЛИСЬ 1
ВСЕ 3	МАЛЕНЬКИЙ 1	ПРИЛИЧИЯ 1	ЧЕМ 1
ГОРОДСКИХ 1	МЕЖДУ 1	ПЯТНАДЦАТИЛЕТНЯ ЧЕРТА 1	
ГОСТИНОЙ 1	МЕНЬШОЙ 1	Я 1	ЧТО 1
ГРАНИЦАХ 1	МИХАЙЛОВНЫ 1	РАЗГОВОРЫ 1	ЭТО 1
ГРАФА 2	МОЛОДОЕ 1	РАЗМЕСТИЛИСЬ 1	
ДРУГ 1	НА 1	СМЕХА 1	
ДРУГА 1	НИКОЛАЙ 1	СОНЯ 1	

Проверив, что все слова присутствуют и количество их вхождений совпадает с данными исходного текста, убеждаемся, что программа работает верно.

Заключение

Выполняя данную работу, я закрепил свои знания и умения в программировании на процедурных языках программирования - Turbo Pascal и Borland C, а также в создании схем алгоритмов. При написании программ я использовал операторы условия, операторы цикла, массивы, записи и структуры, файлы текстовые и типизированные, динамическую память, списки и деревья, а также и многие другие средства, предоставляемые данными замечательными языками. Работая, я применял различные математические и алгоритмические методы. Были написаны три полнофункциональные программы. Первая программа находит значение функции, заданной графически, на интервале, заданном пользователем. Вторая программа вычисляет таблицу значений функции, заданной в виде разложения в ряд. Третья программа предназначена для сортировки главной диагонали матрицы. Четвертая составляет алфавитный частотный словарь. Все задачи имеют различную сложность, но тем не менее, были решены одинаково успешно.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Список используемых источников

1. Абрамов А.Г., Трифонов Н.П., Трифонова Г.Н. Введение в язык Паскаль. М., Наука, 1988.
2. Епанешников А., Епанешников В. Программирование в среде Turbo Pascal 7.0, М., Диалог-Мифи, 1993
3. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. М.: "Нолидж", 2000.
4. Рапаков Г.Г., Ржеуцкая С.Ю. Turbo Pascal для студентов и школьников – Спб.:БХВ-Петербург, 2007.
5. Информатика: Учебник. - 3-е перераб. изд. /под ред. Н.В. Макаровой – М.:Финансы и статистика, 2005.

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

Вариант №4

Лист

161