

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**

## Содержание

Содержание.....	4
Введение.....	6
1. Вычисление определенного интеграла для функции, заданной графически.....	13
1.1. Постановка задачи.....	13
1.2. Математическая формулировка задачи.....	13
1.3. Метод левых прямоугольников.....	13
1.4. Разработка структур данных.....	16
1.5. Схема алгоритма.....	17
1.6. Инструкция пользователю.....	20
1.7. Инструкция программисту.....	21
1.8. Текст программы.....	22
1.9. Тестовый пример.....	25
2. Вычисление таблицы значений функции, заданной в виде разложения в ряд.....	27
2.1. Постановка задачи.....	27
2.2. Математическая формулировка задачи.....	27
2.3. Численный метод решения задачи.....	27
2.4. Разработка структур данных.....	28
2.5. Схема алгоритма.....	29
2.6. Текст программы.....	32
2.7. Инструкция программисту.....	34
2.8. Инструкция пользователю.....	34
2.9. Тестовый пример.....	36
3. Вычисление средних арифметических значений положительных элементов каждой строки матрицы.....	39
3.1. Постановка задачи.....	39
3.2. Математическая формулировка задачи.....	39

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. Численный метод решения задачи.....	39
3.4. Разработка структур данных.....	39
3.5. Схема алгоритма.....	40
3.6. Текст программы.....	43
3.7. Инструкция программисту.....	45
3.8. Инструкция пользователю.....	46
3.9. Тестовый пример.....	47
<b>Заключение.....</b>	<b>49</b>
<b>Список используемых источников.....</b>	<b>50</b>

## **Введение**

Войти в ХХI 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.

Программирование — одно из направлений информатики. Информатика (нем. Informatik, фр. Informatique, англ. computer science — компьютерная наука — в США, англ. computing science — вычислительная наука — в Великобритании) — наука о способах получения, накопления, хранения, преобразования, передачи и использования информации. Она включает дисциплины, относящиеся к обработке информации в вычислительных машинах и вычислительных сетях: как абстрактные, вроде анализа алгоритмов, так и довольно конкретные, например, разработка языков программирования.

Для того, чтобы научиться программированию, сначала нужно овладеть навыками алгоритмизации. Компьютер — сложное электронное устройство, но его "мозг" — центральный процессор — умеет выполнять лишь простейшие команды. Поэтому для решения сложных задач обработки информации программист должен составить алгоритм — подробное описание последовательности арифметических и логических действий, расположенных в строгом логическом порядке и позволяющих решить конкретную задачу. Составление такого пошагового описания процесса решения задачи и называется алгоритмизацией. Слово алгоритм, по существу, является синонимом таких слов, как способ, рецепт и т. п.

Требования, предъявляемые к алгоритму:

Изм.	Лист	№ докум.	Подп.	Дата

**Вариант №4**

Лист

4

- однозначность (понятность) — предлагаемые действия должны быть "понятны" компьютеру, а порядок исполнения этих действий должен быть единственным возможным, любая неопределенность или двусмысленность недопустимы;
- массовость — пригодность алгоритма для решения не только данной задачи, а множества родственных задач, относящихся к общему классу;
- детерминированность — повтор результатата при повторе исходных данных;
- корректность — способность алгоритма давать правильные результаты решения задачи при различных исходных данных;
- дискретность — алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов. При этом для выполнения каждого шага алгоритма требуется конечный отрезок времени, то есть преобразование исходных данных в результат осуществляется во времени дискретно;
- результативность — решение задачи должно быть получено за конечное число шагов алгоритма, "зацикливание" недопустимо;
- эффективность — для успешного решения задачи должны использоваться ограниченные ресурсы конкретного компьютера (время работы процессора, объем оперативной памяти, быстродействие жесткого диска и др.).

Естественно, что для каждой задачи необходим свой собственный способ решения, однако было замечено, что практически любой алгоритм можно разбить на меньшие алгоритмы, схожие между собой. Таких базовых алгоритмических структур можно выделить 3 типа:

- алгоритмические структуры, которые выполняются по очереди в порядке записи, — следование. Программную реализацию такой алгоритмической структуры называют линейной программой. Линейные программы обычно предназначены для решения простейших задач, в которых не предусмотрен выбор из нескольких возможных направлений хода программы или циклическое повторение операций.

Иzm.	Лист	№ докум.	Подп.	Дата

- возможность альтернативного выбора при выполнении программы представляют ветвления при выполнении которых алгоритм может пойти по одной из двух возможных ветвей в зависимости от справедливости проверяемого условия. Иногда выделяют также обход, который представляет собой пропуск нескольких шагов алгоритма при выполнении или невыполнении какого-либо условия.

- цикл представляет собой многократно повторяющуюся последовательность шагов алгоритма.

Комбинируя эти три базовых типа алгоритмов, можно решить практически любую задачу. Однако для «рождения» программы алгоритма недостаточно — он лишь «дух» программы. Необходимо теперь «воплотить» алгоритм. Как известно, команды, выполняемые процессором ПК, являются электрическими сигналами, которые можно представить в виде последовательностей нулей и единиц. Каждой команде соответствует свое число. Таким образом, процессор имеет дело с машинным кодом. Написать программу на нем может только очень опытный программист, хорошо знающий архитектуру процессора (его устройство) и систему команд (набор допустимых инструкций). Большинство программ создаются при помощи "посредников", в качестве которых выступают языки программирования высокого уровня.

Совокупность средств и правил представления алгоритма в виде, пригодном для выполнения вычислительной машиной, называется языком программирования.

Поэтому можно сказать, что программа — это запись (реализация) алгоритма на языке программирования.

В процессе создания любой программы можно выделить несколько этапов.

- Постановка задачи — выполняется специалистом в предметной области на естественном языке (русском, английском и т. д.). Необходимо определить цель задачи, ее содержание и общий подход к решению. Возможно, что задача решается точно (аналитически), и без компьютера можно обойтись. Уже на этапе постановки надо учитывать эффективность алгоритма решения задачи на ЭВМ,

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

ограничения, накладываемые аппаратным и программным обеспечением (АО и ПО).

- Анализ задачи и моделирование — определяются исходные данные и результат, выявляются ограничения на их значения, выполняется формализованное описание задачи и построение (выбор) математической модели, пригодной для решения на компьютере.
- Разработка или выбор алгоритма решения задачи — выполняется на основе ее математического описания. Многие задачи можно решить различными способами. Программист должен выбрать оптимальное решение. Неточности в постановке, анализе задачи или разработке алгоритма могут привести к скрытой ошибке — программист получит неверный результат, считая его правильным.
- Проектирование общей структуры программы — формируется модель решения с последующей детализацией и разбивкой на подпрограммы, определяется "архитектура" программы, способ хранения информации (набор переменных, массивов и т. п.).
- Кодирование — запись алгоритма на языке программирования.
- Отладка и тестирование программы. Под отладкой понимается устранение ошибок в программе. Тестирование позволяет вести их поиск и, в конечном счете, убедиться в том, что полностью отлаженная программа дает правильный результат. Тестирование должно охватывать все возможные ветвления в программе, т. е. проверять все ее инструкции, и включать такие исходные данные, для которых решение невозможно. Проверка особых, исключительных ситуаций, необходима для анализа корректности. В ответственных проектах большое внимание уделяется так называемой "защите от дурака", подразумевающей устойчивость программы к неумелому обращению пользователя. Использование специальных программ — отладчиков, которые позволяют выполнять программу по отдельным шагам, просматривая при этом значения переменных, значительно упрощает этот этап.
- Анализ результатов — если программа выполняет моделирование какого-либо известного процесса, следует сопоставить результаты вычислений с

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

результатами наблюдений. В случае существенного расхождения необходимо изменить модель.

- Публикация результатов работы, передача заказчику для эксплуатации.
- Сопровождение программы — включает консультации представителей заказчика по работе с программой и обучение персонала. Недостатки и ошибки, замеченные в процессе эксплуатации, должны устраняться.

Как видно, запись программы — лишь один из многих этапов её разработки. Однако выбор на данном этапе языка программирования для записи текста программы может оказать значительное влияние на все последующие.

Как известно, языки программирования могут быть реализованы как компилируемые и интерпретируемые.

Программа на компилируемом языке при помощи специальной программы компилятора преобразуется (компилируется) в набор инструкций для данного типа процессора (машинный код) и далее записывается в исполняемый модуль, который может быть запущен на выполнение как отдельная программа. Другими словами, компилятор переводит исходный текст программы с языка программирования высокого уровня в двоичные коды инструкций процессора. К достоинствам этого типа можно отнести более высокую скорость выполнения программ.

Если программа написана на интерпретируемом языке, то интерпретатор непосредственно выполняет (интерпретирует) исходный текст без предварительного перевода. Выполнение по шагам сильно упрощает отладку и тестирование программы. При этом программа остаётся на исходном языке и не может быть запущена без интерпретатора. Можно сказать, что процессор компьютера — это интерпретатор машинного кода.

Язык, выбранный для написания задач данной работы — 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 задуманный как язык для обучения, и на сегодняшний день остается одним из самых удобных средств для

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

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.

Как уже было сказано, в ходе данной работы на языке Паскаль были разработаны 3 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) нахождение среднего арифметического положительных элементов каждой строки матрицы  $A(C,D)$ ,  $C \leq 50$ ,  $D \leq 100$ .

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.

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

Для решения первой задачи используется метод разбиения на прямоугольники по итерационной формуле левых прямоугольников. Вычисления надо закончить при выполнении условия  $|I_n - I_{2n}| < \epsilon$ , где  $\epsilon > 0$  – 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.

В третьей задаче необходимо найти среднее арифметическое положительных элементов каждой строки матрицы. Для этого необходимо пройти по всем строкам матрицы, находя в каждой количество и сумму положительных элементов. Затем сумму этих элементов надо поделить на их количество и записать это значение во вспомогательный массив.

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

Итого, в работе 48 страниц, 11 рисунков, 3 таблицы и в общей сложности около 51000 символов.

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  $\epsilon$  на интервале  $[a, b]$  для функции, заданной на рисунке 1.1.

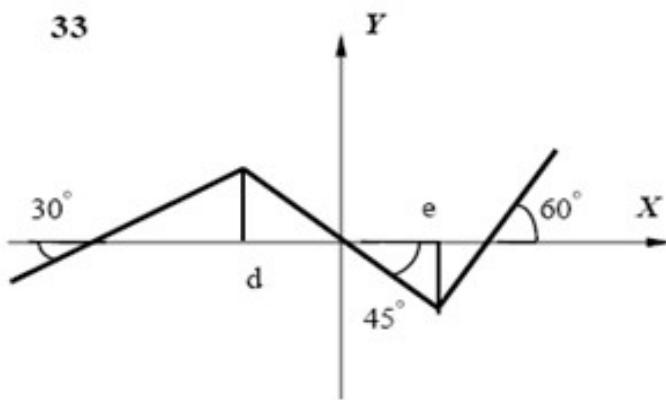


Рисунок 1.1 – График анализируемой функции

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  $-c$ .

Наклон самой левой линии к оси ОХ равен  $60^\circ$ , значит, прямая параллельная ей и проходящая через начало координат, задается уравнением  $y = \sqrt{3}x$ . Сама же

Изм.	Лист	№ докум.	Подп.	Дата

линия сдвинута от нее вверх и влево на  $-d$ , значит, она задается уравнением  $y = \sqrt{3}(x + (-d)) + (-d) = \sqrt{3}(x - d) - d$ . Проводя аналогичные рассуждения для правой части функции, получаем, что она является сдвинутой вправо-вниз функцией

$y = \frac{\sqrt{3}}{3}$ , и соответственно, задается уравнением  $y = \frac{\sqrt{3}}{3}(x - c) - c$ .

Итак, запишем уравнение кусочной функции:

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

### 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:

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

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

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

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, что площадь криволинейной трапеции будет меньше, так как  $(|x|-1)^2$  на интервале  $[-2,2]$  есть сдвинутый вправо и отраженный относительно оси ОУ кусочек

Изм.	Лист	№ докум.	Подп.	Дата

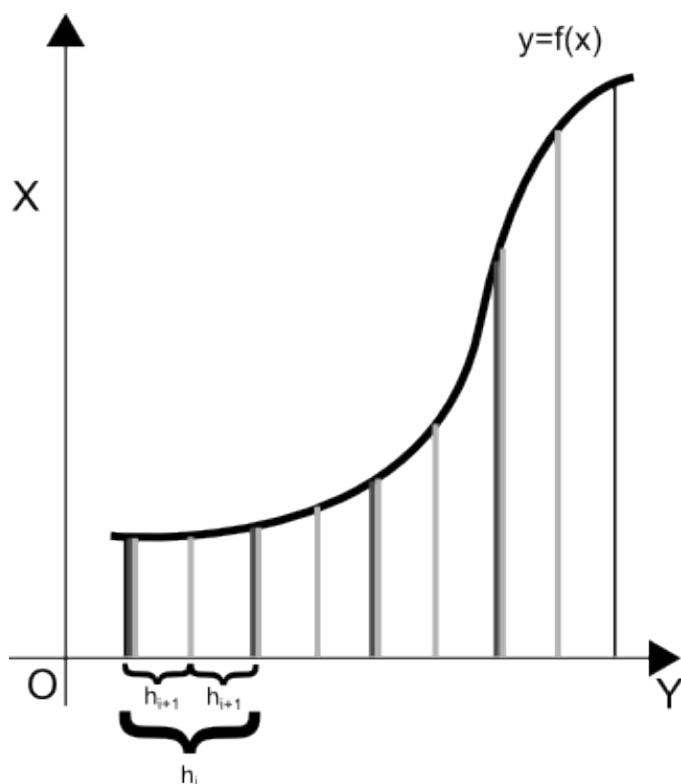
параболы [-1;1], т. е. отрезок функции получается вписанным в прямоугольник 2x4 (который получается на первой итерации) и делит его на две части, из которых нижняя и является значением интеграла. Итак программа работает неправильно. Для решения этой проблемы можно задать некоторое минимальное количество 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:

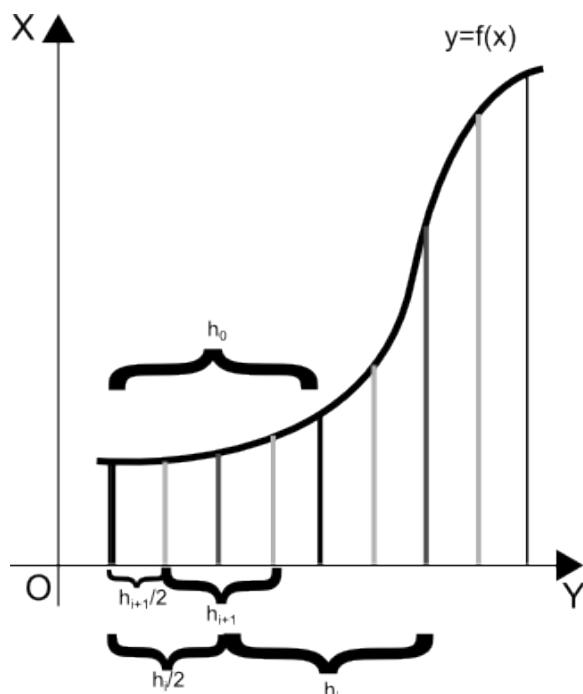
$I_n = \frac{I_{n-1}}{2} + \frac{h_n}{2} \cdot [f(x_0) + f(x_1) + \dots + f(x_{n-1})]$ . Наглядно идею можно понять, рассмотрев рисунки 1.3 и 1.4.

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 1.3 — 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 1.4 - 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 — параметр анализируемой функции, определяет функцию на самом левом отрезке, где она имеет вид  $y=d$ ;

r — также параметр анализируемой функции, определяет функцию на центральном отрезке, где она имеет вид  $y = -\sqrt{R^2 - x^2}$  (нижняя полуокружность);

c — последний параметр анализируемой функции, определяет функцию на правом отрезке, где она имеет вид  $y=c$  ;

a — нижняя граница интервала интегрирования;

b — соответственно, верхняя граница интервала интегрирования;

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

n — количество разбиений отрезка интегрирования, для которых ведется расчет значения функции (так как функция рассчитывается на каждом втором отрезке, то реальное количество разбиений в два раза больше);

h - текущий шаг изменения аргумента анализируемой функции

res0 — значение интеграла на предыдущей итерации;

res - 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.

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

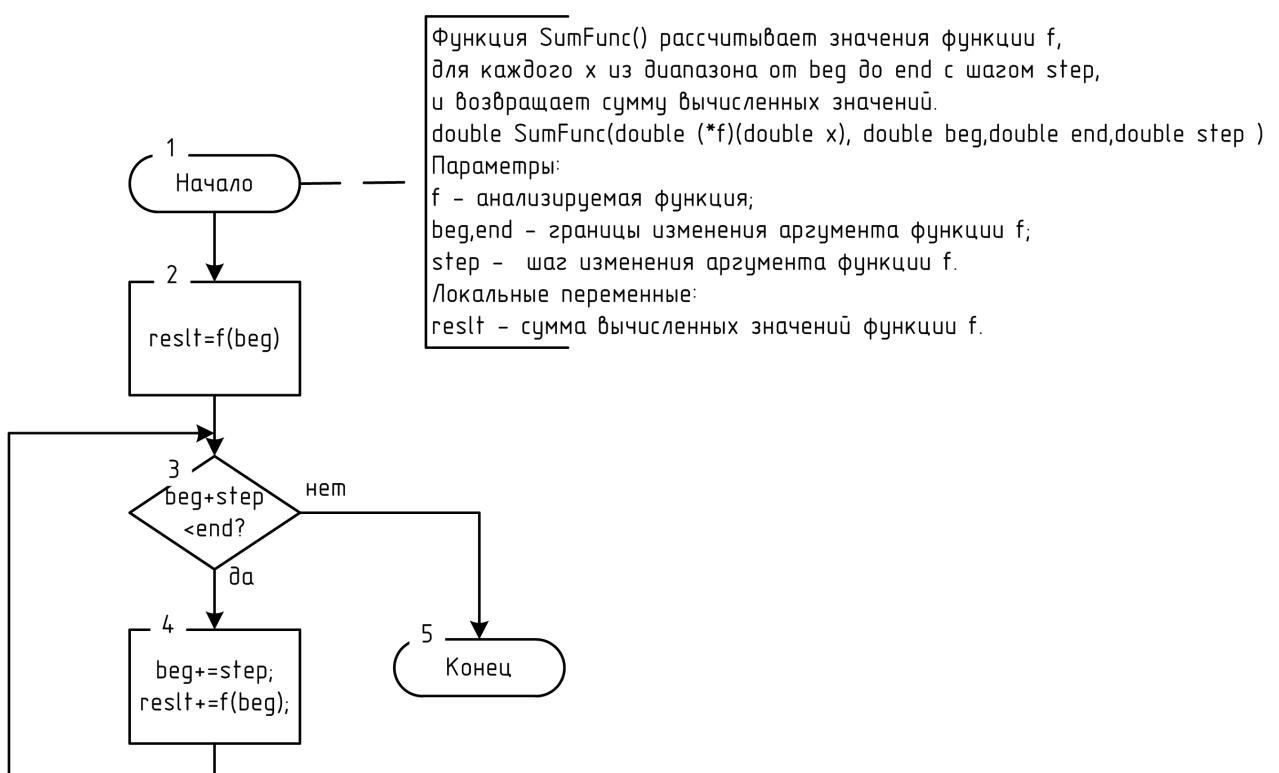
code - сюда записывается значение кода ошибки при извлечении данных из буфера , если ошибки нет, то code равно 0.

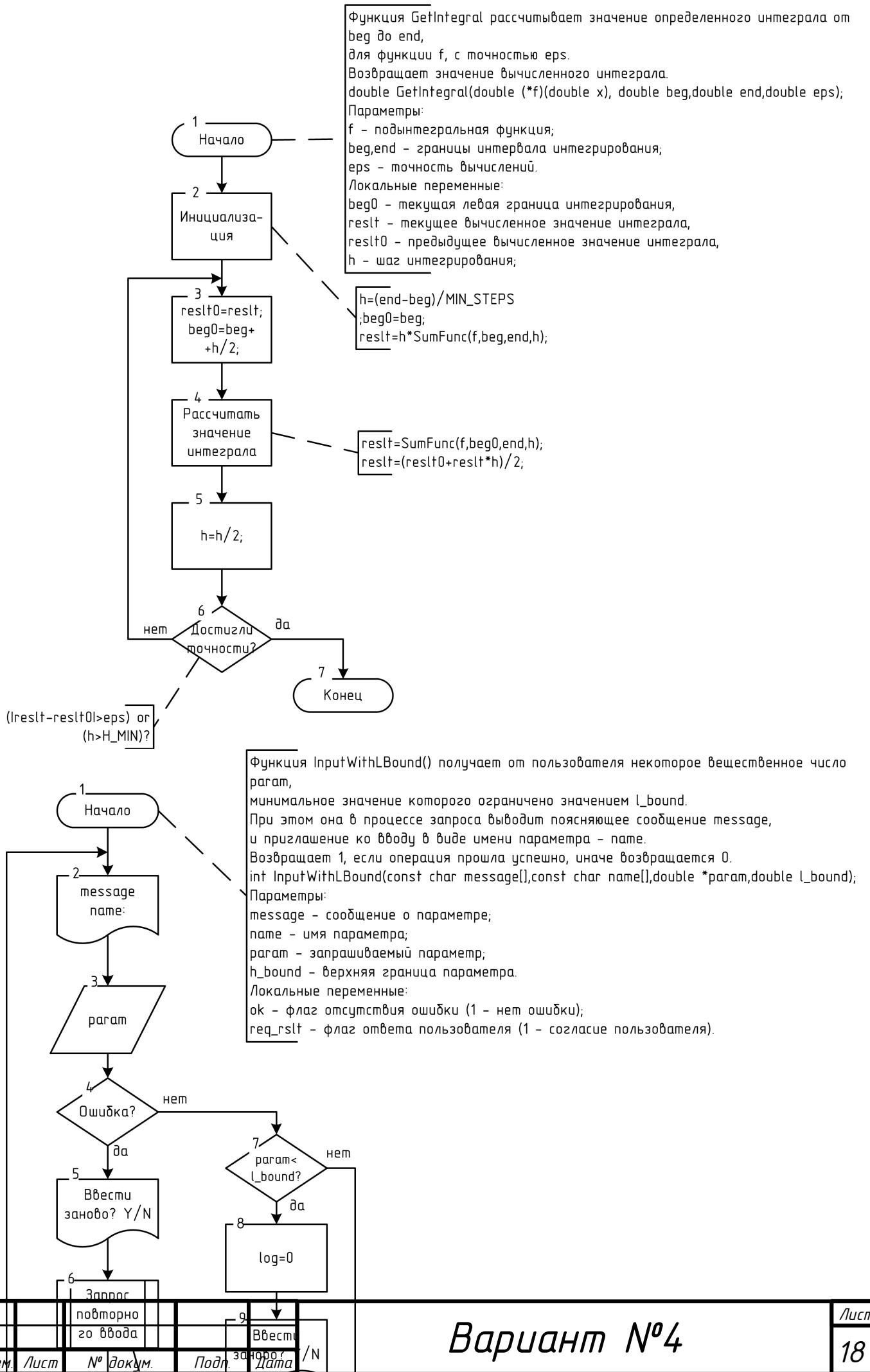
ch - содержит ответ пользователя на запрос программы о завершении, повторном вводе данных и завершении работы.

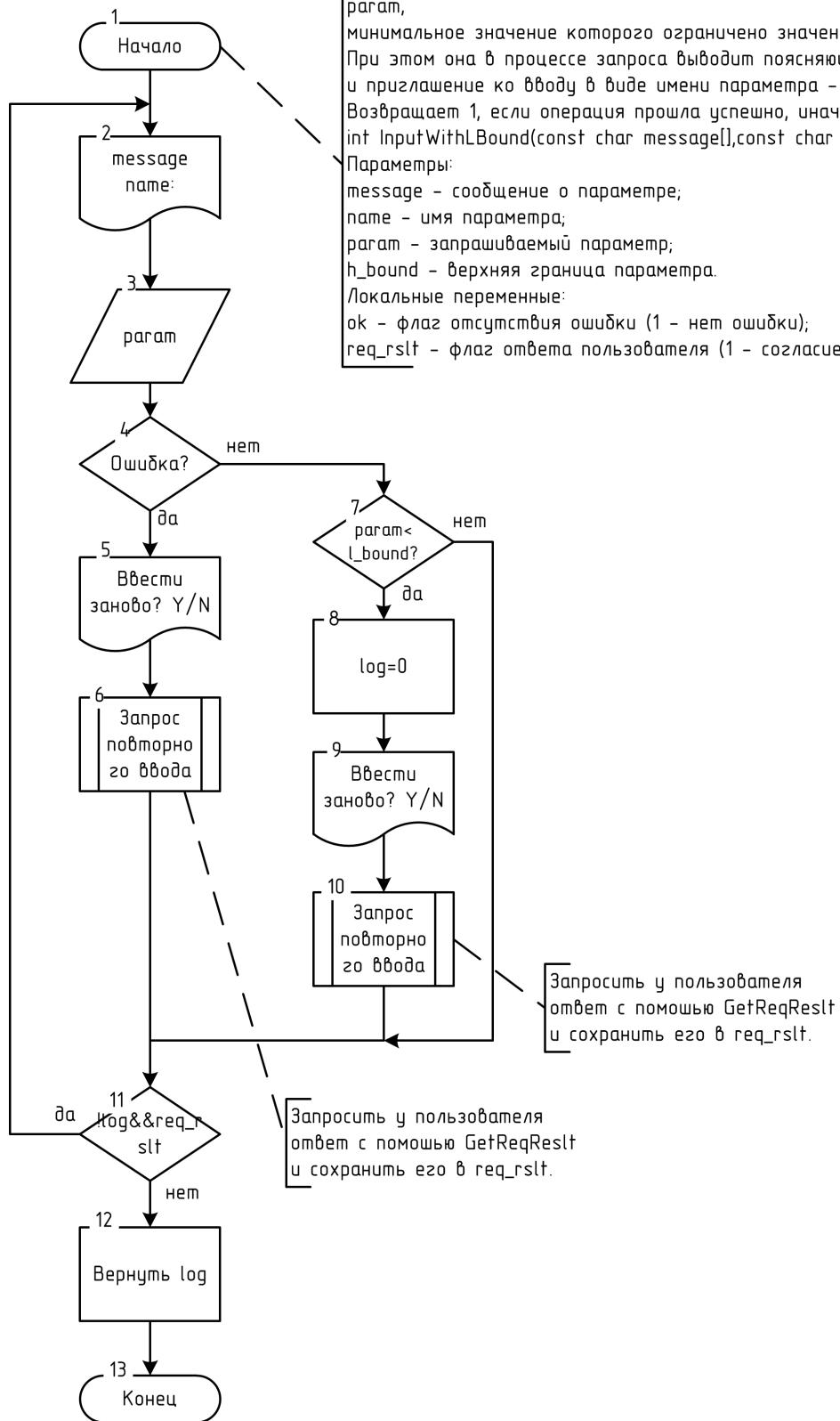
Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

## 1.5. Разработка структуры алгоритма решения задачи

На рисунках 1.2 и 1.3 представлен алгоритм вычисления интеграла функции (1.2) по формуле левых прямоугольников.







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

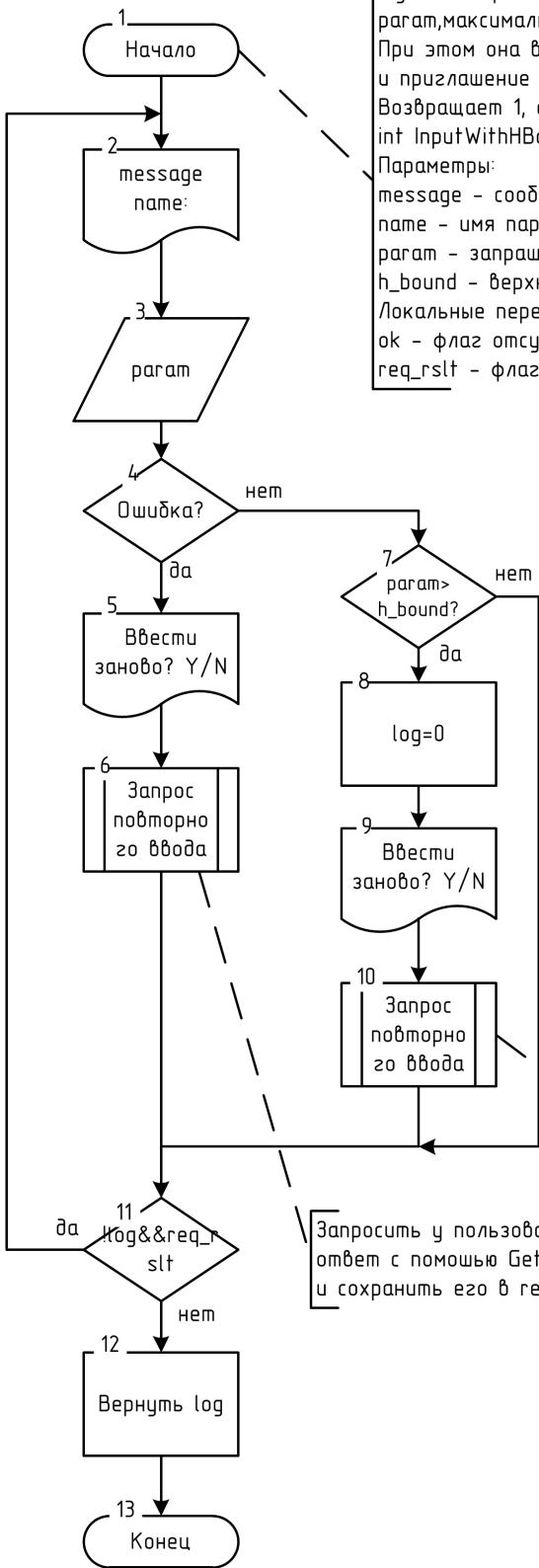
`int InputWithLBound(const char message[], const char name[], double *param, double l_bound);`

Параметры:

- message – сообщение о параметре;
- name – имя параметра;
- param – запрашиваемый параметр;
- l\_bound – верхняя граница параметра.

Локальные переменные:

- ok – флаг отсутствия ошибки (1 – нет ошибки);
- req\_rslt – флаг ответа пользователя (1 – согласие пользователя).



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.

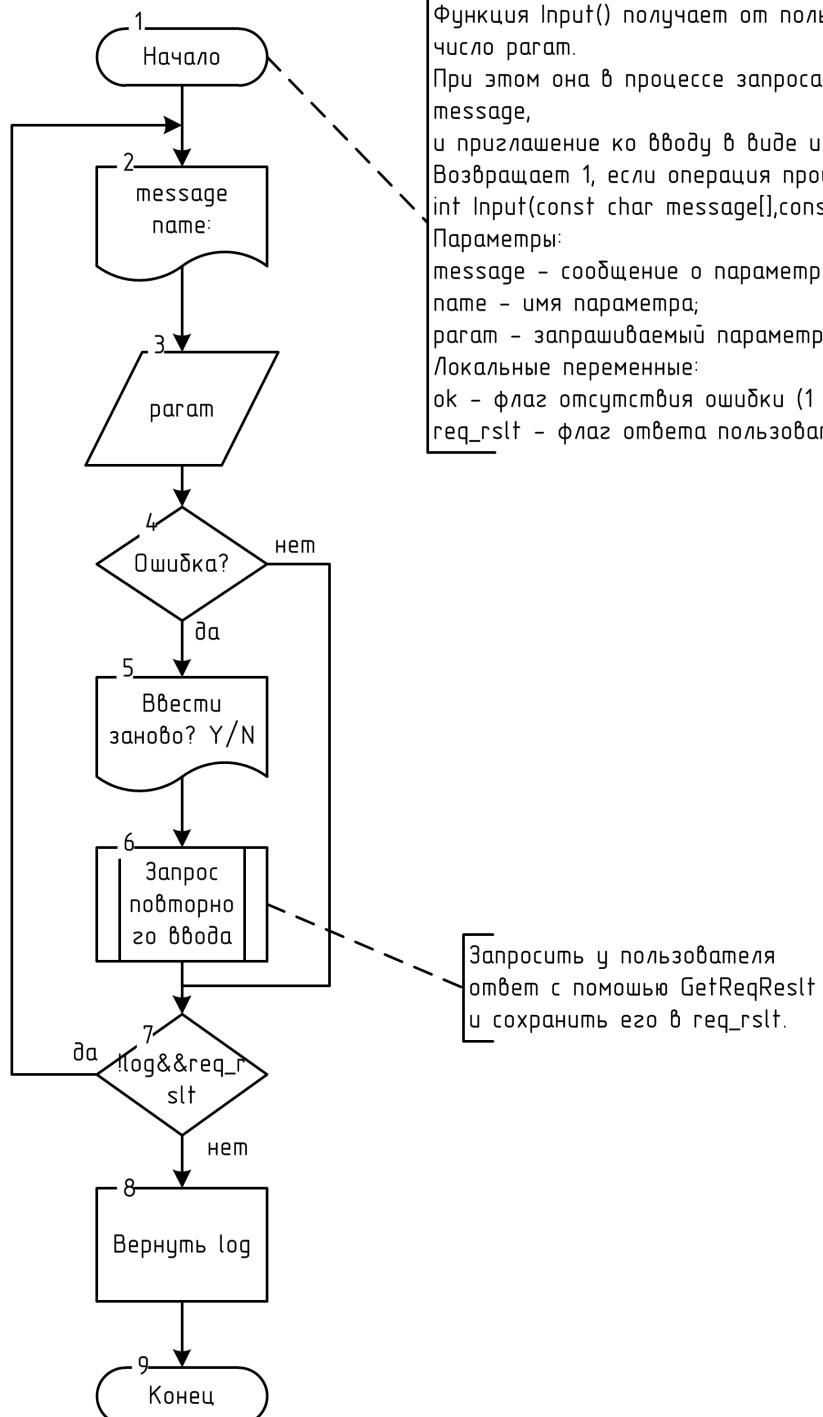
`int InputWithHBound(const char message[], const char name[], double *param, double h_bound);`

Параметры:

- message – сообщение о параметре;
- name – имя параметра;
- param – запрашиваемый параметр;
- h\_bound – верхняя граница параметра.

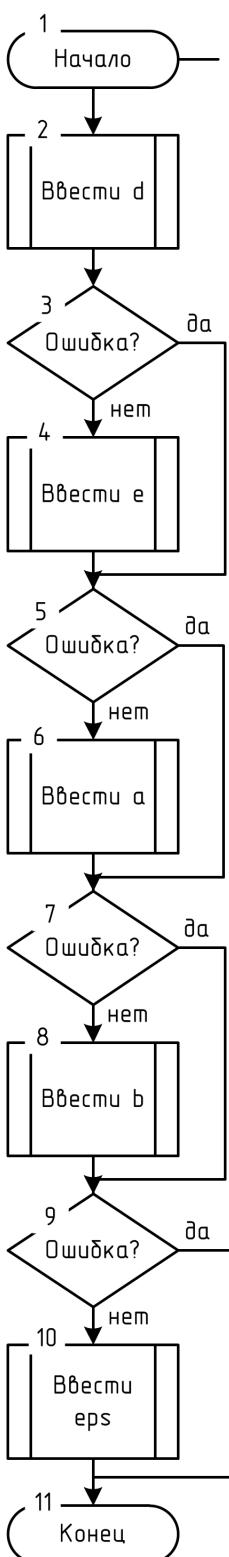
Локальные переменные:

- ok – флаг отсутствия ошибки (1 – нет ошибки);
- req\_rslt – флаг ответа пользователя (1 – согласие пользователя).

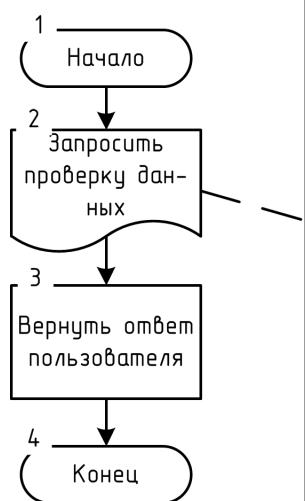


Функция `Input()` получает от пользователя некоторое вещественное число `param`.  
При этом она в процессе запроса выводит поясняющее сообщение `message`,  
и приглашение ко вводу в виде имени параметра - `name`.  
Возвращает 1, если операция прошла успешно, иначе возвращается 0.  
`int Input(const char message[], const char name[], double *param);`  
Параметры:  
`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).

Запросить у пользователя  
ответ с помощью `GetReqReslt`  
и сохранить его в `req_rslt`.



Функция InputData позволяет запросить у пользователя параметры функции  $d$  и  $e$ ,  
 границы интегрирования  $a$  и  $b$ , и точность вычислений  $\text{eps}$ .  
 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;  
 $\text{eps}$  – точность вычислений.  
 Локальные переменные:  
 $\text{log}$  – флаг отсутствия ошибки (1 – нет ошибки).



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.

```

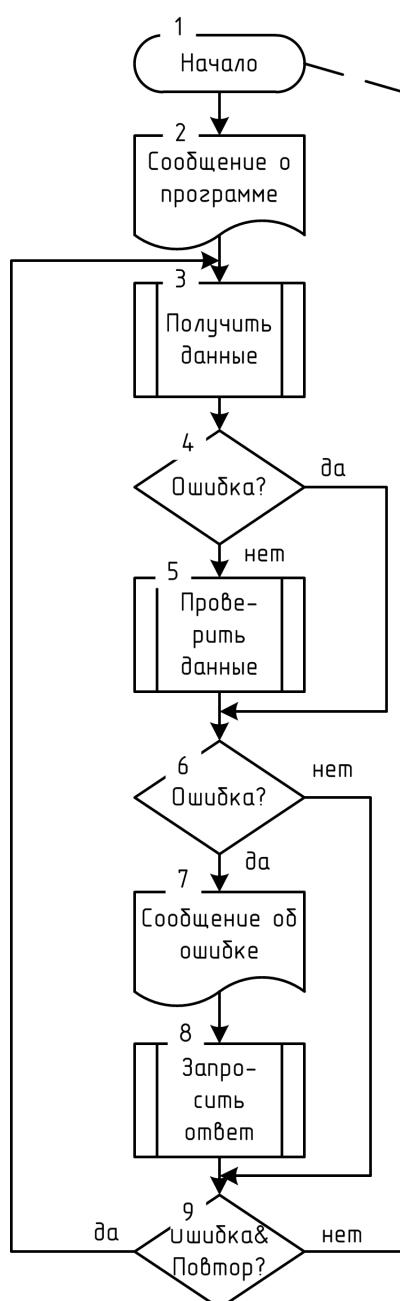
int CheckData(
    const double d,
    const double e,
    const double a,
    const double b,
    const double eps
);

```

Параметры:

d,e - параметры функции;  
a,b - границы интервала интегрирования;  
eps - точность вычислений.

Локальные переменные:  
отсутствуют.



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);

Рисунок 1.2 — Схема расчета определённого интеграла(начало)

Рисунок 1.3 — Схема алгоритма расчета определённого интеграла(продолжение)

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

Вариант №4

Лист

24

## **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: параметры c, d, r — вещественные неотрицательные числа; границы интегрирования a, b — вещественные числа, причём  $b > a$ ; точность eps — вещественное положительное число. При неверном вводе ввод придется повторить. Можно прервать программу сочетанием клавиш Control+C. После ввода данных программа предложит их проверить, чтобы начать вычисления, нажмите 'y' и Enter, чтобы завершить программу — 'a', повторить ввод — 'r'. После окончания расчетов программа выведет искомое значение интеграла.

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

*Вариант №4*

Лист

25

## 1.7. Инструкция программисту

При создании программы расчета определённого интеграла были предприняты следующие действия.

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

Таблица 2.24 - Параметры функции очистки буфера

имя	тип	предназначение
a,b	double	границы интервала интегрирования,
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:

- Подынтегральная функция f.
- 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 2.1, локальные переменные — в таблице 2.2.

Таблица 2.1 - Параметры функции очистки буфера

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

Таблица 2.2 - Локальные переменные функции очистки буфера

имя	тип	предназначение
count	long	Счетчик считанных символов.

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

Заголовок функции:

```
int GetReqResult();
```

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

Локальные переменные функции представлены в таблице 2.3.

Таблица 2.3 - 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 2.4, локальные переменные — в таблице 2.5.

Таблица 2.4 - Параметры функции получения варианта

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

Таблица 2.5 - 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:

Таблица 2.27 - Параметры функции получения вещественного числа

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

Локальные переменные:

Таблица 2.5 - 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:

Таблица 2.27 - Параметры функции получения варианта

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

Локальные переменные:

Таблица 2.5 - Локальные переменные функции получения файла для записи

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:

Таблица 2.27 - Параметры функции получения варианта

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

Локальные переменные:

Таблица 2.5 - Локальные переменные функции получения файла для записи

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:

Изм.	Лист	№ докум.	Подп.	Дата

Таблица 2.27 - Параметры функции получения варианта

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

Локальные переменные:

Таблица 2.5 - Локальные переменные функции получения файла для записи

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:

Таблица 2.27 - Параметры функции получения варианта

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:

Таблица 2.27 - Параметры функции получения варианта

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

Локальные переменные:

Таблица 2.5 - Локальные переменные функции получения файла для записи

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:

Таблица 2.27 - Параметры функции получения варианта

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

Локальные переменные:

Таблица 2.5 - Локальные переменные функции получения файла для записи

имя	тип	предназначение
reslt	double	сумма вычисленных значений функции f

## 1.8. Текст программы

В этом разделе приведен исходный текст программы на языке Borland Pascal 7, 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
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;
l_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,
    double *a, //a,b - границы отрезка поиска корня;

```

Изм.	Лист	№ докум.	Подп.	Дата

```

        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\
            \"          \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);
        log=!clearline(stdin)&&log;
    }
}

```

Изм.	Лист	№ докум.	Подп.	Дата

```

    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];
    log=InputWithHBound("Введите d - точка пересечения линий (1) и (2) -\n"
                        "вещественное число. d<0.\n", "d", d, 0);
}

```

Изм.	Лист	№ докум.	Подп.	Дата

```

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,0);
    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 нечетными 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 ещё 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'
            &&toupper(answer) !='N'){
        printf("Неправильный ответ! Допустимо:\n\""

```

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

```

        "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
------	------	----------	-------	------

## 1.9. Тестовый пример

Ниже на рисунке 1.4 показан пример работы программы вычисления определённого интеграла для параметров функции  $d=4$ ,  $c=3$ ,  $r=2$ , границами интегрирования  $[a,b]$ , с точностью  $\text{eps}$ .

```
Программа находит значение определённого интеграла на промежутке от a до b
методом левых прямоугольников для функции
    / d,           d>=0, x<=-R
y=| -√(R^2-x^2), -R<x<R
    \ c,           c>=0, x>=R
Введите d - вещественное число
d>=0
d:4
Введите c - вещественное число
c>=0
c:3
Введите r - вещественное число
r>=0
r:2
Введите a - вещественное число
a:-50
Введите b - вещественное число
b>a
b:50
Введите точность вычислений (eps) - вещественное число
eps>0
eps:0.0001
Программа находит значение определённого интеграла
на промежутке от -50.0000 до 50.0000
для функции
    / 4.0000,      x<=-2.0000
y=| -√(2.0000^2-x^2), -2.0000<x<2.0000
    \ 3.0000,      x>=2.0000
Продолжить работу?
'a' - завершить программу, 'r' - повторить ввод
'y' (или любая другая клавиша) - продолжить работу,
y
Подождите.....Ура!!!
Значение интеграла: 329.7168
Нажмите <Enter>...
```

Рисунок 1.4 — Результат работы программы вычисления интеграла  
Рассчитаем интеграл вручную.

Исходную криволинейную трапецию можно разбить на 3 части.

Первая фигура — прямоугольник длиной  $|a|-R=48$  и шириной  $d=4$ , значит, его площадь — 192.

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

Следующий участок — полуокружность радиусом R=2. Её площадь -

$$\frac{\pi R^2}{2} = \frac{4\pi}{2} = 2\pi = 6.28319\dots$$
 Так как она лежит ниже оси X, это значение следует

$$\text{вычесть: } 192 - 6.28319 = 185.71681$$

Последний участок — прямоугольник длиной b-R=48 и шириной c=3, его площадь — 144. Суммируем и получаем  $185.71681 + 144 = 329.71681$ . Программа работает верно, требуемая точность достигнута.

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

Вариант №4

Лист

40

## 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  $\varepsilon$ :  $|a_k| < \varepsilon$ .

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:

$y$  - значение функции для текущего аргумента;

$x$  - текущий аргумент функции;

$k$  - порядковый номер вычисляемого элемента последовательности;

$y_k$  - вычисляемый элемент последовательности;

Изм.	Лист	№ докум.	Подп.	Дата

*nk* - количество вычисляемых значений функции (количество строк в таблице значений функции);

*n* - номер по порядку в таблице значений вычисляемого значения функции;

*sqrх* - квадрат текущего аргумента .

Далее описаны переменные, используемые для служебных нужд:

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

*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.

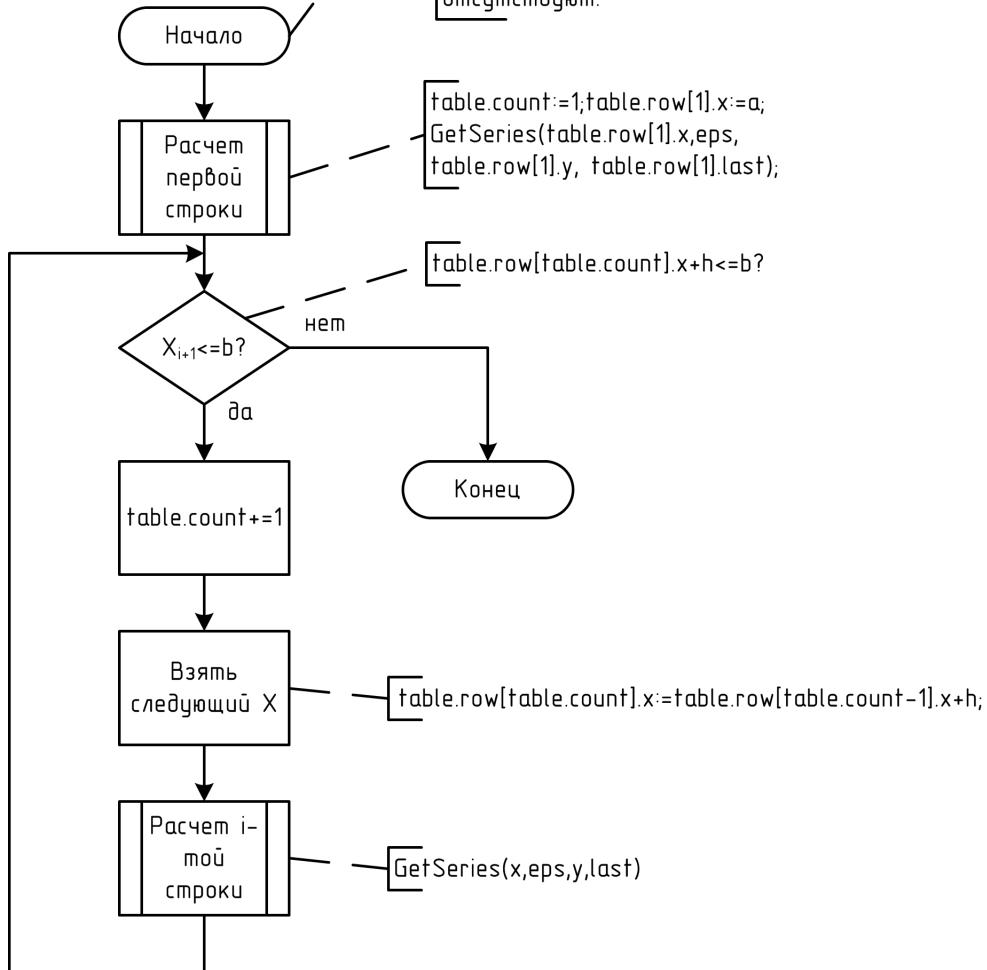
## **2.5. Разработка структуры алгоритма решения задачи**

На рисунках 2.1 и 2.2 представлен алгоритм вычисления таблицы значений функции (2.1).

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

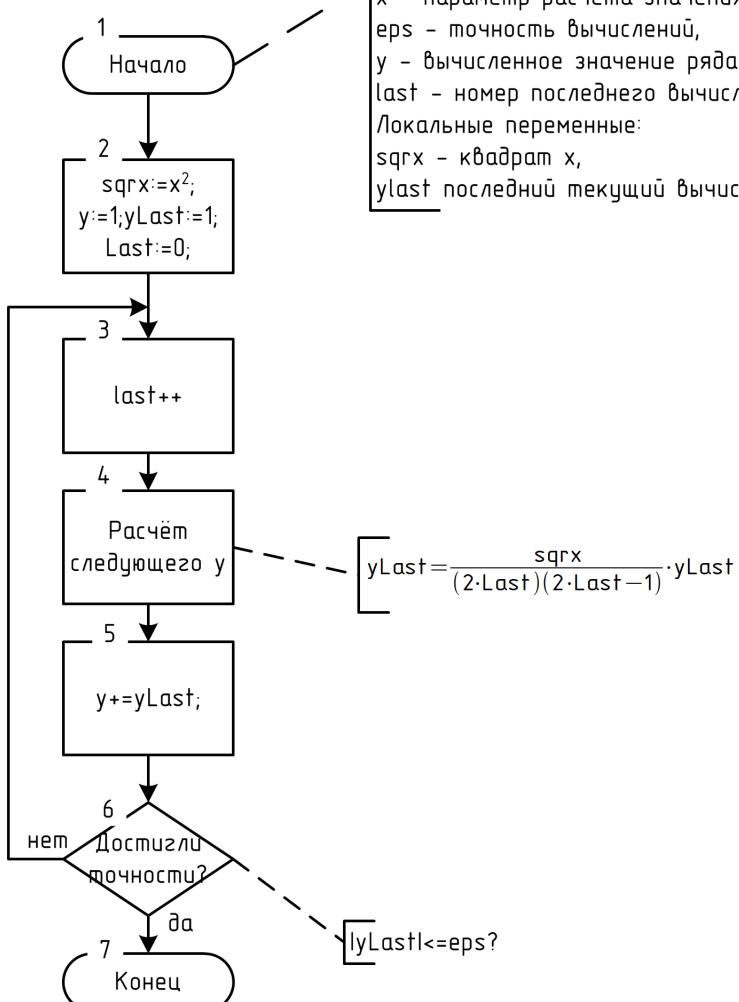
Процедура GetTable получает таблицу значений ряда table, при изменении параметра расчета от a до b с шагом h, с точностью вычислений eps.

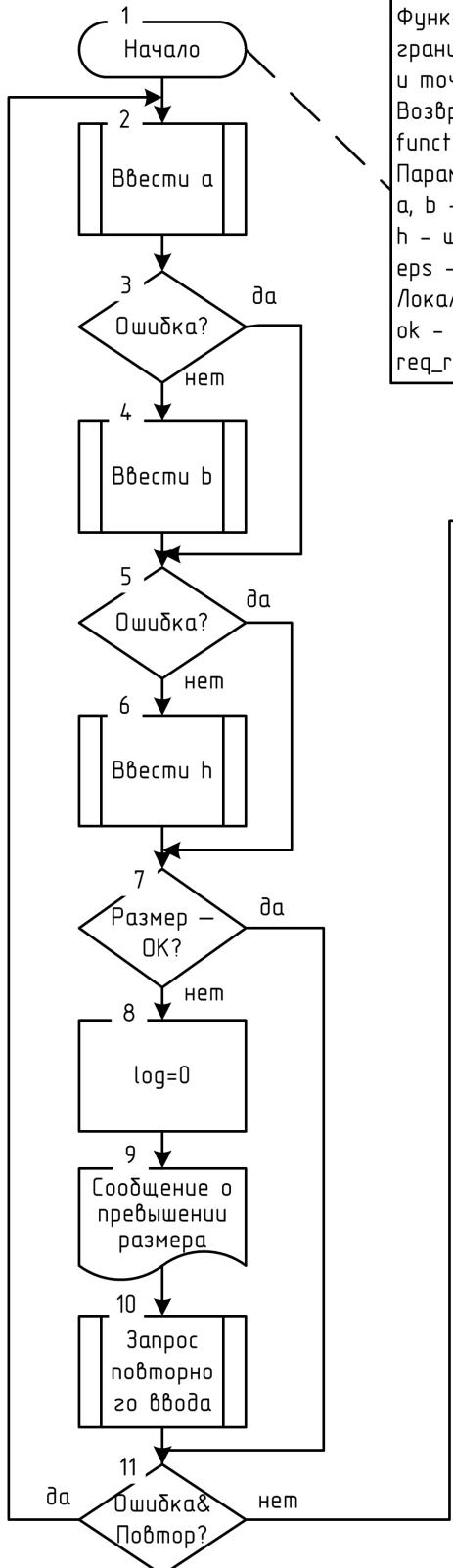
```
procedure GetTable(a,b,h,eps:real;var table:TTable)
Параметры:
a, b - границы изменения параметра расчета,
h - шаг изменения параметра,
eps - точность вычислений,
table - вычисленная таблица значений ряда.
Локальные переменные:
отсутствуют.
```



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:
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.
```



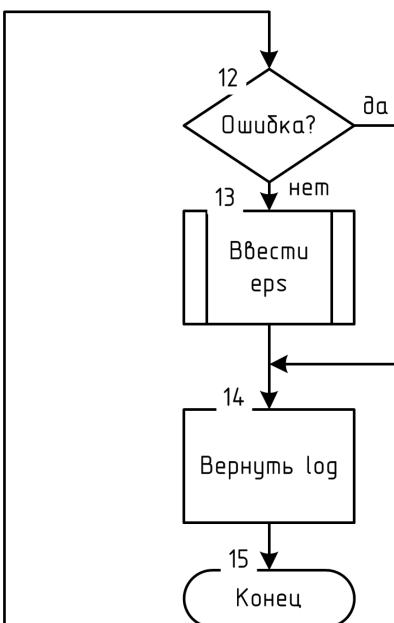


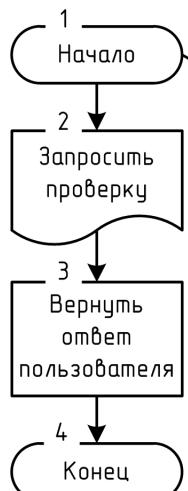
Функция InputConsole() позволяет пользователю считать с клавиатуры границы изменения параметра расчета  $a$  и  $b$ , шаг изменения параметра  $h$ , и точность вычислений  $\text{eps}$ . Возвращает true, если операция прошла успешно, иначе возвращается false.

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

**Параметры:**  
 $a, b$  – границы изменения параметра расчета,  
 $h$  – шаг изменения параметра,  
 $\text{eps}$  – точность вычислений.

**Локальные переменные:**  
 $\text{ok}$  – флаг отсутствия ошибки (true – нет ошибки);  
 $\text{req_rslt}$  – флаг ответа пользователя (true – согласие пользователя).



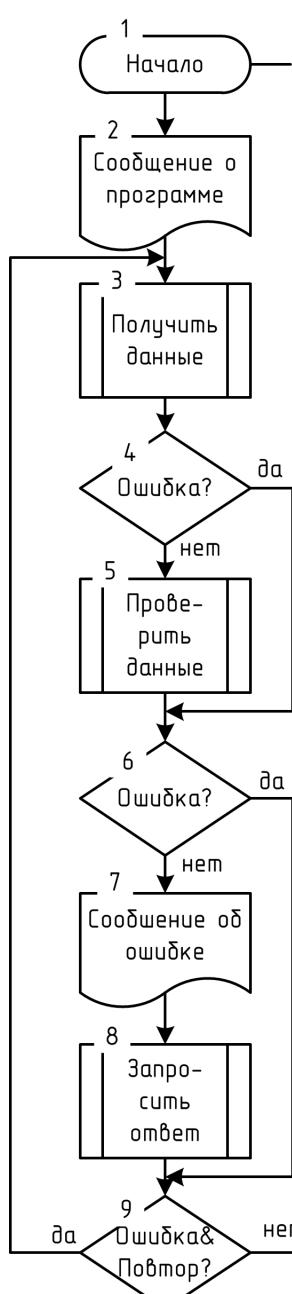


Функция CheckData() позволяет пользователю проверить введенные данные: границы параметра расчета  $a$  и  $b$ , шаг изменения параметра  $h$ , и точность вычислений  $eps$ . Возвращает true, если пользователь подтвердил правильность данных, иначе возвращается false.

```
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.

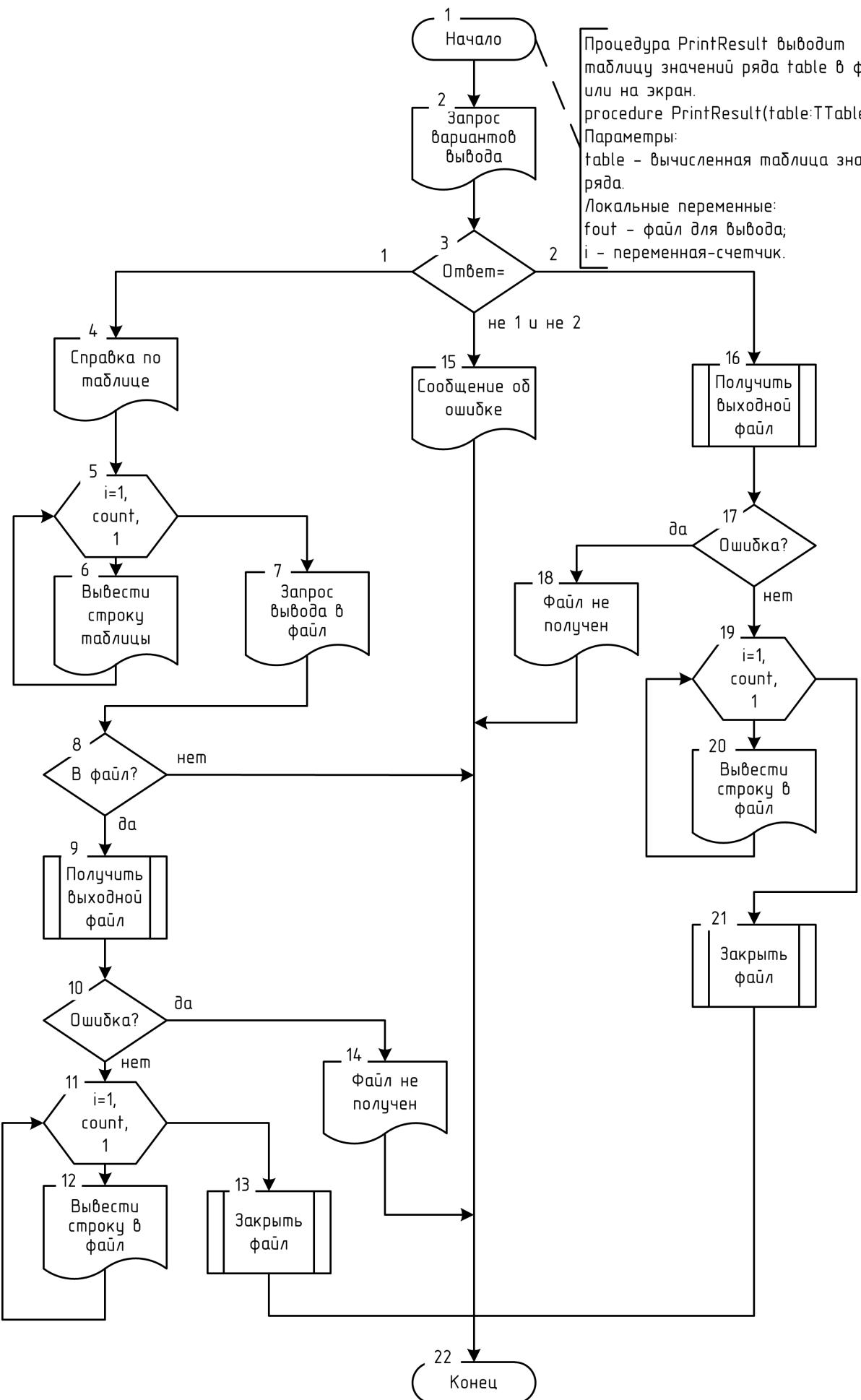


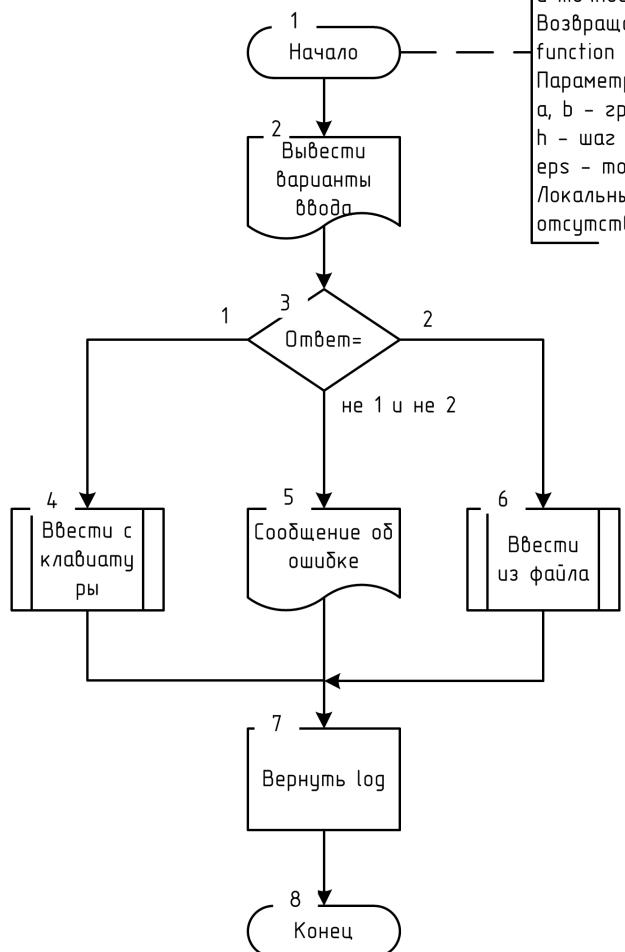
Основная часть программы расчета таблицы значений.

Переменные:

$a$ ,  $b$  - границы изменения параметра расчета,  
 $h$  - шаг изменения параметра,  
 $eps$  - точность вычислений,  
 $ok$  - флаг отсутствия ошибки (true - нет ошибки);  
 $req_rslt$  - флаг ответа пользователя (true - согласие пользователя),  
 $table$  - вычисленная таблица значений ряда.

14 Конец





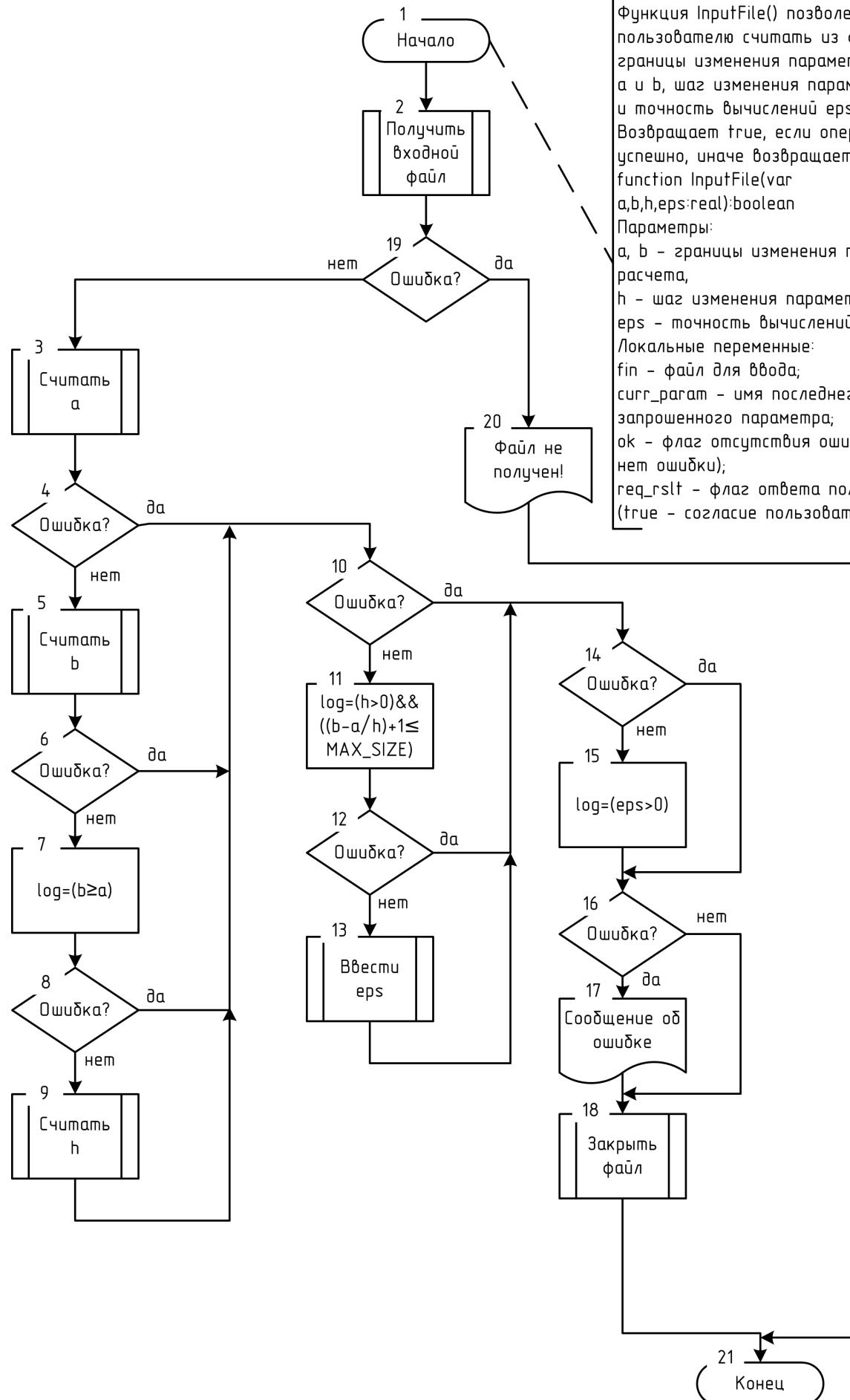
Функция InputData() позволяет пользователю выбрать метод ввода (с клавиатуры или из файла) и ввести соответствующие значения: границы изменения параметра расчета  $a$  и  $b$ , шаг изменения параметра  $h$ , и точность вычислений  $\text{eps}$ . Возвращаем `true`, если операция прошла успешно, иначе возвращается `false`.

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

Параметры:

- $a, b$  – границы изменения параметра расчета,
- $h$  – шаг изменения параметра,
- $\text{eps}$  – точность вычислений.

Локальные переменные:  
отсутствуют.



Функция InputFile() позволяет пользователю считать из файла границы изменения параметра расчета  $a$  и  $b$ , шаг изменения параметра  $h$ , и точность вычислений  $\text{eps}$ . Возвращает `true`, если операция прошла успешно, иначе возвращается `false`.

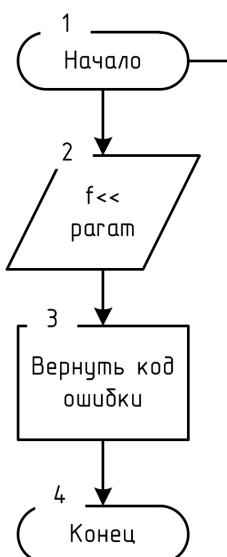
```
function InputFile(var a,b,h,eps:real):boolean
```

Параметры:

- $a, b$  – границы изменения параметра расчета,
- $h$  – шаг изменения параметра,
- $\text{eps}$  – точность вычислений.

Локальные переменные:

- `fin` – файл для ввода;
- `curre_param` – имя последнего запрошенного параметра;
- `ok` – флаг отсутствия ошибки (`true` – нет ошибки);
- `req_rslt` – флаг ответа пользователя (`true` – согласие пользователя).



Функция GetQuiet считывает параметр param из файла f.  
 Возвращаем true, если операция прошла успешно, иначе возвращается false.  
 function GetQuiet(var f:text; var param:real):boolean  
 Параметры:  
 f - файл для ввода  
 param - считываемый параметр.  
 Локальные переменные:  
 отсутствуют.

Рисунок 2.1 — Схема алгоритма вычисления таблицы значений функции(начало)

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

Вариант №4

Лист  
52

Рисунок 2.2 — Схема алгоритма вычисления таблицы значений функции(продолжение)

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

Вариант №4

Лист  
53

## 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;

(* **** *)
(*
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:

x - параметр расчета значения ряда,

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

eps - точность вычислений,  
 у - 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.	Дата
------	------	----------	-------	------

```

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
  sqrX:=sqr(x); {запомним x^2}

```

Изм.	Лист	№ докум.	Подп.	Дата

```

y:=1; ylast:=1; last:=0; {инициализация}
repeat
    inc(last);
    ylast:=ylast*sqr(x)/(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
    ok:=true; req_rslt:=false;

```

Изм.	Лист	№ докум.	Подп.	Дата

```

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, 0);
        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, 0);
        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';
        if ok then

```

Изм.	Лист	№ докум.	Подп.	Дата

```

        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);
            for i:=1 to table.count do

```

Изм.	Лист	№ докум.	Подп.	Дата

```

        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.

```

## 2.7. Инструкция программисту

При создании программы вычисления таблицы значений были предприняты следующие действия.

Объявлена константа MAX\_COUNT — максимальное количество строк в таблице значений;

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

Таблица 2.1 - Локальные переменные функции получения ответа

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 - Локальные переменные функции получения ответа

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 - Локальные переменные функции получения ответа

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', '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 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

имя	тип	предназначение
sqrX	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. 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, и точность вычисления значений, которая больше 0. Все параметры вещественные числа. Отказаться от ввода и завершить программу можно сочетанием клавиш Control-C. 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 выведена таблица значений для аргументов из указанного промежутка, изменяющихся с указанным шагом.

## 2.9. Тестовый пример

Ниже на рисунке 2.3 показан пример работы программы вычисления таблицы значений функции (2.1) для  $x=2$  и точностью 0.01.

```
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)!

Ведите минимальное x (a) - вещественное число
a:2
Ведите максимальное x (b) - вещественное число
b>=a
b:2
Ведите шаг изменения x (h) - вещественное число
h>=0
h:0
Ведите точность вычислений (eps) - вещественное число
eps>0
eps:0.01
Вычисление значений функции
на интервале от 2.0000 до 2.0000 с шагом 0.0000. Точность - 0.0100
Продолжить работу?
'a' - завершить программу, 'g' - повторить ввод
'y' (или любая другая клавиша) - продолжить работу,
y
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
Нажмите <Enter>...
```

Рисунок 2.3 — Результат запуска программы для  $x=2$  и точностью 0.01

Проверим этот результат, рассчитав значение функции вручную:

1. Первый элемент последовательности  $a_1 = \frac{4}{2} = 2$ ,  $2 > 0.01$ ,  $f(x) = 3$ , значит,

вычисляем следующий элемент ряда...

2.  $a_2 = \frac{16}{2 \cdot 3 \cdot 4} = 0.666667$ ,  $f(x) = 3.666667$ ,  $0.666667 > 0.01$ , продолжаем

вычисления...

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

$$3. a_3 = \frac{64}{24 \cdot 5 \cdot 6} = 0.088889, \quad f(x)=3.755556, \quad 0.088889 > 0.01, \text{ продолжаем}$$

вычисления...

$$4. a_4 = \frac{256}{720 \cdot 7 \cdot 8} = 0.00634, \quad f(x)=3.761896, \quad 0.00634 < 0.01, \text{ текущий элемент}$$

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 2.4.

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

```

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)!

Ведите минимальное x (a) - вещественное число
a:-5
Ведите максимальное x (b) - вещественное число
b>=a
b:5
Ведите шаг изменения x (h) - вещественное число
h>=0
h:0.5
Ведите точность вычислений (eps) - вещественное число
eps>0
eps:0.0001
Вычисление значений функции
на интервале от -5.0000 до 5.0000 с шагом 0.5000. Точность - 0.0001
Продолжить работу?
'a' - завершить программу, 'г' - повторить ввод
'y' (или любая другая клавиша) - продолжить работу,
y
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        -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

```

Нажмите <Enter>...

Рисунок 2.4 — Результат запуска программы для x из промежутка [-5;5] с шагом 0.5 и точностью 0.0001

Изм.	Лист	№ докум.	Подп.	Дата

### 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 указаны исходные последовательности.

Номер шага	$i \longrightarrow \quad \longleftarrow j$								Примечание
	40	11	83	57	32	21	75	64	
1	40							64	Исходный список $k_0 < k_j$
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	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  $u$ , то среднее число операций обмена равно числу элементов в левой части сегмента, а именно  $u$ , умноженному на вероятность того, что элемент попал на свое место посредством обмена. Обмен произошел, если элемент принадлежал правой части; вероятность этого равна  $(n-u)/n$ .

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 эфек-

Изм.	Лист	№ докум.	Подп.	Дата

тивность 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  $n^2$ .

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 лучше. 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

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

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  $\log(n)$ .

### 3.4. Разработка структур данных

Согласно численному методу, необходимо ввести следующие переменные:

$A$  - матрица, в которой выполняется подсчет средних арифметических значений положительных элементов каждой строки;

$n, m$  - определенные пользователем количество строк и количество столбцов матрицы соответственно;

Эти величины должны вводится пользователем. Далее описываются не вводимые пользователем переменные:

$sra$  - вспомогательный массив для хранения средних арифметических значений положительных элементов строк матрицы ;

$i$  — переменная-счетчик, используемая для прохождения по строкам данной матрицы и элементам вспомогательного массива;

$j$  - переменная-счетчик, используемая для прохождения по столбцам данной матрицы (для выбора элемента матрицы в текущей строке).

Далее описаны переменные, используемые для служебных нужд:

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

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

*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;

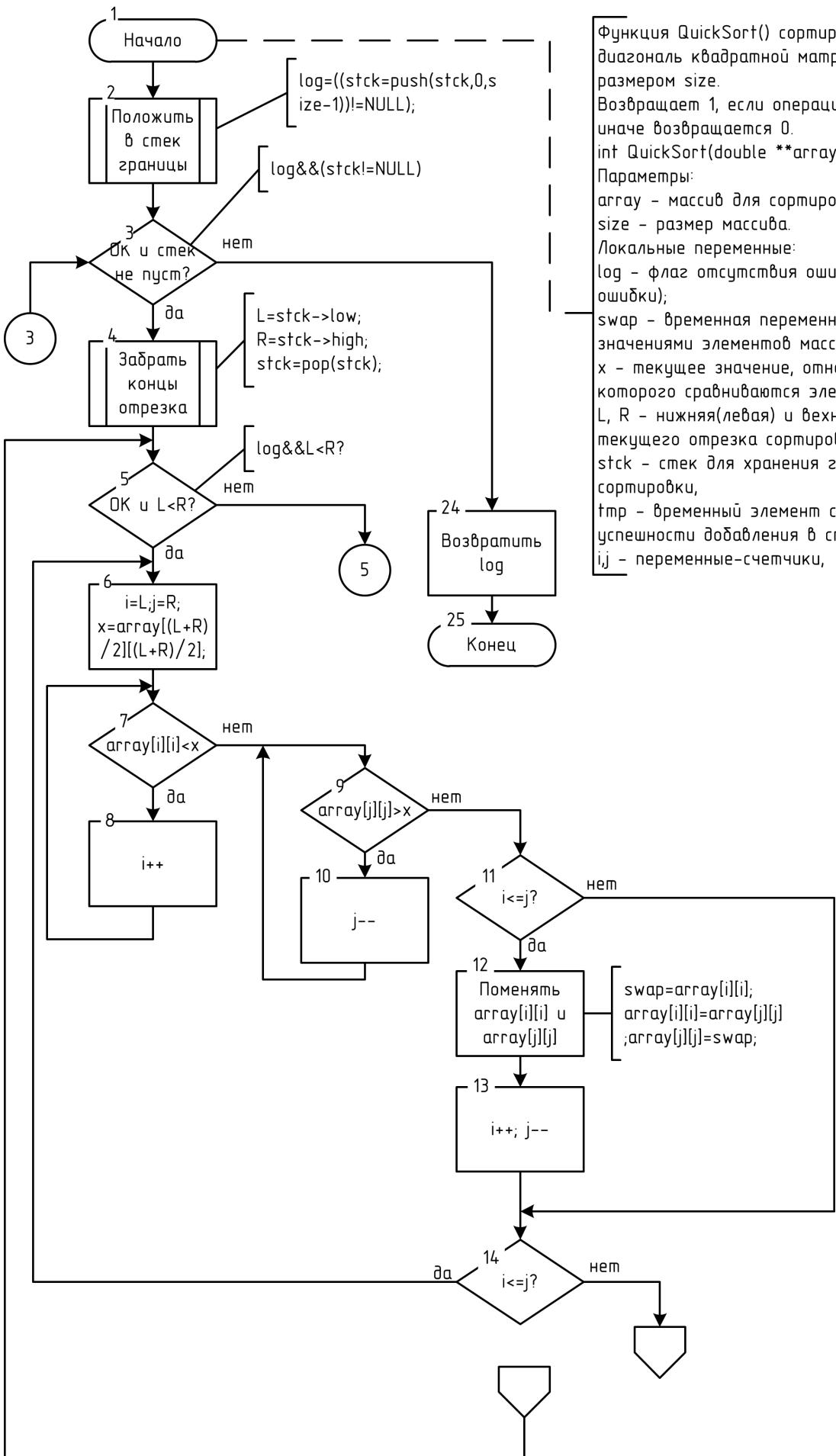
*ch* - содержит ответ пользователя на запрос программы о завершении, повторном вводе данных и завершении работы.

*nn,mm* - максимальные размеры обрабатываемой матрицы — соответственно, максимальное количество строк и максимальное количество столбцов.

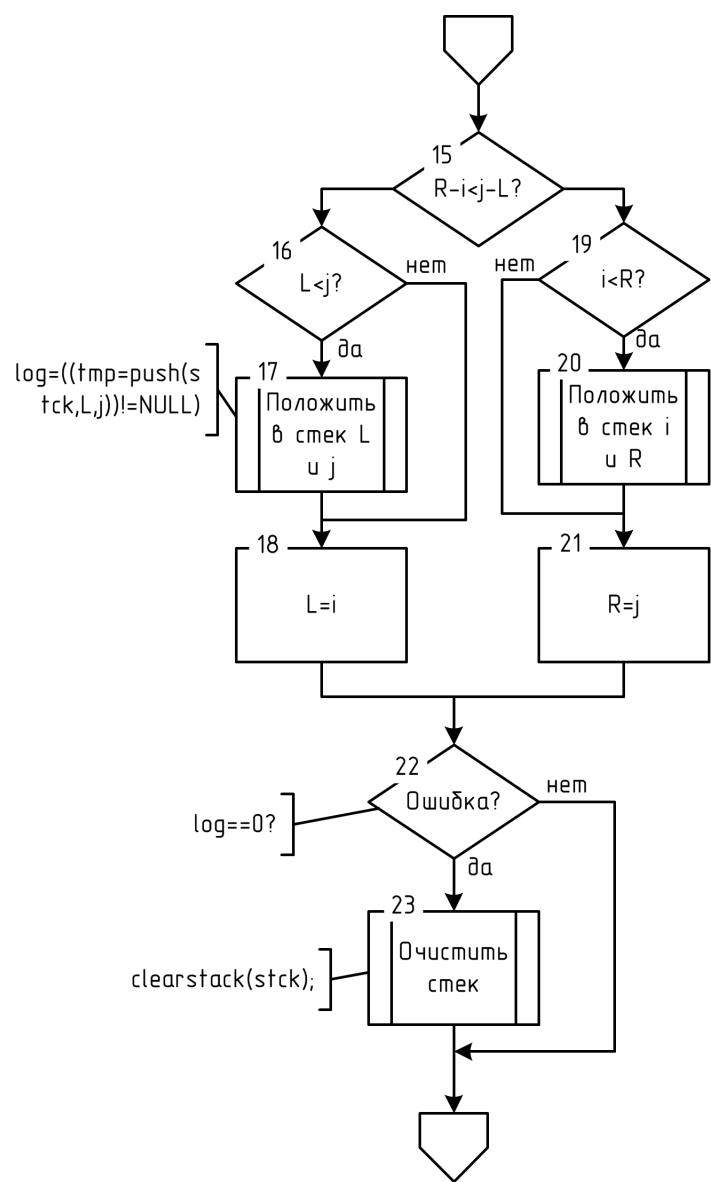
### **3.5. Разработка структуры алгоритма решения задачи**

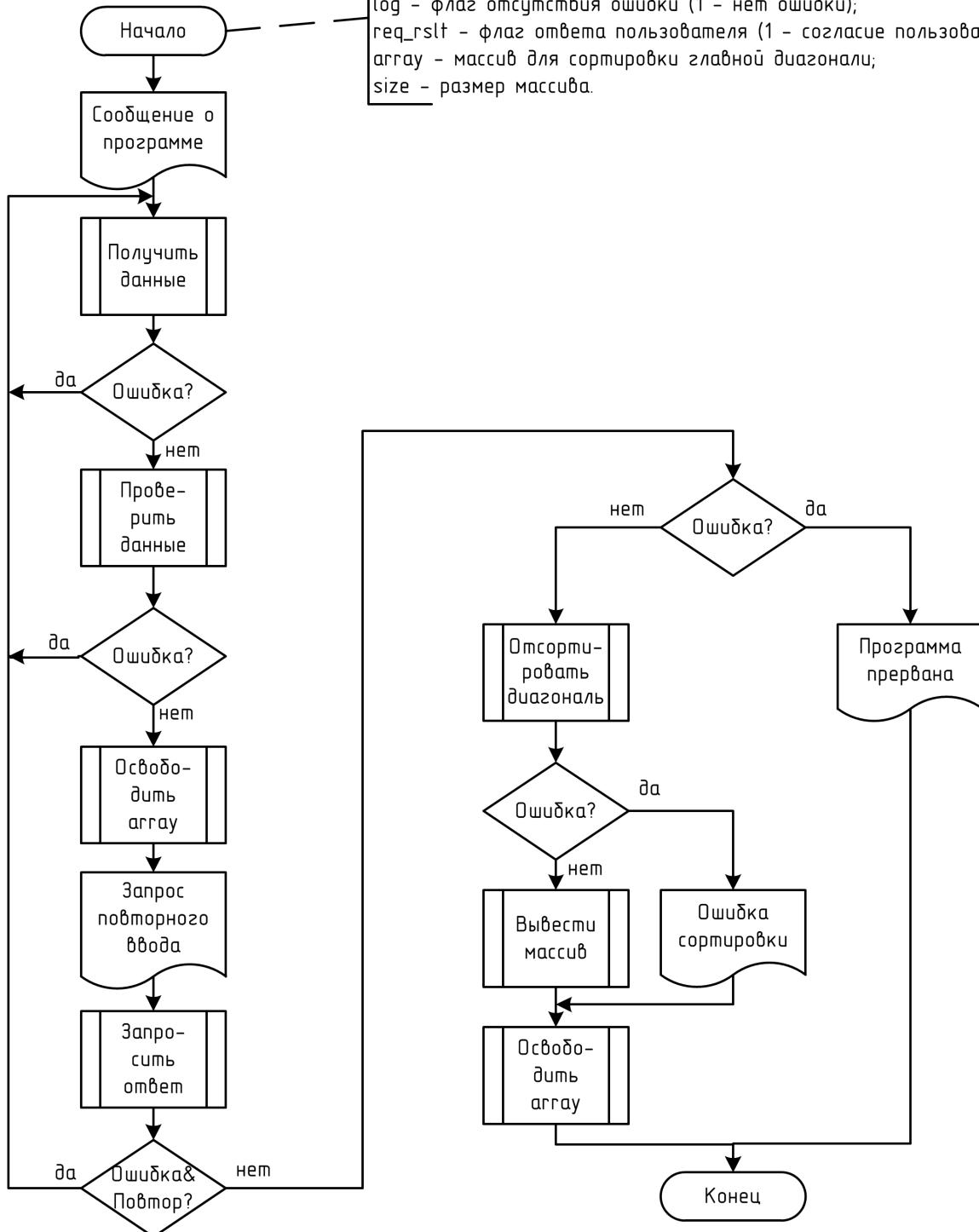
На рисунках 3.1 и 3.2 представлен алгоритм вычисления средних арифметических значений положительных элементов каждой строки матрицы, заданной пользователем.

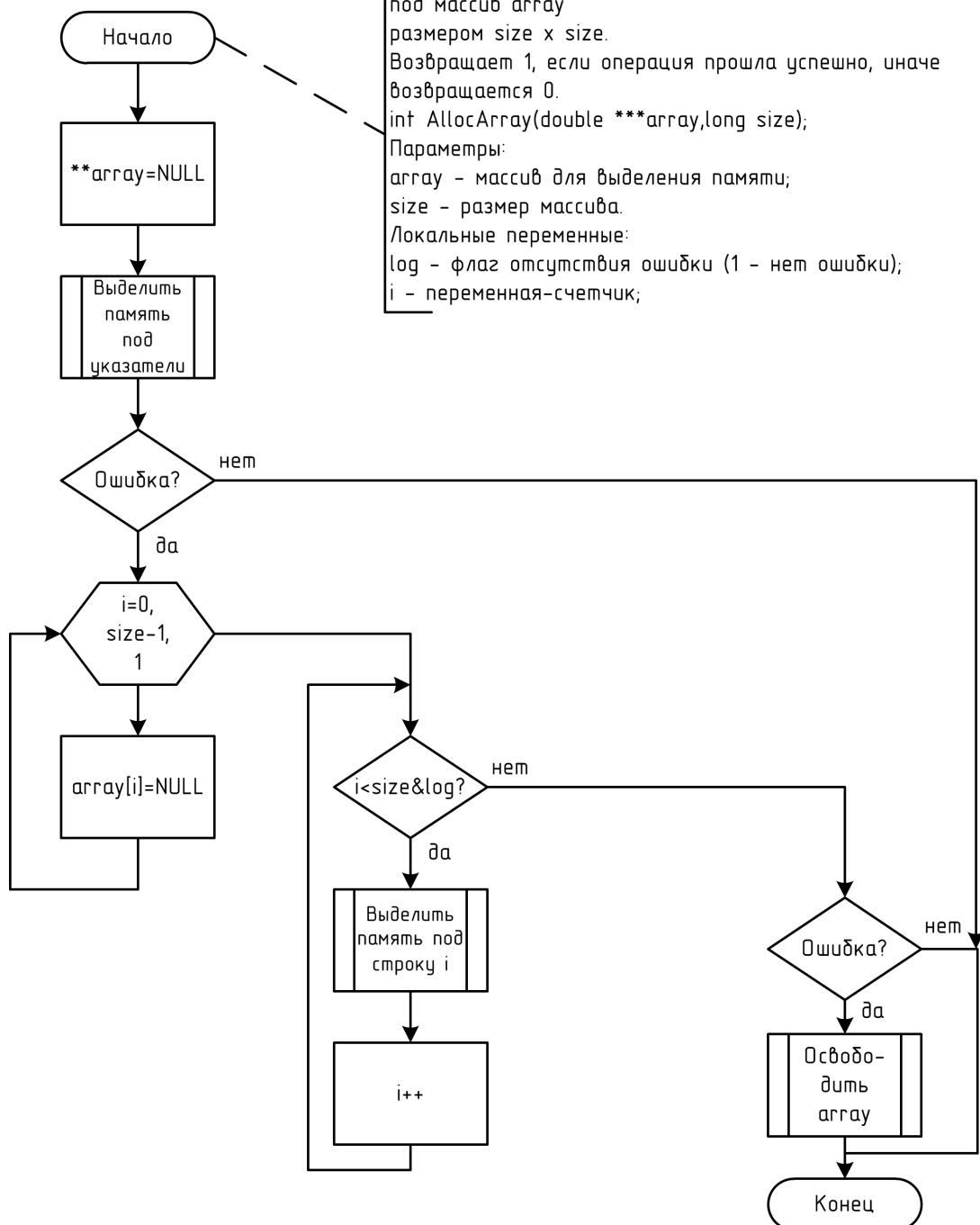
Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

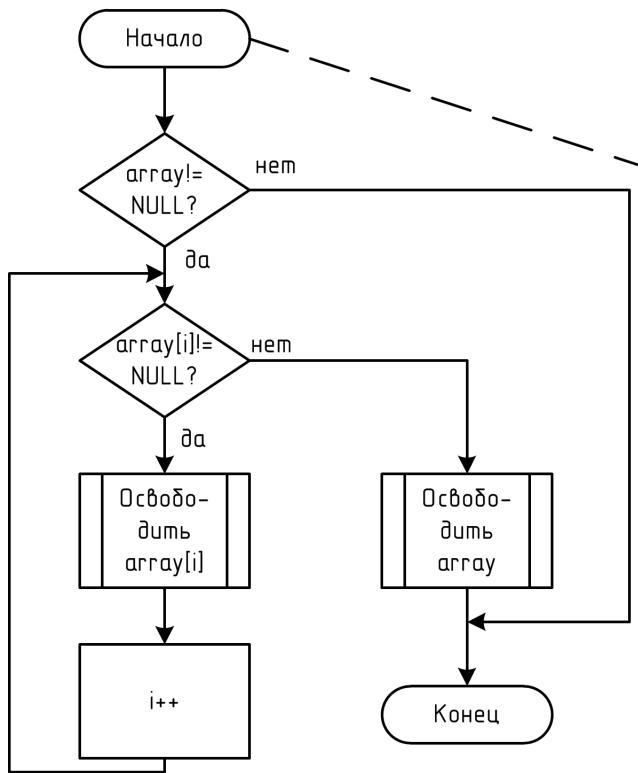


Функция QuickSort() сортирует главную диагональ квадратной матрицы array размером size.  
Возвращаем 1, если операция прошла успешно, иначе возвращается 0.  
int QuickSort(double \*\*array, long size)  
Параметры:  
array - массив для сортировки;  
size - размер массива.  
Локальные переменные:  
log - 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,









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

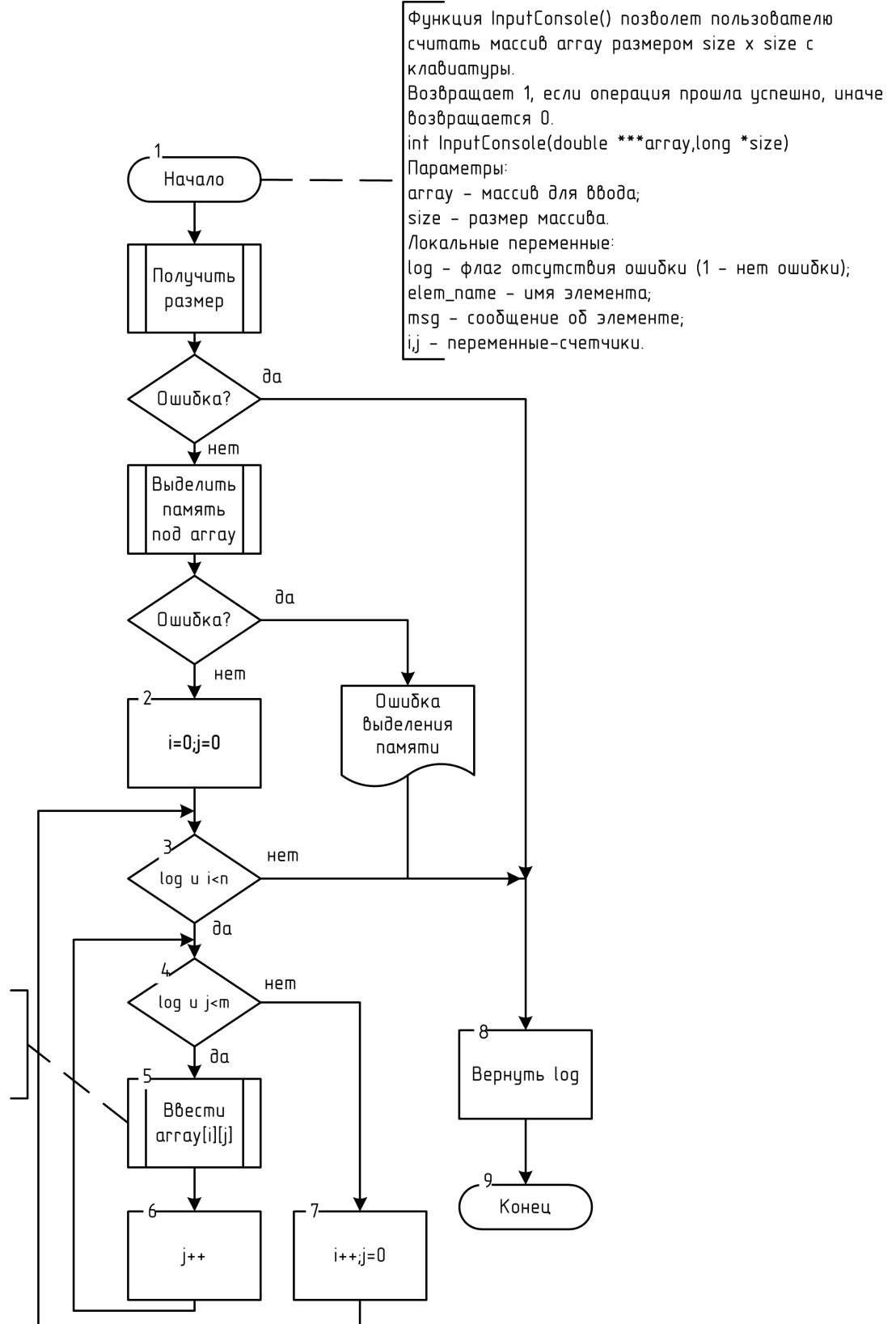
```
void FreeArray(double ***array, long size)
```

Параметры:

array - массив для освобождения памяти;  
size - размер массива.

Локальные переменные:  
i - переменная-счетчик;

Ввести  $array[i][j]$  с помощью функции Input, результат записать в log.



Получить файл f с помощью функции  
fopen, стартуя  
операции записи  
лог

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:

array – массив для ввода;

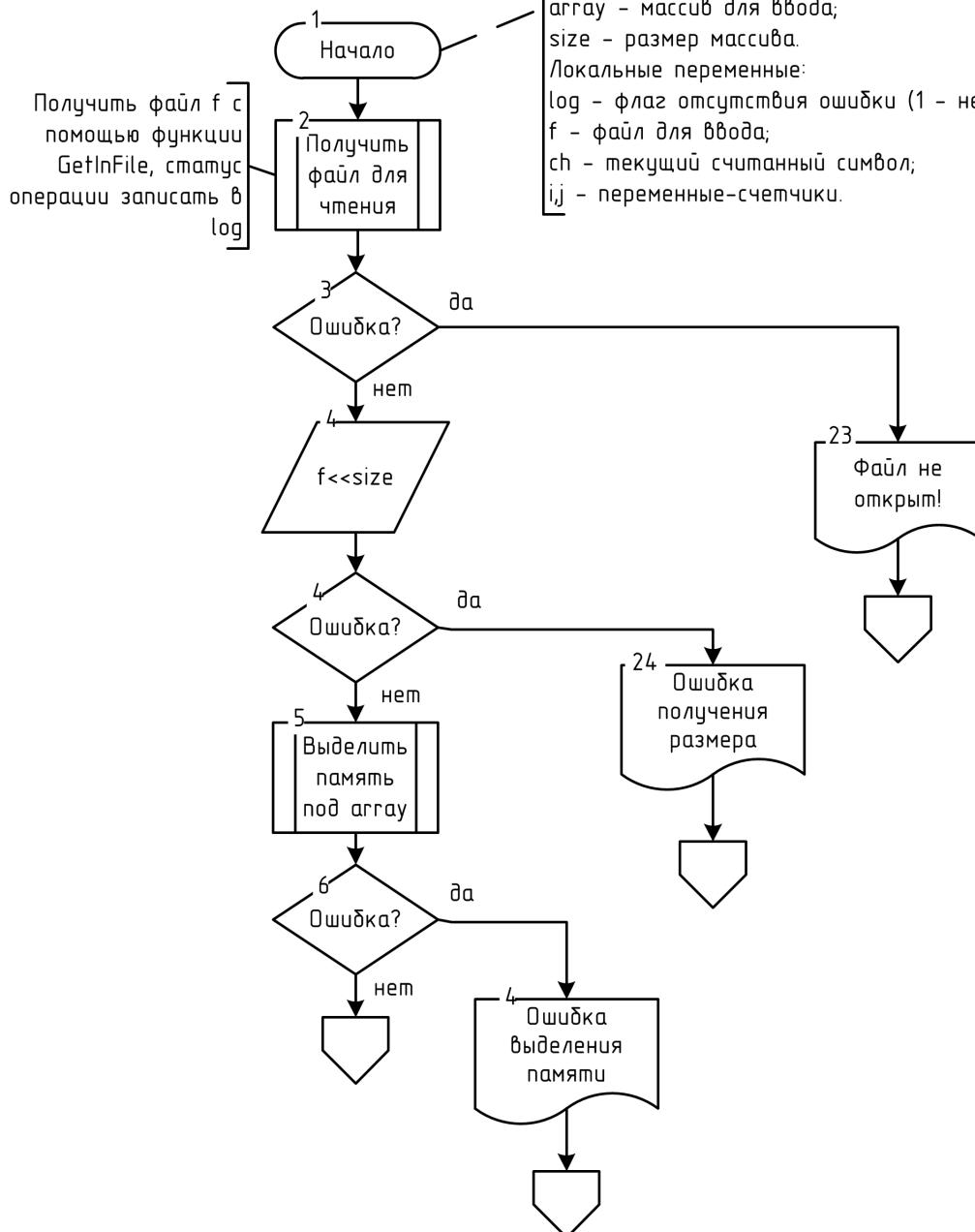
size – размер массива.

Локальные переменные:

log – флаг отсутствия ошибки (1 – нет ошибки);  
f – файл для ввода;

ch – текущий считанный символ;

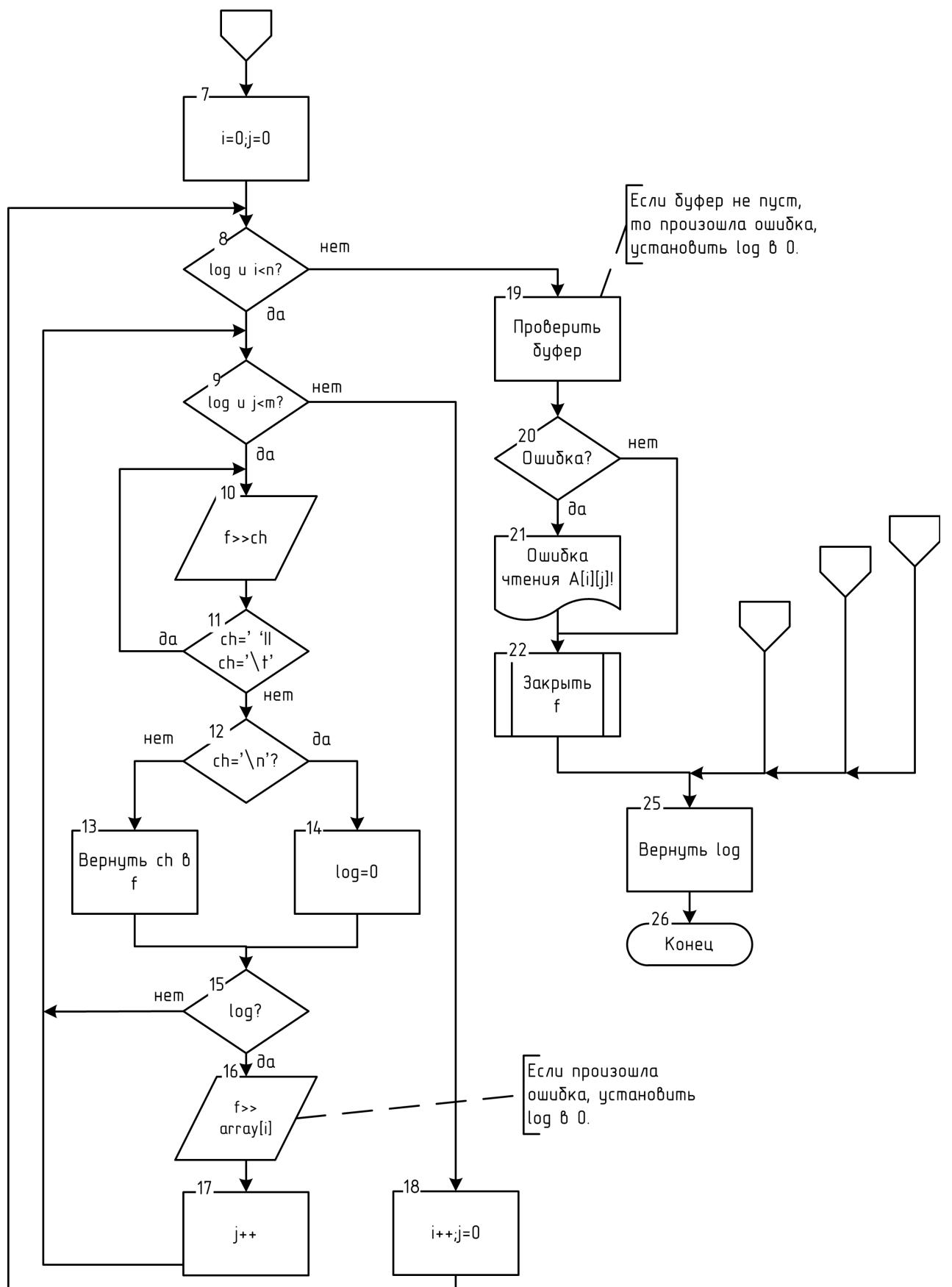
i,j – переменные-счетчики.

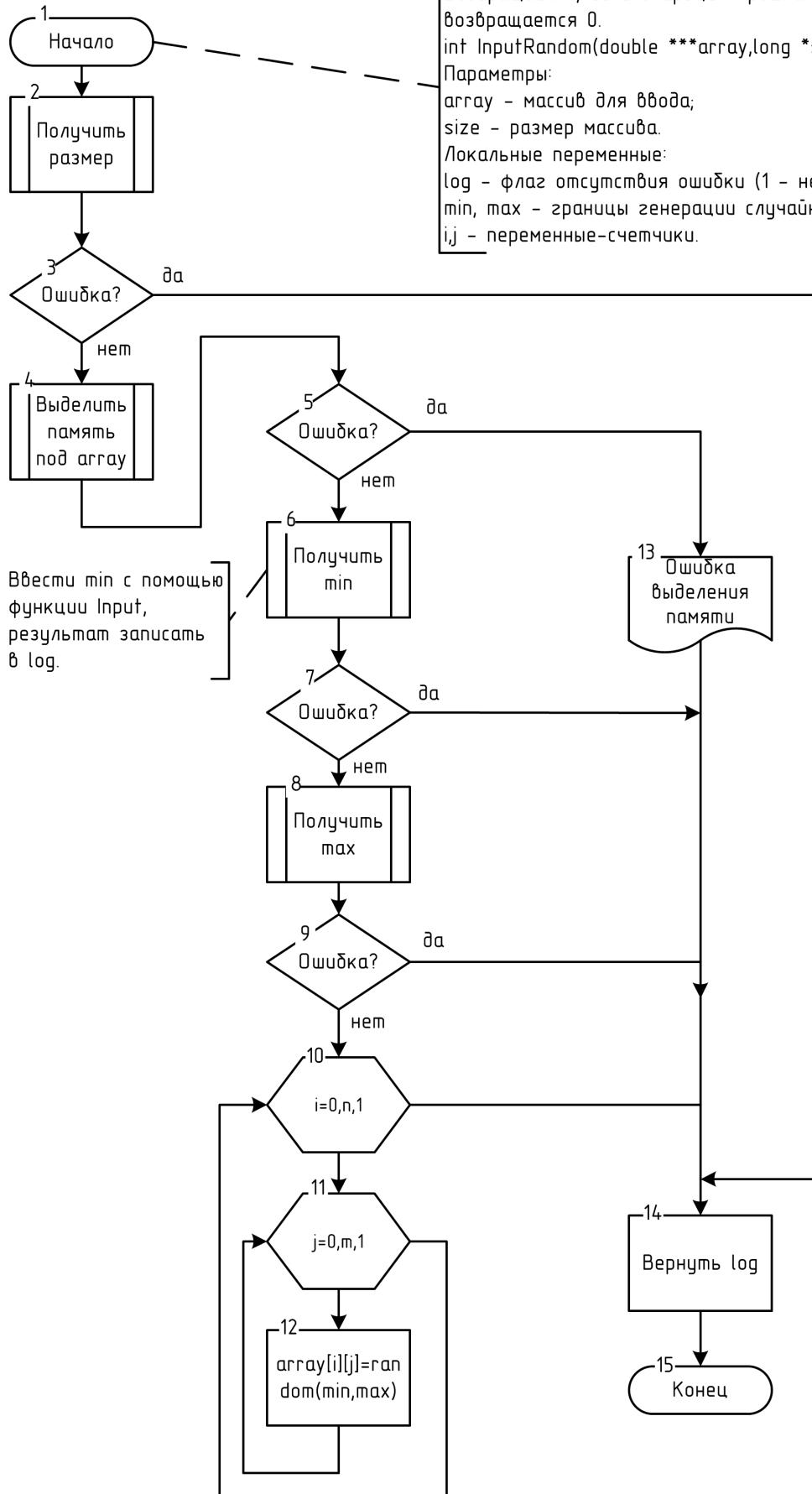


Функция InputFile() позволяет пользователю считать массив `array` размером `size` x `size` из файла.  
Возращает 1, если операция прошла успешно, иначе возвращается 0.

```
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.





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.

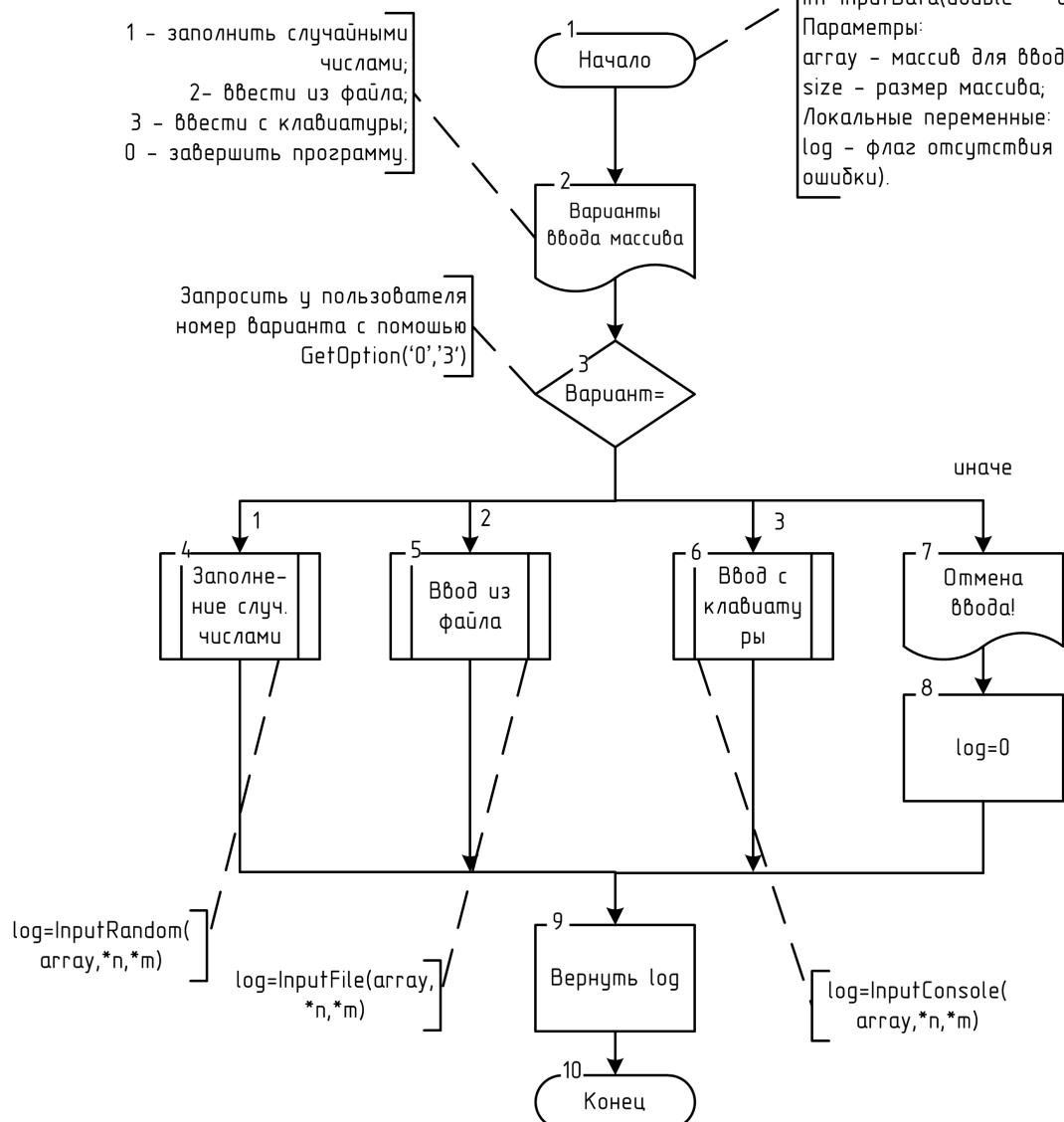
Локальные переменные:

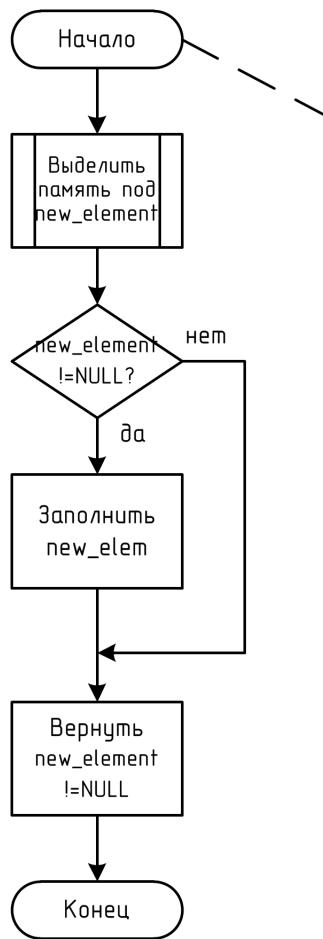
- `log` – флаг отсутствия ошибки (1 – нет ошибки);
- `min, max` – границы генерации случайных чисел;
- `i, j` – переменные-счетчики.

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.

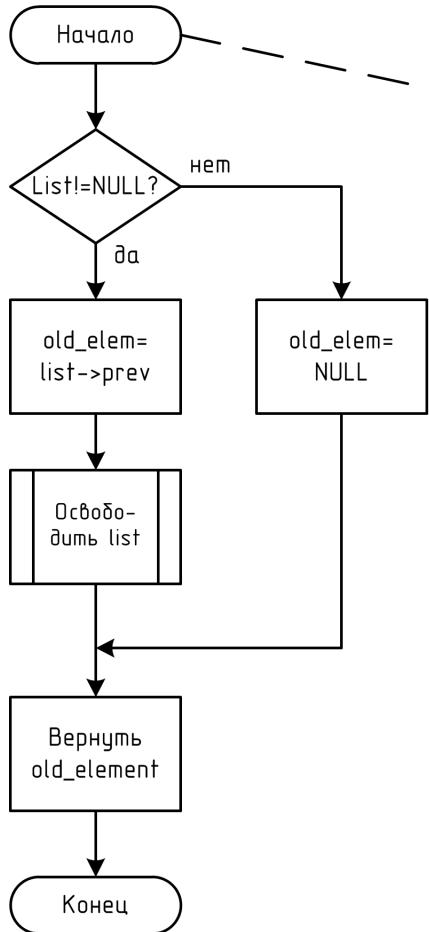
```
int InputData(double ***array, long *size);
```

Параметры:  
array - массив для ввода;  
size - размер массива;  
Локальные переменные:  
log - флаг отсутствия ошибки (1 - нет ошибки).

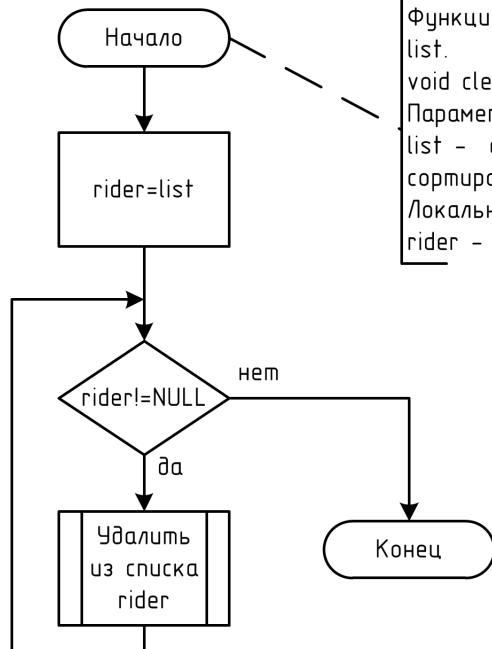




Функция push добавляет границы low и high отрезка сортировки в стек list.  
 Возвращает новый указатель на верхний(последний) элемент стека.  
 Если память под новый элемент не была получена, возвращается NULL.  
`STACK* push(STACK *list, long low, long high);`  
 Параметры:  
 high,low - верхняя и нижняя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

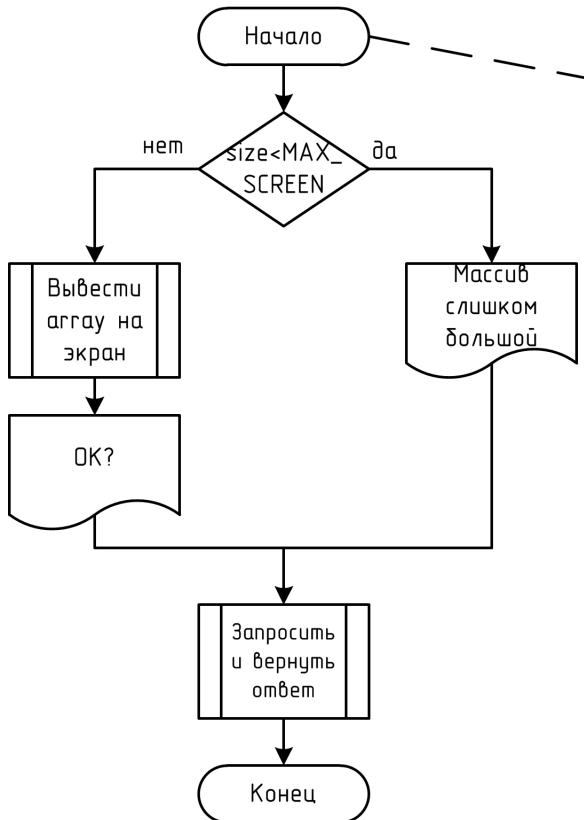


Функция `pop` удаляет последний элемент стека `list`.  
 Возвращает новый указатель на предыдущий элемент стека.  
 Если удаленный элемент был первым элементом стека, возвращается `NULL`.  
`STACK* pop(STACK *list);`  
 Параметры:  
`list` – стек для хранения границ отрезков сортировки.  
 Локальные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



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.





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 3.1 — Схема алгоритма вычисления средних арифметических значений положительных элементов каждой строки матрицы(начало)

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

Вариант №4

Лист  
94

Рисунок 3.2 — Схема алгоритма вычисления средних арифметических значений положительных элементов каждой строки матрицы(продолжение)

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

Вариант №4

Лист  
95

### 3.6. Текст программы

Ниже приведен исходный текст программы на языке Turbo Pascal 7, находящей средние арифметические значения положительных элементов каждой строки заданной 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.
*/
```

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

```

int fexists(char *fname);

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

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

/*
Основная часть программы 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 - верхняя 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.
*/
int CheckData(double **array, long size);

/*
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:

```

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

```

log - флаг отсутствия ошибки (1 - нет ошибки).
*/
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:

```

Изм.	Лист	№ докум.	Подп.	Дата

```

i,j - переменные-счетчики.
*/
void PrintMatrix(FILE *f,double **array,long size);

/*
Функция Input() получает от пользователя некоторое вещественное число param.
При этом она в процессе запроса выводит поясняющее сообщение message,
и приглашение ко вводу в виде имени параметра - name.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
message - сообщение о параметре;
name - имя параметра;
param - запрашиваемый параметр.
Локальные переменные:
ok - флаг отсутствия ошибки (1 - нет ошибки);
req_rslt - флаг ответа пользователя (1 - согласие пользователя).
*/
int Input(const char message[], //paramName - имя запрашиваемого параметра;
           const char name[], //paramCond - дополнительная информация о
параметре;
           double *param
           );

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

/*
Функция InputWithLBound() получает от пользователя некоторое вещественное число
param,
минимальное значение которого ограничено значением l_bound.
При этом она в процессе запроса выводит поясняющее сообщение message,
и приглашение ко вводу в виде имени параметра - name.
Возвращает 1, если операция прошла успешно, иначе возвращается 0.
Параметры:
message - сообщение о параметре;
name - имя параметра;
param - запрашиваемый параметр;
h_bound - верхняя граница параметра.
Локальные переменные:
ok - флаг отсутствия ошибки (1 - нет ошибки);
req_rslt - флаг ответа пользователя (1 - согласие пользователя).
*/
int InputWithLBound(const char message[],const char name[],double *param,double
l_bound);

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);

```

Изм.	Лист	№ докум.	Подп.	Дата

```

    clearline(stdin);
    if (fexists(fileName)) {
        error=1;
        printf("ВНИМАНИЕ! Указанное имя файла занято!\n");
        printf("ПЕРЕЗАПИСАТЬ ФАЙЛ? (Y/N)\n");
        error=!GetReqResult();
    }
    if (!error) {
        *f=fopen(fileName,"w");
        error=*f==NULL;
    }
    if (error) {
        printf("Неправильное имя файла! \n");
        printf( "Хотите повторить ввод? (Y/N)\n");
        req_rslt=GetReqResult();
    };
    } while (req_rslt&&error);
    return !error;
};

int main(){
    int log,req_rslt;
    double **array; 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");
        }
    }
}

```

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

```

    if (GetReqResult()) {
        if ((log=GetOutFile(&f))) {
            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{

```

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

```

        while (array[i][i]<x) i++;
        while (x<array[j][j]) j--;
        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;

```

Изм.	Лист	№ докум.	Подп.	Дата

```

do{
    req_rslt=0;
    printf("%s",message);
    printf("%s: ",name);
    log=scanf("%lf",param);
    log=!clearline(stdin)&&log;
    if (!log){ //введено не вещественное число?
        printf("Введено не число!\n"
               "Повторить ввод? Y/N\n");
        req_rslt=GetReqResult();
    }
    if (log&&(*param<l_bound)){
        log=0;
        printf("Ошибка! %s меньше %lf!\n"
               "Повторить ввод(Y/N)?\n",name,l_bound);
        req_rslt=GetReqResult();
    }
} while(!log&&req_rslt);
return log;
}

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

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

```

Изм.	Лист	№ докум.	Подп.	Дата

```

int log; FILE *f; long i,j; char ch;
log=/*1; f=stdin; */GetInFile(&f);
if (log){
    //log=InputFileLong(f,size)&&(size>0);
    log=fscanf(f,"%ld",size)&&!clearline(f);
    if (!log){
        printf("Размер массива не указан в файле!\n");
    }
} else {
    log=AllocArray(array,*size);
    if (log){
        for (i=0;(i<=*size-1)&&log;i++) {
            for (j=0;(j<=*size-2)&&log;j++) {
                //log=InputFile(f,(*array)[i]+j);
                while((ch=getc(f))==' '||ch=='\t');
                if (ch!='\n')
                    ungetc(ch,f);
                else
                    log=0;
                if (log)
                    log=fscanf(f,"%lf",(*array)[i]+j);
            }
        }
        if (log){
            while((ch=getc(f))==' '||ch=='\t');
            if (ch!='\n')
                ungetc(ch,f);
            else
                log=0;
            if (log)
                log=fscanf(f,"%lf",(*array)[i]+j)&&!clearline(f); j++;
        }
    };
    if (!log){
        //i--; j--;
        printf("Ошибка при чтении из файла элемента %ld,%ld.\n",i,j);
    }
    if (f!=stdin)
        fclose(f);
    if (!log) FreeArray(array,*size);
} else
    printf("Ошибка выделения памяти!\n");
}
} else printf("Неправильное имя файла! \n");

return log;
}

int InputRandom(double ***array,long *size){
int log=1; double min,max; long i,j; char msg[150];
sprintf(msg,"Введите m - размер матрицы. 1<=m<=%ld.\n",MAX_SIZE);
log=InputIndex(msg,"m",size,MAX_SIZE);
if (log)
    log=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)rand() / RAND_MAX * (max-min) +
min);
    }
}
}

```

Изм.	Лист	№ докум.	Подп.	Дата

```

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

int CheckData(double **array, long size) {
    printf("Будет выполнена сортировка главной диагонали матрицы %ldx%ld\n",
           size,size);
    if (size<=MAX_SCREEN_ELEM) {
        PrintMatrix(stdout,array,size);
        printf("Продолжить? (Y/N)\n");
    } else {
        printf("Массив %ldx%ld слишком большой для вывода на экран!\n",size,size);
        printf("Вы уверены в правильности введенных данных? (Y/N)\n");
    }

    return GetReqResult();
}

```

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 3.5:

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 3.8 - 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 3.7:

Таблица 3.9 - Локальные переменные функции вывода матрицы в файл или на экран

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

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 3.8:

Таблица 3.10 - Параметры функции вывода матрицы в файл

имя	тип	предназначение

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

Локальные переменные функции представлены в таблице 3.11:

Таблица 3.9 - 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 3.12:

Таблица 3.12 - Параметры функции получения вещественного числа

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

Локальные переменные функции представлены в таблице 3.13:

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 2.24 - 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 удаляемым 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**

Программа предназначена для нахождения средних арифметических значений положительных элементов строк матрицы  $A[n,m]$ . При вводе параметров необходимо соблюдать следующие условия: параметр  $n$  — натуральное число и  $1 \leq n \leq 100$ , параметр  $m$  — натуральное число и  $1 \leq m \leq 50$ , любой элемент  $A[i,j]$  — вещественное число. При неверном вводе ввод придется повторить. Можно прервать программу сочетанием клавиш Control+C. После ввода данных программа предложит их проверить, чтобы начать вычисления, нажмите 'y' и Enter, чтобы завершить программу — 'a', повторить ввод — 'r'. После окончания расчетов программа выведет искомые средние арифметических значений положительных элементов строк в столбик, под номерами, соответствующие номерам исходных строк.

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

### 3.9. Тестовый пример

Ниже на рисунке 3.3 показан пример работы программы нахождения средних арифметических значений положительных элементов строк матрицы для данных

$$m=4, n=4 \text{ и } A = \begin{matrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 0 & 0 \\ -5 & 5 & -5 & 5 \\ -5 & 0 & -4 & 2 \end{matrix}.$$

Как можно заметить, программа вывела правильные результаты;

- строка 1 — искомое значение равно  $\frac{1+2+3+4}{4} = \frac{10}{4} = 2,5$  ;
- строка 2 — так как 0 не является положительным числом — в строке нет положительных элементов;
- строка 3 — искомое значение равно  $\frac{5+5}{2} = \frac{10}{2} = 5$  ;
- строка 4 — искомое значение равно  $\frac{2}{1} = 2$  ;

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

```

Программа находит в заданном массиве A[k,l]
средние арифметические значения положительных элементов каждой строки.
Введите количество строк массива(k) - натуральное число
1<=k<=100
k:4
Введите количество столбцов массива(l) - натуральное число
1<=l<=50
l:4
Введите A[1,1] - вещественное число
A[1,1]:1
Введите A[1,2] - вещественное число
A[1,2]:2
Введите A[1,3] - вещественное число
A[1,3]:3
Введите A[1,4] - вещественное число
A[1,4]:4
Введите A[2,1] - вещественное число
A[2,1]:0
Введите A[2,2] - вещественное число
A[2,2]:0
Введите A[2,3] - вещественное число
A[2,3]:0
Введите A[2,4] - вещественное число
A[2,4]:0
Введите A[3,1] - вещественное число
A[3,1]:-5
Введите A[3,2] - вещественное число
A[3,2]:5
Введите A[3,3] - вещественное число
A[3,3]:-5
Введите A[3,4] - вещественное число
A[3,4]:5
Введите A[4,1] - вещественное число
A[4,1]:-5
Введите A[4,2] - вещественное число
A[4,2]:0
Введите A[4,3] - вещественное число
A[4,3]:-4
Введите A[4,4] - вещественное число
A[4,4]:2
Проверьте правильность данных:
  1.0000  2.0000  3.0000  4.0000
  0.0000  0.0000  0.0000  0.0000
 -5.0000  5.0000  -5.0000  5.0000
 -5.0000  0.0000  -4.0000  2.0000
Продолжить работу?
'a' - завершить программу, 'г' - повторить ввод
'y'(или любая другая клавиша) - продолжить работу,
y
Среднее арифметическое значение положительных элементов строки...
1:2.5000
2:Положительных не было...
3:5.0000
4:2.0000
Нажмите <Enter>...

```

Рисунок 3.3 - Пример работы программы нахождения среднего арифметического значения положительных элементов строки матрицы

Изм.	Лист	№ докум.	Подп.	Дата

## 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:

Изм.	Лист	№ докум.	Подп.	Дата

```

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

```

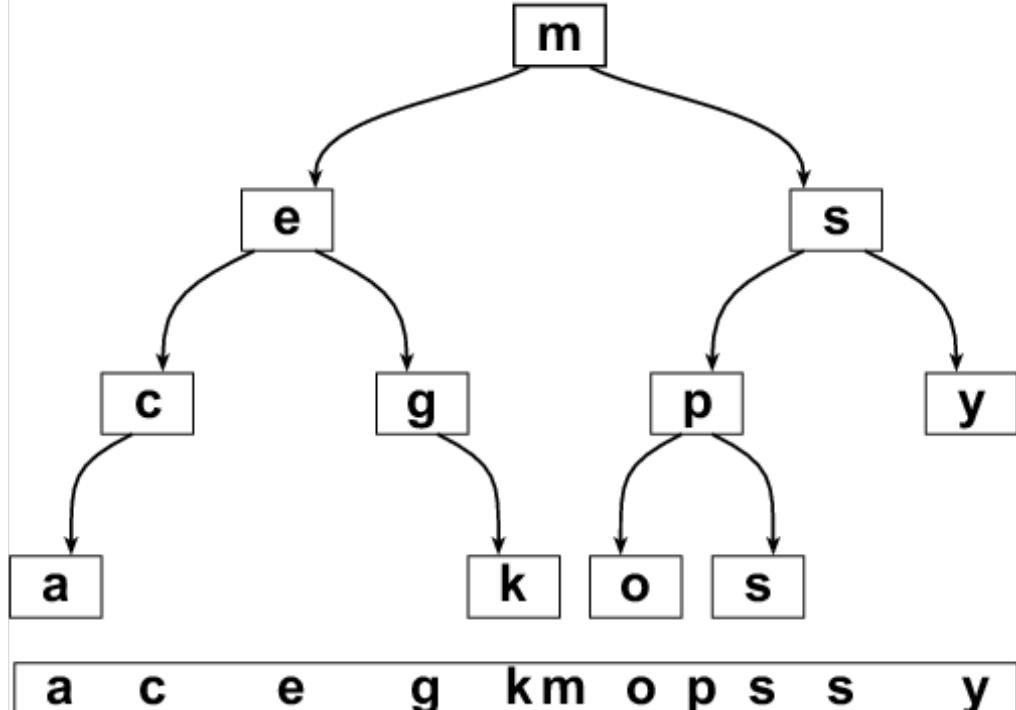


Рис. 4.1 - Двоичное дерево поиска, в котором ключами являются латинские символы упорядоченные по алфавиту

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)  $\text{left}$  и  $\text{right}$  и, соответственно, два под дерева (левое и правое) с корнями  $\text{left}$  и  $\text{right}$ .

**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.

Изм.	Лист	№ докум.	Подп.	Дата

- 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.

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

## 2. Добавление элемента (INSERT)

Дано: дерево Т и пара (K,V).

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

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  $K \geq X$ , 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 Т с корнем n и ключом K.

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

Алгоритм:

1. Если дерево Т 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

Изм.	Лист	№ докум.	Подп.	Дата

- а) найдём узел  $m$ , являющийся самым левым узлом правого поддерева с корневым узлом  $\text{Right}(n)$ ;
- б) скопируем данные (кроме ссылок на дочерние элементы) из  $m$  в  $n$ ;
- 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:

Изм.	Лист	№ докум.	Подп.	Дата

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

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

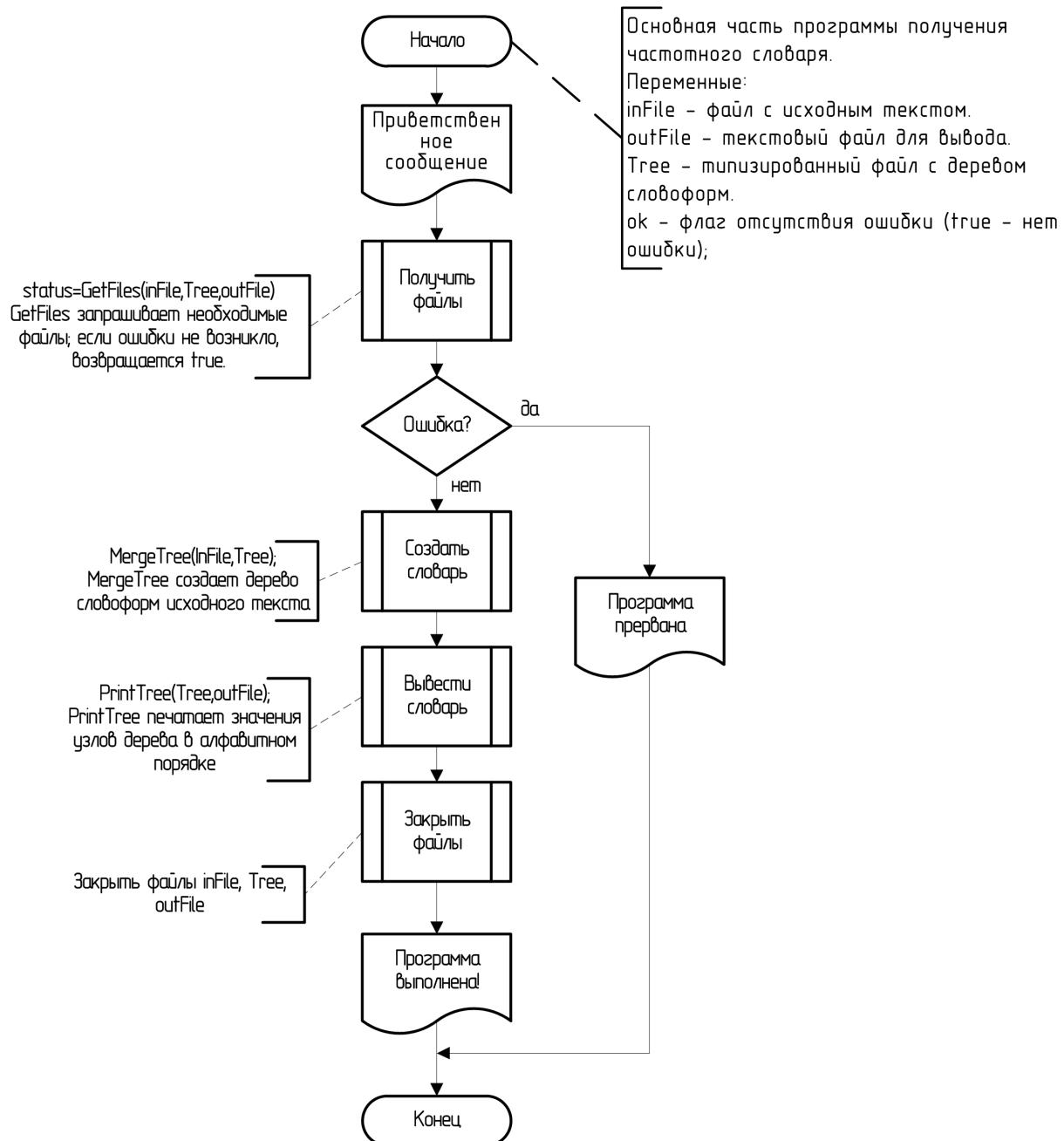
Изм.	Лист	№ докум.	Подп.	Дата

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.

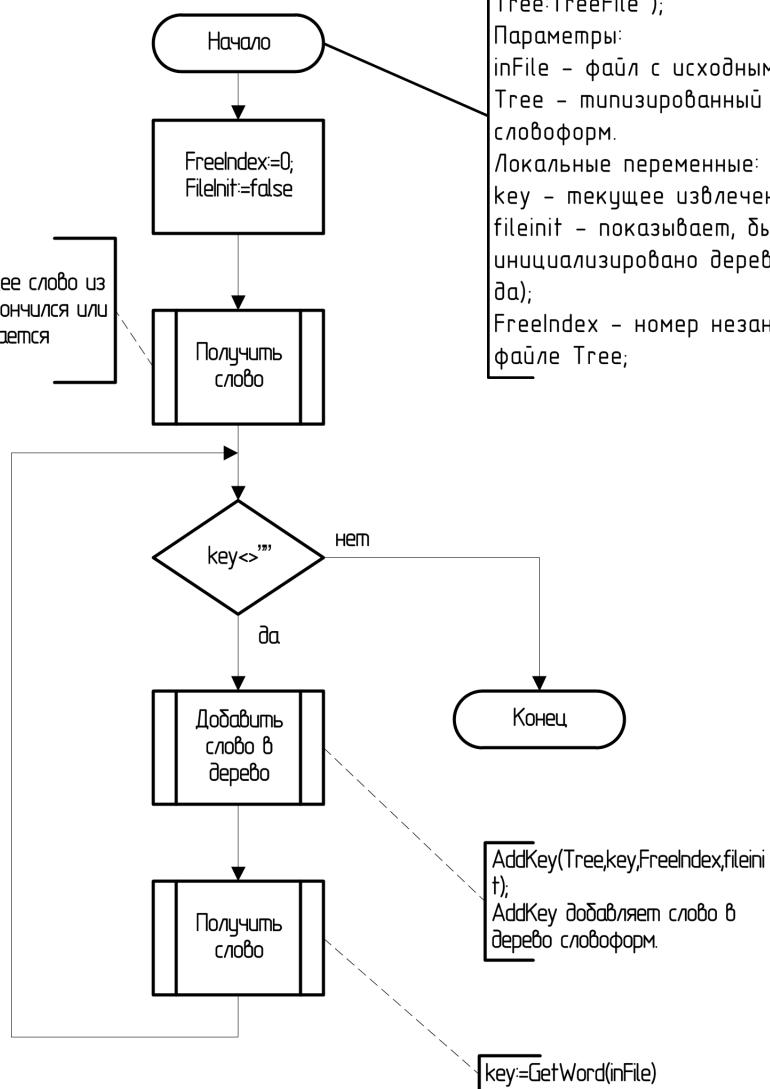
Итак, взвесив плюсы и минусы использования дерева, опишем метод решения задачи формирования алфавитного частотного словаря. Пусть ключом дерева будут слова из данного текста. Также каждый узел дерева должен зранить количество этого слова в тексте. Создадим подобное двоичное дерево, из данных, содержащихся 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.

Иzm.	Лист	№ докум.	Подп.	Дата

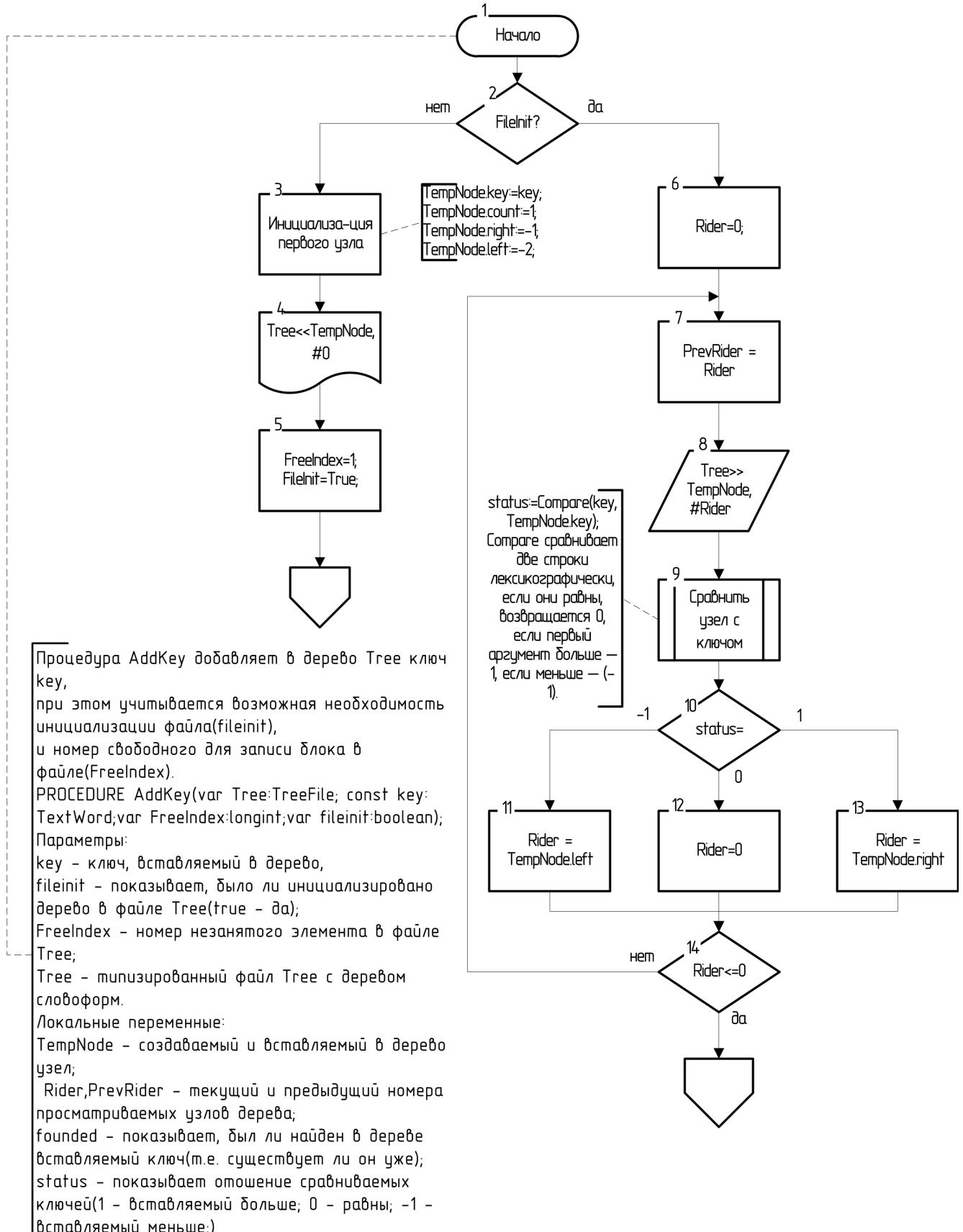
## 4.5. Разработка структуры алгоритма решения задачи

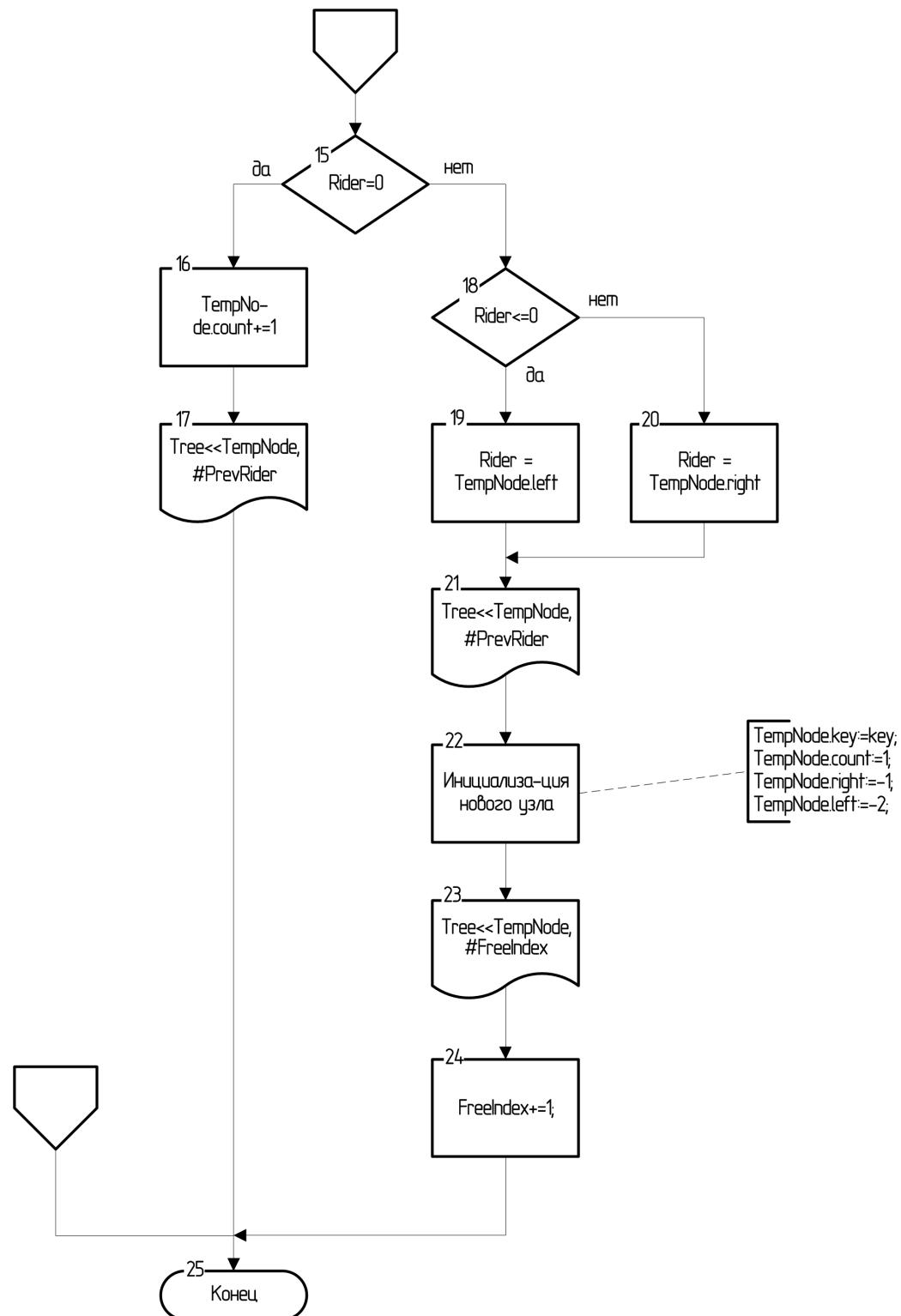


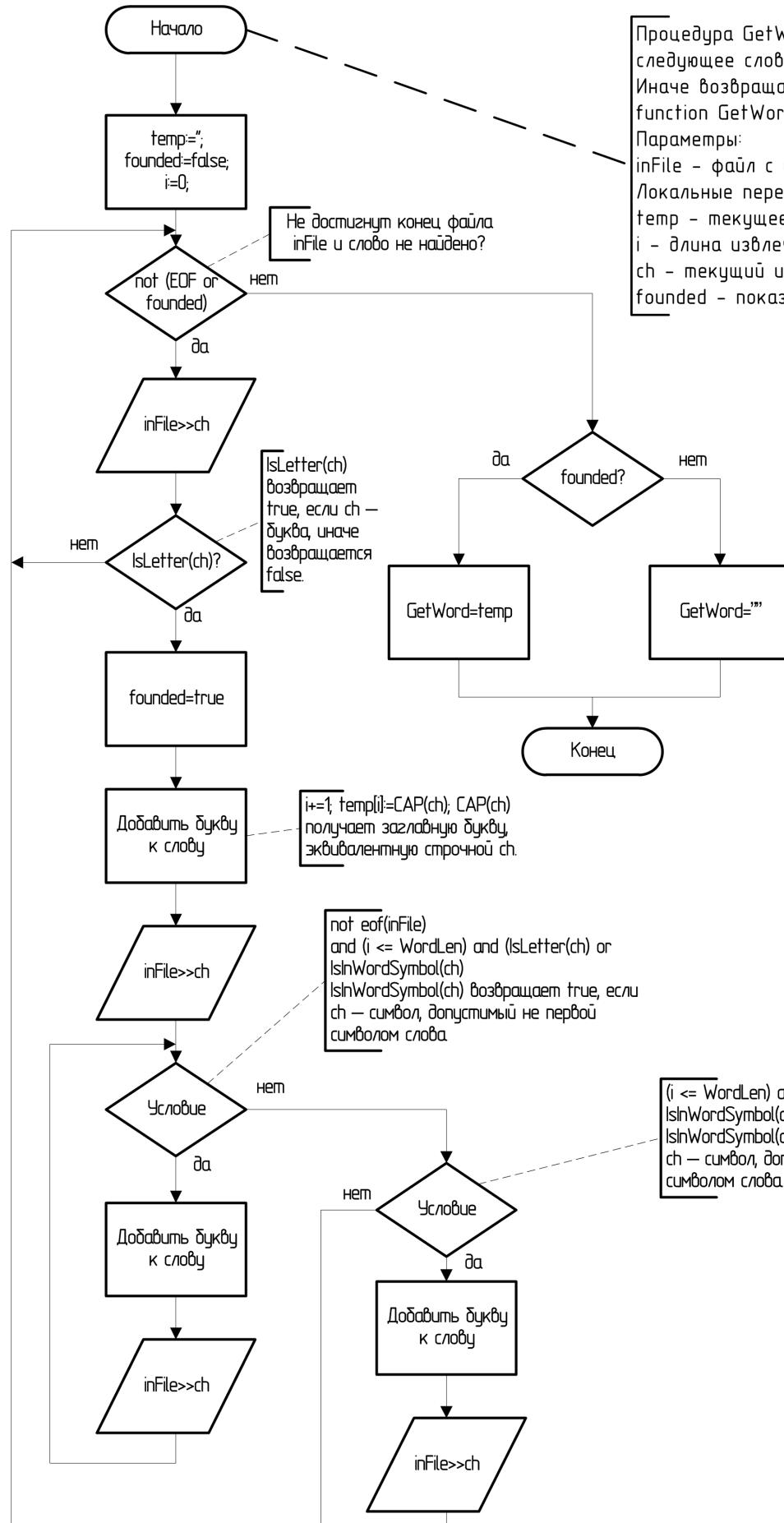
key:=GetWord(inFile)  
GetWord получает следующее слово из файла inFile; если файл закончился или слово больше нет, возвращается пустая строка.

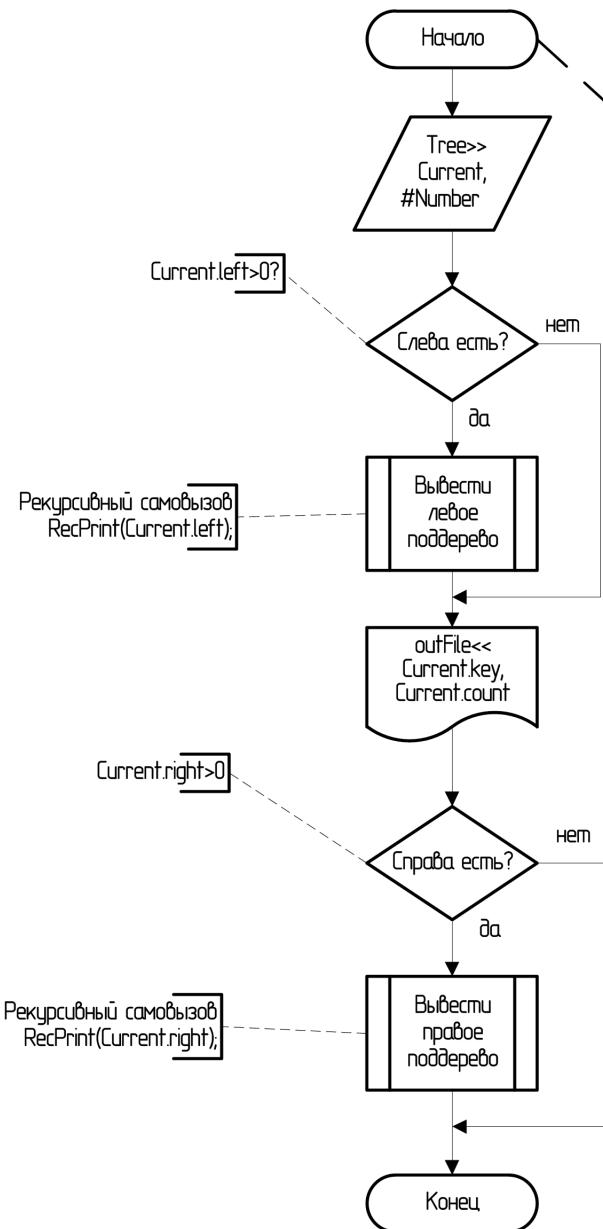


Процедура MergeTree создает в типизированном файле Tree дерево словоформ, находящихся в файле-источнике inFile.  
**PROCEDURE** MergeTree( VAR inFile: text; var Tree:TreeFile );  
**Параметры:**  
inFile – файл с исходным текстом.  
Tree – типизированный файл Tree с деревом словоформ.  
**Локальные переменные:**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









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

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

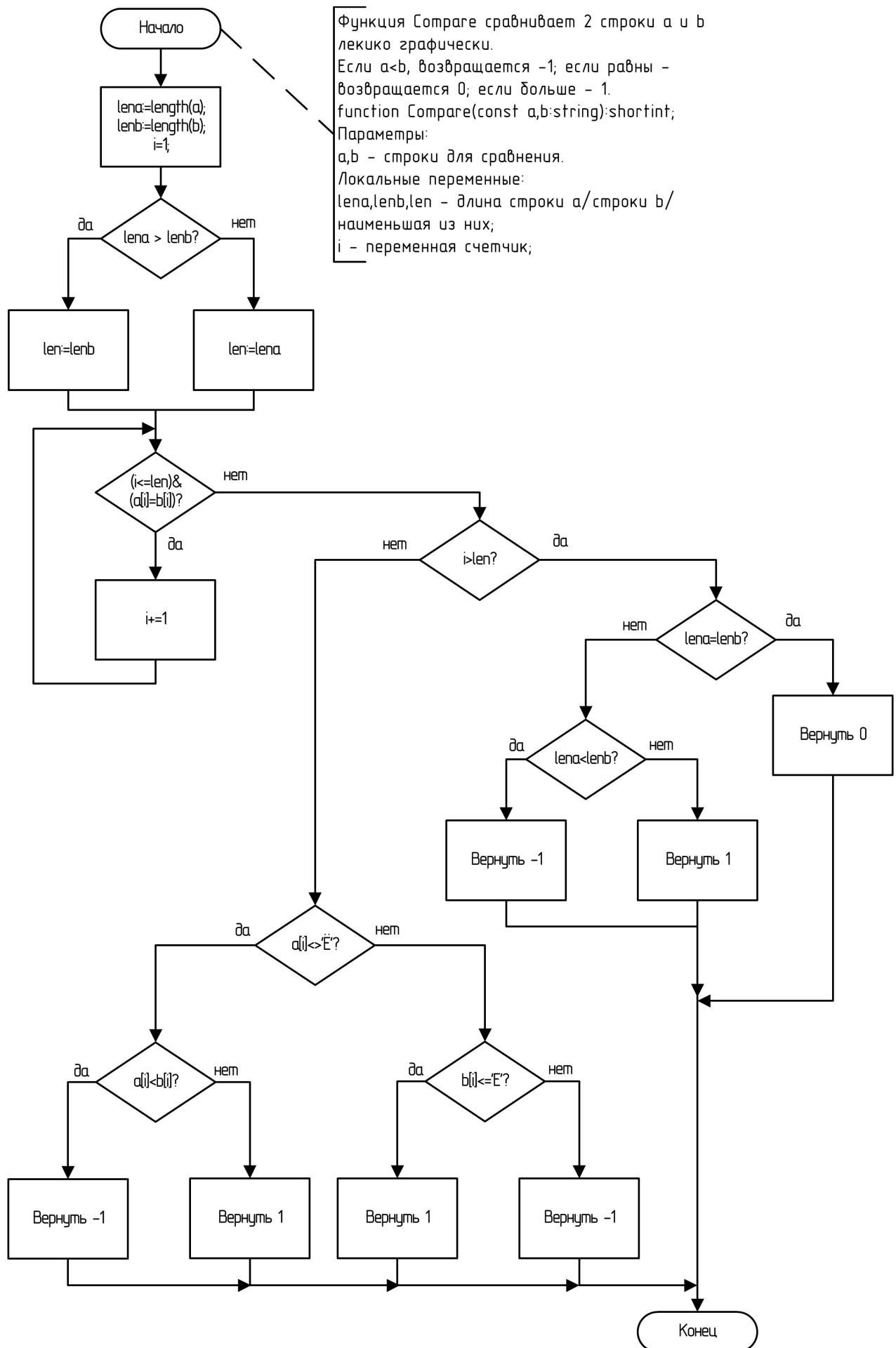
`procedure RecPrint(Number: Longint);`

Параметры:

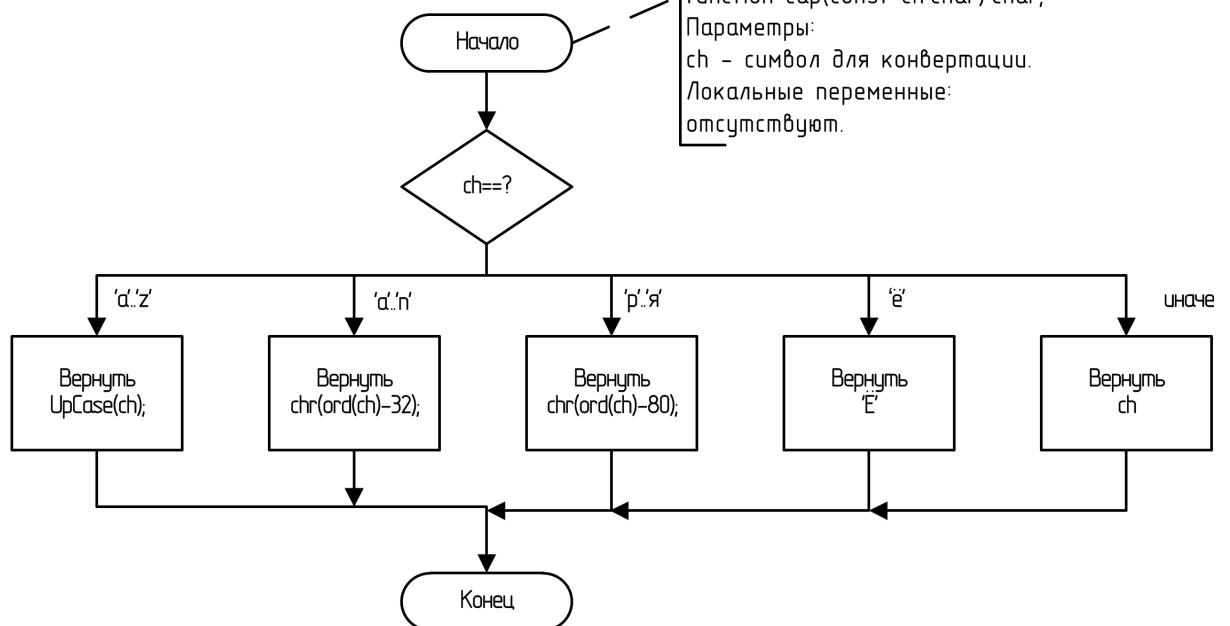
Number - номер текущего элемента для печати.

Локальные переменные:

Current - текущий элемент для печати.



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



## 4.6. Текст программы

```
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
       ('A'<=ch) and (ch<='п') or
       ('п'<=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:
отсутствуют.
*)
function IsInWordSymbol(ch:char) :boolean;
begin
```

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

```

if ('0'<=ch) and (ch<='9')
    or (ch='''') or (ch='-') or (ch='_')
        then IsInWordSymbol:=true
        else IsInWordSymbol:=false;
end;

(*
Функция cap 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 лекико графически.
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]<='E' then
                Compare:=1
            else
                Compare:=-1;
    end;
end;

```

```

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

```

Изм.	Лист	№ докум.	Подп.	Дата

```

        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 отношение 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
        {**Root node position**}
        if FileInit then begin {**File initialized**}
            {**Read root node**}
            Rider:=0;

```

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

```

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
end {FileExist};

function FileExists(var f:text):boolean;
var exist:boolean;

```

Изм.	Лист	№ докум.	Подп.	Дата

```

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:
error - флаг ошибки (true - ошибка);
req_rslt - флаг ответа пользователя (true - согласие пользователя).
*)
function GetTmpFile(var f:TreeFile) :boolean;

```

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

```

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

```

Изм.	Лист	№ докум.	Подп.	Дата

```

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

Объявлена константа WordLen – максимальная длина слова.

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

Объявлен тип-запись Node – узел дерева. Описание его полей

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.

Функция IsLetter проверяет, является ли символ ch буквой.

Если да, возвращает true, иначе – false.

function IsLetter(ch:char):boolean;

Параметры:

Таблица 2.27 - Параметры функции получения варианта

имя	тип	предназначение
ch	char	символ для проверки.

Локальные переменные:

отсутствуют.

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

Если да, возвращает true, иначе - false.

function IsInWordSymbol(ch:char):boolean;

Параметры:

Таблица 2.27 - Параметры функции получения варианта

имя	тип	предназначение
ch	char	символ для проверки.

Локальные переменные:

отсутствуют.

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:

Таблица 2.27 - Параметры функции получения варианта

имя	тип	предназначение
ch	char	символ для конвертации.

Локальные переменные:

отсутствуют.

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:

Таблица 2.27 - Параметры функции получения варианта

имя	тип	предназначение
a,b	string	строки для сравнения.

Локальные переменные:

Таблица 2.5 - 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;

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:

Таблица 2.27 - Параметры функции получения варианта

имя	тип	предназначение

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

Локальные переменные:

отсутствуют.

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:

Таблица 2.27 - Параметры функции получения варианта

имя	тип	предназначение
Number	LongInt	номер текущего элемента для печати.

Локальные переменные:

Таблица 2.5 - Локальные переменные функции получения файла для записи

имя	тип	предназначение
Current	Node	текущий элемент для печати.

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

Иначе возвращается пустая строка.

function GetWord(var inFile:text):TextWord;

Параметры:

Таблица 2.27 - Параметры функции получения варианта

имя	тип	предназначение
inFile	text	файл с исходным текстом.

Локальные переменные:

Таблица 2.5 - 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.

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

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

Параметры:

Таблица 2.27 - Параметры функции получения варианта

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

Локальные переменные:

Таблица 2.5 - 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;

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:

Таблица 2.27 - 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:

Таблица 2.5 - 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;)

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:

Таблица 2.27 - Параметры функции получения варианта

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

Локальные переменные:

Таблица 2.5 - Локальные переменные функции получения файла для записи

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

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 2.4:

Таблица 2.4 - Параметры функции получения типизированного файла

имя	тип	предназначение
f	TreeFile	запрашиваемый файл.

Локальные переменные функции представлены в таблице 2.5:

Таблица 2.5 - Локальные переменные функции получения типизированного файла

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

6. 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 2.2:

Таблица 2.2 - Параметры функции получения файла для чтения

<b>имя</b>	<b>тип</b>	<b>предназначение</b>
f	text	запрашиваемый файл.

Локальные переменные функции представлены в таблице 2.3:

Таблица 2.3 - Локальные переменные функции получения файла для чтения

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

7. 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 2.4:

Таблица 2.4 - Параметры функции получения файла для записи

<b>имя</b>	<b>тип</b>	<b>предназначение</b>
f	text	запрашиваемый файл.

Локальные переменные функции представлены в таблице 2.5:

Таблица 2.5 - Локальные переменные функции получения файла для записи

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

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 2.26.

Таблица 2.4 - Локальные переменные функции получения ответа

имя	тип	предназначение
answer	char	Ответ пользователя.

## **Заключение**

Выполняя данную работу, я закрепил свои знания и умения в программировании на языке Turbo Pascal 7.0, а также в создании схем алгоритмов. При написании программ я использовал операторы условия, операторы цикла, массивы и другие средства языка программирования Turbo Pascal 7.0. Работая, я применял различные математические методы. Были написаны три функциональные 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 нахождения средних арифметических значений положительных элементов строк матрицы, заданной пользователем.

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

*Вариант №4*

Лист

153

## **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

154