Четвериков Александр Владимирович

Учебное пособие
по алгоритмам и их графическому представлению

Белгород 2020

Содержание

Введение 3
Алгоритм и правила построения блок-схем 4
Понятие и свойства алгоритма 4
Описание алгоритма с помощью блок-схемы 5
Правила оформления блок-схем 8
Примеры оформления простейших блок-схем 9
Примеры алгоритмов 14
Поиск минимума(максимума) в последовательности 14
Вычисление значений функции 16
Сумма элементов массива 18
Примеры сложных алгоритмов 19
Обработка двумерных массивов, разбиение алгоритма на функции 19
Рекурсивная обработка последовательности значений 21
Рекурсивная обработка одномерного массива 22

Введение

Данное учебное пособие предназначено для учащихся, планирующих представлять свои алгоритмы в структурированном графическом виде.

Алгоритм и правила построения блок-схем

Понятие и свойства алгоритма

Слово *«алгоритм»* происходит от algorithm — латинского написания имени аль-Хорезми, под которым в средневековой Европе знали величайшего математика из Хорезма (город в современном Узбекистане) Мухаммеда бен Мусу, жившего в 783-850 гг., который сформулировал правила выполнения 4 арифметических действий над многозначными числами.

Алгоритм – это конечная последовательность однозначных предписаний исполнителю, позволяющая решить задачу.

Свойства алгоритма:

1. **Дискретность** – алгоритм должен представлять процесс решения задачи как последовательность простых шагов.
2. **Определенность** – каждая команда алгоритма должна быть четкой и однозначной.
3. **Результативность** – алгоритм должен приводить к решению за конечное число шагов.
4. **Массовость** – алгоритм позволяет решать целый класс однотипных задач, различающихся лишь исходными данными.
5. **Конечность** – каждое из действий и весь алгоритм в целом обязательно завершаются.

Основные виды алгоритмов:

1. **Линейные**. Предполагают последовательное выполнение действий друг за другом.
2. **Ветвления**. Содержат хотя бы одну проверку условия, в результате которой обеспечивается переход на один из возможных вариантов решения.

3. **Циклические**. Предусматривают многократное повторение одной и той же последовательности действий. Количество повторений обусловливается исходными данными или условием задачи.

Любая алгоритмическая конструкция может содержать в себе другую конструкцию того же или иного вида, т. е. алгоритмические конструкции могут быть вложенными.

Способы задания алгоритмов:

1. Словесно-формульный
2. Графический
3. Табличный
4. Блок-схема
5. Описание на каком-либо языке программирование (программа).

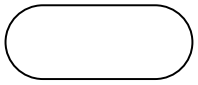
Одной из форм записи алгоритма является графическое представление в виде структурных блок-схем.

Описание алгоритма с помощью блок-схемы

Блок-схема – это графическое представление алгоритма, в котором операции изображены с помощью различных геометрических фигур, причем каждому типу операций соответствует своя фигура. Содержимое операции записывается внутри фигуры, а последовательность их выполнения изображается в виде линий, соединяющих соответствующие фигуры.

Используемые обозначения в блок-схемах:

1. Терминатор – блок начала или конца алгоритма. Может включать входные или выходные параметры при указании в начале или конце алгоритма соответственно.

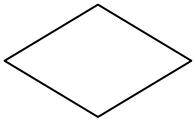


2. Данные – отображает ввод/вывод данных (например, получение значения переменной, вывод результатов на экран, считывание/запись информации в файл).

3. Процесс – отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации).

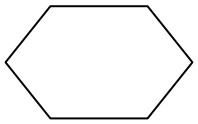
4. Предопределенный процесс – отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле).

5. Решение – отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа. Соответствующие результаты вычисления могут быть записаны по соседству с линиями, отображающими эти пути.



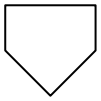
6. Подготовка – отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (установка

переключателя, модификация индексного регистра или инициализация программы).

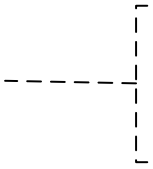


7. Соединитель – отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.

8. Межстраничный соединитель – связывает блоки, находящиеся на разных листах. Первая строка внутри межстраничного соединителя определяет номер листа, вторая – уникальное обозначение.



9. Комментарий – используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов. Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры.



Правила оформления блок-схем

1. Правила применения символов
   1.1. Блоки в схеме должны быть расположены равномерно. Следует придерживаться разумной длины соединений и минимального числа длинных линий.
   1.2. Большинство блоков задумано так, чтобы дать возможность включения текста внутри символа. Блоки должны быть, по возможности, одного размера.
   1.3. Минимальное количество текста, необходимого для понимания функции данного блока, следует помещать внутри данного блока.
   1.4. Если объем текста, помещаемого внутри блока, превышает его размеры, следует использовать символ комментария. Если использование комментария может запутать или разрушить ход схемы, текст следует помещать на отдельном листе и давать перекрестную ссылку на блок.
   1.5. Содержимое блоков должно записываться математически или словесно. Не должно быть конструкций языка (например «==», «++», «+=» из С-подобных языков).
   1.6. Описание переменных никак не указывается в блок-схеме (например, если описана переменная в начале программы, а используется в середине).
2. Правила выполнения соединений
   2.1. Потоки данных или потоки управления в схемах показываются линиями. Направление потока слева направо и сверху вниз считается стандартным.
   2.2. В случаях, когда необходимо внести большую ясность в схему (например, при соединениях), на линиях используются стрелки.

Если поток имеет направление, отличное от стандартного, стрелки должны указывать это направление.

2.3. В схемах следует избегать пересечения линий. Пересекающиеся линии не имеют логической связи между собой, поэтому изменения направления в точках пересечения не допускаются.

2.4. Линии в схемах должны подходить к символу либо слева, либо сверху, а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа.

2.5. Для блоков «решение» (ромб) исходящие линии **всегда** должны иметь подпись, которая поясняет в каком случае выбирается именно эта ветвь. В случае бинарного ветвления линии должны исходить вправо и влево, в случае множественного ветвления – исходить снизу и разделяться на несколько (Рис. 2, Рис. 3). В случае организации цикла с условием одна из линий должна исходить вниз, другая влево или вправо (Рис. 5). Необходимо придерживаться указанного в примерах способа

2.6. При необходимости линии в схемах следует разрывать для избегания излишних пересечений или слишком длинных линий, а также, если схема состоит из нескольких страниц.

2.7. Ссылки к страницам могут быть приведены совместно с символом комментария для их соединителей (в случае использования блока «соединитель» – обязательное требование).

Примеры оформления простейших блок-схем

Пример использования блока «терминатор» для обозначения границ алгоритмов:

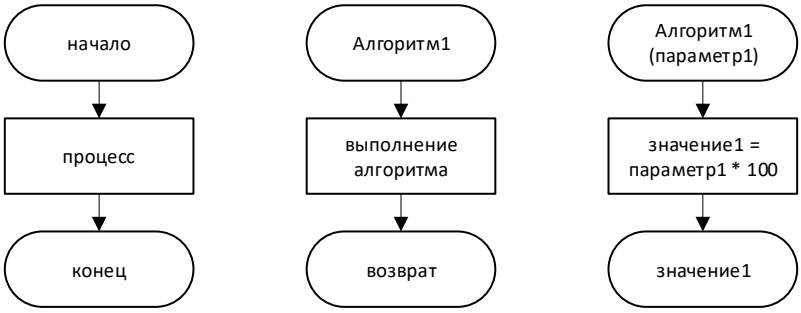


Рис. 1. Использование блоков «терминатор» и «процесс»

Примеры использования блока «Решение» для организации бинарного или множественного ветвления:

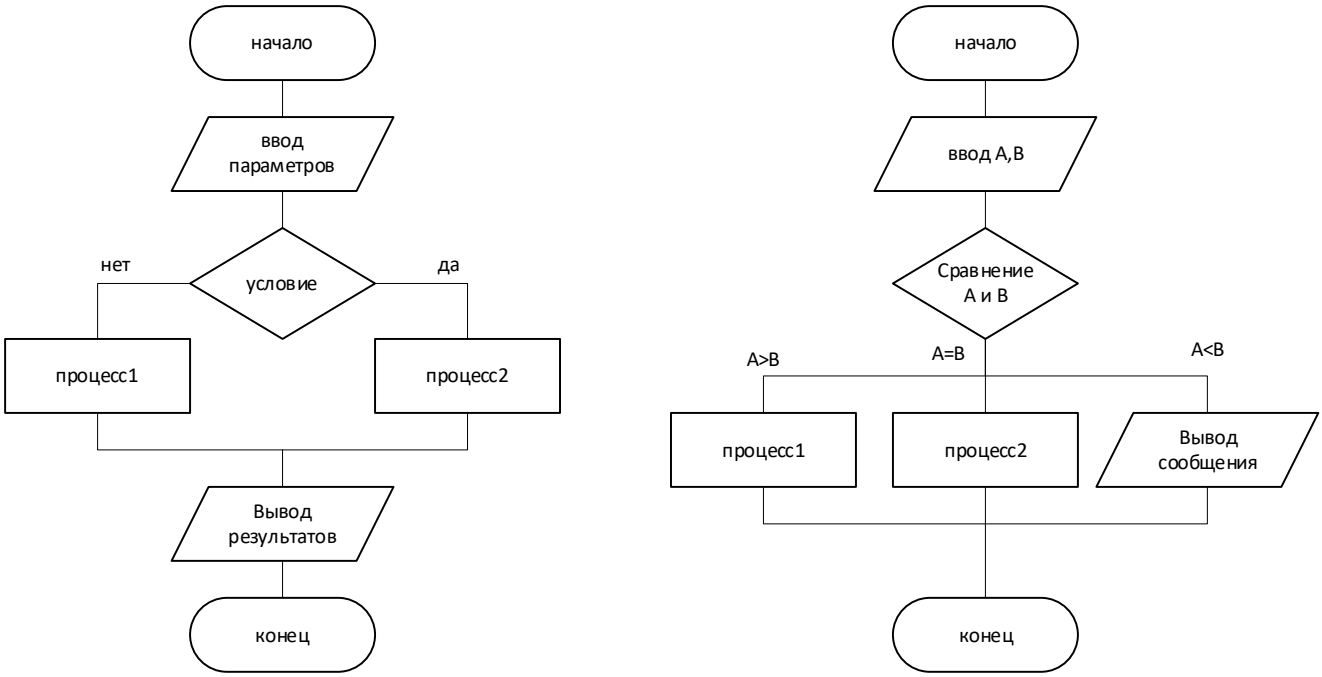


Рис. 2. Использование блока «Решение»

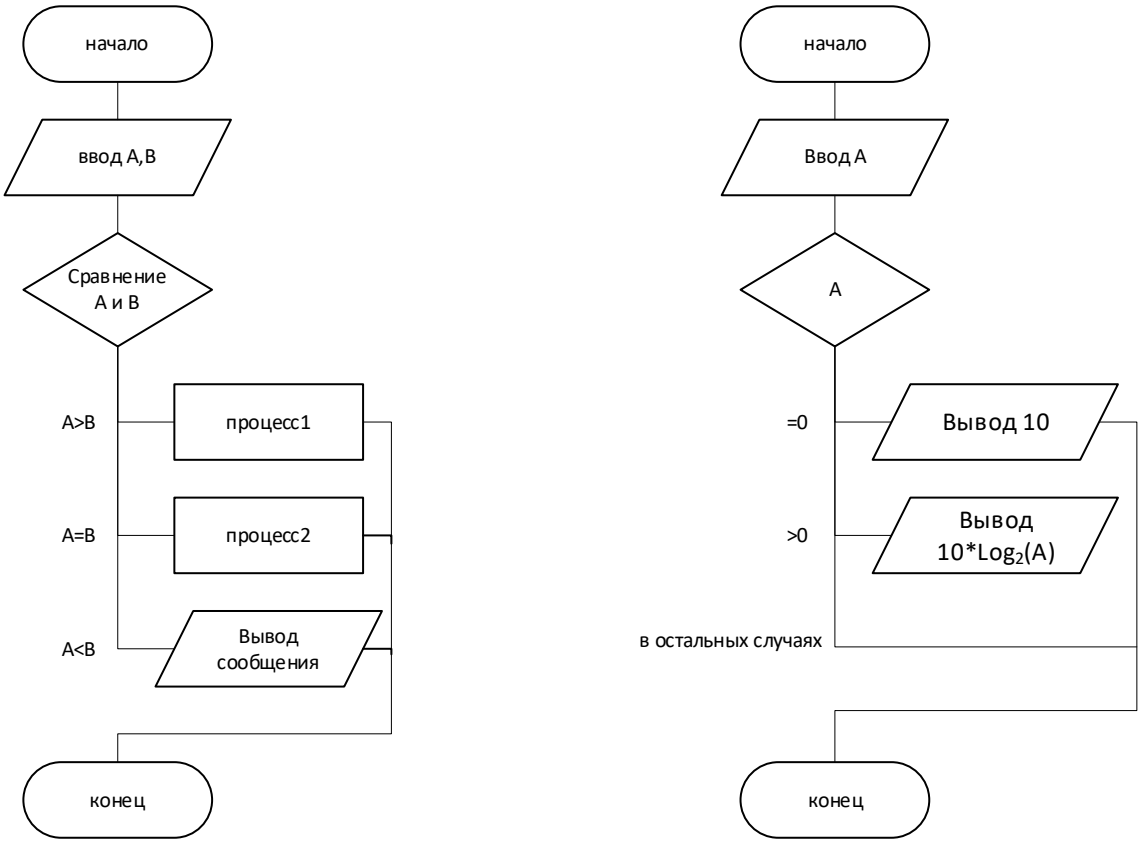


Рис. 3. Множественное ветвление (2 способ)

Пример использования блока «предопределённый процесс», а также пример использования вспомогательных алгоритмов в основном алгоритме:

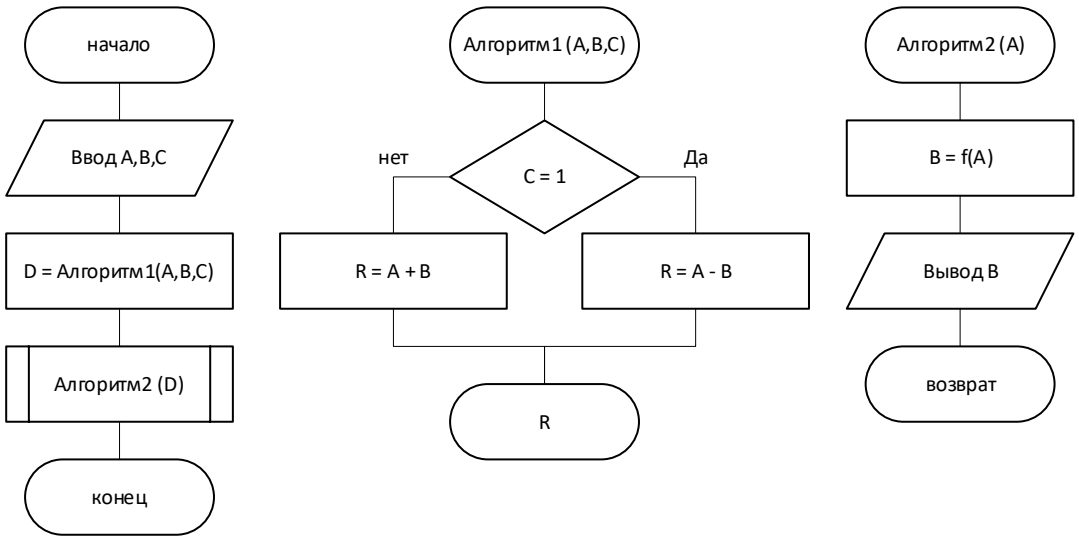


Рис. 4. Вспомогательные алгоритмы и блок «предопределённый процесс»

Пример использования блока «решение» для организации циклов с предусловием и постусловием:

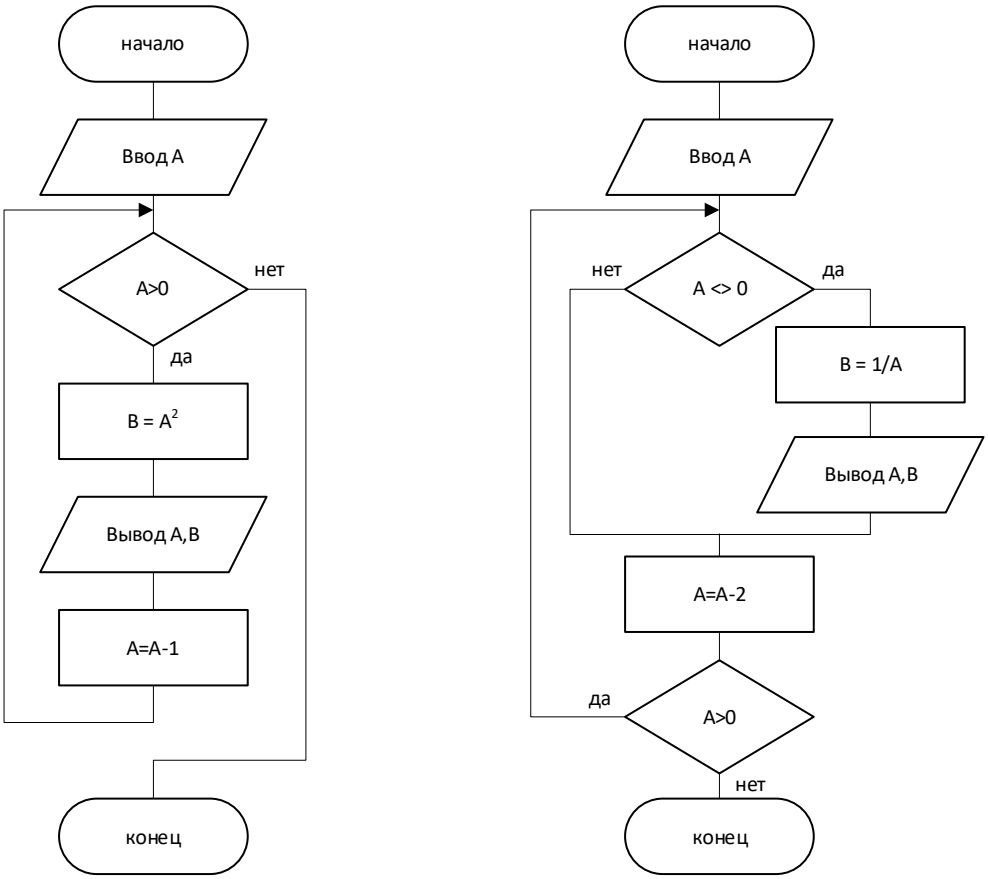


Рис. 5. Блок-схема цикла с предусловием (слева) и постусловием (справа)

Пример использования блоков «подготовка» и «соединитель» для организации цикла с фиксированным числом шагов (цикл со счётчиком), а также использования комментариев:

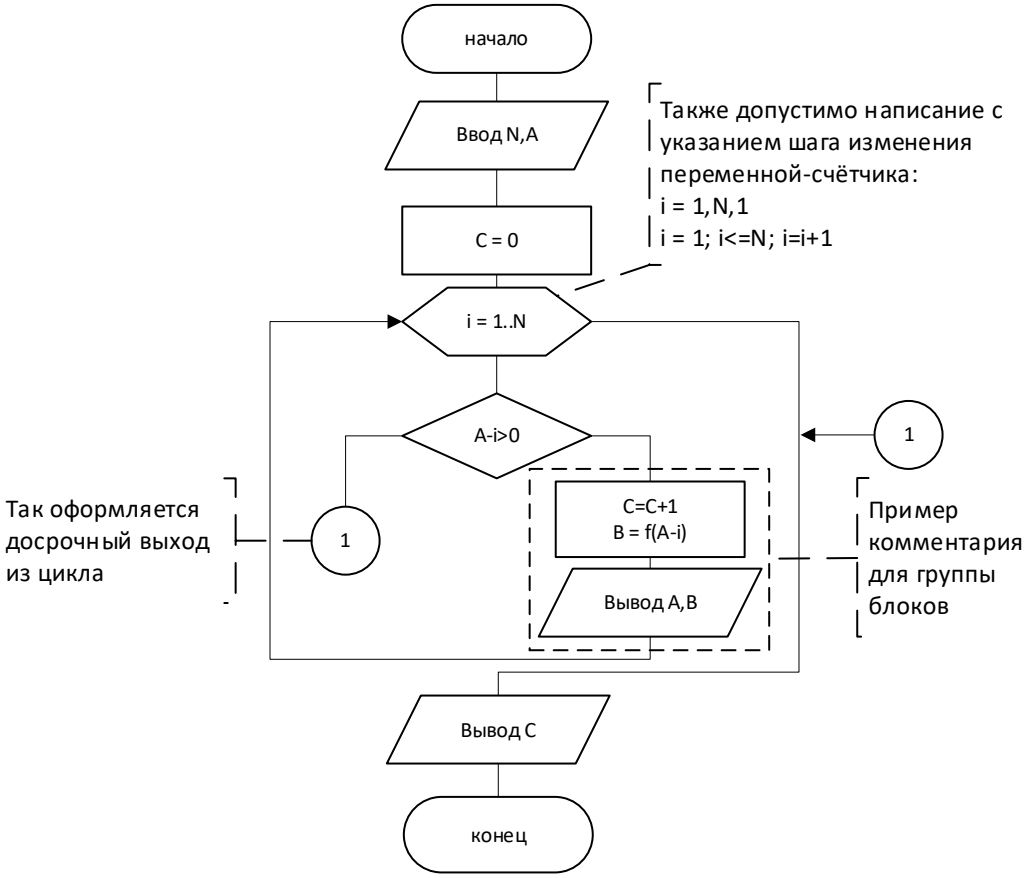


Рис. 6. Блок-схема цикла со счётчиком

Пример использования блоков «соединитель» и «межстраничный соединитель» для организации межстраничных переходов (для блока «соединитель» обязательно использовать комментарии)

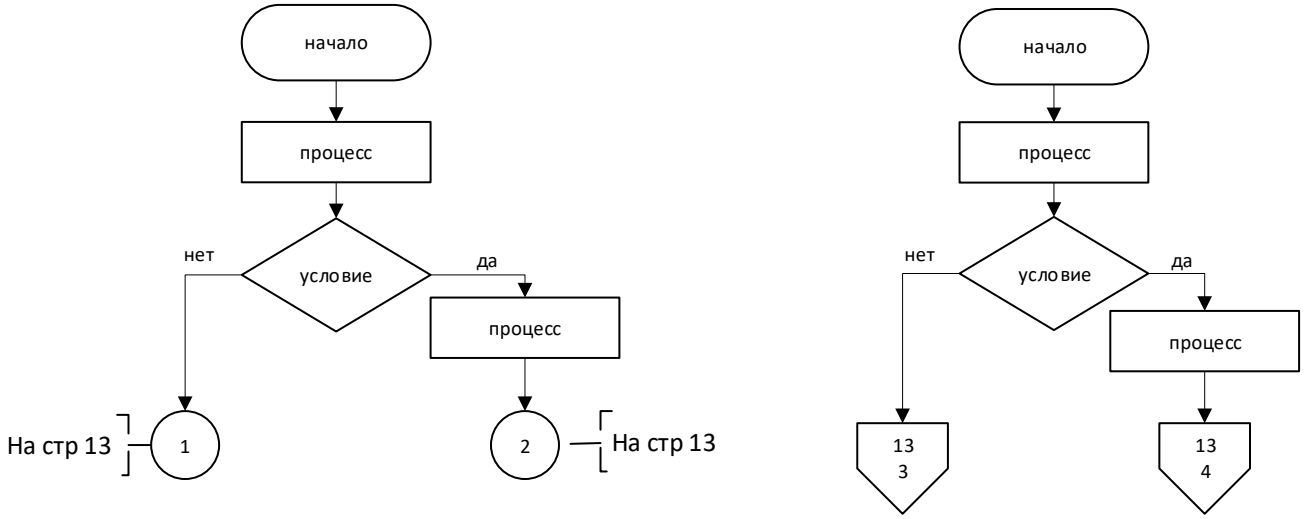


Рис. 7. Блок-схема алгоритма с переносом на другую страницу (часть 1)

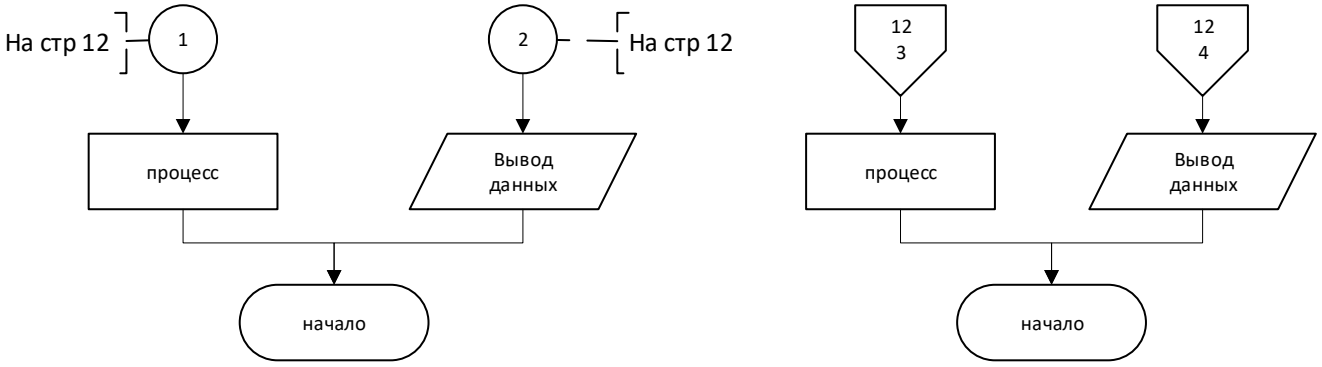


Рис. 8. Блок-схема алгоритма с переносом на другую страницу (часть 2)

Примеры алгоритмов

Поиск минимума(максимума) в последовательности

Пример поиска **минимального чётного** значения среди последовательности из n целых чисел. Данную задачу можно выполнить двумя способами. Первый способ (Рис. 9) предполагает большее количество сравнений в случае наличия хотя бы одного чётного числа, чем второй способ (Рис. 10):

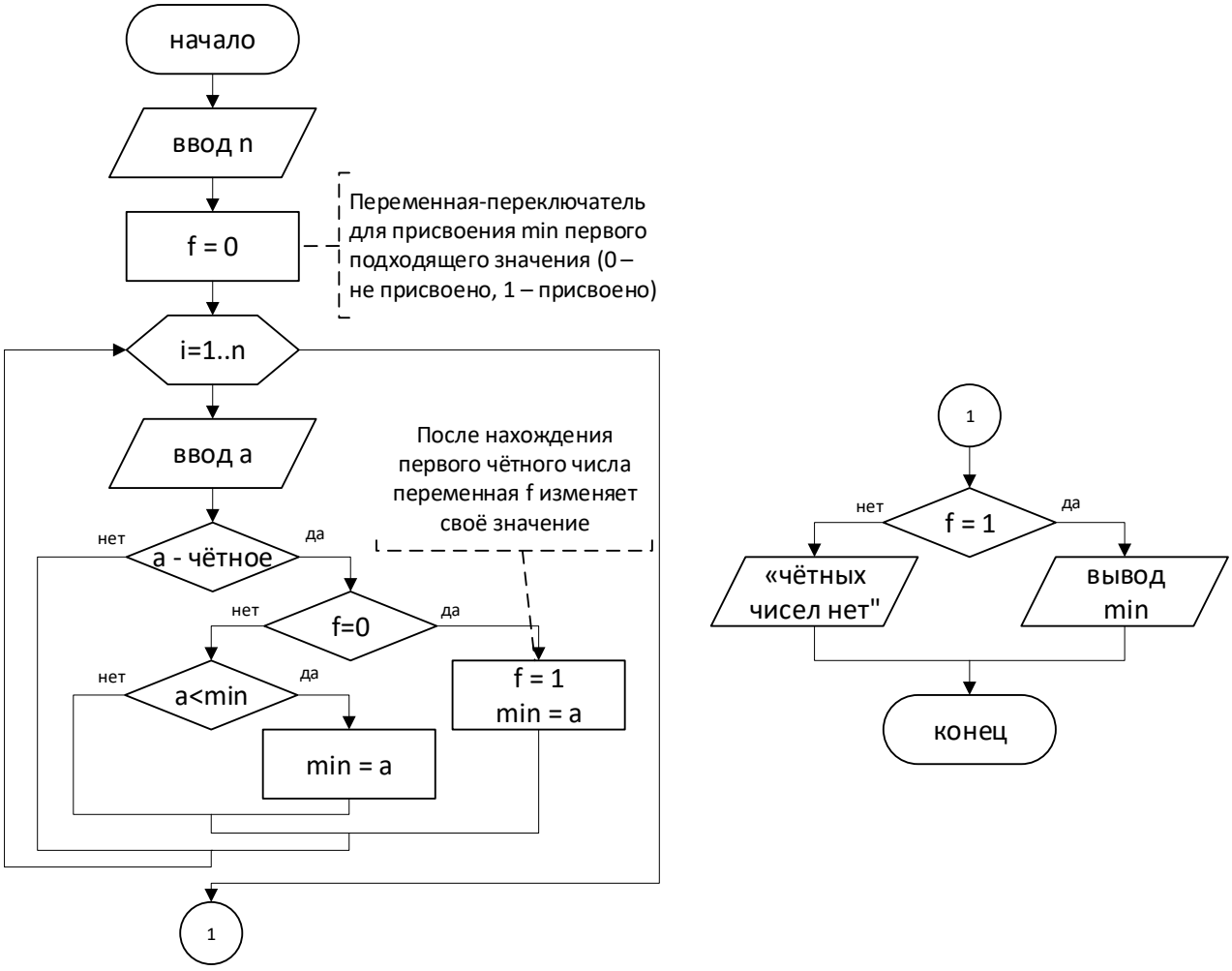


Рис. 9. Блок-схема алгоритма поиска минимального целого (1 способ)

Второй способ заключается в организации двух циклов: первый цикл с предусловием для поиска первого значения, подходящего в качестве минимального, второй цикл со счётчиком – проверка остальных значений на чётность и сравнение с минимумом.

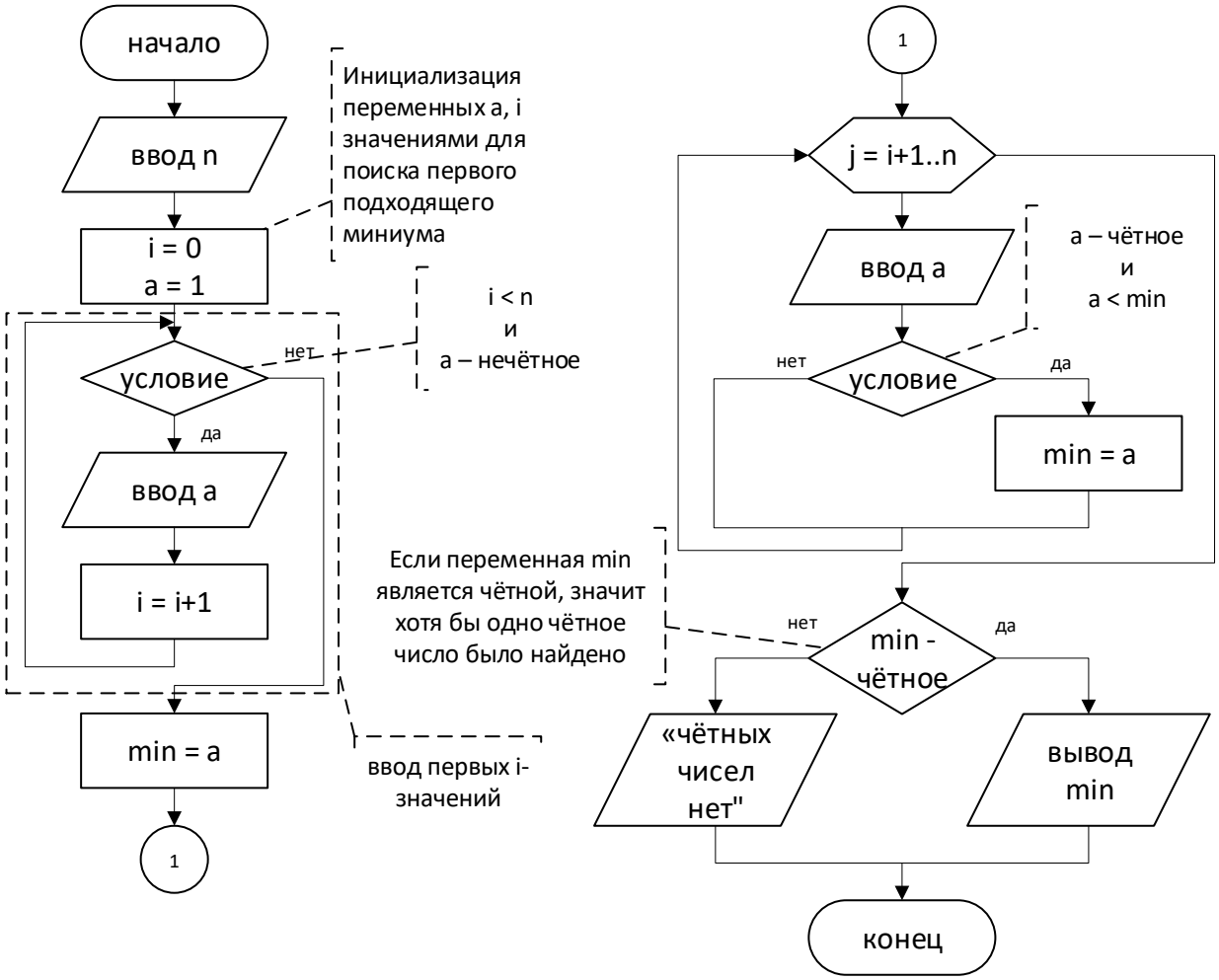


Рис. 10. Блок-схема алгоритма поиска минимального целого (2 способ)

Поиск максимума осуществляется аналогичным образом

Вычисление значений функции

Пример вычисления и вывода на экран **значений функции** y=f(x) в n-**точках** из интервала [a,b].

Перед решением данной задачи необходимо понять, что в случае, если функция f(x) в точке x_i **не определена**, необходимо вывести соответствующее сообщение **вместо** попытки вычисления $f(x_i)$. Также необходимо предусмотреть обработку некоторых выходных данных, в данном случае параметр n. Сам параметр n в рамках данной задачи может пониматься либо как количество точке на интервале [a,b], либо как количество разбиений интервала [a,b] (получится n+1 точка). Для вычисления шага h в этих двух случаях используется похожая формула.

Пример:

Входные данные a=0, b=1, n=3.

Если n – количество точек, то $h = \frac{(b-a)}{(n-1)}$, точки: x_1=0, x_2=0.5, x_3=1.

Если n – количество интервалов, то точек получится уже 4, $h = \frac{(b-a)}{n}$, сами точки: x_1=0, x2=0,33(3), x3=0,66(6), x4=1.

Для данного примера будем считать n количеством точек в интервале [a,b], а значит n должно быть не меньше 2. В качестве f(x) возьмём

$$f(x) = \frac{\ln(x+2)}{x^2-9} + e^{\sqrt{\sin(x*\pi)}} + x^2$$

В данной функции существует 3 потенциальных места, которые влияют на ОДЗ и возможность вычисления: выражение под натуральным логарифмом, выражение в знаменателе и выражение под корнем. Для первых двух легко найти конкретные значения аргумента и проверить их. Для подкоренного выражения непосредственная проверка аргумента затруднена, поэтому можно проверить подходит ли значение выражения $\sin(x * pi)$ нашему ОДЗ (результат должен быть больше или равен 0).

Графическое представление f(x) на Рис. 11.

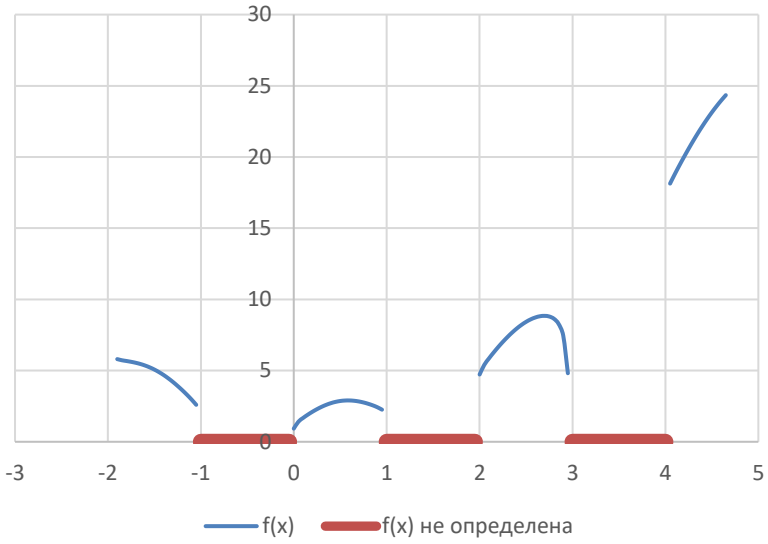


Рис. 11. График функции f(x)

Пример алгоритма в виде блок-схемы на Рис. 12:

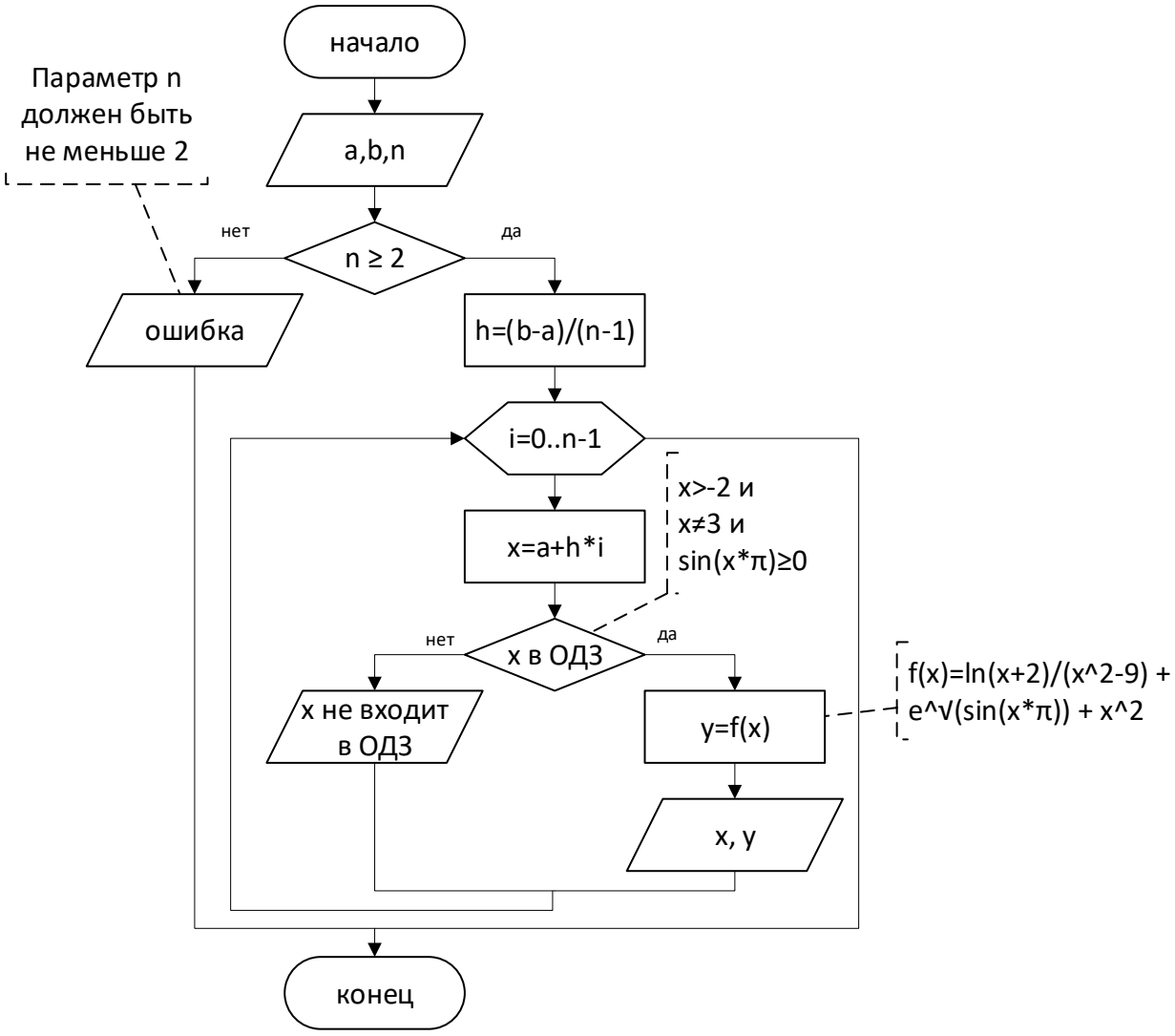


Рис. 12. Блок-схема алгоритма вычисления значений функции

Описанную выше задачу можно дополнить нахождением **среднего значения** всех вычисленных значений функции, а также **максимума** среди этих значений. Для этого необходимо скомбинировать соответствующие алгоритмы (Рис. 9, Рис. 12).

Сумма элементов массива

Пример описания действий над массивами (считывание, обработка, вывод). Вводится последовательность вещественных чисел, необходимо вычислить сумму элементов последовательности, которые меньше среднего значения всех элементов последовательности. Для решения этой задачи последовательность необходимо сохранить в массив:

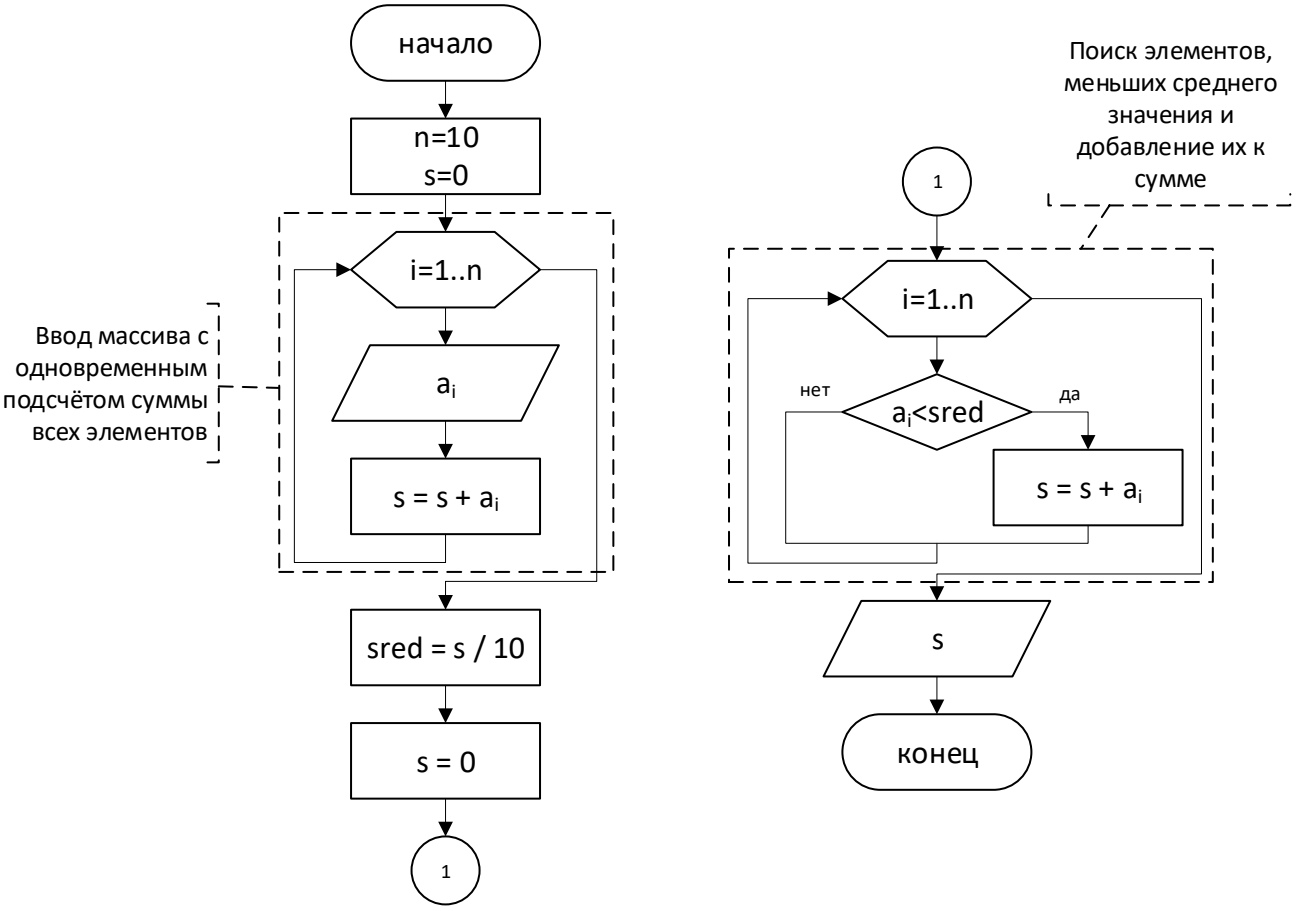


Рис. 13. Блок-схема алгоритма обработки массива

Примеры сложных алгоритмов

Обработка двумерных массивов, разбиение алгоритма на функции

Пример описания действий над двумерными массивами, файлами и выбором функции для дальнейшего использования (например, через указатель в случае языка C/C++). Задание:

Дана целочисленная матрица A(5x5). Определить массив X из пяти элементов, равных элементам побочной диагонали матрицы A. Элементы матрицы по выбору пользователя либо вводятся с клавиатуры, либо генерируются случайным образом (Рис. 14, Рис. 15, Рис. 16, Рис. 17).

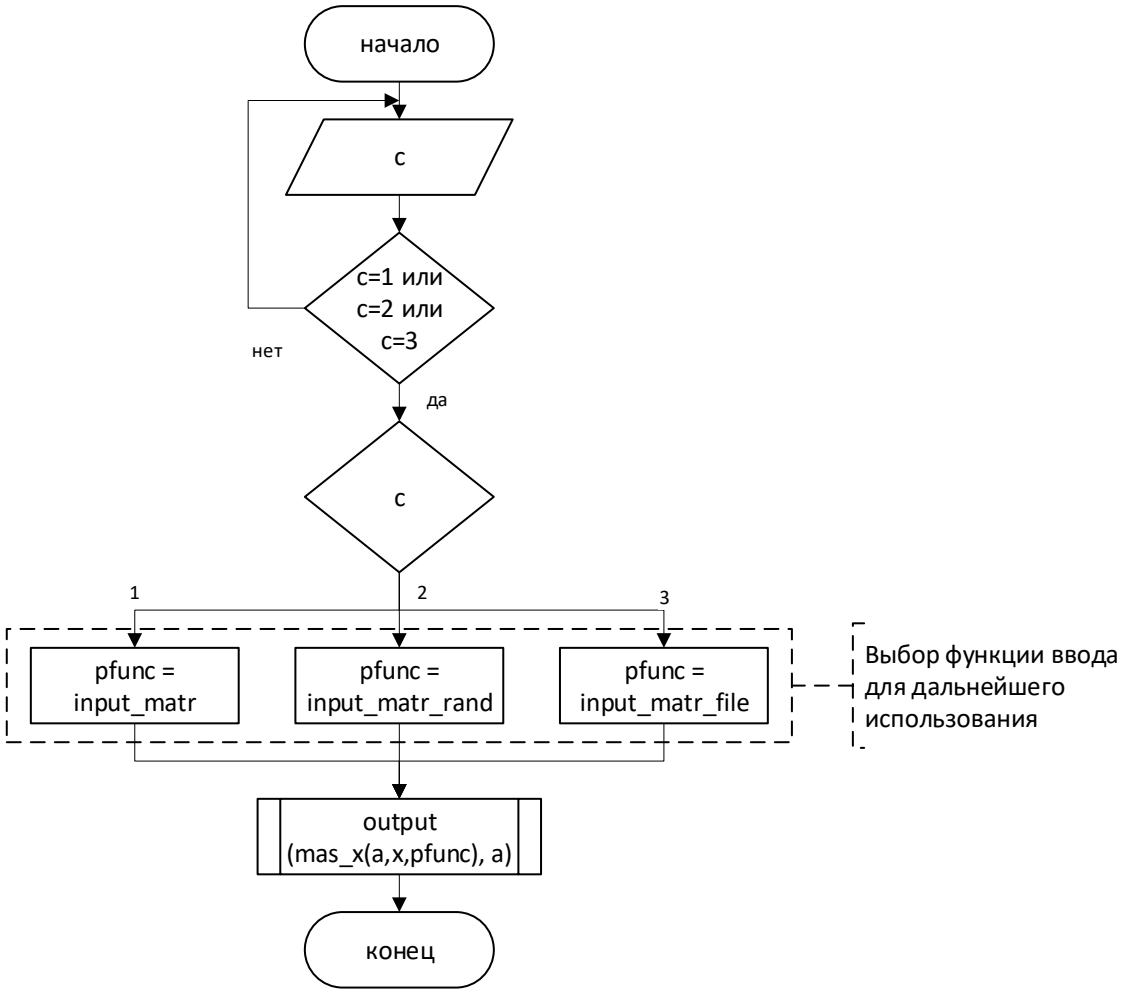


Рис. 14. Основной алгоритм программы

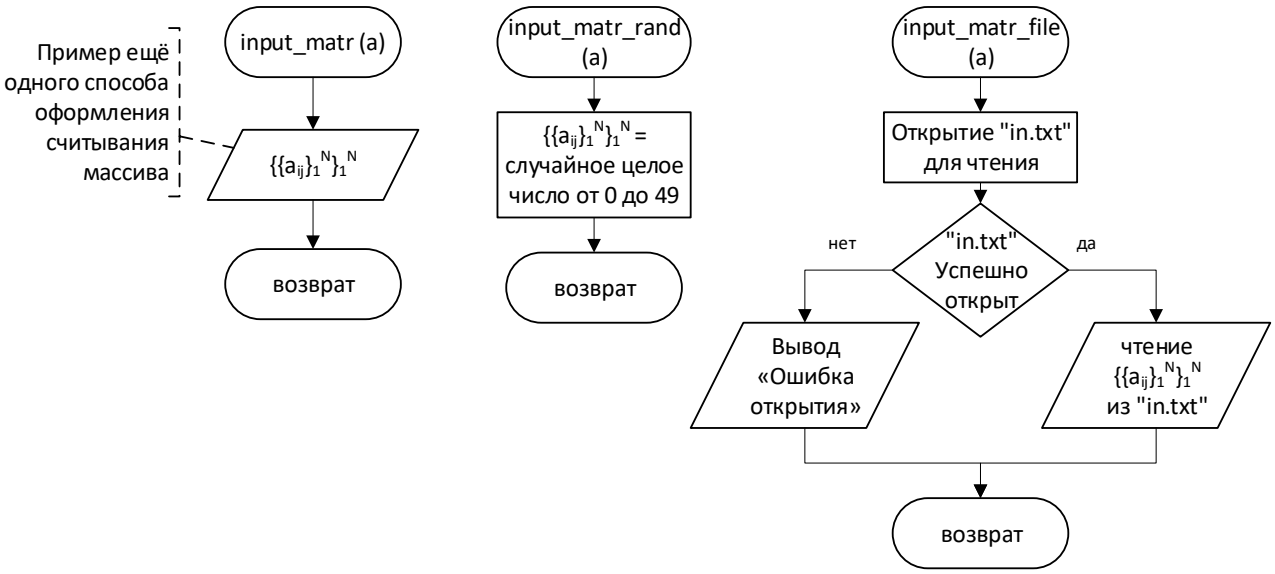


Рис. 15. Блок-схемы алгоритмов ввода двумерного массива

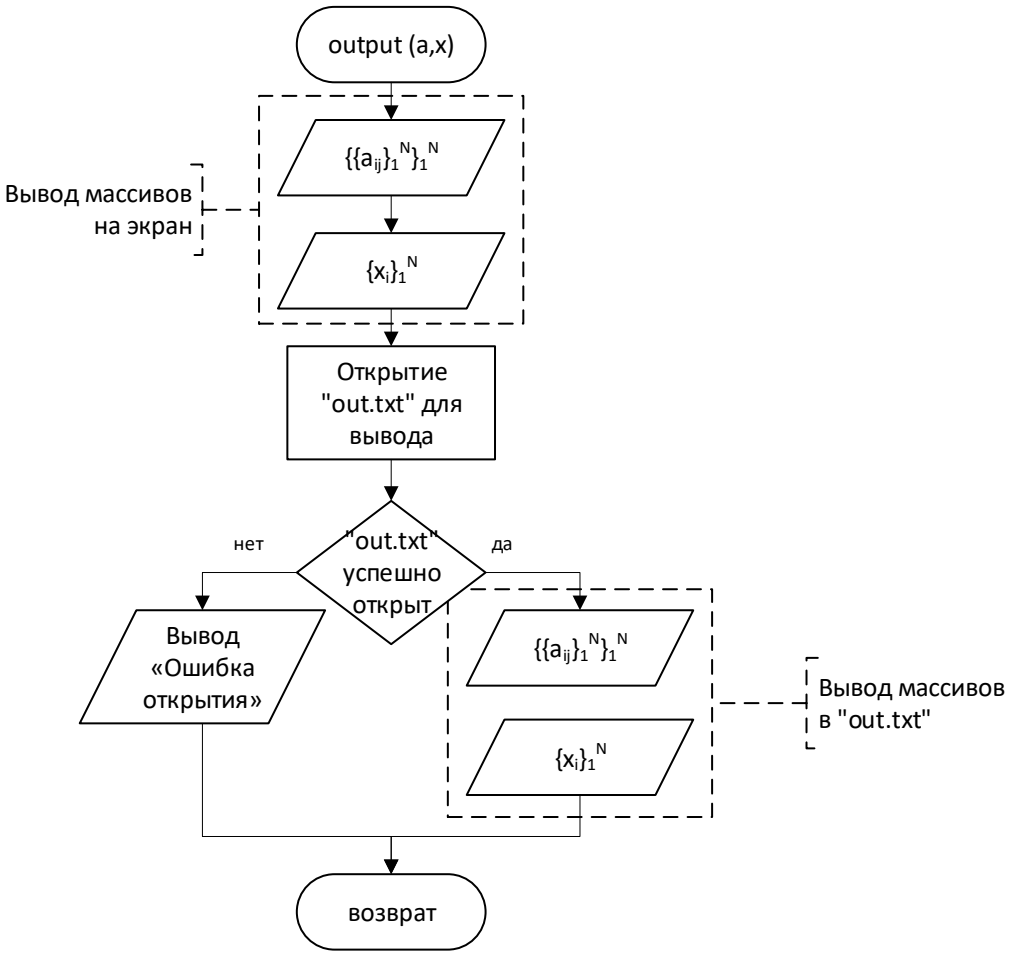


Рис. 16. Блок-схема алгоритма вывода массивов на экран и в файл

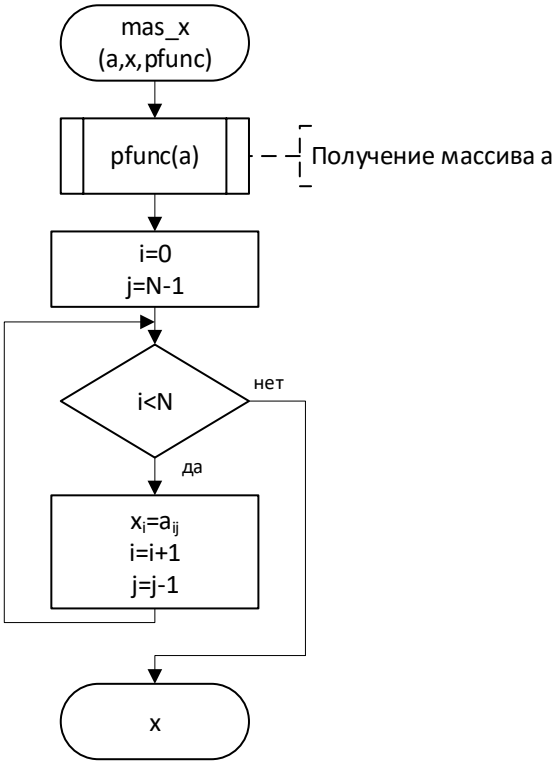


Рис. 17. Блок-схема алгоритма вычисления элементов массива x.

Рекурсивная обработка последовательности значений

Пример оформления рекурсивного алгоритма. Задание: вводится последовательность из N вещественных чисел, вычислить сумму элементов, меньших среднего значения всей последовательности (Рис. 18, Рис. 19).

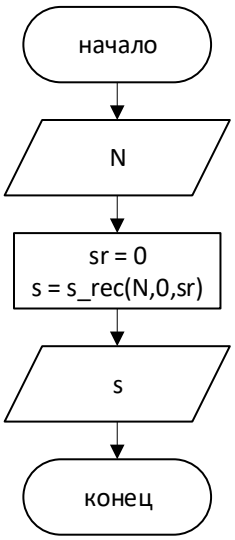


Рис. 18. Блок-схема основного алгоритма

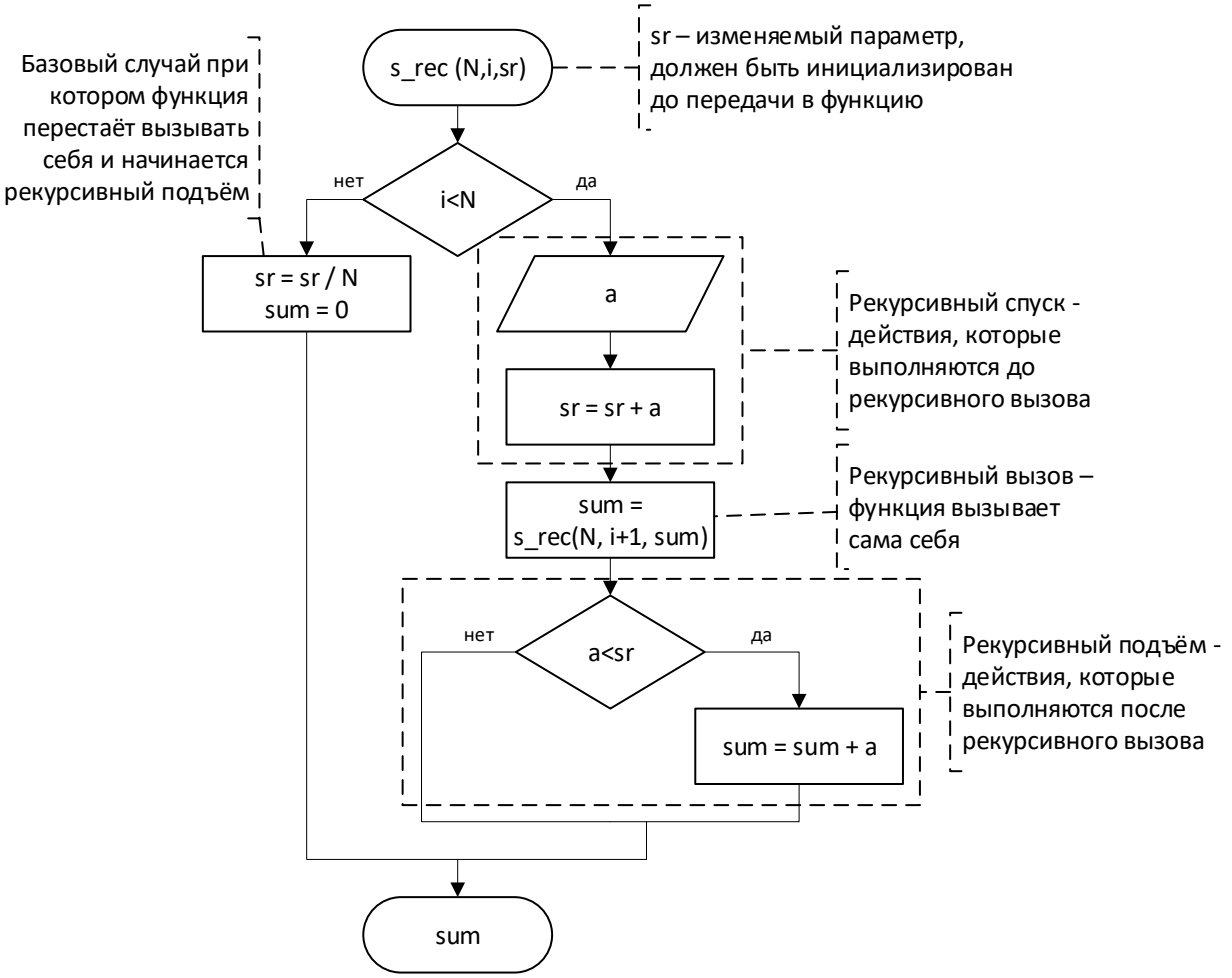


Рис. 19. Блок-схема рекурсивного алгоритма

Рекурсивная обработка одномерного массива

Пример использования рекурсии для обработки массива целочисленных значений.

Задание: в массиве из 10 целых чисел заменить все элементы, стоящие между минимальным и максимальным элементом на 0 (Рис. 20, Рис. 21).

Пример входных данных:

3 5 -2 **-3** -1 6 8 **10** 9 -2

Результат:

3 5 -2 **-3** 0 0 0 **10** 9 -2

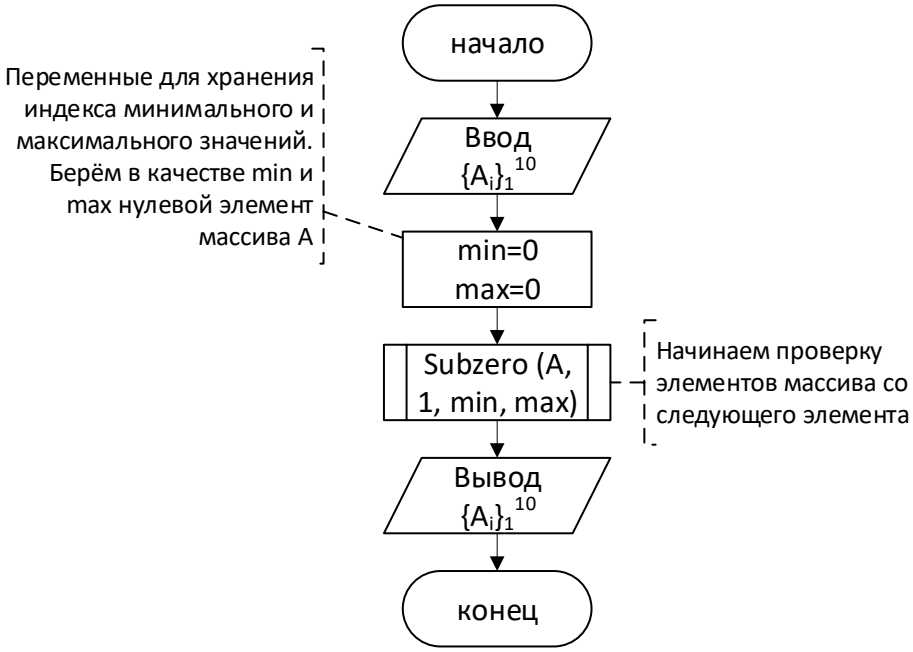


Рис. 20. Блок-схема основного алгоритма

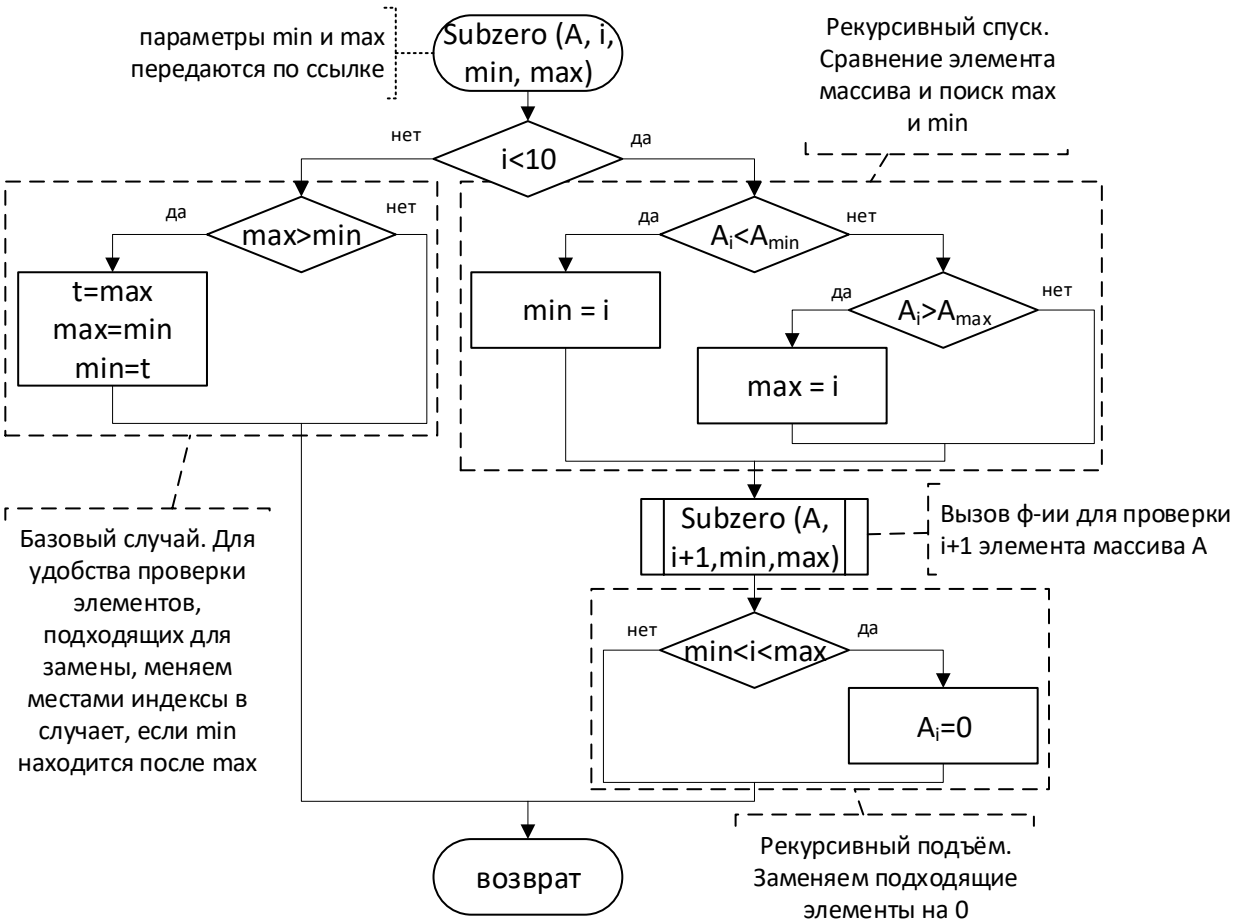


Рис. 21. Блок-схема рекурсивного алгоритма