ЛЕКЦИЯ № 2. Алгоритмы циклической структуры.

Цель лекции: *Знакомство с понятием алгоритма циклической структуры. Приобретение навыков построения алгоритмов циклической структуры.*

5. Алгоритмы циклической структуры

Циклом называют повторение одних и тех же действий (шагов). Последовательность действий, которые повторяются в цикле, называют *телом цикла*. Существует несколько типов *алгоритмов циклической структуры*. На рис. 5.1 изображен цикл с предусловием, а на рис. 5.2 – цикл с постусловием, которые называют *условными циклическими алгоритмами*. Нетрудно заметить, что эти циклы взаимозаменяемы и обладают некоторыми отличиями.

- в цикле с предусловием условие проверяется до тела цикла, в цикле с постусловием – после тела цикла;
- в цикле с постусловием тело цикла выполняется хотя бы один раз, в цикле с предусловием тело цикла может не выполниться ни разу;
- в цикле с предусловием проверяется условие продолжения цикла, в цикле с постусловием – условие выхода из цикла.

Рис. 5.1. Алгоритм циклической структуры с предусловием

Рис. 5.2. Алгоритм циклической структуры с постусловием

При написании условных циклических алгоритмов следует помнить следующее. Во-первых, чтобы цикл имел шанс когда-нибудь закончиться, содержимое его тела должно обязательно влиять на условие цикла. Во-вторых, условие должно состоять из корректных выражений и значений, определенных еще до первого выполнения тела цикла.

Кроме того, существует так называемый безусловный циклический алгоритм (рис. 5.3), который удобно использовать, если известно, сколько раз необходимо выполнить тело цикла.

Выполнение безусловного циклического алгоритма начинается с присвоения переменной **i** стартового значения **in**. Затем следует проверка, не превосходит ли переменная **i** конечное значение **iк**. Если превосходит, то цикл считается завершенным, и управление передается следующему за телом цикла оператору. В противном случае выполняется тело цикла, и переменная **i** меняет свое значение в соответствии с указанным шагом **di**. Далее, снова производится проверка значения переменной **i** и алгоритм повторяется. Понятно, что безусловный

циклический алгоритм можно заменить любым условным. Например, так как показано на рис. 5.4.

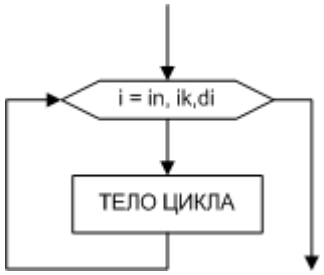

Рис. 5.3. Алгоритм циклической структуры без условия

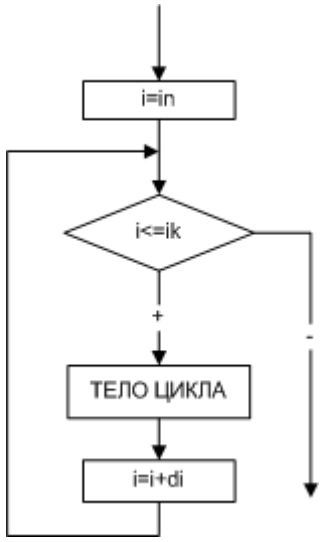

Рис. 5.4. Условный циклический алгоритм с известным числом повторений

Отметим, что переменную **i** называют *параметром цикла*, так как это переменная, которая изменяется внутри цикла по определенному закону и влияет на его окончание.

Рассмотрим использование алгоритмов циклической структуры на конкретных примерах.

ПРИМЕР 5.1. Найти наибольший общий делитель (НОД) двух натуральных чисел A и B.

Входные данные**:** `A` и `B`. Выходные данные**:** `A` – НОД**.**

Для решения поставленной задачи воспользуемся алгоритмом Евклида: будем уменьшать каждый раз большее из чисел на величину меньшего до тех пор, пока оба значения не станут равными, так, как показано в таблице 5.1.

Таблица 5.1. Поиск НОД для чисел A=25 и B=15.

Исходные данные	**Первый шаг**	**Второй шаг**	**Третий шаг**	**НОД(A,B)=5**
A=25	A=10	A=10	A=5	
B=15	B=15	B=5	B=5	

В блок–схеме решения задачи, представленной на рис. 5.5, для решения поставленной задачи используется цикл с предусловием, то есть тело цикла повторяется до тех пор, пока A не равно B.

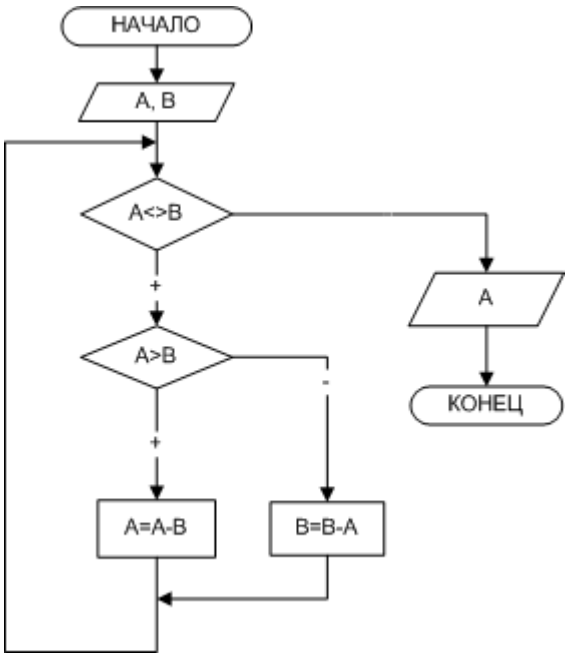

Рис. 5.5. Поиск наибольшего общего делителя двух чисел

ПРИМЕР 5.2. Вводится последовательность чисел, 0 – конец последовательности. Определить, содержит ли последовательность хотя бы два равных соседних числа.

Входные данные**:** `X0` – текущий член последовательности, `X1` – следующий член последовательности.

Выходные данные**:** сообщение о наличии в последовательности двух равных соседних элементов**.**

Вспомогательные переменные: `Fl` – логическая переменная, сохраняет значение «истина», если в последовательности есть равные рядом стоящие члены и «ложь» - иначе.

Блок–схема решения задачи приведена на рис. 5.6. Применение здесь цикла с постусловием обосновано тем, что необходимо вначале сравнить два элемента последовательности, а затем принять решение об окончании цикла.

В приведенных примерах условие задачи таково, что неизвестно, сколько раз повторится тело цикла. Такие циклы называют *циклами с неизвестным числом* повторений (вообще говоря, неизвестно, закончится ли цикл вообще). Цикл, количество повторений которого известно заранее или его можно определить по исходным данным, называют *циклом с известным числом повторений*.

Рассмотрим несколько примеров с использованием таких циклов.

ПРИМЕР 5.3. Составить таблицу значений функции $y = e^{\sin(x)}\cos(x)$ на отрезке $[0;\pi]$ с шагом 0.1.

Входные данные**:** начальное значение аргумента – 0, конечное значение аргумента – π, шаг изменения аргумента – 0.1.

Выходные данные**:** множество значений аргумента `X` и соответствующее им множество значений функции `Y`**.**

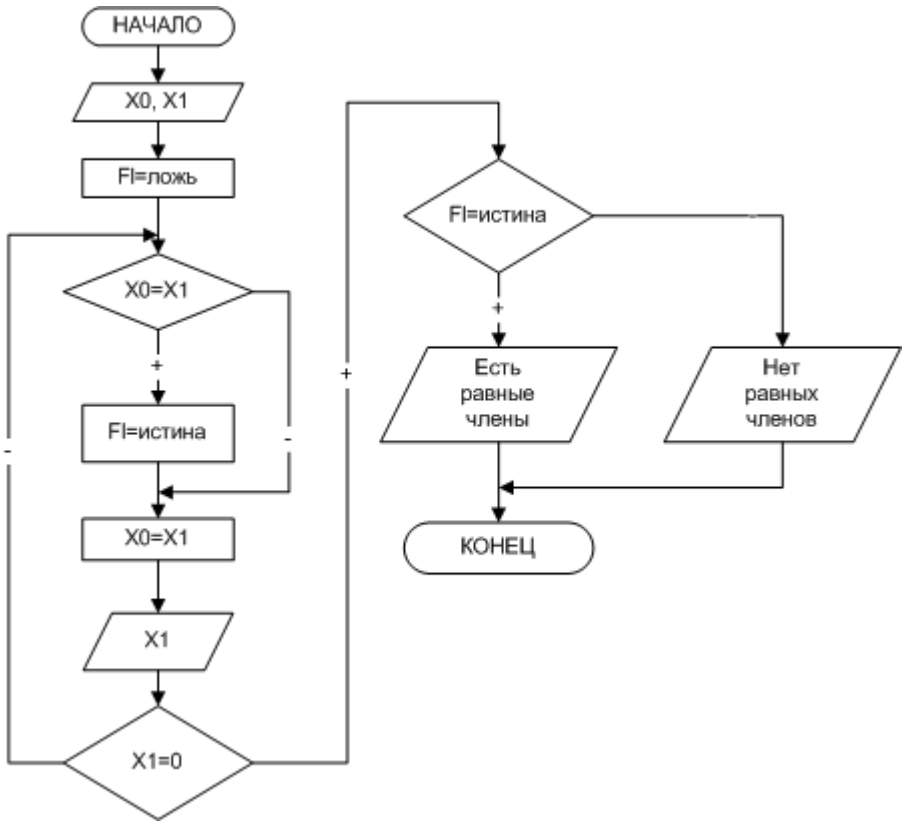

Рис. 5.6. Поиск равных соседних элементов последовательности

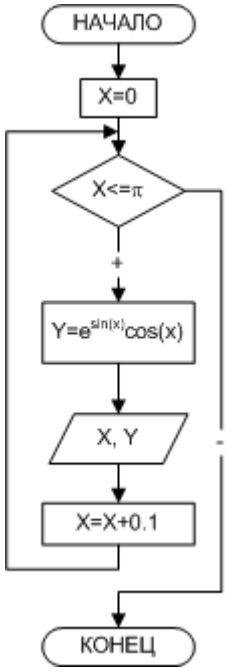

Рис. 5.7. Создание таблицы значений функции $y = e^{sin(x)}cos(x)$(1–й способ)

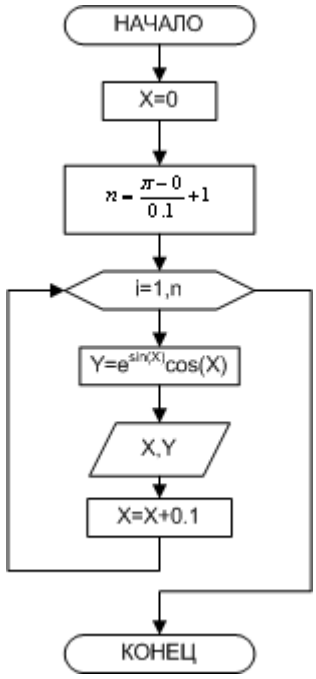

Рис. 5.8. Создание таблицы значений функции $y = e^{sin(x)}cos(x)$(2–й способ)

В условии задачи количество повторений цикла явно не задано, поэтому решить ее можно, используя цикл с предусловием (рис. 5.7). С другой стороны известно, как изменяется параметр цикла `X` и каковы его начальное и конечное значения, следовательно, предварительно определив количество повторений тела цикла *n*, так как показано на рис. 5.8, можно воспользоваться безусловным циклическим оператором. Итак, если параметр цикла `x` принимает значения в

диапазоне от xn до xk, изменяясь с шагом dx, то количество повторений тела цикла можно определить по формуле:

$$n = \frac{xk - xn}{dx} + 1 , \qquad (5.1)$$

округлив результат деления до целого числа.

ПРИМЕР 5.4. Вычислить факториал числа N (N!=1·2·3 …·*N*).

Входные данные: N– целое число, факториал которого необходимо вычислить.

Выходные данные: factorial– значение факториала числа N, произведение чисел от 1 до N, целое число.

Промежуточные данные: i– целочисленная переменная, принимающая значения от 2 до N с шагом 1, параметр цикла. Блок-схема приведена на рис. 5.9.

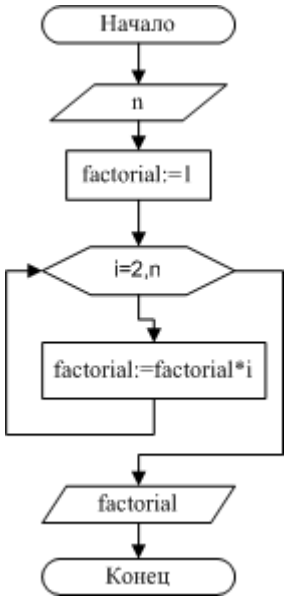

Рис. 5.9. Вычисление факториала

Итак, вводится число N. Переменной factorial**,** предназначенной для хранения значения произведения последовательности чисел, присваивается начальное значение, равное единице. Затем организуется цикл, параметром которого выступает переменная i. Если значение параметра цикла меньше или равно N, то выполняется оператор тела цикла, в котором из участка памяти с именем factorial считывается предыдущее значение произведения, умножается на текущее значение параметра цикла, а результат снова помещается в участок памяти с именем factorial. Когда параметр i становится больше N**,** цикл заканчивается, и на печать выводится значение переменой factorial**,** которая была вычислена в теле цикла.

ПРИМЕР 5.5. Вычислить a^n (n>0).

Входные данные: a – вещественное число, которое необходимо возвести в целую положительную степень n.

Выходные данные: p (вещественное число) **–** результат возведения вещественного числа a в целую положительную степень n.

Промежуточные данные: i– целочисленная переменная, принимающая значения от 1 до n с шагом 1, параметр цикла.

Блок-схема приведена на рис. 5.10.

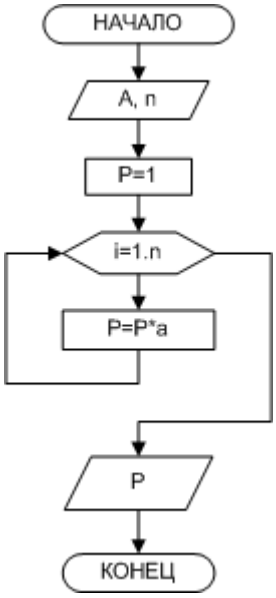

Рис. **5**.10. Возведение вещественного числа в целую степень

Известно, что для того, чтобы получить целую степень `n` числа `a`, нужно умножить его само на себя `n` раз. Результат этого умножения будет храниться в участке памяти с именем `p`. При выполнении очередного цикла из этого участка предыдущее значение будет считываться, умножаться на основание степени `a` и снова записываться в участок памяти `p`. Цикл выполняется `n` раз.

В таблице 5.2 отображен протокол выполнения алгоритма при возведении числа 2 в пятую степень: `a`=2, `n`=5. Подобные таблицы, заполненные вручную, используются для тестирования – проверки всех этапов работы программы.

Таблица 5.2. Процесс возведения числа `a` в степень `n`

i		1	2	3	4	5
P	1	2	4	8	16	32

ПРИМЕР 5.6. Вычислить сумму натуральных четных чисел, не превышающих N**.**

Входные данные**:** `N` – целое число.

Выходные данные**:** `S` **–** сумма четных чисел**.**

Промежуточные данные**:** `i` – переменная, принимающая значения от 2 до `N` с шагом 2, следовательно, также имеет целочисленное значение.

При сложении нескольких чисел необходимо накапливать результат в определенном участке памяти, каждый раз считывая из этого участка предыдущее значение суммы и прибавляя к нему следующее слагаемое. Для выполнения первого оператора накапливания суммы из участка памяти необходимо взять такое число, которое не влияло бы на результат сложения. Т.е. перед началом цикла переменной, предназначенной для накапливания сумы, необходимо присвоить значение нуль. Блок-схема решения этой задачи представлена на рис. 5.11.

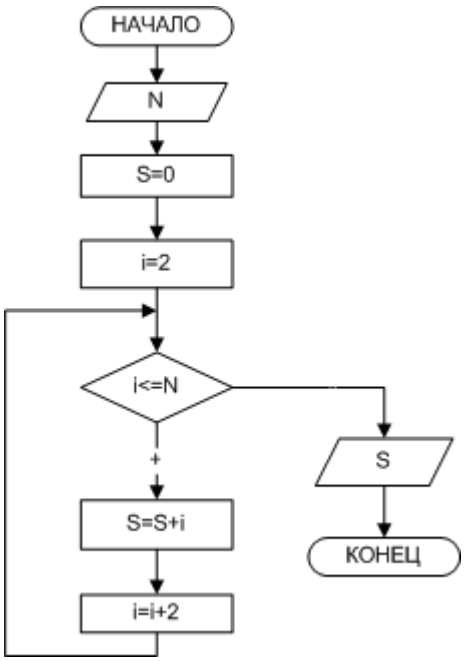

Рис. 5.11. Вычисление суммы четных, натуральных чисел

В таблице 5.3 приведены результаты тестирования алгоритма для `n`=7. Несложно заметить, что при нечетных значениях параметра цикла значение переменной, предназначенной для накапливания суммы, не изменяется.

Таблица 5.3. Суммирование четных чисел

i		1	2	3	4	5	6	7
S	0	0	2	2	6	6	12	12

ПРИМЕР 5.7. Дано натуральное число *N*. Определить *K* – количество делителей этого числа, не превышающих его (*N*=12, его делители 1, 2, 3, 4, 6, *K*=5)**.**

Входные данные**:** `N` – целое число.

Выходные данные**:** целое число `K` **–** количество делителей `N`**.**

Промежуточные данные**:** `i` – параметр цикла, возможные делители числа `N`.

В блок-схеме, изображенной на рис. 5.12, реализован следующий алгоритм: в переменную `K`, предназначенную для подсчета количества делителей заданного числа, помещается значение, которое не влияло бы на результат, т.е. нуль. Далее организовывается цикл, в котором изменяющийся параметр `i` выполняет роль возможных делителей числа `N`. Если заданное число делится нацело на параметр цикла, это означает, что `i` является делителем `N`; и значение переменной `K` следует увеличить на единицу. Цикл необходимо повторить `N/2` раз.

В таблице 5.4 отображены результаты тестирования алгоритма при определении делителей числа `N=12`.

Таблица 5.4. Определение делителей числа `N`

i		1	2	3	4	5	6
K	0	1	2	3	4	4	5

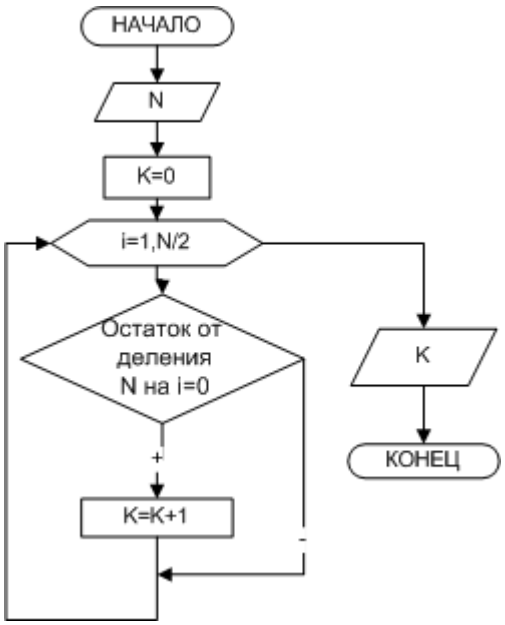

Рис. 5.12. Определение количества делителей натурального числа

ПРИМЕР 5.8. Дано натуральное число *N*. Определить, является ли оно простым. Натуральное число N называется *простым*, если оно делится нацело без остатка только на единицу и N. Число 13 – простое, так как делится только на 1 и 13, *N*=12 не является простым, так как делится на 1, 2, 3, 4, 6 и 12**.**

Входные данные**:** `N` – целое число. Выходные данные**:** сообщение**.**

Промежуточные данные**:** `i` – параметр цикла, возможные делители числа `N`.

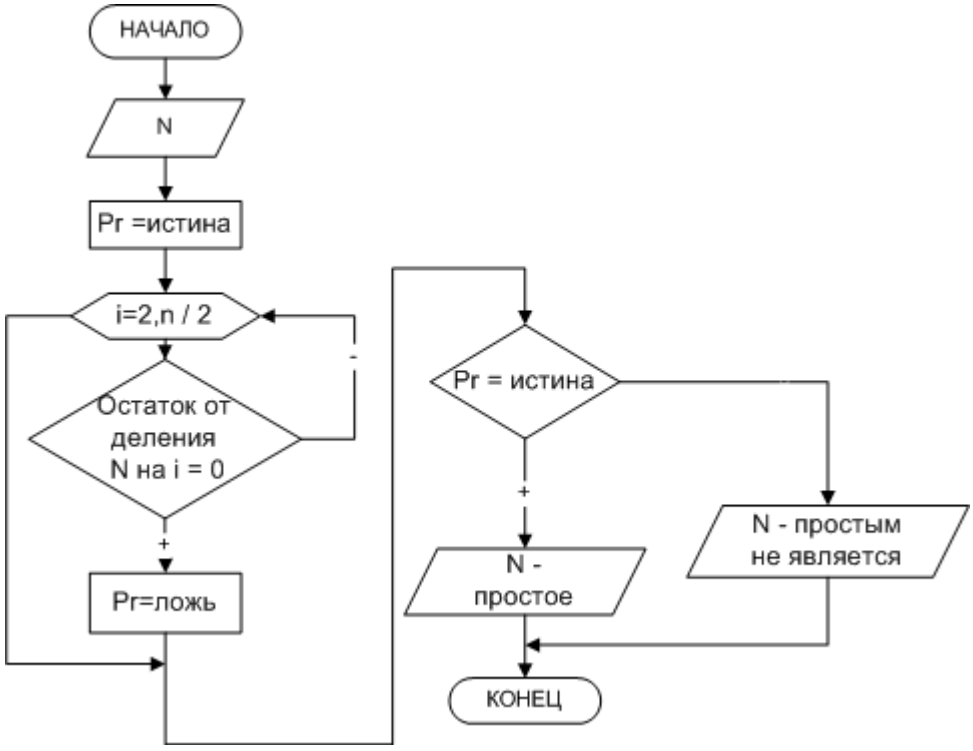

Рис. 5.13. Определение простого числа

Алгоритм решения этой задачи (рис. 5.13) заключается в том, что число `N` делится на параметр цикла `i`, изменяющийся в диапазоне от 2 до `N/2`. Если среди значений параметра не найдется ни одного числа, делящего заданное число нацело, то `N` – простое число, иначе оно таковым не является. Обратите внимание на то, что в алгоритме предусмотрено два выхода из цикла. Первый –

естественный, при исчерпании всех значений параметра, а второй - досрочный. Нет смысла продолжать цикл, если будет найден хотя бы один делитель из указанной области изменения параметра.

ПРИМЕР 5.9. Определить количество простых чисел в интервале от *N* до *M*, где *N* и *M* – натуральные числа.

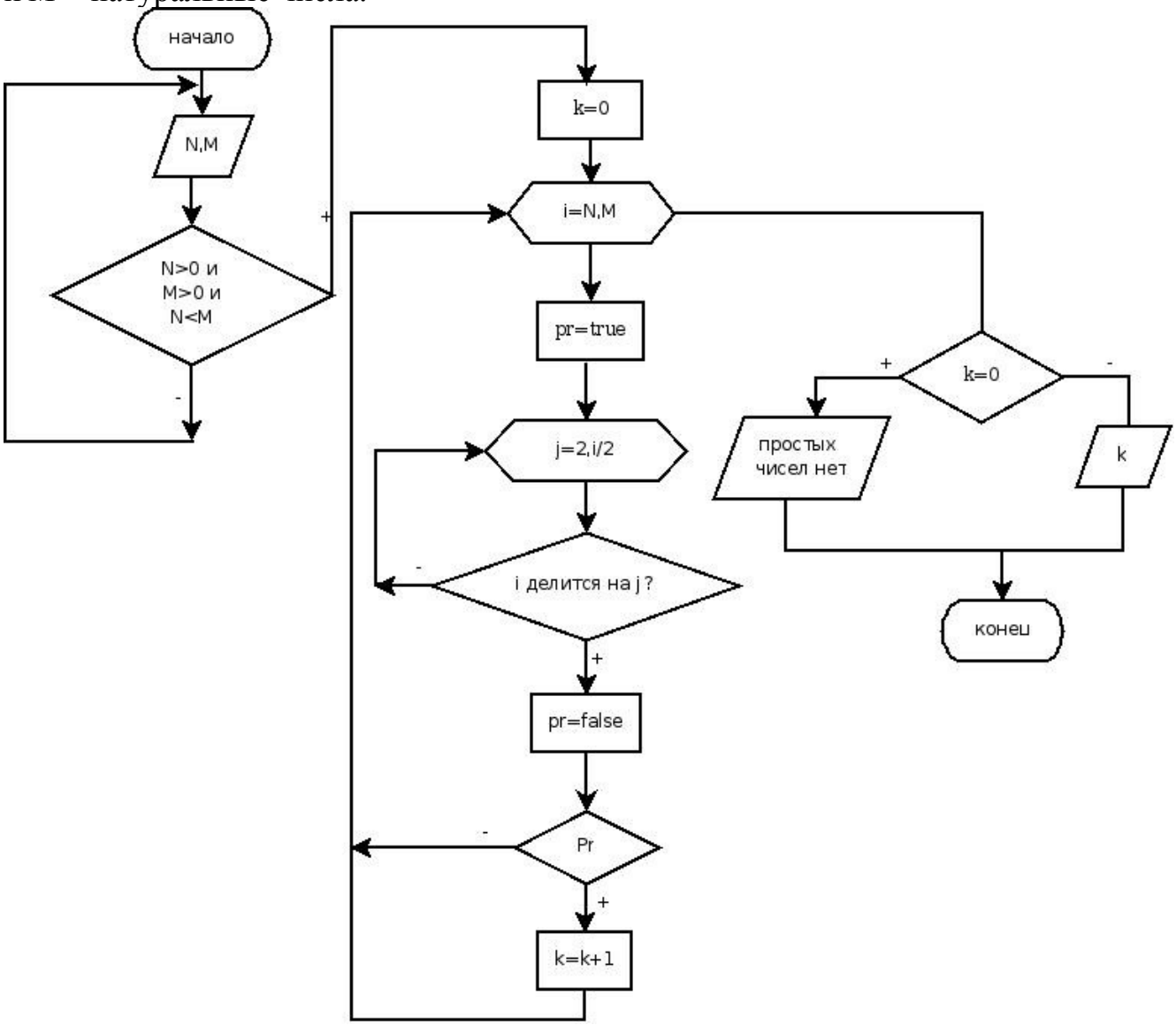

Рис. 5.14 Поиск простых чисел в заданном интервале

ПРИМЕР 5.10. Вычислить значения `y`, соответствующие каждому значению `x` (`xn≤x≤xk, dx`) по формуле:

$$y = \begin{matrix} e^{-x}\sin(x), |x| \le a \\ e^{-x^2}\cos(x), |x| > a \end{matrix}$$

Найти максимальное и минимальное значение `y`.

Входные данные: `xn`, `xk`, `dx`, `a`.

Выходные данные*:* множество значений `y`, `ymax` — максимальное значение `y`, `ymin` — минимальное значение `y`.

Алгоритм поиска минимального (максимального) значения переменной `y` можно описать следующим образом. Пусть минимальное (максимальное) значение `y` хранится в переменной `ymin` (`ymax`). До начала цикла этой переменной присваивается очень большое (маленькое) значение, например, 10^{20} (-10^{20}). Затем в

цикле сравниваются значения y и ymin (ymax), и если встречается значение **y** меньшее (большее), чем хранится в ymin (ymax), то его необходимо записать в переменную ymin (ymax). Таким образом, все значения **y** неявно сравниваются между собой, а по окончании цикла в переменной ymin (ymax) будет храниться наименьшее (наибольшее) среди значений y.

Блок-схема алгоритма изображена на рис. 5.15.

ПРИМЕР 5.11**.** Вычислить значения z, которые соответствуют каждому значению **x** (*x = 1*; *dx = 0.5*) по формуле: $z=lnx\sqrt{(\frac{x}{x^3+1})}$. Считать z до тех пор, пока подкоренное выражение больше или равно 0.02. Определить k — количество вычисленных z.

Входные данные: x, dx.

Выходные данные: множество значений z , k — количество вычисленных z.

Здесь условие окончания цикла явно указано в условии примера. Цикл окончится, когда подкоренное выражение станет меньше 0.02. Блок-схема алгоритма изображена на рис. 5.16.

ПРИМЕР 5.12**.** Вычислить отрицательный корень уравнения $x^3-x+0.5=0$, используя рекуррентную формулу: $x_k+1=\sqrt[3]{x_k-0.5}$ *(k=0,1,2,...). $x_0=-1.3$; $\varepsilon=10^{-4}$.* Определить количество итераций (шагов цикла); вычисления прекратить при выполнении условия $|x_{k+1}-x_k|<\varepsilon$.

Входные данные: x_0, E (точность ε).

Выходные данные: k — количество итераций; x — корень.

Промежуточные данные: x_1 — последующее значение **x**.

В данном примере вычисляется множество значений $x_1,x_2,x_3,...x_к$. Сколько их заранее неизвестно, но конечное значение переменной **x** и является корнем уравнения. В каждом цикле необходимо сравнивать текущее и предыдущее значение. Поэтому для расчета можно использовать всего две переменные: x0 – предыдущее значение и x1 – текущее значение. Далее необходимо организовать цикл, который окончится, если модуль разности текущего и предыдущего значений станет меньше заданной точности. В теле цикла на каждом новом шаге необходимо в переменную x0 записывать текущее значение и пересчитывать x1 по формуле. Блок-схема алгоритма изображена на рис. 5.17.

ПРИМЕР 5.13. Вычислить значения функции $y=chx=\sum_{k=1}^{\infty}U_k$, используя рекуррентную формулу $U_{k+1}=\frac{x^2}{(2k-1)2k}U_k$, (*k = 1,2,3,...).* Вычисления прекратить при выполнении условия $|U_{k+1}|<\varepsilon$. Определить количество итераций. Начальные значения параметров равны $U_1 = 1$; *x = 5.5*; точность ε принять равной 10^{-4}.

Входные данные: U – начальное приближение, x, E – точность (ε).

Выходные данные: y — значение вычисленной функции, k — количество итераций. Блок-схема алгоритма изображена на рис. 5.18.

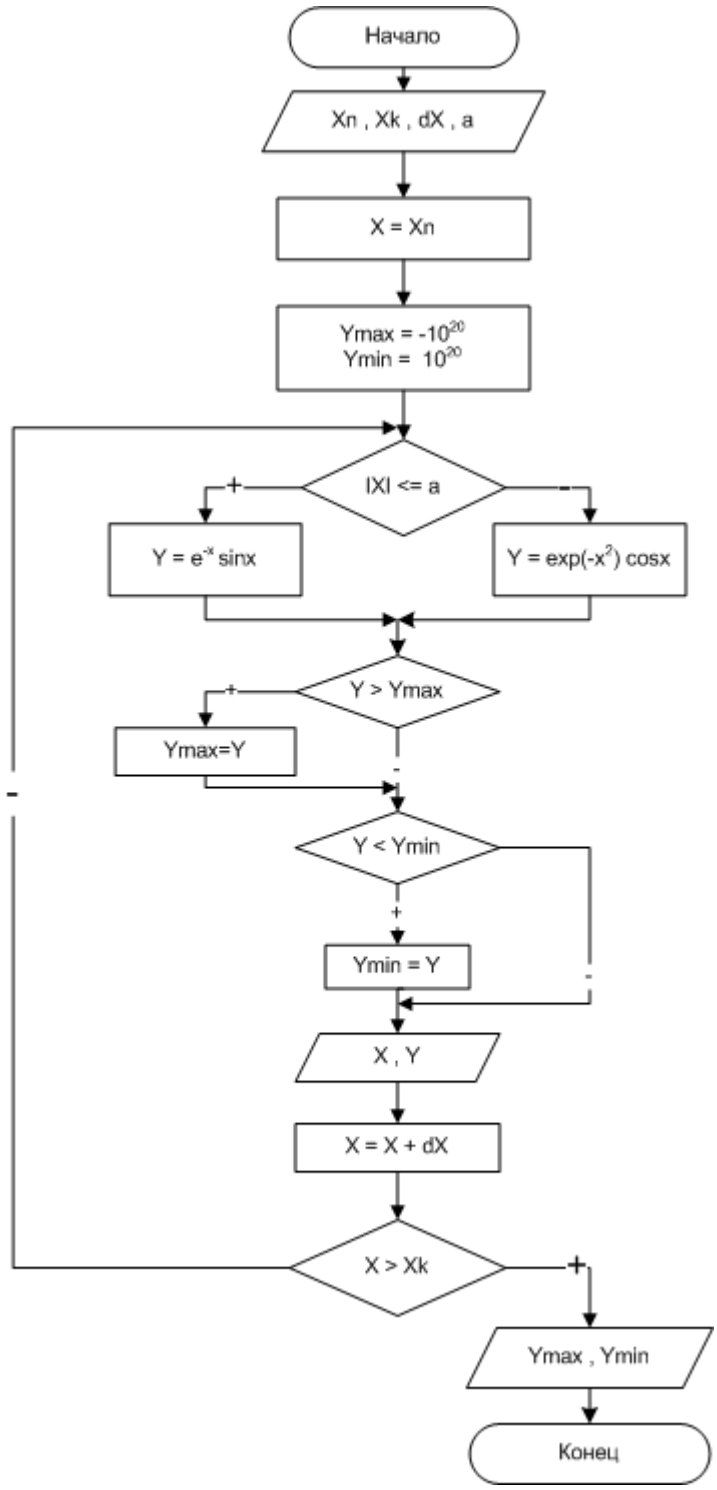

Рис. 5.15. Блок-схема алгоритма примера 5.10

В отличие от предыдущего примера, здесь достаточно одной переменной `U`, в которой будет храниться текущее значение `y`. На каждом шаге циклического алгоритма необходимо пересчитывать значение `y` и к нему добавлять `U`. Цикл окончится, когда абсолютное значение `U` станет меньше `E`. По окончании цикла выводится `y` – сумма значений `U`.

ПРИМЕР 5.14. Вводится последовательность целых чисел, 0 – конец последовательности. Найти минимальное среди положительных, если таких значений несколько, определить, сколько их.

Блок-схема решения задачи приведена на рис. 5. 19.

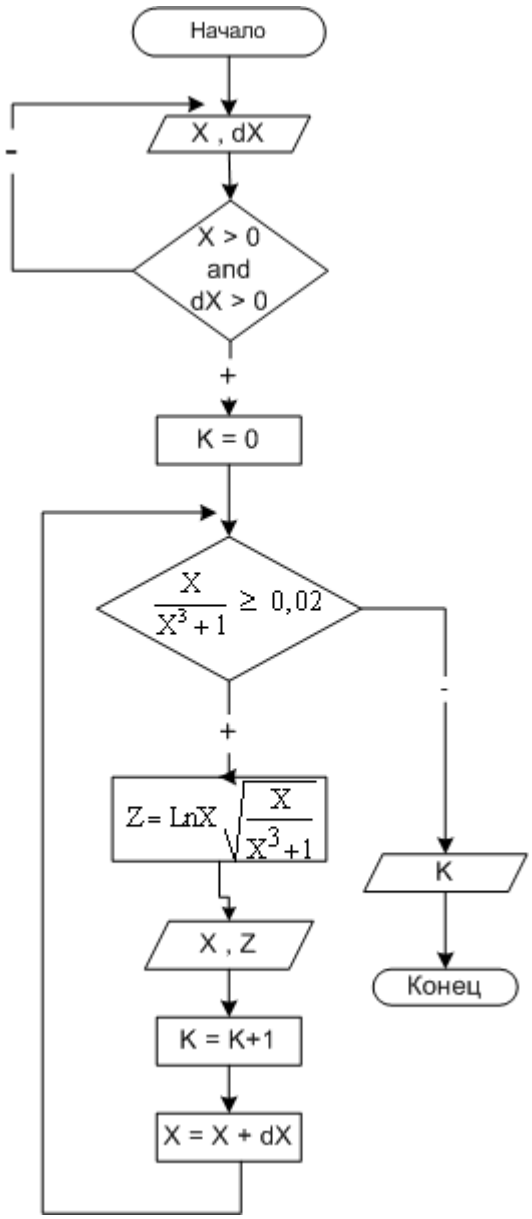

Рис. 5.16. Блок-схеме алгоритма примера 5.11

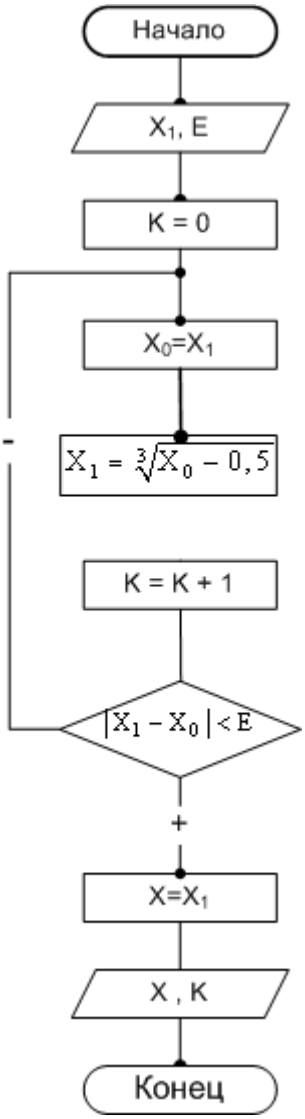

Рис. 5.17. Блок-схема алгоритма примера 5.12

При решении этой задачи используется логическая переменная `Pr`, которая определяет наличие положительных чисел в последовательности (`Pr=false`, если положительных чисел нет). Основной цикл работает до тех пор, пока `N≠0`. При каждом входе в цикл проверяем, знак `N`. Если `N>0`, а `pr=false`, то это говорит о том, что это первое положительное число. В этом случае это число объявляем минимальным, количество минимальных чисел (`k`) равным 1, а признак `pr=true`. Если `N>0`, а `pr` не равно `false`, то сравниваем текущее значение N с min. Если `N<min`, это `N` объявляем минимальным, количество минимальных чисел (`k`) равным 1, если `N=min`, то количество минимальных значений увеличиваем на 1 (`k=k+1`). Последним действием в цикле является ввод очередного значения `N`.

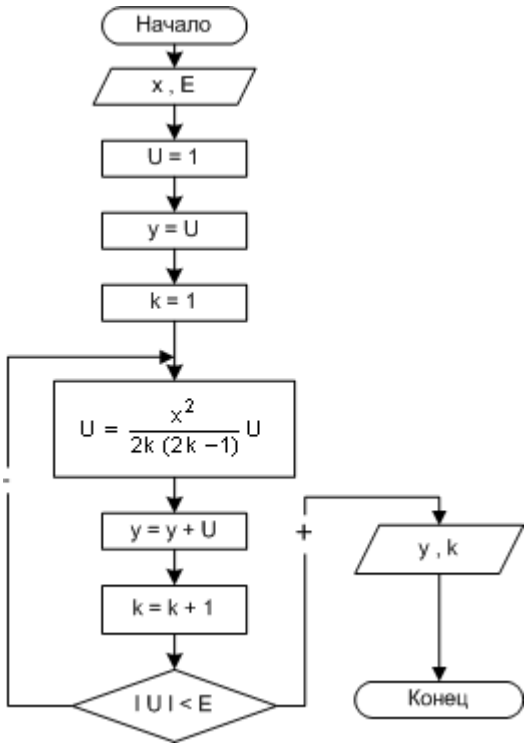

Рис. 5.18. Блок-схема алгоритма примера 5.13

ПРИМЕР 5.15. Дано натуральное число `N`. Определить самую большую цифру и ее позицию в числе (`N`=573863, наибольшей является цифра 8, ее позиция – четвертая слева)**.**

Входные данные**:** `N` – целое число.

Выходные данные**:** `max` – значение наибольшей цифры в числе, `pos` – позиция этой цифры в числе**.**

Промежуточные данные**:** `i` – параметр цикла, `kol` – количество цифр в числе, `M` – переменная для временного хранения значения `N`**.**

Разобьем решение этой задачи на два этапа. Вначале найдем количество цифр в заданном числе (рис. 5.20), а затем определим наибольшую цифру и ее позицию (рис. 5.21).Для того, чтобы подсчитать количество цифр в числе, необходимо определить, сколько раз заданное число можно разделить на десять нацело. Например, пусть `N=12345`, тогда количество цифр `kol` = 5. Результаты вычислений сведены в таблицу 5.5.

Таблица 5.5. Определение количества цифр числа

kol	**N**
1	12345
2	12345 div10=1234
3	1234 div 10=123
4	123 div 10=12
5	12 div 10=1
	1 div 10=0

Процесс определения текущей цифры числа `N`=12345 представлен в таблице 5.6.

Таблица 5.6. Определение текущей цифры числа

i	**Число M**	**Цифра**

1	12345	12345 mod 10 = 5
2	12345 div10=1234	1234 mod 10 = 4
3	1234 div 10=123	123 mod 10 = 3
4	123 div 10=12	12 mod 10 = 2
5	12 div 10=1	1 mod 10 = 1

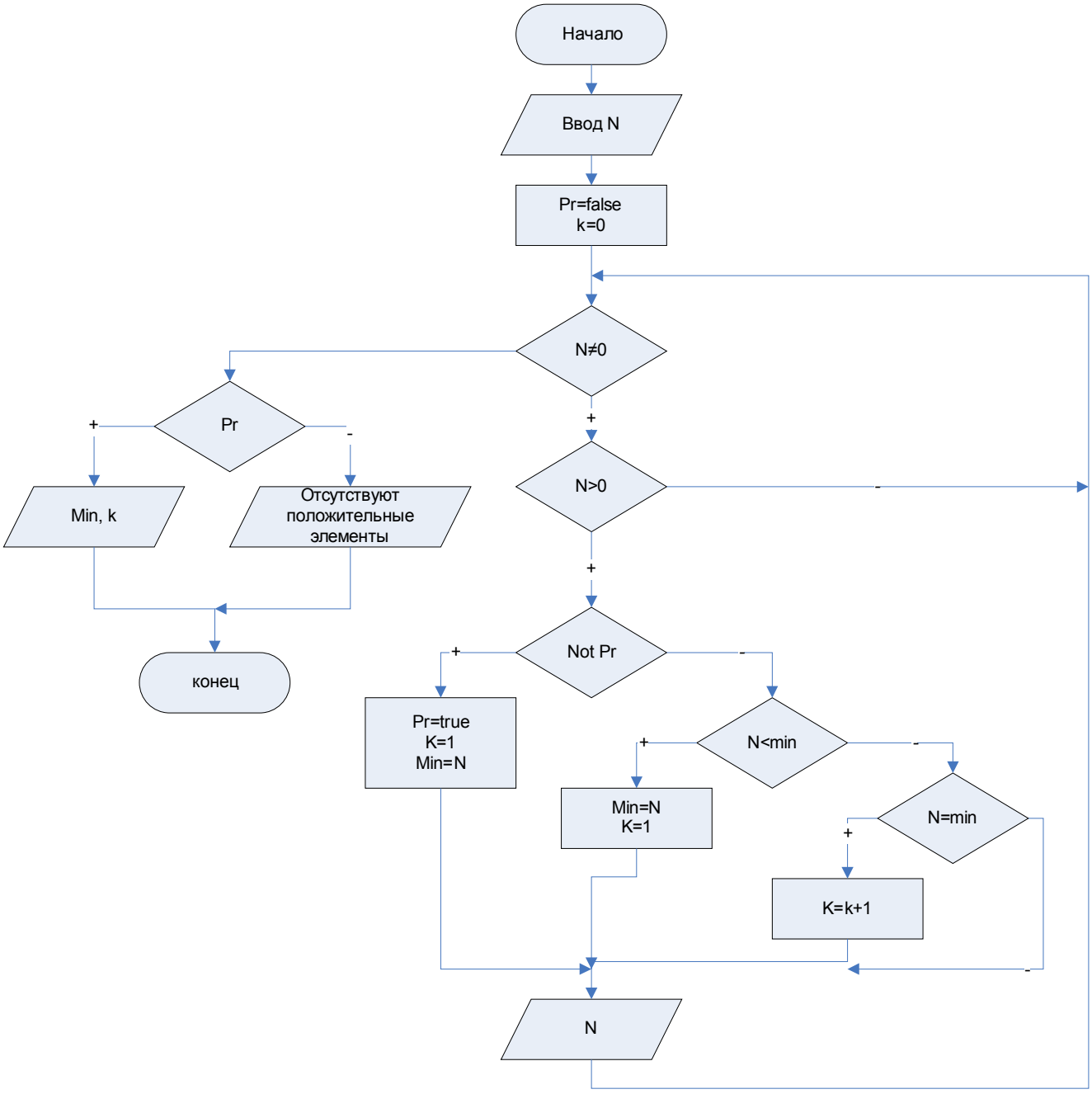

Рис. 5.19. Блок-схема нахождения минимального положительного числа последовательности

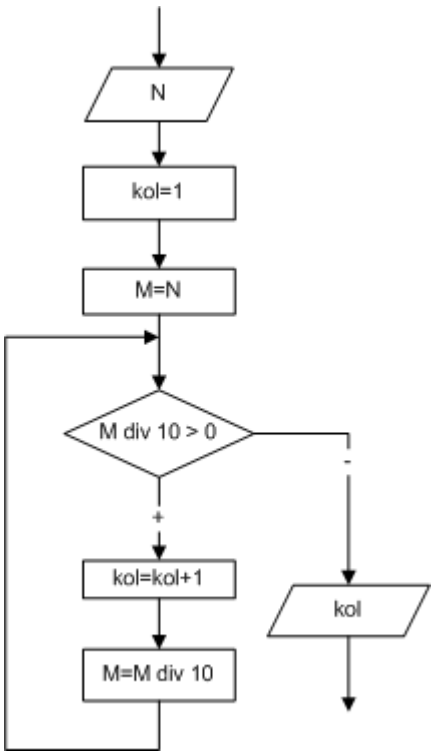

Рис. 5.20. Определение количества цифр в числе

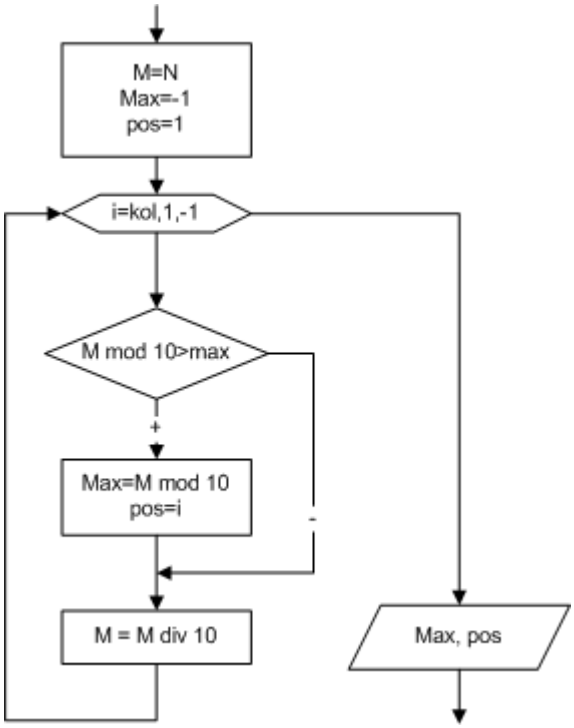

Рис. 5.21. Определение максимальной цифры в числе и ее позиции

Если цифры числа известны, определить наибольшую из них не составит труда. Алгоритм поиска максимального значения в некоторой последовательности цифр заключается в следующем. В ячейку, в которой будет храниться максимальный элемент (`max`), записывают значение, меньшее любого из элементов последовательности (в нашем случае `max`=-1, так как цифры числа находятся в диапазоне от 0 до 9). Затем сравнивают элементы последовательности со значением ячейки `max`, если найдется элемент, превышающий значение

предполагаемого максимума, то ячейке ~~max~~ необходимо присвоить значение этого элемента и, соответственно, запомнить его номер в последовательности (в нашем случае переменной `pos` присваивается значение параметра цикла ~~i~~).

В алгоритме поиска минимума вначале в переменную `min` записываем значение, заранее большее любого элемента последовательности. Затем все элементы последовательности сравниваем с `min`, если встретится значение меньшее, чем `min`, переписываем его в переменную `min`.