3 Операторы управления

В этой главе изложена методика составления алгоритмов с помощью блок-схем и описаны основные операторы языка: условный оператор `if`, оператор выбора `case`, операторы цикла `while..do`, `repeat..until`, `for..do`. Приводится большое количество примеров составления программ различной сложности.

3.1 Основные конструкции алгоритма

Как правило, созданию программы предшествует разработка алгоритма[36]. *Алгоритм* — это четкое описание последовательности действий, которые необходимо выполнить для того, чтобы при соответствующих исходных данных получить требуемый результат. Одним из способов представления алгоритма является *блок-схема*. При ее составлении все этапы решения задачи изображаются с помощью различных геометрических фигур. Эти фигуры называют блоками и, как правило, сопровождают надписями. Последовательность выполнения этапов указывают при помощи стрелок, соединяющих эти блоки.

Типичные этапы решения задачи изображаются следующими геометрическими фигурами:

- *блок начала* (*конца*) (рис. 3.1). Надпись внутри блока: «начало» («конец»);
- *блок ввода* (*вывода*) данных (рис. 3.2). Надпись внутри блока: ввод (вывод) и список вводимых (выводимых) переменных;
- *блок решения,* или *арифметический* (рис. 3.3). Внутри блока записывается действие, вычислительная операция или группа операций;
- *условный блок* (рис. 3.4). Логическое условие записывается внутри блока. В результате проверки условия осуществляется выбор одного из возможных путей (ветвей) вычислительного процесса.

Рассмотренные блоки позволяют описать три *основные конструкции алгоритма*: линейный, разветвляющийся и циклический процессы.

36 Алгоритм - от algorithmi, algorismus, первоначально латинская транслитерация имени математика аль-Хорезми.

Рисунок 3.1: Блок начала (конца) алгоритма

Рисунок 3.2: Блок ввода (вывода)

Рисунок 3.3: Арифметический блок

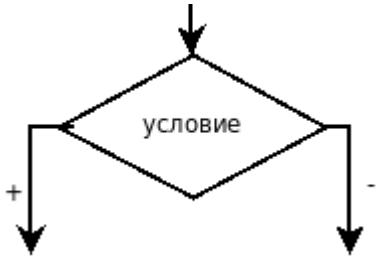

Рисунок 3.4: Условный блок

Линейный процесс — это конструкция, представляющая собой последовательное выполнение двух или более блоков (рис. 3.5).

Разветвляющийся процесс задает выполнение одного или другого оператора в зависимости от выполнения условия (рис. 3.6).

Циклический процесс задает многократное выполнение оператора или группы операторов (рис. 3.7).

Нетрудно заметить, что каждая из основных конструкций алгоритма имеет один вход и один выход. Это позволяет вкладывать конструкции друг в друга произвольным образом и составлять алгоритмы для решения задач любой сложности.

Рассмотрим операторы языка программирования Free Pascal, реализующие основные конструкции алгоритма.

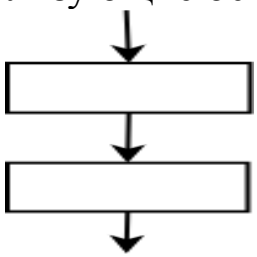

Рисунок 3.5: Линейный процесс

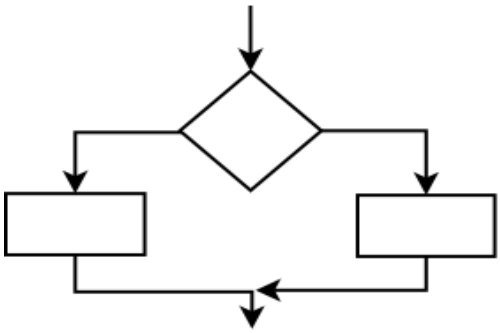

Рисунок 3.6: Разветвляющийся процесс

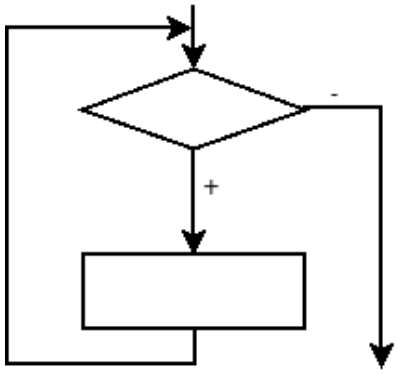

Рисунок 3.7: Циклический процесс

3.2 Оператор присваивания

Оператор присваивания в языке Free Pascal состоит из двух символов: двоеточия и знака равенства. Символы `:=` всегда пишут слитно. Пробелы допускаются перед символом двоеточия и после символа равенства.

В общем случае оператор присваивания имеет вид:

```
имя_переменной := значение;
```

где значение – это выражение, переменная, константа или функция. Выполняется оператор так. Сначала вычисляется значение выражения, указанного в правой части оператора, а затем его результат записывается в область памяти (переменную), имя которой указано слева. Например, запись a:=b означает, что переменной a присваивается значение b.

Типы переменных a и b должны совпадать или быть совместимыми для присваивания, то есть тип, к которому принадлежит переменная b, должен находиться в границах типа переменной a.

3.3 Составной оператор

Составной оператор – группа операторов языка Free Pascal, отделенных друг от друга точкой с запятой, начинающихся со служебного слова begin и заканчивающихся служебным словом end:

```
begin
оператор_1;
...
оператор_n
end;
```

Транслятор воспринимает составной оператор как один оператор.

3.4 Условные операторы

В языке Free Pascal одна из основных конструкций алгоритма, *разветвляющийся процесс,* реализована двумя условными операторами: if и case. Рассмотрим каждый из них.

3.4.1 Условный оператор if…then…else

При решении большинства задач порядок вычислений зависит от определенных условий, например, от исходных данных или от промежуточных результатов, полученных на предыдущих шагах программы. Для организации вычислений в зависимости от какого-либо условия в языке Free Pascal используется *условный оператор* if…then…else, который в общем виде записывается так:

```
if условие then оператор_1 else оператор_2;
```

где if…then…else – зарезервированные слова, условие – выражение логического типа[37], оператор_1 и оператор_2 – любые операторы языка Free Pascal.

37 Логическое выражение может принимать одно из двух значений: истина или ложь.

Работа условного оператора организована следующим образом. Сначала вычисляется выражение, записанное в условии. Если оно имеет значение истина (True), то выполняется оператор_1. В противном случае, когда выражение имеет значение ложь (False), оператор_1 игнорируется и управление передается оператору_2. Алгоритм, который реализован в условном операторе if…then…else, представлен на рис. 3.8.

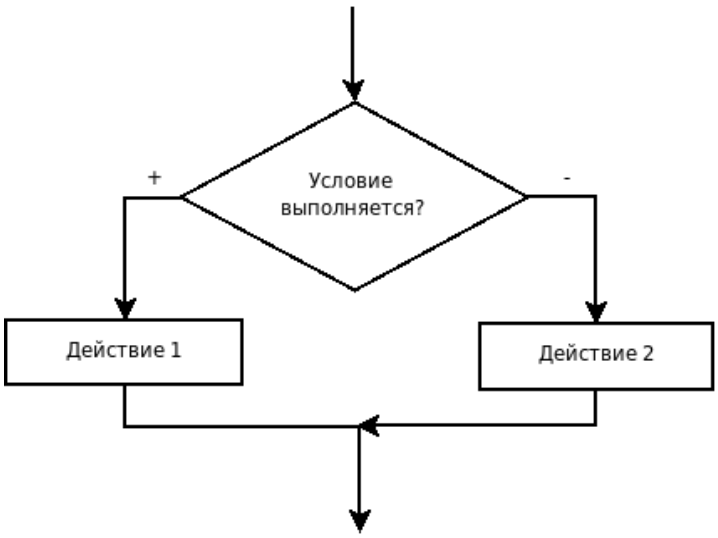

Рисунок 3.8: Алгоритм условного оператора if...then...else

Например, чтобы сравнить значения переменных x и y, нужно создать следующий программный код:

```
write('x='); readln(x);
write('y='); readln(y);
if x=y then
writeln('значение x равно значению y')
else
writeln('значение x не равно значению y');
```

Если в задаче требуется, чтобы в зависимости от значения условия выполнялся не один оператор, а несколько, их необходимо заключать в операторные скобки как составной оператор:

```
if условие then
 begin
     оператор_1;
     оператор_2;
     …
 end
```

```
else
 begin
     оператор_1;
     оператор_2;
     …
 end;
```

Альтернативная ветвь `else` в условном операторе может отсутствовать, если в ней нет необходимости:

```
if условие then оператор;
```

или

```
if условие then
 begin
     оператор_1;
     оператор_2;
     …
     оператор_n;
 end;
```

В таком «усеченном» виде условный оператор работает так: оператор (группа операторов) либо выполняется, либо пропускается, в зависимости от значения выражения, представляющего условие. Алгоритм этого условного процесса представлен на рис. 3.9.

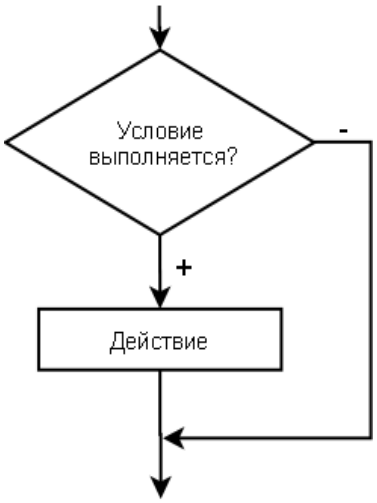

Рисунок 3.9: Алгоритм условного оператора if без альтернативной ветви else

Пример применения условного оператора, без альтернативной ветви `else` может быть таким:

```
write('x=');
readln(x);
write('y=');
readln(y);
z:=0;
{Значение с изменяется только при
условии,что x не равно y.}
if (x<>y) then    z:=x+y;
{Вывод на экран значения перемен-
ной z выполняется в любом случае.}
writeln('Значение z=',z);
```

Условные операторы могут быть вложены друг в друга. При вложениях условных операторов всегда действует правило: альтернатива `else` считается принадлежащей ближайшему `if`, имеющему ветвь `else`. Например, в записи

```
if условие_1 then
 if условие_2 then
     оператор_А
 else оператор_Б;
```

`оператор_Б` относится к `условию_2`, а в конструкции

```
if условие_1 then
 begin
     if условие_2 then
         оператор_А;
     end
 else оператор_Б;
```

он принадлежит оператору `if` с `условием_1`.

Для сравнения переменных в условных выражениях применяют *операции отношения*: =, <>, <, >, <=, >=. Условные выражения составляют с использованием логических операций `and`, `or` и `not`. В языке Free Pascal приоритет *операций отношения* меньше, чем у *логических операций*, поэтому составные части сложного логического выражения заключают в скобки.

Допустим, нужно проверить, принадлежит ли переменная `x` интервалу [`a`, `b`]. Условный оператор будет иметь вид:

```
if (x>=a) and (x<=b) then….
```

Запись

```
if x>=a and x<=b then…
```

не верна, так как фактически будет вычисляться значение выражения

```
x>= (a and x) <=b.
```

Рассмотрим использование оператора `if` на примерах[38].

ЗАДАЧА 3.1. Дано вещественное число `x`. Для функции, график которой приведен на рис. 3.10, вычислить $y=f(x)$.

38 В задачах этой главы мы не будем уделять много внимания интерфейсу создаваемых программ, чтобы у читателя была возможность разобраться в алгоритмах и способах их записи на языке Free Pascal.

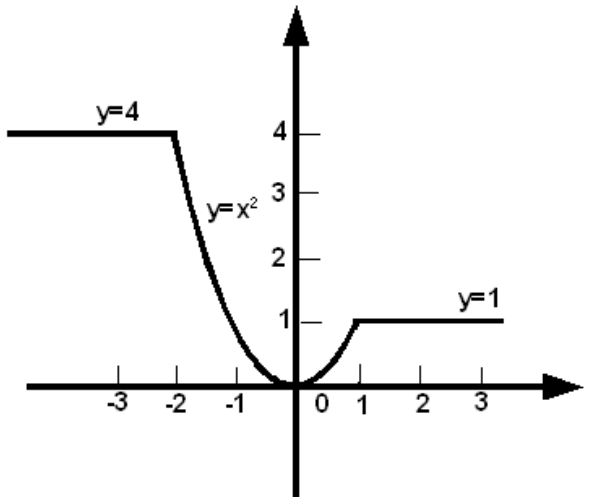

Рисунок 3.10: Графическое представление задачи 3.1

Аналитически функцию, представленную на рис. 3.10, можно записать так:

$$y(x)=\begin{cases} 4, & x\leqslant -2 \\ 1, & x\geqslant 1 \\ x^2, & -2<x<1 \end{cases}$$

Составим словесный алгоритм решения этой задачи:

1. Начало алгоритма.
2. Ввод числа `x` (аргумент функции).
3. Если значение `x` меньше либо равно `-2`, то переход к п. 4, иначе переход к п. 5.
4. Вычисление значения функции: `y=4`, переход к п. 8.
5. Если значение `x` больше либо равно `1`, то переход к п. 6, иначе переход к п. 7.
6. Вычисление значения функции: `y=1`, переход к п. 8.
7. Вычисление значения функции: $y=x^2$.
8. Вывод значений аргумента `x` и функции `y`.
9. Конец алгоритма.

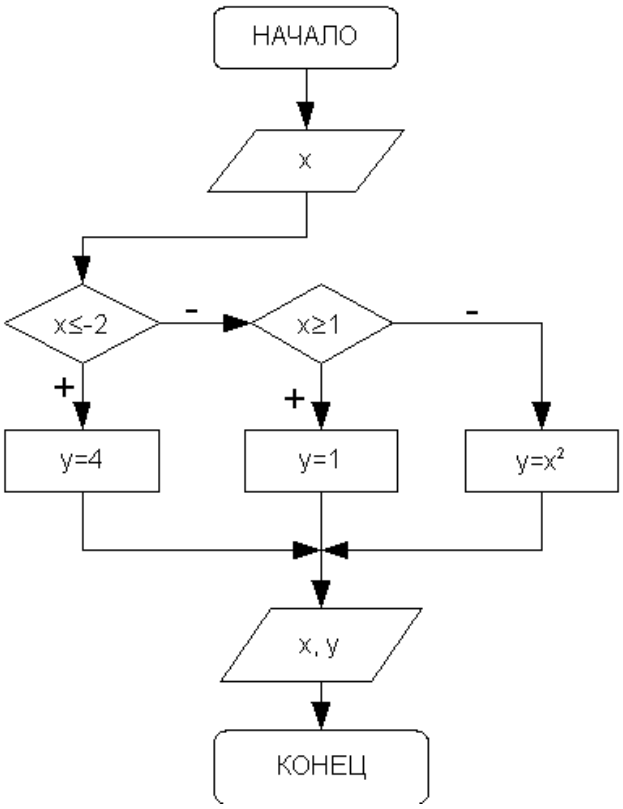

Рисунок 3.11: Блок-схема алгоритма решения задачи 3.1

Блок-схема, соответствующая описанному алгоритму, представлена на рис. 3.11.

Текст программы на языке Free Pascal будет иметь вид:

```
var x,y: real; begin
 write('x='); readln(x);
 if x<=-2 then y:=4 else if x>=1 then y:=1
 else y:=sqr(x);
writeln('x=',x:5:2,'    y=',y:5:2);end.
```

ЗАДАЧА 3.2. Даны вещественные числа x и y. Определить принадлежит ли точка с координатами (x; y) заштрихованной части плоскости (рис. 3.12).

Как показано на рис. 3.12, плоскость ограничена линиями x=-1, x=3, y=-2 и y=4. Значит точка с координатами (x; y) будет принадлежать этой плоскости, если будут выполняться следующие условия: $x \geq -1$, $x \leq 3$, $y \geq -2$ и $y \leq 4$. Иначе точка лежит за пределами плоскости.

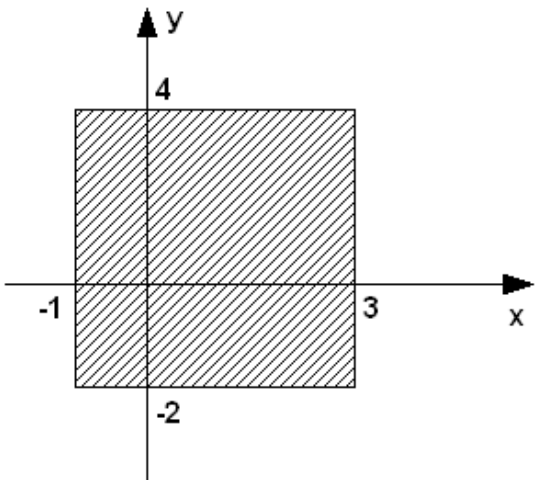

Рисунок 3.12: Графическое представление задачи 3.2

Блок-схема, описывающая алгоритм решения данной задачи, представлена на рис. 3.13. Текст программы к задаче 3.2 приведен далее.

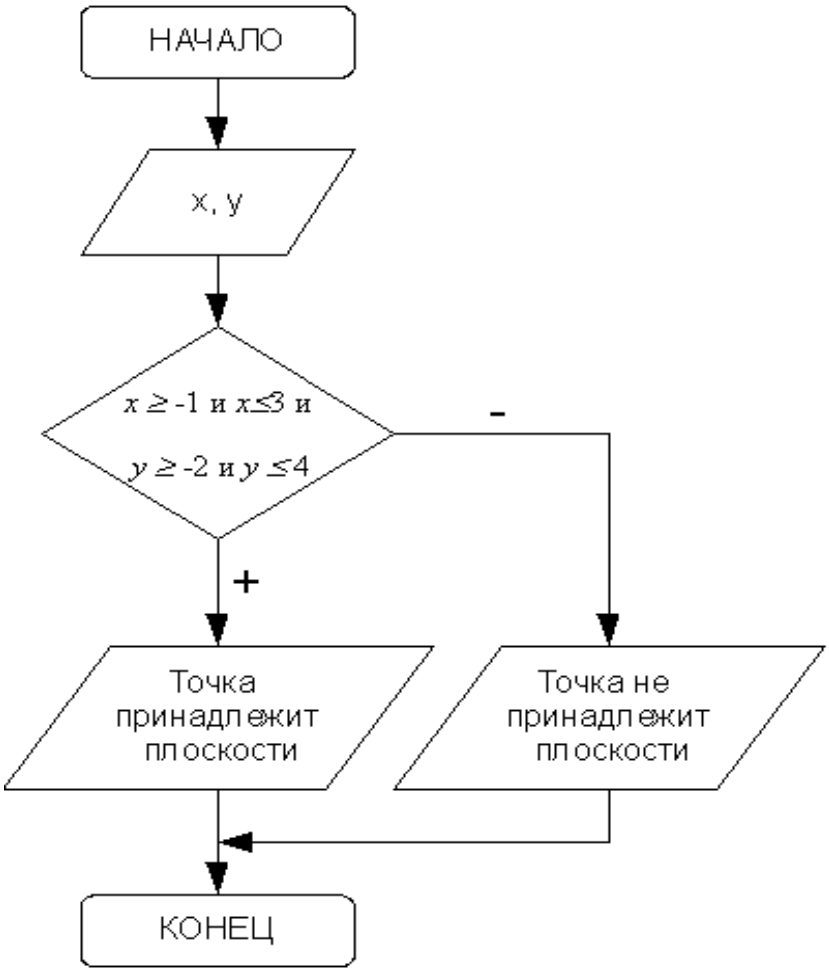

Рисунок 3.13: Алгоритм решения задачи 3.2

```
var
    x,y:real;
begin
    write('x=');
    readln(x);
    write('y=');
    readln(y);
    if (x>=-1) and (x<=3) and (y>=-2) and (y<=4)
                                                then
        writeln('Точка принадлежит плоскости')
    else
        writeln('Точка не принадлежит плоскости');
end.
```

ЗАДАЧА 3.3. Написать программу решения квадратного уравнения $ax^2+bx+c=0$.

Исходные данные: вещественные числа a, b и c – коэффициенты квадратного уравнения.

Результаты работы программы: вещественные числа x1 и x2 – корни квадратного уравнения либо сообщение о том, что корней нет.

Вспомогательные переменные: вещественная переменная d, в которой будет храниться дискриминант квадратного уравнения.

Составим словесный алгоритм решения этой задачи.

1. Начало алгоритма.
2. Ввод числовых значений переменных a, b и c.
3. Вычисление значения дискриминанта d по формуле $d = b^2 - 4ac$.
4. Если d<0, то переход к п.5, иначе переход к п.6.
5. Вывод сообщения Вещественных корней нет и переход к п.8.
6. Вычисление корней $x1 = \frac{-b+\sqrt{d}}{2a}$ и $x2 = \frac{-b-\sqrt{d}}{2a}$.
7. Вывод значений x1 и x2 на экран.
8. Конец алгоритма.

Блок-схема, соответствующая этому описанию, представлена на рис. 3.14.

Текст консольной программы, которая реализует решение квадратного уравнения:

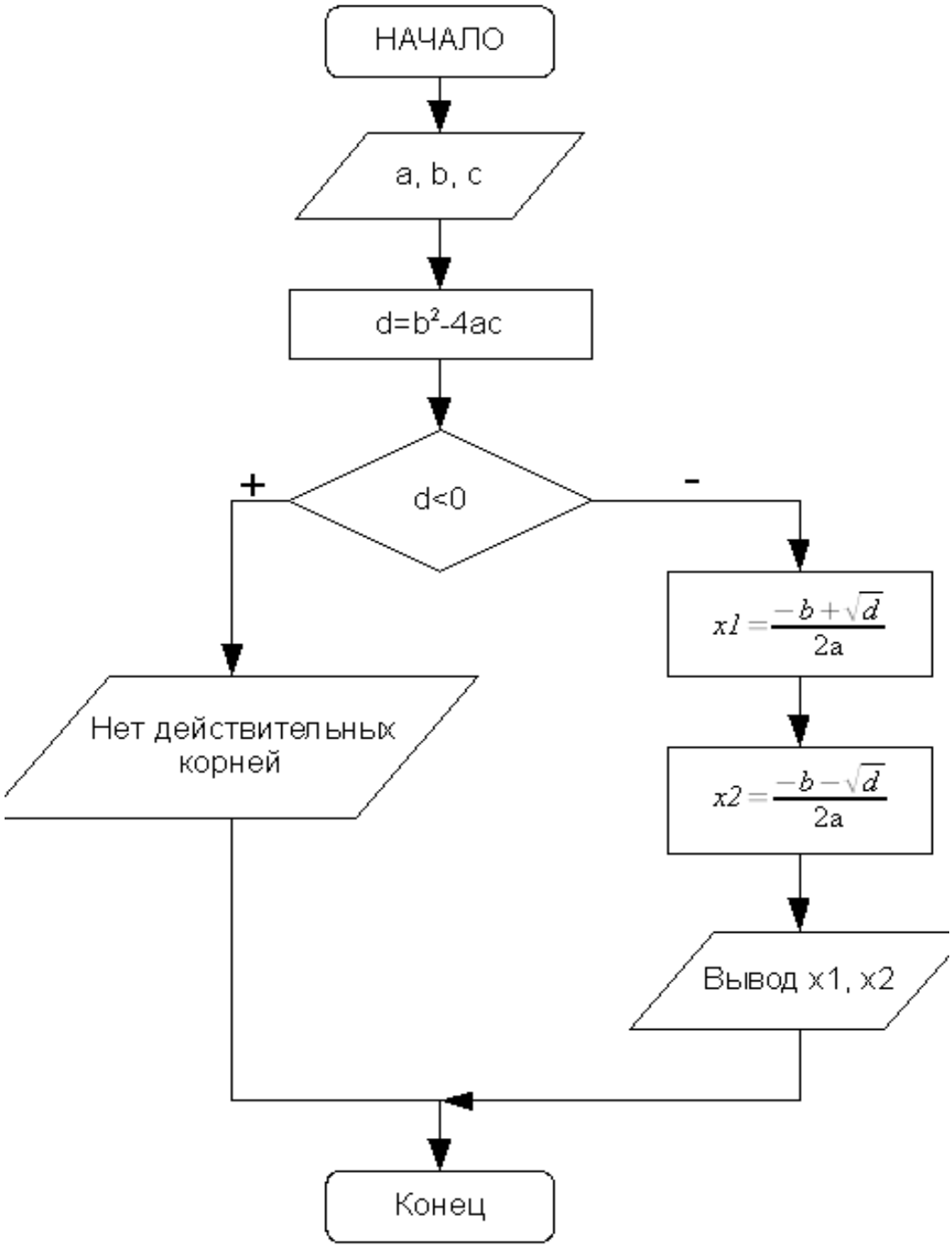

Рисунок 3.14: Алгоритм решения квадратного уравнения

```
{Описание переменных.}
var a,b,c,d,x1,x2: real;
begin
 {Ввод коэффициентов квадратного уравнения.}
 writeln('Введите коэффициенты уравнения');
 readln(a,b,c);
 {Вычисление дискриминанта.}
```

```
d:=b*b-4*a*c;
{Если дискриминант отрицателен,}
if d<0 then
    {то вывод сообщения, что корней нет,}
    writeln('Корней нет')
else
begin
    {иначе вычисление корней x1, x2}
    x1:=(-b+sqrt(d))/2/a;
    x2:=(-b-sqrt(d))/(2*a);
    {и вывод их на экран.}
    writeln('X1=',x1:6:3, ' X2=',x2:6:3)
end
end.
```

ЗАДАЧА 3.4. Составить программу нахождения действительных и комплексных корней квадратного уравнения $ax^2+bx+c=0$.

Исходные данные: вещественные числа `a`, `b` и `c` – коэффициенты квадратного уравнения.

Результаты работы программы: вещественные числа `x1` и `x2` – действительные корни квадратного уравнения либо `x1` и `x2` – действительная и мнимая части комплексного числа.

Вспомогательные переменные: вещественная переменная `d`, в которой будет храниться дискриминант квадратного уравнения.

Можно выделить следующие этапы решения задачи:

1. Ввод коэффициентов квадратного уравнения `a`, `b` и `c`.
2. Вычисление дискриминанта d по формуле $d = b^2 - 4ac$.
3. Проверка знака дискриминанта. Если $d \geq 0$, то вычисление действительных корней:

$$x1 = \frac{-b+\sqrt{d}}{2a} \quad \text{и} \quad x2 = \frac{-b-\sqrt{d}}{2a}$$

и вывод их на экран. При отрицательном дискриминанте выводится сообщение о том, что действительных корней нет, и вычисляются комплексные корни[39] $\frac{-b}{2a} + i\frac{\sqrt{|d|}}{2a}$, $\frac{-b}{2a} - i\frac{\sqrt{|d|}}{2a}$.

У обоих комплексных корней действительные части одинаковые,

39 Комплексные числа записываются в виде a+ib, где a – действительная часть комплексного числа, b – мнимая часть комплексного числа, i – мнимая единица $\sqrt{-1}$.

а мнимые отличаются знаком. Поэтому можно в переменной `x1` хранить действительную часть числа $\frac{-b}{2a}$, в переменной `x2` – модуль мнимой части $\frac{\sqrt{|d|}}{2a}$, а в качестве корней вывести `x1+ix2` и `x1-ix2`.

На рис. 3.15 изображена блок-схема решения задачи.

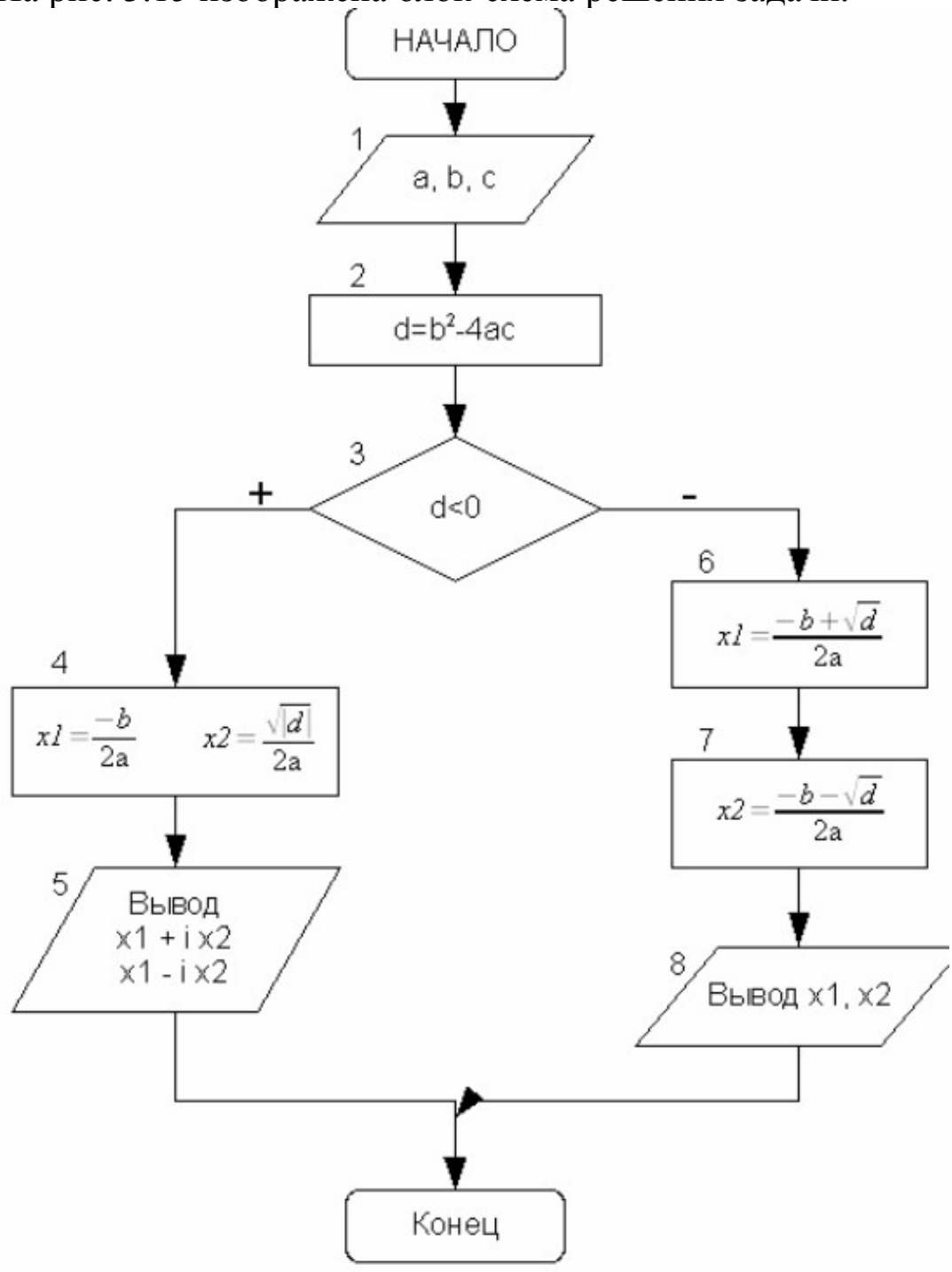

Рисунок 3.15: Алгоритм решения задачи 3.4

Описание блок-схемы, изображенной на рис. 3.15. Блок 1 предназначен для ввода коэффициентов квадратного уравнения. В блоке 2 осуществляется вычисление дискриминанта. Блок 3 осуществляет проверку знака дискриминанта, если дискриминант отрицателен, то корни комплексные, их расчет происходит в блоке 4 (действительная часть корня записывается в переменную `x1`, модуль мнимой – в переменную `x2`), а вывод – в блоке 5 (первый корень `x1+ix2`, второй – `x1-ix2`). Если дискриминант положителен, то вычисляются действительные корни уравнения (блоки 6, 7) и выводятся на экран (блок 8).

Текст программы, реализующей поставленную задачу:

```
var a,b,c,d,x1,x2: real;
begin
 writeln('Введите коэффициенты уравнения');
 readln(a,b,c);
 d:=b*b-4*a*c;
 if d<0 then
 begin
 {Если дискриминант отрицателен,
 то вывод сообщения,
 что действительных корней нет, и вычисление
 комплексных корней.}
     writeln('Действительных корней нет');
     {Вычисление действительной части
         комплексных корней.}
     x1:=-b/(2*a);
     {Вычисление модуля мнимой части
         комплексных корней.}
     x2:=sqrt(abs(d))/(2*a);
     writeln('Комплексные корни уравнения ',
         a:1:2,'x^2+',b:1:2,'x+',c:1:2,'=0');
     {Вывод значений комплексных корней в виде
      x1±ix2}
     writeln(x1:1:2,'+i*(',x2:1:2,')');
     writeln(x1:1:2,'-i*(',x2:1:2,')');
 end
 else
 begin
     {иначе вычисление действительных
```

```
        корней x1, x2}
        x1:=(-b+sqrt(d))/2/a;
        x2:=(-b-sqrt(d))/(2*a);
        {и вывод их на экран.}
        writeln('Действительные корни уравнения ',
             a:1:2,'x^2+',b:1:2,'x+',c:1:2,'=0');
        writeln('X1=',x1:1:2, ' X2=',x2:1:2)
    end
end.
```

ЗАДАЧА 3.5. Составить программу для решения кубического уравнения $ax^3+bx^2+cx+d=0$.

Кубическое уравнение имеет вид $ax^3+bx^2+cx+d=0$ (3.1)

После деления на a уравнение (3.1) принимает канонический вид:

$$x^3+rx^2+sx+t=0 \quad (3.2)$$

где $r=\frac{b}{a}$, $s=\frac{c}{a}$, $t=\frac{d}{a}$. В уравнении (3.2) сделаем замену $x=y-\frac{r}{3}$ и получим приведенное уравнение: $y^3+py+q=0$ (3.3),

где $p=\frac{3s-r^2}{3}$, $q=\frac{2r^3}{27}-\frac{rs}{3}+t$.

Число действительных корней приведенного уравнения (3.3) зависит от знака дискриминанта (табл. 3.1): $D=(\frac{p}{3})^3+(\frac{q}{2})^2$.

Таблица 3.1. Количество корней кубического уравнения

Дискриминант	**Количество действительных корней**	**Количество комплексных корней**
$D\geqslant 0$	1	2
$D<0$	3	-

Корни приведенного уравнения могут быть рассчитаны по формулам Кардано:

$$\begin{aligned} y_1&=u+v \\ y_2&=\frac{-u+v}{2}+\frac{u-v}{2}i\sqrt{3} \\ y_3&=\frac{-u+v}{2}-\frac{u-v}{2}i\sqrt{3} \end{aligned} \quad (3.4)$$

где $u=\sqrt[3]{\frac{-q}{2}+\sqrt{D}}$, $v=\sqrt[3]{\frac{-q}{2}-\sqrt{D}}$.

При отрицательном дискриминанте уравнение (3.1) имеет три действительных корня, но они будут вычисляться через вспомогательные комплексные величины. Чтобы избавиться от этого, можно воспользоваться формулами:

$$y_1=2\sqrt[3]{\rho}\cos(\frac{\phi}{3}) ,$$

$$y_2=2\sqrt[3]{\rho}\cos(\frac{\phi}{3}+\frac{2\pi}{3}) , \quad y_3=2\sqrt[3]{\rho}\cos(\frac{\phi}{3}+\frac{4\pi}{3}) , \qquad (3.5)$$

где $\rho=\sqrt{\frac{-p^3}{27}}$, $\cos(\phi)=\frac{-q}{2\rho}$.

Таким образом, при положительном дискриминанте кубического уравнения (3.3) расчет корней будем вести по формулам (3.4), а при отрицательном – по формулам (3.5). После расчета корней приведенного уравнения (3.3) по формулам (3.4) или (3.5) необходимо по формулам $x_k=y_k-\frac{r}{3}$, $k=1,2,3...$ перейти к корням заданного кубического уравнения (3.1).

Блок-схема решения кубического уравнения представлена на рис.3.16 - 3.17.

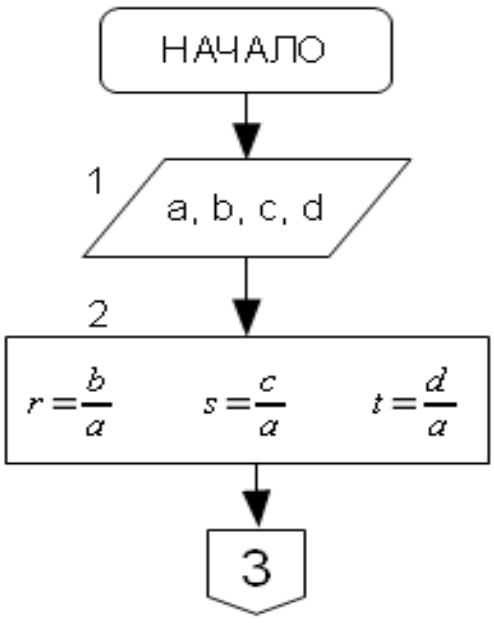

Рисунок 3.16: Алгоритм решения кубического уравнения

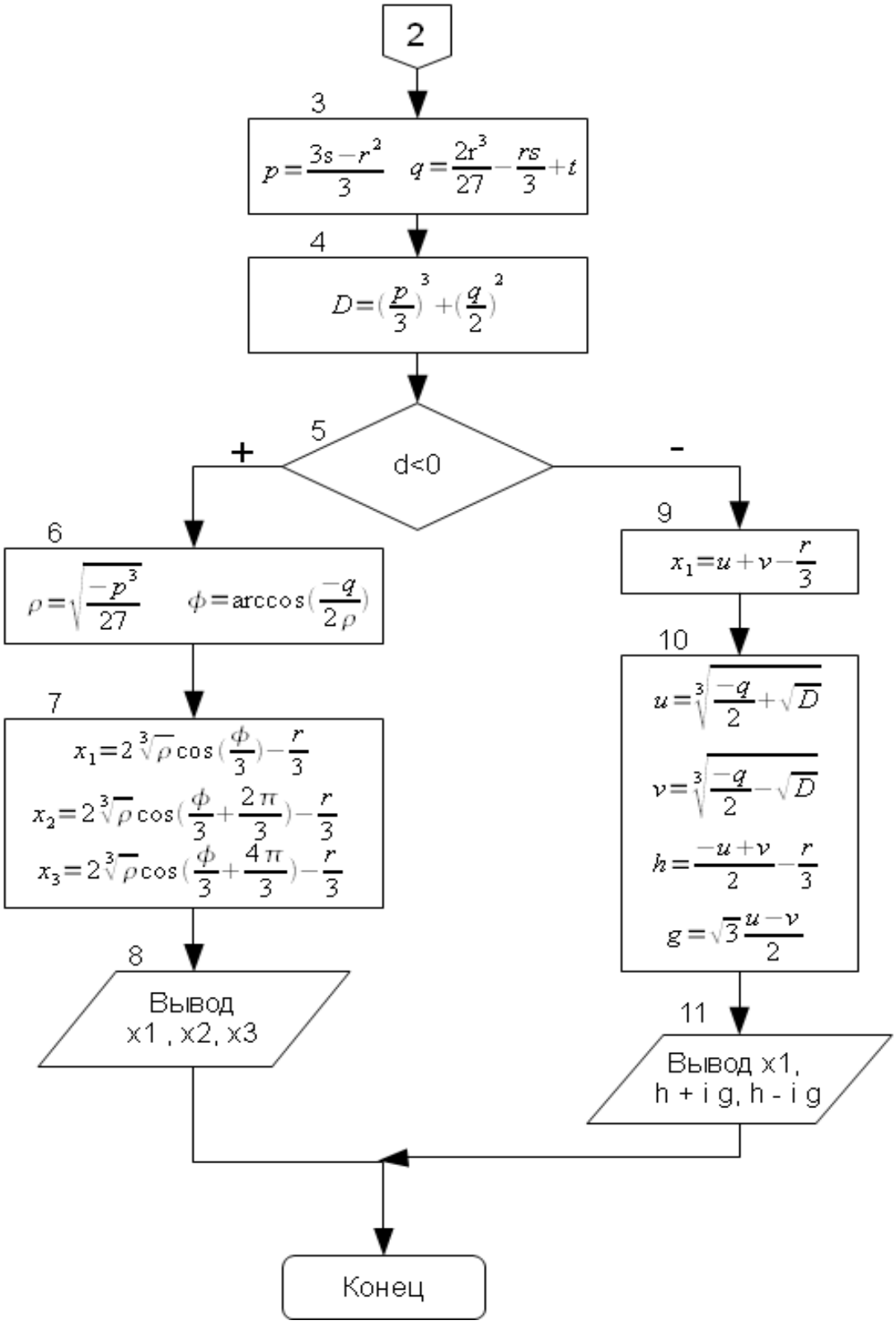

Рисунок 3.17: Алгоритм решения кубического уравнения (продолжение)

Описание алгоритма решения кубического уравнения (рис. 3.16 - 3.17). В блоке 1 вводятся коэффициенты кубического уравнения, в

блоках 2-3 рассчитываются коэффициенты канонического и приведенного уравнений. Блок 4 предназначен для вычисления дискриминанта. В блоке 5 проверяется знак дискриминанта кубического уравнения. Если он отрицателен, то корни вычисляются по формулам (3.5) (блоки 6-7). При положительном значении дискриминанта расчет идет по формулам (3.4) (блок 9, 10). Блоки 8 и 11 предназначены для вывода результатов на экран.

Текст программы приведен далее[40].

```
var
 a,b,c,d,r,s,t,p,q,ro,fi,x1,x2,x3,u,v,h,g:real;
begin
 {Ввод коэффициентов кубического уравнения.}
 write('a=');readln(a);
 write('b=');readln(b);
 write('c=');readln(c);
 write('d=');readln(d);
 {Расчет коэффициентов канонического
    уравнения по (3.2).}
 r:=b/a; s:=c/a; t:=d/a;
 {Вычисление коэффициентов
    приведенного уравнения (3.3).}
 p:=(3*s-r*r)/3;
 q:=2*r*r*r/27-r*s/3+t;
 {Вычисление дискриминанта
    кубического уравнения.}
 d:=(p/3)*sqr(p/3)+sqr(q/2);
 {Проверка знака дискриминанта,
 ветка then реализует формулы (3.5),
 ветка else - формулы (3.4)}
 if d<0 then
 begin
    ro:=sqrt(-p*p*p/27);
```

40 При расчете величин u и v в программе предусмотрена проверка значения подкоренного выражения.

Если $\frac{-q}{2}\mp\sqrt{D}>0$, *то* $u=\sqrt[3]{\frac{-q}{2}+\sqrt{D}}, a \;\; v=\sqrt[3]{\frac{-q}{2}-\sqrt{D}}$.

Если $\frac{-q}{2}\mp\sqrt{D}<0$, *то* $u=-\sqrt[3]{|\frac{-q}{2}+\sqrt{D}|}, a \;\; v=-\sqrt[3]{|\frac{-q}{2}-\sqrt{D}|}$.

Соответственно, при нулевом значении подкоренного выражения u и v обращаются в ноль.

```
    {Следующие два оператора реализуют
    расчет угла fi, сначала вычисляется
    величина косинуса угла, затем вычисляется
    его арккосинус через арктангенс.}
    fi:=-q/(2*ro);
    fi:=pi/2-arctan(fi/sqrt(1-fi*fi));
    {Вычисление действительных
    корней уравнения x1, x2 и x3}
    x1:=2*exp(1/3*ln(ro))*cos(fi/3)-r/3;
    x2:=2*exp(1/3*ln(ro))*
                  cos(fi/3+2*pi/3)-r/3;
    x3:=2*exp(1/3*ln(ro))*
                  cos(fi/3+4*pi/3)-r/3;
    writeln('x1=',x1:1:3,
           ' x2=',x2:1:3,' x3=',x3:1:3);
end
else
begin
    {Вычисление u и v с проверкой знака
              подкоренного выражения. }
    if -q/2+sqrt(d)>0 then
       u:=exp(1/3*ln(-q/2+sqrt(d)))
    else
       if -q/2+sqrt(d)<0 then
          u:=-exp(1/3*ln(abs(-q/2+sqrt(d))))
       else
          u:=0;
    if -q/2-sqrt(d)>0 then
       v:=exp(1/3*ln(-q/2-sqrt(d)))
    else
       if -q/2-sqrt(d)<0 then
          v:=-exp(1/3*ln(abs(-q/2-sqrt(d))))
       else
          v:=0;
    {Вычисление действительного
       корня кубического уравнения.}
    x1:=u+v-r/3;
    {Вычисление действительной и
```

```
        мнимой частей комплексных корней.}
        h:=-(u+v)/2-r/3;
        g:=(u-v)/2*sqrt(3);
        writeln('x1=',x1:1:3,' x2=',h:1:3,'+i*',
                g:1:3,' x3=',h:1:3,'-i*',g:1:3);
 end
end.
```

ЗАДАЧА 3.6. Заданы коэффициенты *a*, *b* и *c* биквадратного уравнения $ax^4+bx^2+c=0$. Найти все его действительные корни.

Входные данные: a, b, c.

Выходные данные: x1, x2, x3, x4.

Для решения биквадратного уравнения необходимо заменой $y=x^2$ привести его к квадратному уравнению $ay^2+by+c=0$ и решить это уравнение.

Опишем алгоритм решения этой задачи (рис. 3.18):

1. Ввод коэффициентов биквадратного уравнения a, b и c (блок 1).
2. Вычисление дискриминанта уравнения d (блок 2).
3. Если d < 0 (блок 3), вывод сообщения, что корней нет (блок 4), а иначе определяются корни соответствующего квадратного уравнения y1 и y2 (блок 5).
4. Если y1<0 и y2<0 (блок 6) , то вывод сообщения, что корней нет (блок 7).
5. Если y1≥0 и y2≥0 (блок 8), то вычисляются четыре корня по формулам $\pm\sqrt{y_1}, \pm\sqrt{y_2}$ (блок 9) и выводятся значения корней (блок 10).
6. Если условия 4) и 5) не выполняются, то необходимо проверить знак y1. Если y1≥0 (блок 11), то вычисляются два корня по формуле $\pm\sqrt{y_1}$ (блок 12), иначе (если y2≥0) вычисляются два корня по формуле $\pm\sqrt{y_2}$ (блок 13). Вывод вычисленных корней (блок 14).

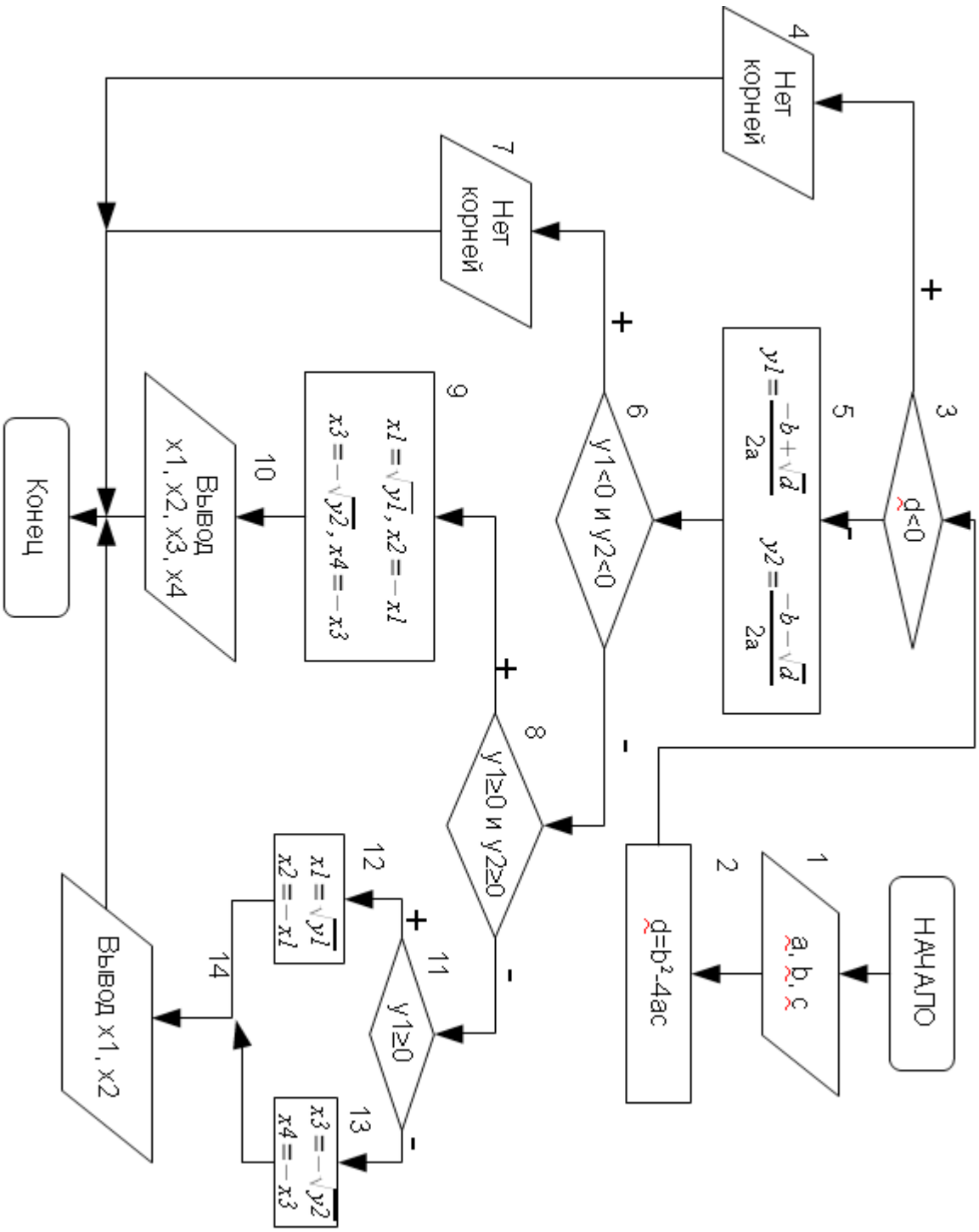

Рисунок 3.18: Алгоритм решения биквадратного уравнения

Текст программы на языке Free Pascal с комментариями:

```
var
    a,b,c,d,x1,x2,x3,x4,y1,y2: real;
begin
 {Ввод коэффициентов уравнения.}
 writeln('Введите коэффициенты уравнения');
 readln(a,b,c);
 {Вычисление дискриминанта.}
 d:=b*b-4*a*c;
 {Если он отрицателен,}
 if d<0 then
    {вывод сообщения «Корней нет»}
    writeln('Корней нет')
 {Если дискриминант ≥0,}
 else
{Описание переменных:
a,b,c – коэффициенты биквадратного уравнения,
d – дискриминант,
x1,x2,x3,x4 – корни биквадратного уравнения,
y1,y2 – корни уравнения ay^2+by+c=0.}
 begin
    {вычисление корней квадратного уравнения.}
    y1:=(-b+sqrt(d))/2/a;
    y2:=(-b-sqrt(d))/(2*a);
    {Если оба корня квадратного уравнения <0,}
    if (y1<0) and (y2<0) then
       {вывод сообщения «Корней нет».}
       writeln('Корней нет')
    {Если оба корня квадратного уравнения ≥0,}
    else if (y1>=0) and (y2>=0) then
    begin
       {вычисление четырех корней уравнения.}
       x1:=sqrt(y1);  x2:=-x1;
       x3:=sqrt(y2);  x4:=-sqrt(y2);
       {Вывод корней уравнения на экран.}
       writeln('X1=',x1:6:3,' X2=',x2:6:3);
       writeln('X3=',x3:6:3,' X4=',x4:6:3);
    end
```

```
        {Если не выполнились оба условия
            1.  y1<0 И y2<0
            2.  y1>=0 И y2>=0,
        то проверяем условие y1>=0}
        else if (y1>=0) then
        {Если оно истинно}
        begin
            x1:=sqrt(y1);  x2:=-x1;
            writeln('X1=',x1:6:3,'   X2=',x2:6:3);
        end
        else
        {Если условие y1>=0 ложно, то}
        begin
            x1:=sqrt(y2);  x2:=-x1;
            writeln('X1=',x1:6:3,'   X2=',x2:6:3);
        end
    end
end.
```

3.4.2 Оператор варианта case

Оператор варианта `case` необходим в тех случаях, когда в зависимости от значений какой-либо переменной надо выполнить те или иные операторы.

```
case переменная of
 набор_значений_1: оператор_1;
 набор_значений_2: оператор_2;
 …
 набор_значений_N: оператор_N
else
 альтернативный_оператор
end;
```

Оператор работает следующим образом. Если `переменная` принимает значение из `набора_значений_1`, то выполняется `оператор_1`. Если `переменная` принимает значение из `набора_значений_2`, то выполняется `оператор_2` и так далее. Если `переменная` не принимает значений из имеющихся наборов, то выполняется `альтернативный_оператор`, расположенный после ключевого слова `else`.

Тип переменной должен быть только перечислимым (включая char и boolean), диапазоном или целочисленным. Набор_значений – это конкретное значение управляющей переменной или выражение, при котором необходимо выполнить соответствующий оператор, игнорируя остальные варианты. Значения в каждом наборе должны быть уникальны, то есть они могут появляться только в одном варианте. Пересечение наборов значений для разных вариантов является ошибкой.

Альтернативная ветвь else может отсутствовать, тогда оператор имеет вид:

```
case переменная of
 набор_значений_1: оператор_1;
 набор_значений_2: оператор_2;
 …
 набор_значений_N: оператор_N;
end;
```

Кроме того, в операторе case допустимо использование составного оператора. Например:

```
case переменная of
 набор_значений_1: begin
                     оператор_A;
                     оператор_B;
                   end;
 набор_значений_2: begin
                     оператор_C;
                     оператор_D;
                     оператор_E;
                   end;
 …
 набор_значений_N: оператор_N;
end;
```

ЗАДАЧА 3.7. Вывести на печать название дня недели, соответствующее заданному числу D, при условии, что в месяце 31 день и первое число – понедельник.

Для решения задачи воспользуемся операцией mod, позволяющей вычислить остаток от деления двух чисел, и условием, что 1-е число – понедельник. Если в результате остаток от деления (обозначим его R)

заданного числа D на семь будет равен единице, то это понедельник, двойке – вторник, тройке – среда и так далее. Следовательно, при построении алгоритма необходимо использовать семь условных операторов, как показано рис. 3.19.

Решение задачи станет значительно проще, если при написании программы воспользоваться оператором варианта:

```
var d:byte;
begin
 write('Введите число D='); readln(D);
{Вычисляется остаток от деления D на 7.}
```

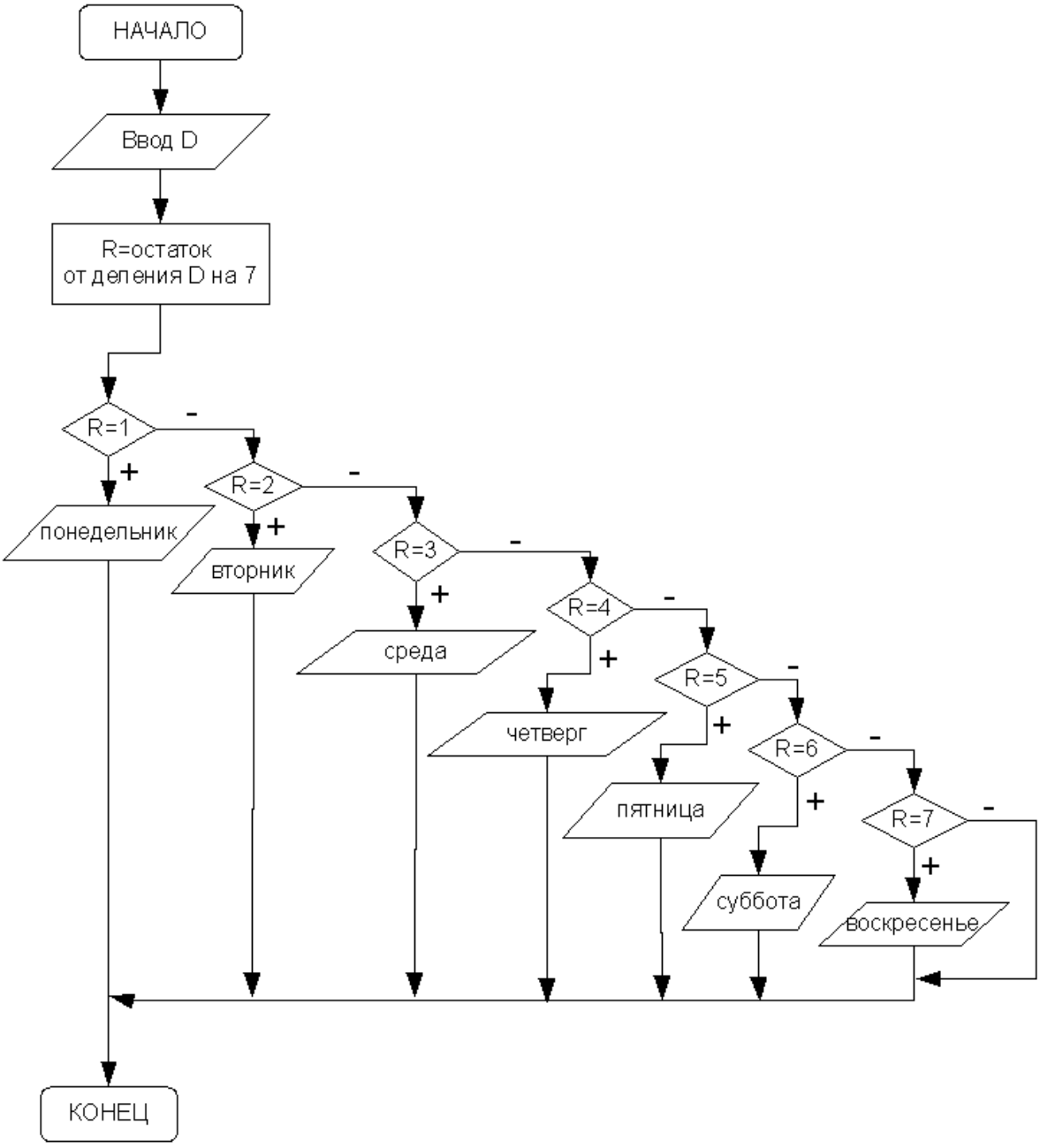

Рисунок 3.19: Алгоритм решения задачи 3.7

```
case D mod 7 of
    {В зависимости от полученного значения}
    {на печать выводится название дня недели}
    1:  writeln('ПОНЕДЕЛЬНИК');
    2:  writeln('ВТОРНИК');
    3:  writeln('СРЕДА');
    4:  writeln('ЧЕТВЕРГ');
    5:  writeln('ПЯТНИЦА');
    6:  writeln('СУББОТА');
    0:  writeln('ВОСКРЕСЕНЬЕ');end;end.
```

В предложенной записи оператора варианта отсутствует ветвь `else`. Это объясняется тем, что переменная `R` может принимать только одно из указанных значений, т.е. 1, 2, 3, 4, 5, 6 или 0.

ЗАДАЧА 3.8. По заданному номеру месяца `m` вывести на печать название времени года.

Для решения данной задачи необходимо проверить выполнение четырех условий. Если заданное число `m` равно 12, 1 или 2, то это зима, если `m` попадает в диапазон от 3 до 5, то – весна, лето определяется принадлежностью числа `m` диапазону от 6 до 8 и, соответственно, при равенстве переменной `m` 9, 10 или 11 – это осень. Понятно, что область возможных значений переменной `m` находится в диапазоне от 1 до 12 и если пользователь введет число, не входящее в этот интервал, то появится сообщение об ошибке. Для этого в операторе `case` программы предусмотрена альтернативная ветка `else`.

```
Var m:byte;
begin
 write('Введите номер месяца m=');
 readln(m);
 {Проверка допустимых значений переменной m.}
 case m of
    {В зависимости от значения m на печать}
    {выводится название времени года.}
    12,1,2:writeln('ЗИМА');
    3..5:  writeln('ВЕСНА');
    6..8:  writeln('ЛЕТО');
    9..11: writeln('ОСЕНЬ')
```

```
    {Если значение переменной m выходит за пределы
    области допустимых значений, то выдается
    сообщение об ошибке.}
        else
            writeln('ОШИБКА ПРИ ВВОДЕ!!!');
        end
end.
```

3.5 Обработка ошибок. Вывод сообщений в среде Lazarus

Понятно, что чем меньше в программе ошибок, тем она лучше. В очень хорошей программе ошибок нет вообще. А это значит, что программист должен не только основательно продумать алгоритм, поставленной задачи, но и предугадать ошибки, которые может допустить пользователь, работая с программой.

Если пользователь допустил ошибку, например, при вводе данных, его необходимо проинформировать об этом. Для этого можно воспользоваться функцией `MessageDlg`, которая выводит *сообщение* в отдельном окне. В общем виде функцию записывают так:

```
MessageDlg(сообщение, тип_сообщения,
                        [список_кнопок], справка);
```

где

- `сообщение` – текст, который будет отображен в окне сообщения;
- `тип_ сообщения` – определяет внешний вид окна (табл. 3.2);
- `список_кнопок` – константы (перечисляются через запятую), определяющие тип кнопок окна сообщения (табл. 3.3);
- `справка` – номер окна справочной системы, которое будет выведено на экран, если нажать **F1**, параметр равен нулю, если использование справки не предусмотрено.

Таблица. 3.2. Тип окна сообщения.

Параметр	Тип окна сообщения
mtInformation	информационное
mtWarning	предупредительное
mtError	сообщение об ошибке
mtConfirmation	запрос на подтверждение
mtCustom	обычное

Таблица. 3.3. Тип кнопки в окне сообщения.

Константа	Кнопка в окне сообщения
mbYes	Да
mbNo	Нет
mbOk	Ок
mbCancel	Отмена
mbAbort	Прервать
mbRetry	Повторить
mbIgnore	Пропустить
mbHelp	Помощь

Вернемся к задаче решения квадратного уравнения (задача 3.3). Нами был рассмотрен алгоритм решения этой задачи и написана программа на языке программирования Free Pascal. Реализуем эту задачу в среде Lazarus. Создадим новый проект[41] (рис. 3.20).

Рисунок 3.20: Форма для решения квадратного уравнения

Для организации ввода коэффициентов уравнения внедрим на форму четыре объекта типа надпись (Label1, Label2, Label3, Label4) и три поля ввода (Edit1, Edit2, Edit3). Корни уравнения или сообщение об их отсутствии будем выводить в надпись Label5[42].

Все действия по вычислению корней квадратного уравнения будут выполняться при нажатии кнопки Button1.

При вводе данных в программе могут возникнуть следующие ошибки:

- в поле ввода оказалась строка, которую невозможно преобразовать в число;
- значение коэффициента a равно нулю[43].

Для того чтобы не допустить подобных ошибок необходимо контролировать данные, вводимые пользователем. Применим для этой цели встроенную процедуру Val(S,X,Kod), которая преобразовывает строку S в целое или вещественное число X. Если преоб-

41 Подробно о создании проекта см. в первой главе.
42 На этапе конструирования формы Label5.Visible:=false.
43 В этом случае при вычислении корней произойдет деление на ноль.

разование прошло успешно, то параметр Kod принимает значение, равное нулю, а результат преобразования записывается в переменную x. В противном случае Kod содержит номер позиции в строке S, где произошла ошибка, и содержимое переменной X не меняется. Далее приведен фрагмент программы с подробными комментариями:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 a,b,c,d,x1,x2: real;
 kod1,kod2,kod3:integer;
begin
 //Ввод значений коэффициентов уравнения.
 //Из поля ввода Edit1 считывается строка
//символов и преобразовывается в вещественное
//число, если преобразование прошло успешно,
//то kod1=0 и полученное число записывается
//в переменную a.
 val(Edit1.Text,a,kod1);
 val(Edit2.Text,b,kod2);
 val(Edit3.Text,c,kod3);
 //Если преобразования прошли успешно, то
 if (kod1=0) and (kod2=0) and (kod3=0) then
     //проверить чему равен первый коэффициент.
     //Если значение первого коэффициента
     //равно нулю, то
     if a=0 then
         //выдать соответствующее сообщение.
         MessageDlg('Введите не нулевое
         значение a', mtInformation,[mbOk],0)
     else
     //иначе перейти к решению уравнения
     begin
         d:=b*b-4*a*c;
         Label5.Visible:=true;
         if d<0 then
             Label5.Caption:='В уравнении'+
             chr(13)+'нет действительных корней'
         else
         begin
```

```
            x1:=(-b+sqrt(d))/2/a;
            x2:=(-b-sqrt(d))/(2*a);
        Label5.Caption:='X1='+
                    FloatToStr(x1)+chr(13)+
                    'X2='+FloatToStr(x2);
        end;
    end
 else
//Преобразование не выполнено,
//выдать сообщение.
    MessageDlg('Введите числовое значение',
                    mtInformation,[mbOk],0);
end;
```

Результаты работы программы показаны на рис. 3.21 - 3.24.

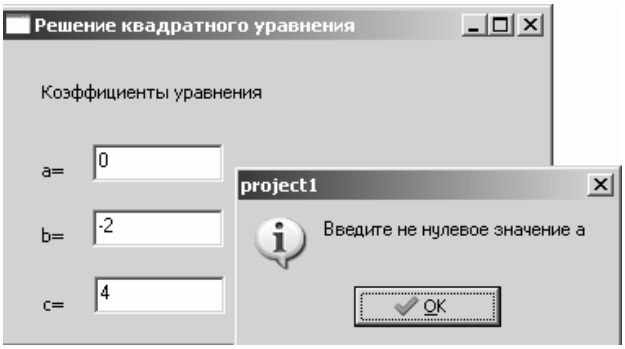

Рисунок 3.21: Обработка ошибки ввода данных — коэффициент a равен 0

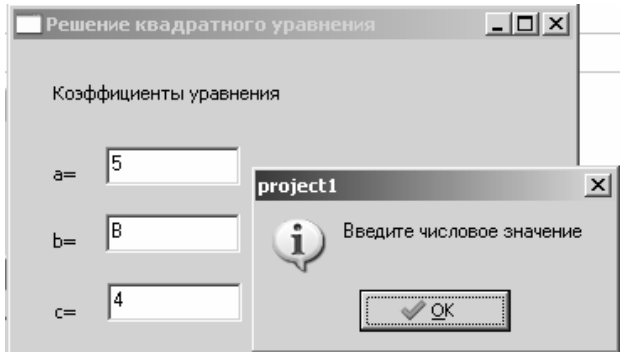

Рисунок 3.22: Обработка ошибки ввода данных - в поле ввода строка, которую невозможно преобразовать в число (коэффициент равен символу B)

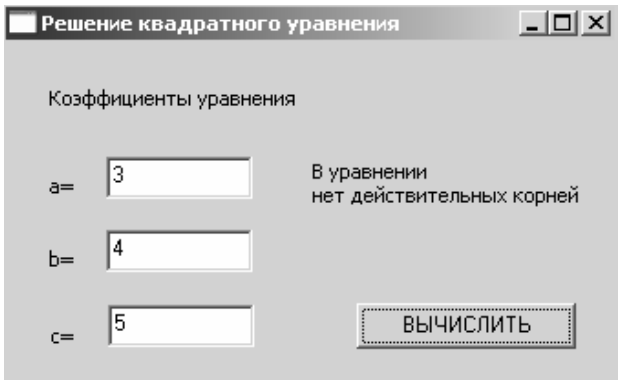

Рисунок 3.23: Решение квадратного уравнения $3x^2+4x+5=0$ (корней нет)

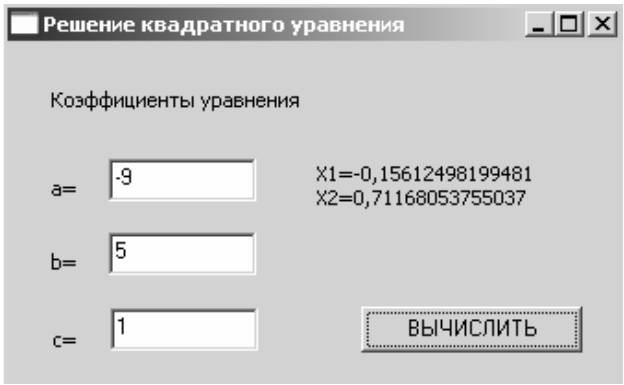

Рисунок 3.24: Вычисление корней квадратного уравнения

Метод обработки ошибок ввода, изложенный в этом примере, не единственный. Можно попытаться контролировать символы, поступающие в поля ввода. Набор символов, которые могут быть преобразованы в число, невелик. Это все цифры от 0 до 9, знак минус и символ запятой. Кроме того, пользователь должен иметь возможность удалять символы из поля ввода. Для этого ему понадобится клавиша `BackSpace` (`#8` – символ с кодом восемь). Представим набор перечисленных символов в виде множества, и если окажется, что введенный символ не принадлежит ему, то будем выводить сообщение об ошибке при вводе.

Возможность контролировать ввод символов обеспечивает событие `OnKeyPress`. Если выделить компонент `Edit2` и на вкладке инспектора объектов дважды щелкнуть возле события нажатия клавиши `OnKeyPress`, то будет создана процедура обработки этого события. Текст процедуры с комментариями:

```
//Обработка события ввод символа в поле Edit2.
procedure TForm1.Edit2KeyPress(Sender: TObject;
                               var Key: Char);
begin       //Если символ не принадлежит
            //множеству допустимых символов,то
 if not(Key in [#8,',','-','0'..'9']) then
 begin             //выдать сообщение и
    MessageDlg('Введите числовое значение',
                       mtInformation,[mbOk],0);
    Abort;  //прервать выполнение подпрограммы.
 end;
end;
```

3.6 Операторы цикла

Циклический процесс, или просто *цикл,* – это повторение одних и тех же действий. Последовательность действий, которые повторяются в цикле, называют *телом цикла*. Один проход цикла называют *шагом*, или *итерацией*[44]. Переменные, которые изменяются внутри цикла и влияют на его окончание, называются *параметрами цикла*.

44 Понятие итерации в математике и программировании несколько отличаются. В математике под итерацией понимают повторение какой-либо математической операции, использующее результат предыдущей аналогичной операции. В программировании итерация — это организация обработки данных, при котором действия повторяются многократно, не приводя при этом к вызовам самих себя (http://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F).

При написании 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.

В языке Free Pascal для удобства программиста предусмотрены три оператора, реализующих циклический процесс: `while`, `repeat... until` и `for`.

3.6.1 Оператор цикла с предусловием while .. do

На рис. 3.25 изображена блок-схема алгоритма *цикла с предусловием*. Оператор, реализующий этот алгоритм в языке Free Pascal, имеет вид:

```
while выражение do оператор;
```

здесь `while .. do` – зарезервированные слова языка Free Pascal, `выражение` – логическая константа, переменная или логическое выражение, `оператор` – любой допустимый оператор языка.

Работает оператор `while` следующим образом. Вычисляется значение `выражения`. Если оно истинно (`True`), выполняется `оператор`. В противном случае цикл заканчивается, и управление передается оператору, следующему за телом цикла. `Выражение` вычисляется перед каждой итерацией цикла. Если при первой проверке `выражение` ложно (`False`), цикл не выполнится ни разу.

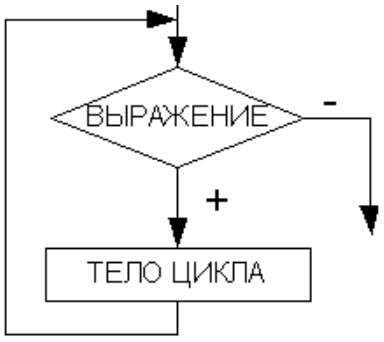

Рисунок 3.25: Алгоритм циклической структуры с предусловием

Если в цикле надо выполнить более одного оператора, необходимо использовать *составной оператор*:

```
while условие do
 begin
     оператор_1;
     оператор_2;
     …
     оператор_n;
 end;
```

Рассмотрим пример.

Пусть необходимо вывести на экран значения функции $y = e^{\sin(x)}\cos(x)$ на отрезке [0;π] с шагом 0.1. Применим *цикл с предусловием*:

```
var  x,y:real;
begin
 {Присваивание параметру цикла
 стартового значения.}
 x:=0;
 {Цикл с предусловием.}
 while x<=pi do     {Пока параметр цикла
        не превышает конечное значение,
        выполнять тело цикла.}
 begin
     {Вычислить значение y.}
     y:=exp(sin(x))*cos(x);
     {Вывод на экран пары x и y.}
     writeln('x=', x,'   y=', y);
     {Изменение параметра цикла -
      переход к следующему значению x.}
     x:=x+0.1;
 end; {Конец цикла.}
end.
```

В результате работы данного фрагмента программы на экран последовательно будут выводиться сообщения со значениями переменных x и y:

```
x= 0; y=1
x= 0.1; y=1.0995
…
x= 3.1; y=-1.0415
```

3.6.2 Оператор цикла с постусловием repeat … until

Если в цикле с предусловием проверка условия осуществляется до тела цикла, то в цикле с постусловием условие проверяется после тела цикла (см. рис. 3.26). Сначала выполняются операторы, являющиеся телом цикла, после чего проверяется условие, если последнее ложно, то цикл повторяется. Выполнение цикла прекратится, если условие станет истинным.

В языке Free Pascal *цикл с постусловием* реализован конструкцией

```
repeat
 оператор;
until выражение;
```

здесь `repeat .. until` – зарезервированные слова языка Free Pascal, `выражение` – логическая константа, переменная или логическое выражение, `оператор` – любой допустимый оператор языка.

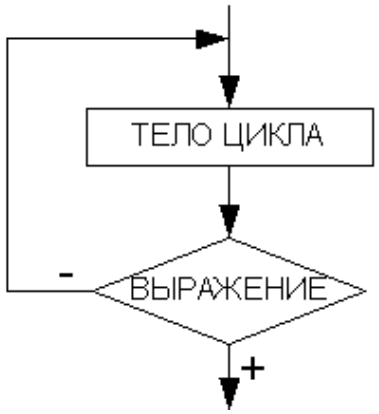

Рисунок 3.26: Алгоритм цикла с постусловием

Если тело цикла состоит более чем из одного оператора, то цикл с постусловием имеет вид:

```
repeat
 оператор_1;
 оператор_2;
 ..............
 оператор_N;
until выражение;
```

Работает цикл следующим образом. В начале выполняется `оператор`, представляющий собой тело цикла. Затем вычисляется значение `выражения`. Если оно ложно (`False`), `оператор` тела цикла выполняется еще раз. В противном случае цикл завершается, и управление передается оператору, следующему за циклом.

Таким образом, нетрудно заметить, что цикл с постусловием всегда будет выполнен хотя бы один раз в отличие от цикла с предусловием, который может не выполниться ни разу.

Если применить цикл с постусловием для создания подпрограммы, которая выводит значения функции $y=e^{\sin(x)}\cos(x)$ на отрезке $[0;\pi]$ с шагом 0.1, получим:

```
var
 x,y:real;
 begin
 {Присваивание параметру цикла
 стартового значения.}
 x:=0;
 {Цикл с постусловием.}
 repeat          {Начало цикла}
     y:=exp(sin(x))*cos(x);
     writeln('x=',x,'   y=',y);
     x:=x+0.1;   {Изменение параметра цикла.}
 until x>pi;     {Закончить работу цикла,
```

```
когда параметр превысит конечное значение.}
end.
```

3.6.3 Оператор цикла for … do

Операторы *цикла с условием* обладают значительной гибкостью, но не слишком удобны для организации «строгих» циклов, которые должны быть выполнены заданное число раз. Оператор цикла `for … do` используется именно в таких случаях:

```
for i:= in to ik do оператор;
for i:= ik downto in do оператор;
```

где `оператор` – любой оператор языка, `i`, `in` и `ik` — переменная целочисленного или перечислимого типов.

Переменную `i` называют *параметром цикла.* Переменные `in` и `ik` — *диапазон* изменения параметра цикла: `in` — начальное значение, а `ik` — конечное значение параметра цикла.

Шаг изменения цикла `for` всегда постоянен и равен интервалу между двумя ближайшими значениями типа параметра цикла (при целочисленном значении параметра цикла шаг равен 1).

В случае если тело цикла состоит более чем из одного оператора, необходимо использовать *составной оператор*:

```
for i:= in to ik do
 begin
     оператор_1;
     оператор_2;
     ..........
     оператор_n;
 end;
```

Опишем (рис.3.27) алгоритм работы *цикла* `for … do`:

1. Параметру цикла `i` присваивается начальное значение `in`.
2. Если значение параметра цикла превосходит конечное значение (`i>ik`), то цикл завершает свою работу. В противном случае выполняется пункт 3.
3. Выполняется `оператор`.
4. Значение параметра цикла `i` изменяется на соответствующий `шаг` и осуществляется переход к п.2 и т. д.

Понятно, что этот алгоритм представляет собой цикл с предусловием.

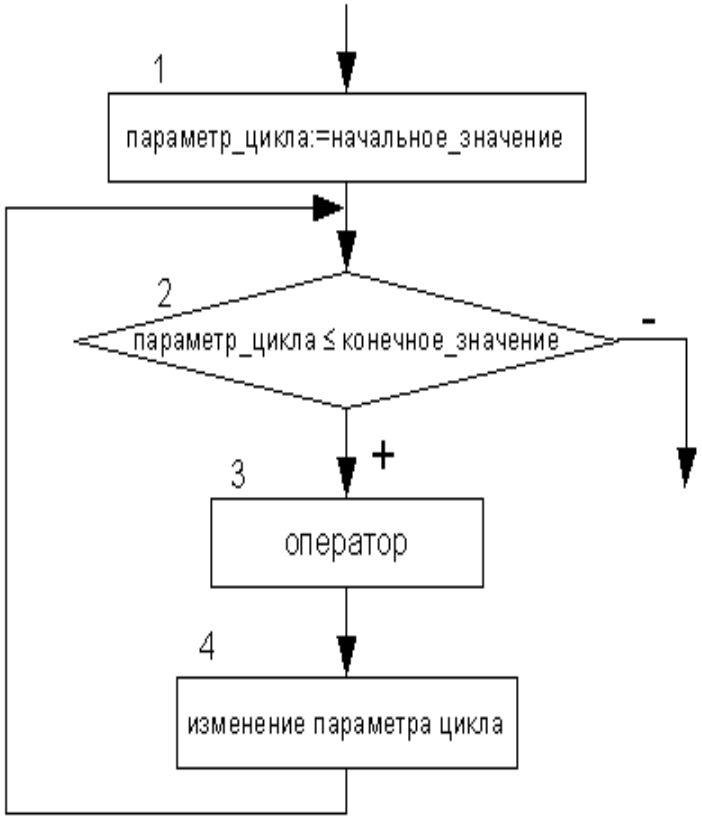

Рисунок 3.27: Алгоритм работы цикла for...do

В дальнейшем, чтобы избежать создания слишком громоздких алгоритмов, в блок-схемах цикл `for` будем изображать так, как показано на рис. 3.28[45].

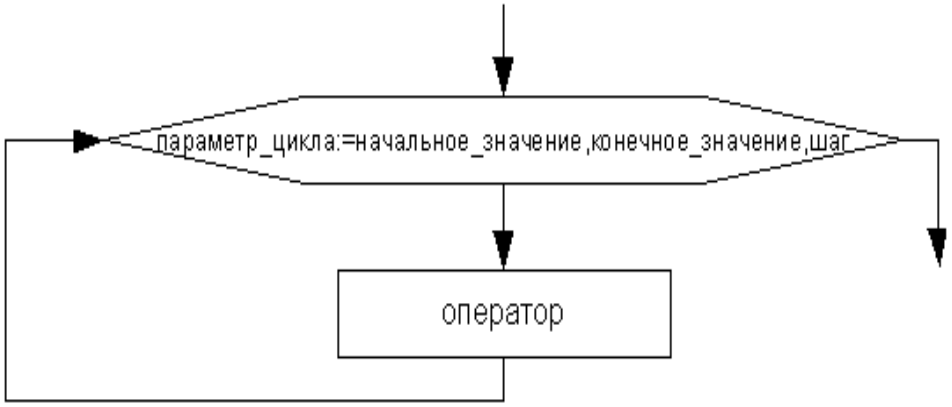

Рисунок 3.28: Представление цикла for...do с помощью блок-схемы

Фрагмент подпрограммы, приведенный далее, демонстрирует применение цикла `for`:

```
var i:integer; c:char;
 begin
```

45 Если шаг изменения параметра цикла равен единице, его в блок-схемах можно не указывать.

```
    {Вывод на экран чисел от 1 до 10.}
    for i:=1 to 10 do  writeln(i);
    {Вывод на экран чисел от 10 до -10.}
    for i:=10 downto -10 do writeln(i);
    {Вывод на экран символов от a до r.}
    for c:='a' to 'r' do writeln(c);
end;
```

Вернемся к задаче вывода значений функции $y=e^{\sin(x)}\cos(x)$ на отрезке [0;π] с шагом 0.1. Как видим, здесь количество повторений цикла явно не задано. Однако это значение, можно легко вычислить. Предположим, что параметр цикла x принимает значения в диапазоне от xn до xk, изменяясь с шагом dx, тогда количество повторений тела цикла можно определить по формуле:

$$n=\frac{xk-xn}{dx}+1 \quad , \qquad (3.6)$$

округлив результат деления до целого числа. Следовательно, фрагмент программы вывода значений функции $y=e^{\sin(x)}\cos(x)$ на отрезке [0;π] с шагом 0.1 будет иметь вид:

```
var i,n:integer; x,y:real;
 begin
 {Количество повторений цикла:}
 {n=(xk-xn)/dx+1; xk=pi, xn=0, dx=0.1}
 n:=round((pi-0)/0.1)+1;
 x:=0;   {Начальное значение аргумента.}
 {Цикл с известным числом повторений,}
 {i – параметр цикла,
 изменяется от 1 до n с шагом 1.}
 for i:=1 to n do
 begin              {Начало цикла.}
    {Вычисление значения функции }
    y:=exp(sin(x))*cos(x);
    {для соответствующего значения аргумента.}
    writeln('x=',x,'   y=',y);
    x:=x+0.1;       {Вычисление нового
                    значения аргумента.}
 end;               {Конец цикла.}
end.
```

3.7 Операторы передачи управления

Операторы передачи управления принудительно изменяют порядок выполнения команд. В языке Free Pascal таких операторов пять: `goto`, `break`, `continue`, `exit` и `halt`.

Оператор `goto метка`, где `метка` обычный идентификатор, применяют для безусловного перехода, он передает управление оператору с меткой:

```
метка: оператор;
```
[46].

Операторы `break` и `continue` используют только внутри циклов. Так, оператор `break` осуществляет немедленный выход из циклов `repeat`, `while`, `for` и управление передается оператору, находящемуся непосредственно за циклом. Оператор `continue` начинает новую итерацию цикла, даже если предыдущая не была завершена.

Оператор `exit` осуществляет выход из подпрограммы.

Оператор `halt` прекращает выполнение программы.

3.8 Решение задач с использованием циклов

Рассмотрим использование циклических операторов на конкретных примерах.

ЗАДАЧА 3.9. 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 3.4.

Таблица 3.4. Поиск НОД для чисел A=25 и B=15.

Исходные данные	**Первый шаг**	**Второй шаг**	**Третий шаг**	**Значение НОД для A и B равно 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. 3.29, для решения поставленной задачи используется цикл с предусловием, то

46 Обычно применение оператора goto приводит к усложнению программы и затрудняет отладку. Он нарушает принцип структурного программирования, согласно которому все блоки, составляющие программу, должны иметь только один вход и один выход. В большинстве алгоритмов применения этого оператора можно избежать.

есть тело цикла повторяется до тех пор, пока A не равно B. Следовательно, при создании программы воспользуемся циклом while … do.

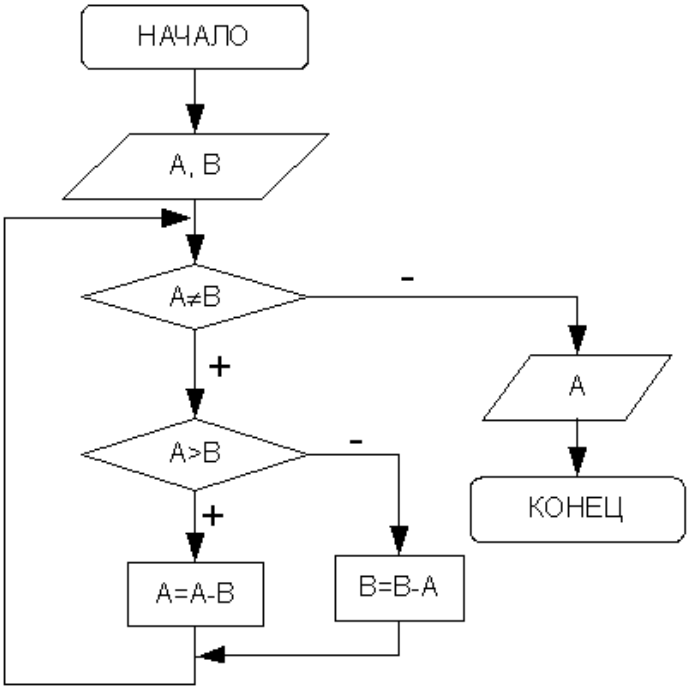

Рисунок 3.29: Алгоритм поиска наибольшего общего делителя двух чисел

Программа на языке Free Pascal, реализующая задачу 3.9:

```
var  a,b:word;
 begin
 writeln('введите два натуральных числа');
 write('A='); readln(A);
 write('B='); readln(B);
 {Если числа не равны, выполнять тело цикла.}
 while a<>b do
 {Если число A больше, чем B, то уменьшить
 его значение на B,}
 if a>b then
        a:=a-b
 {иначе уменьшить значение числа B на A.}
 else
        b:=b-a;
 writeln('НОД=',A);
end.
```

ЗАДАЧА 3.10. Вычислить факториал числа N (N!=1·2·3 …·N).

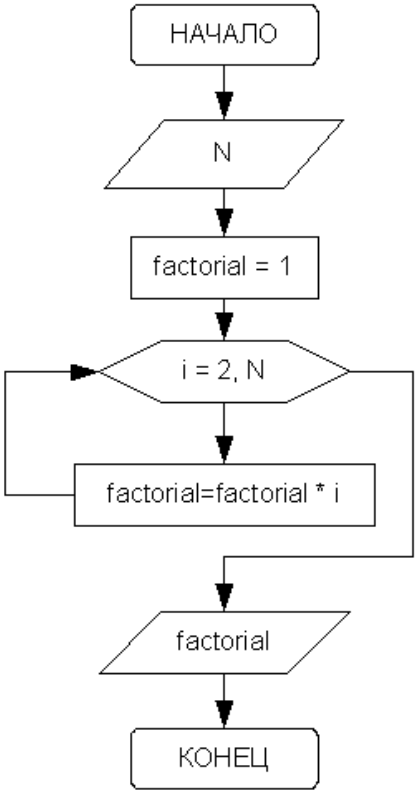

Рисунок 3.30: Алгоритм вычисления факториала

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 переменные: `i` – параметр цикла, целочисленная переменная, последовательно принимающая значения 2, 3, 4 и так далее до `N`. Блок-схема приведена на рис. 3.30.

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 выводится значение переменой `factorial`.

Теперь рассмотрим текст программы вычисления факториала на языке Free Pascal.

```
Var factorial, n, i:integer;
 begin
     write('n='); readln(n);
     factorial:=1;
     for i:=2 to n do
         factorial:=factorial*i;
     writeln(factorial);
 end.
```

ЗАДАЧА 3.11. Вычислить a^n , где n – целое положительное число.

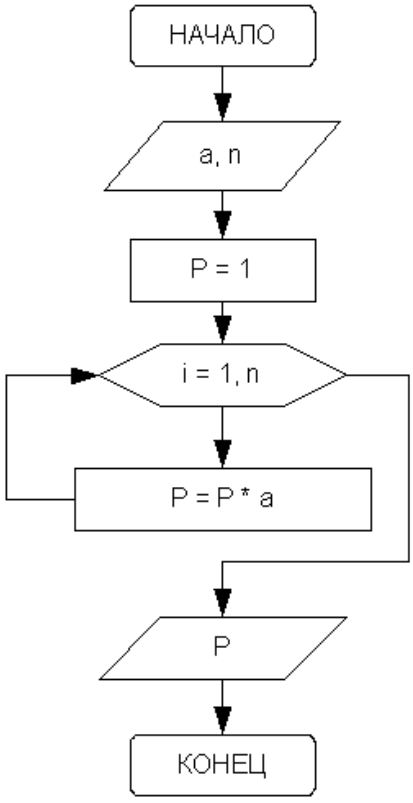

Рисунок 3.31: Алгоритм возведения вещественного числа в целую степень

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. 3.31.

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 3.5 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 3.5. Процесс возведения числа a в степень n

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

Далее приведен текст программы, составленной для решения поставленной задачи.

```
Var  a,p:real; i,n:word;
 begin
     write('Введите основание степени a=');
     readln(a);
     write('Введите показатель степени n=');
     readln(n);
     p:=1;
```

```
        for i:=1 to n do p:=p*a;
        writeln('P=',P:1:3);
    end.
```

ЗАДАЧА 3.12. Вычислить сумму натуральных четных чисел, не превышающих N.

Входные данные: N – целое число.

Выходные данные: S – сумма четных чисел.

Промежуточные переменные: i – параметр цикла, принимает значения 2, 4, 6, 8 и так далее, 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. Перед началом цикла переменной, предназначенной для накапливания суммы, необходимо присвоить значение нуль (s=0).

Блок-схема решения этой задачи представлена на рис. 3.32.Так как параметр цикла i изменяется с шагом 2, в блок–схеме, построенной для решения данной задачи, использован цикл с предусловием, который реализуется при составлении программы с помощью оператора while … do:

```
var
 n,i,S:word;
 begin
     write('n=');
     readln(n);
     S:=0;
     i:=2;
     while i<=n do
     begin
         S:=S+i;
         i:=i+2;
     end;
     writeln('S=',S);
end.
```

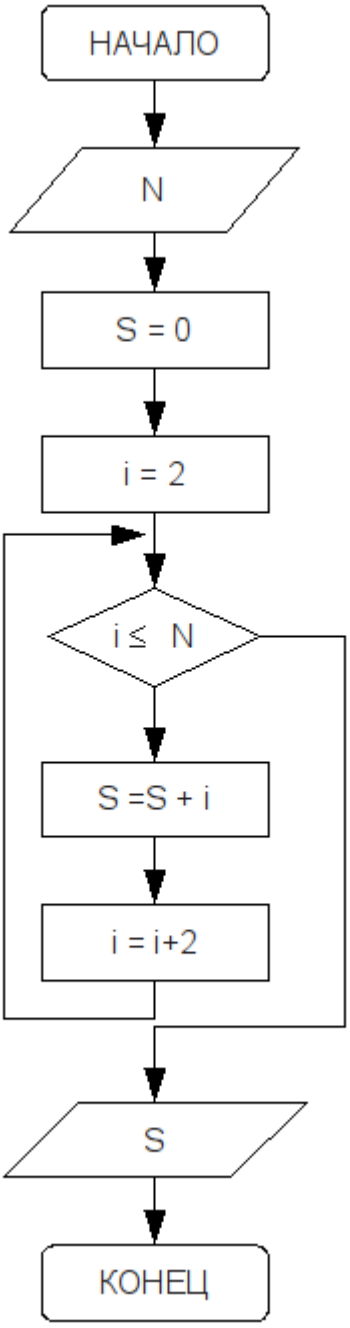

Рисунок 3.32: Алгоритм вычисления суммы четных натуральных чисел

Эту же задачу можно решить иначе, используя цикл `for … do`:

```
var
    n,i,S:word;
begin
    write('n=');
    readln(n);
```

```
    S:=0;
    for i:=1 to n do
    {Если остаток от деления параметра цикла
    на 2 равен 0, то это число четное,
    значит, происходит накапливание суммы.}
    if i mod 2 = 0 then
            S:=S+i;
    writeln('S=',S);
end.
```

В таблице 3.6 приведены результаты тестирования программы для n=7.

Таблица 3.6. Суммирование четных чисел

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

Несложно заметить, что при нечетных значениях параметра цикла значение переменной, предназначенной для накапливания суммы, не изменяется.

ЗАДАЧА 3.13. Дано натуральное число N. Определить К – количество делителей этого числа, не превышающих его (Например, для N=12 делители 1, 2, 3, 4, 6. Количество К=5).

Входные данные: N – целое число.

Выходные данные: целое число K – количество делителей N.

Промежуточные переменные: i – параметр цикла, возможные делители числа N.

В блок-схеме, изображенной на рис. 3.33, реализован следующий алгоритм: в переменную K, предназначенную для подсчета количества делителей заданного числа, помещается значение, которое не влияло бы на результат(k=0).

Далее организовывается цикл, в котором изменяющийся параметр i выполняет роль возможных делителей числа N.

Параметр цикла меняется от 1 до N/2 с шагом 1. 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.

Текст программы, соответствующий описанному алгоритму, приведен далее.

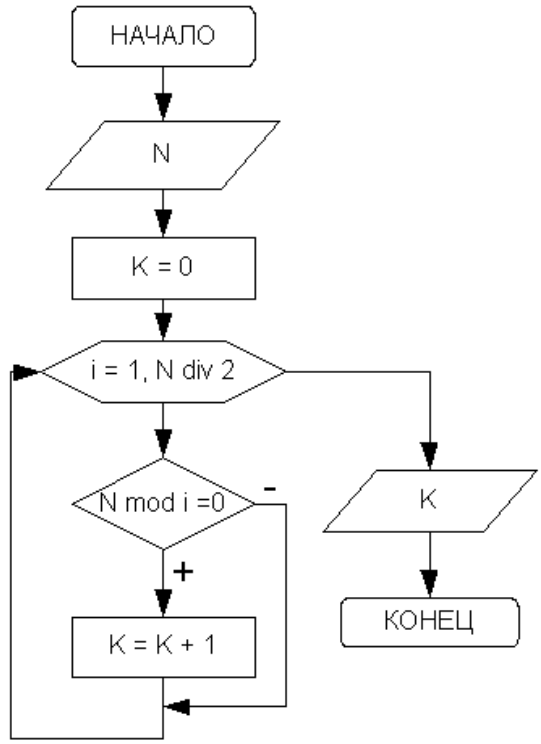

Рисунок 3.33: Алгоритм подсчета делителей натурального числа

```
var
N,i,K:word;
begin
write('N=');
readln(N);
K:=0;
for i:=1 to N div 2 do
{Если N делится
 нацело на i, то}
if N mod i= 0 then
{увеличить счетчик
 на один.}
k:=k+1;
writeln(' K=',K);
end.
```

В таблице 3.7 отображены результаты тестирования алгоритма при определении делителей числа `N=12`.

Таблица 3.7. Определение количества делителей числа `N`

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

ЗАДАЧА 3.14. 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`.

Алгоритм решения этой задачи (рис. 3.34) заключается в том, что необходимо определить, есть ли у числа `N` делители среди чисел от `2` до `N/2`. Если делителей нет — число простое.

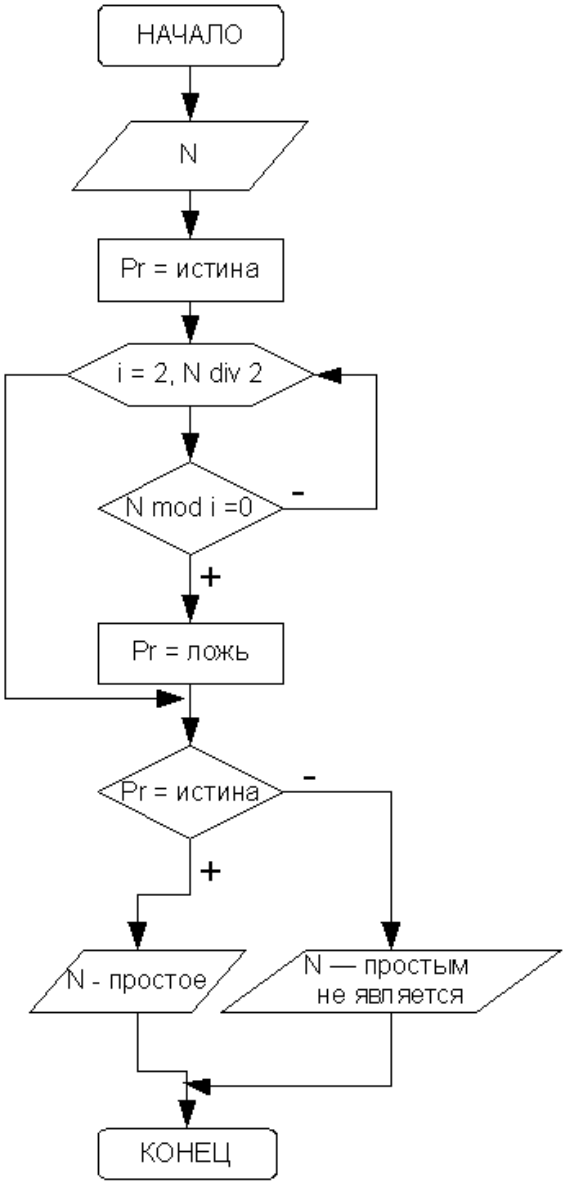

Рисунок 3.34: Алгоритм определения проситого числа

Предположим, что число N является простым (pr:=true). Организуем цикл, в котором переменная i будет изменяться от 2 до N/2.

В цикле будем проверять, делится ли N на i. Если делится, то мы нашли делитель, N не является простым (Pr:=false). Проверка остальных делителей не имеет смысла, поэтому покидаем цикл. В алгоритме предусмотрено два выхода из цикла. Первый – естественный, при исчерпании всех значений параметра, а второй — досрочный.

После выхода из цикла надо проверить значение Pr. Если Pr=true, то число N — простое, иначе N не является простым числом.

При составлении программы на языке Free Pascal досрочный выход из цикла удобно выполнять при помощи оператора break.

Текст программы:

```
var N,i:integer; Pr:boolean;
begin
    write('N='); readln(N);
    Pr:=true;  {Предположим, что число простое.}
    for i:=2 to N div 2 do
    {Если найдется хотя бы один делитель, то}
    if N mod i = 0 then
    begin
        Pr:=false; {число простым не является}
        break;     {досрочный выход из цикла.}
```

```
    end;
{Проверка значения логического параметра и}
if Pr then
{вывод на печать соответствующего сообщения.}
    writeln('Число ',N,' - простое')
else
    writeln('Число ',N,' простым не является');
end.
```

ЗАДАЧА 3.15. Определить количество простых чисел в интервале от N до M, где N и M – натуральные числа, причем N ≤ M.

Алгоритм решения данной задачи представлен на рис. 3.35.

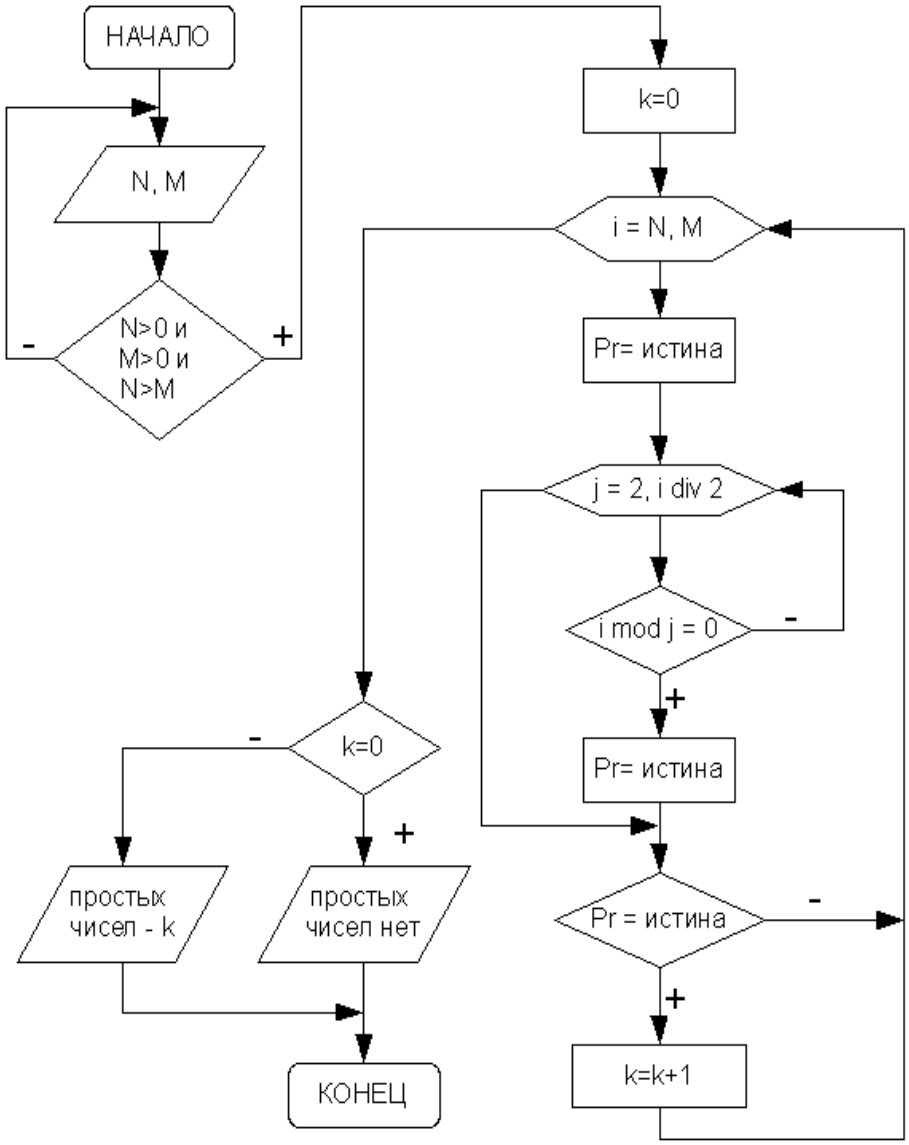

Рисунок 3.35: Алгоритм определения простых чисел в заданном интервале

Обратите внимание, что здесь осуществляется проверка корректности ввода исходных данных. Если границы интервала не положительны или значение N превышает M, ввод данных повторяется в цикле с постусловием до тех пор, пока не будут введены корректные исходные данные.

Далее для каждого числа из указанного интервала (параметр i принимает значения от N до M) происходит проверка. Если число является простым, то количество простых чисел k увеличивается на единицу. Подробно определение простого числа описано в задаче 3.14.

Программа на языке Free Pascal, реализующая алгоритм подсчета количества простых чисел в заданном диапазоне:

```
var N,M,i,j,k: longint; Pr:boolean;
begin
 repeat
     write('N='); readln(N);
     write('M='); readln(M);
 until (N>0) and (M>0) and (N<M);
 k:=0;       {Количество простых чисел}
{Параметр i принимает значения от N до M}
 for i:=N to M do
 begin
     {Определение простого числа.}
     Pr:=true;
     for j:=2 to i div 2 do
         if i mod j = 0 then
         begin
             Pr:=false;
             reak;
         end;
         {Если число простое,
         увеличиваем количество на 1.}
         if Pr then k:=k+1;
 end;
 if k=0 then writeln('Простых чисел нет')
 else writeln('Простых чисел в диапазоне ',k);
end.
```

ЗАДАЧА 3.16. Дано натуральное число N. Определить количество цифр в числе.

Входные данные: `N` – целое число.

Выходные данные: `kol` – количество цифр в числе.

Промежуточные данные: `M` – переменная для временного хранения значения N.

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 3.8.

Таблица 3.8. 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

Алгоритм определения количества цифр в числе представлен на рис. 3.36.Текст программы, реализующей задачу, можно записать так:

```
var M,N:longint; kol:word;
begin
{Так как речь идет о натуральных числах, при вводе предусмотрена проверка. Закончить цикл, если введено положительное число, иначе повторить ввод.}
repeat
    write('N=');readln(N);
until N>0;
M:=N;  {Сохранить значение переменной N. }
kol:=1;{Пусть число состоит из одной цифры.}
while M div 10 > 0 do
{Выполнять тело цикла, пока число делится на 10.}
begin
    kol:=kol+1;    {Счетчик количества цифр.}
    M:=M div 10;   {Изменение числа.}
end;
writeln('kol=', kol); end.
```

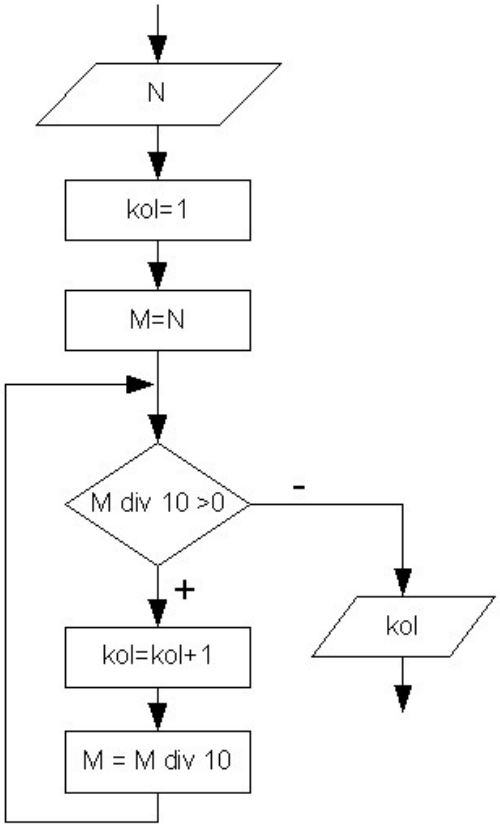

Рисунок 3.36: Алгоритм определения количества цифр в числе

ЗАДАЧА 3.17. Дано натуральное число N. Определить, содержит ли это число нули и в каких разрядах они расположены (например, число 1101111011 содержит ноль в третьем и восьмом разрядах).

Входные данные: `N` – целое число. *Выходные данные*: `pos` – позиция цифры в числе. *Промежуточные данные*: `i` – параметр цикла, `M` – переменная для временного хранения значения `N`.

В связи с тем что разряды в числе выделяются начиная с последнего, то для определения номера разряда в числе, необходимо знать количество цифр в числе[47]. Таким образом, на первом этапе решения задачи необходимо определить `kol` – количество цифр в числе. Затем начинаем выделять из числа цифры, если очередная цифра равна нулю, вывести на экран номер разряда, который занимает эта цифра. Процесс определения текущей цифры числа N=12345 представлен в таблице 3.9.

47 Алгоритм нахождения количества цифр в числе был рассмотрен в предыдущей задаче.

Таблица 3.9. Определение текущей цифры числа

i	Число M	Цифра	Номер позиции
1	120405	120405 mod 10= 5	6
2	12040 div10=1204	12040 mod 10 = 0	5
3	1204 div 10=120	1204 mod 10 = 4	4
4	120 div 10=12	120 mod 10 = 0	3
5	12 div 10=1	12 mod 10 = 2	2
6	1 div = 0	1 div 10 = 1	1

Блок-схема алгоритма решения данной задачи показана на рис. 3.37.

Текст программы, реализующей данный алгоритм:

```
var
 M,N:longint;
 i,pos,kol:word;
begin
 {Так как речь идет о натуральных числах,}
 {при вводе предусмотрена проверка.}
 {Закончить цикл, если введено}
 {положительное число, иначе повторить ввод}
 repeat
     write('N=');
     readln(N);
 until N>0;
//Определение kol - количества разрядов.
 M:=N;  {Сохранить значение переменной N. }
 kol:=1; {Пусть число состоит из одной цифры.}
 while M div 10 > 0 do
 {Выполнять тело цикла,}
 {пока число делится нацело на 10.}
 begin
     kol:=kol+1;    {Счетчик количества цифр.}
     M:=M div 10;   {Изменение числа.}
 end;
 writeln('kol=', kol);
 M:=N;
 pos:=0;        {Пусть в числе нет нулей.}
```

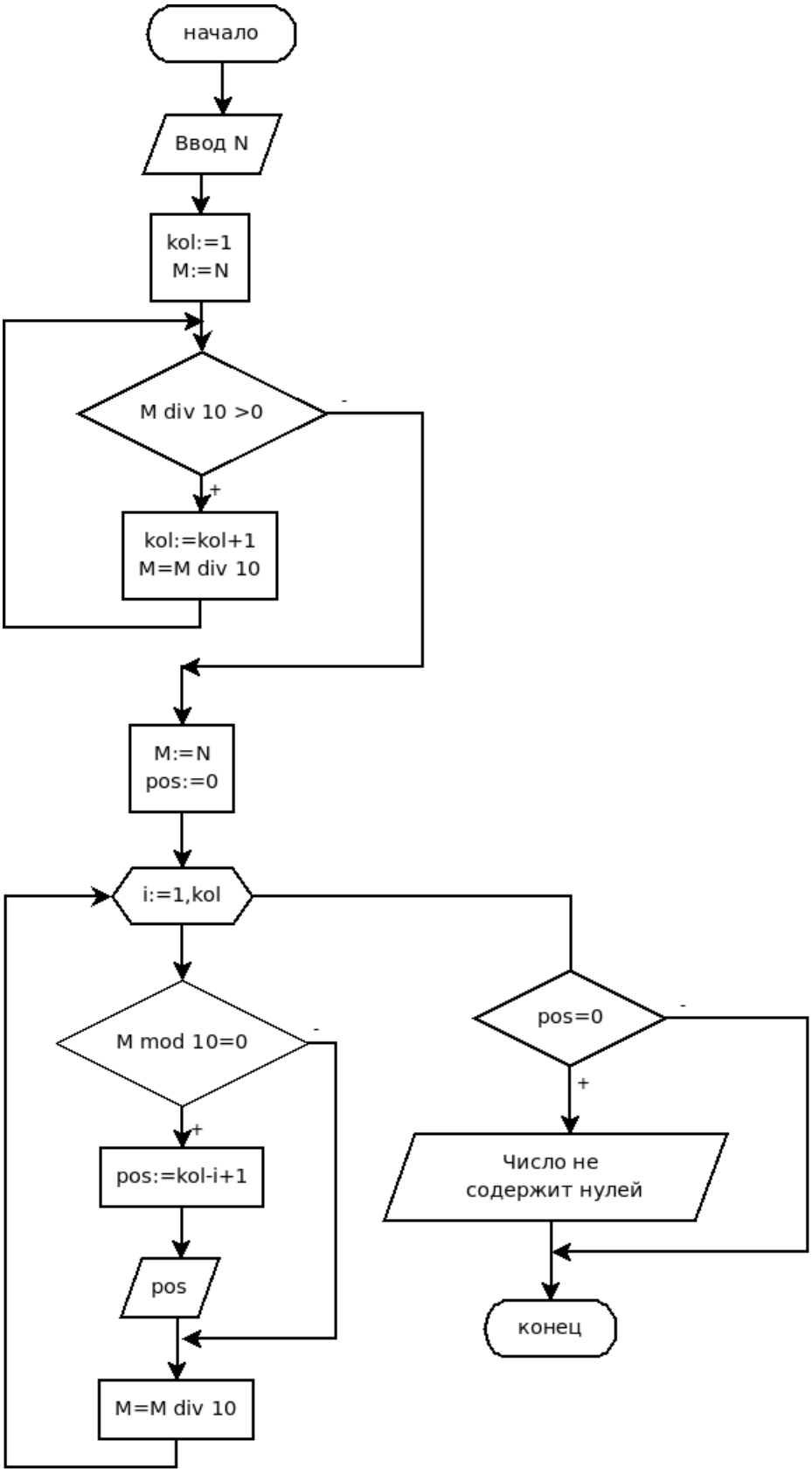

Рисунок 3.37: Алгоритм решения задачи 3.17

```
    for i:=1 to kol do
    begin
    {Выделение цифры из числа и сравнение ее с 0.}
        if (M mod 10 = 0) then
        begin
            pos:=kol-i+1;  {Позиция нуля в числе.}
            writeln('Ноль в ', pos, '-м разряде.');
        end;
        M:=M div 10;       {Изменение числа.}
    end;
    if pos=0 then
        writeln('Число не содержит цифру 0.');
end.
```

3.9 Ввод данных из диалогового окна в среде Lazarus

Окно ввода – это стандартное диалоговое окно, которое появляется на экране в результате вызова функции `InputBox`. В общем виде оператор ввода данных с использованием этой функции записывают так:

```
имя:=InputBox(заголовок_окна,подсказка,значение);
```

где

- `заголовок_окна` – строка, определяющая название окна;
- `подсказка` – текст поясняющего сообщения;
- `значение` – строка, которая будет находиться в поле ввода при появлении окна на экране;
- `имя` — переменная строкового типа, которой будет присвоено `значение` из поля ввода;

После выполнения фрагмента программы

```
var
 S:string;
begin
 S:=InputBox('ЗАГОЛОВОК ОКНА',
 'Подсказка: введите исходные данные',
                    'Данное значение');
end;
```

появится окно, представленное на рис. 3.38. У пользователя есть возможность изменять текст в поле ввода. Щелчок по кнопке **ОК** приве-

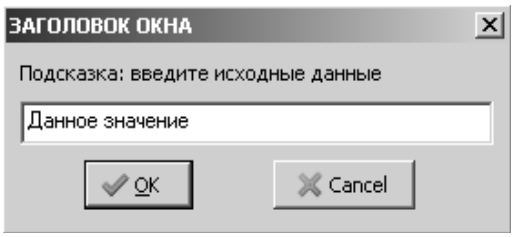

Рисунок 3.38: Окно ввода

дет к тому, что в переменную, указанную слева от оператора присваивания, будет занесена строка, находящаяся в поле ввода. В данном случае в переменную `S` будет записана строка `'Данное значение'`. Щелчок по кнопке **Cancel** закроет окно ввода.

Учитывая, что функция `InputBox` возвращает строковое значение, при вводе числовых данных применяют функции преобразования типов:

```
var S:string; gradus,radian:real;
begin
 S:=InputBox('Ввод данных',
 'Введите величину угла в радианах','0,000');
 gradus:=StrToFloat(S);
 radian:=gradus*pi/180;
 MessageDlg('Величина угла в градусах'
 +FloatToStr(radian),MtInformation,[mbOk],0);
end;
```

Можно применять диалоговое окно при решении задач, обрабатывающих некоторые *числовые последовательности*. Рассмотрим несколько таких задач.

ЗАДАЧА 3.18. Поступает последовательность из N вещественных чисел. Определить наибольший элемент последовательности.

Входные данные: `N` – целое число; `X` – вещественное число, определяет текущий элемент последовательности.

Выходные данные: `Max` – вещественное число, элемент последовательности с наибольшим значением.

Промежуточные переменные: `i` – параметр цикла, номер вводимого элемента последовательности.

Алгоритм поиска наибольшего элемента в последовательности следующий (рис. 3.39). В памяти компьютера отводится ячейка, например с именем `Max`, в которой будет храниться *наибольший элемент* последовательности – *максимум*.

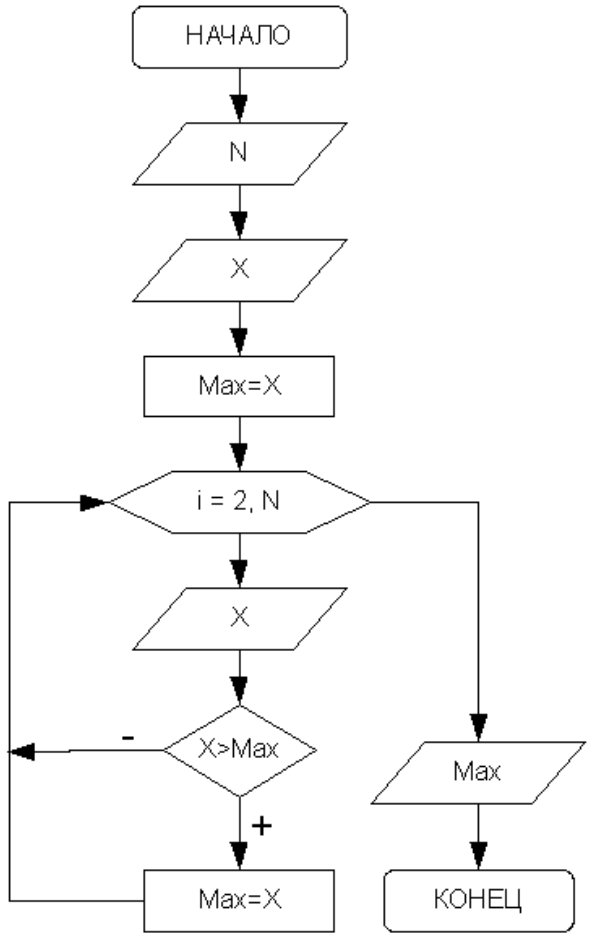

Рисунок 3.39: Алгоритм поиска наибольшего числа в последовательности

Вводим количество элементов последовательности и первый элемент последовательности. Предполагаем, что первый элемент последовательности наибольший и записываем его в `Max`. Затем вводится второй элемент последовательности и сравнивается с предполагаемым максимумом. Если окажется, что второй элемент больше, его записываем в ячейку `Max`. В противном случае никаких действий не предпринимает. Потом переходим к вводу следующего элемента последовательности, и алгоритм повторяется с начала. В результате в ячейке `Max` будет храниться элемент последовательности с наибольшим значением[48].

Разместим на форме объект типа надпись `Label1` и кнопку `Button1` (рис. 3.40).

Щелчок по кнопке приведет к выполнению приведенной далее процедуры.

```
procedure Tform1.Button1Click(
      Sender: TObject);
var
 i,N:integer;
 max,X:real;
 S:string;
begin
//Ввод количества элементов
//последовательности.
S:=InputBox('Ввод',
```

48 Для поиска *наименьшего элемента* последовательности (*минимума*) предполагают, что первый элемент – наименьший, записывают его в ячейку min, а затем среди элементов последовательности ищут число, значение которого будет меньше, чем предполагаемый минимум.

```
'Введите количество элементов в последователь-
ности.','0');
N:=StrToFloat(S);
//Ввод первого элемента последовательности.
    S:=InputBox('Ввод элементов
        последовательности','Введите число.','0');
    X:=StrToFloat(S);
    //Предположим, что 1-й элемент максимальный.
    max:=X;
    //Параметр цикла принимает стартовое
   //значение i=2, т.к. первый элемент уже введен.
    for i:=2 to N do
    begin
    //Ввод следующих элементов последовательности.
        S:=InputBox('Ввод элементов
        последовательности','Введите число.','0');
        X:=StrToInt(S);
        //Если найдется элемент превышающий
        //максимум, записать его в ячейку Max -
        //теперь он предполагаемый максимум.
        if X>max then max:=X;
    end;
    //Вывод наибольшего элемента.
    MessageDlg('Значение наибольшего элемента - '
    +FloatToStr(max),MtInformation,[mbOk],0);
   end;
```

Результаты работы программы представлены на рис. 3.40 - 3.42.

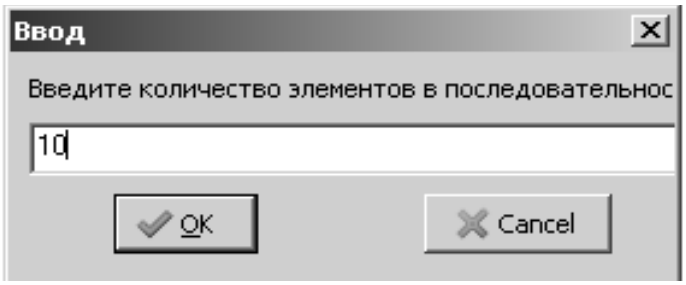

Рисунок 3.40: Второе окно диалога к задаче 3.18

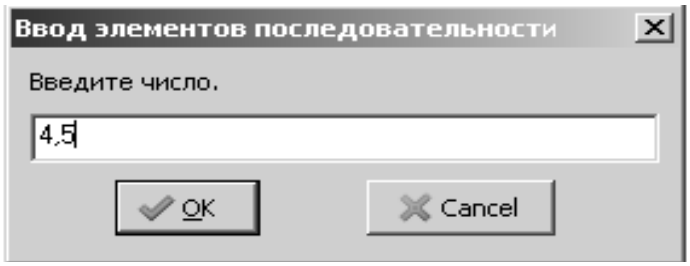

Рисунок 3.41: Третье окно диалога к задаче 3.18

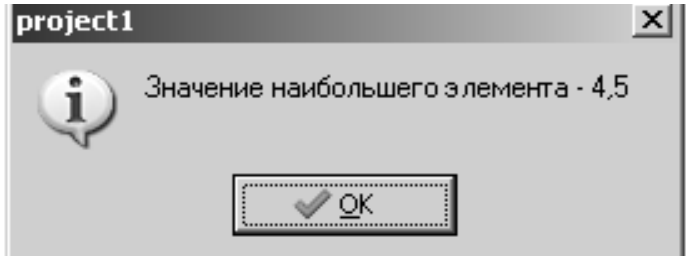

Рисунок 3.42: Результат работы программы к задаче 3.18

ЗАДАЧА 3.19. Вводится последовательность целых чисел, 0 – конец последовательности. Найти наименьшее число среди положительных, если таких значений несколько[49], определить, сколько их.

Блок-схема решения задачи приведена на рис. 3.43.

Далее приведен текст подпрограммы с подробными комментариями[50]. Подпрограмма выполняется при обращении к кнопке `Button1`, предварительно размещенной на форме.

```
procedure TForm1.Button1Click(Sender: TObject);
 var
    N,k,min:integer;
    S:string;
 begin
 //Ввод первого элемента последовательности.
 S:=InputBox('Ввод элементов последовательности', 'введите число. 0 - конец последовательности','0');
 N:=StrToInt(S);
```

49 Предположим, вводится последовательность чисел 11, -3, 5, 12, -7, 5, 8,-9, 7, -6, 10, 5,0.Наименьшим положительным числом является 5. Таких минимумов в последовательности 3.

50 Алгоритм поиска максимального (минимального) элемента последовательности подробно описан в задаче 3.18.

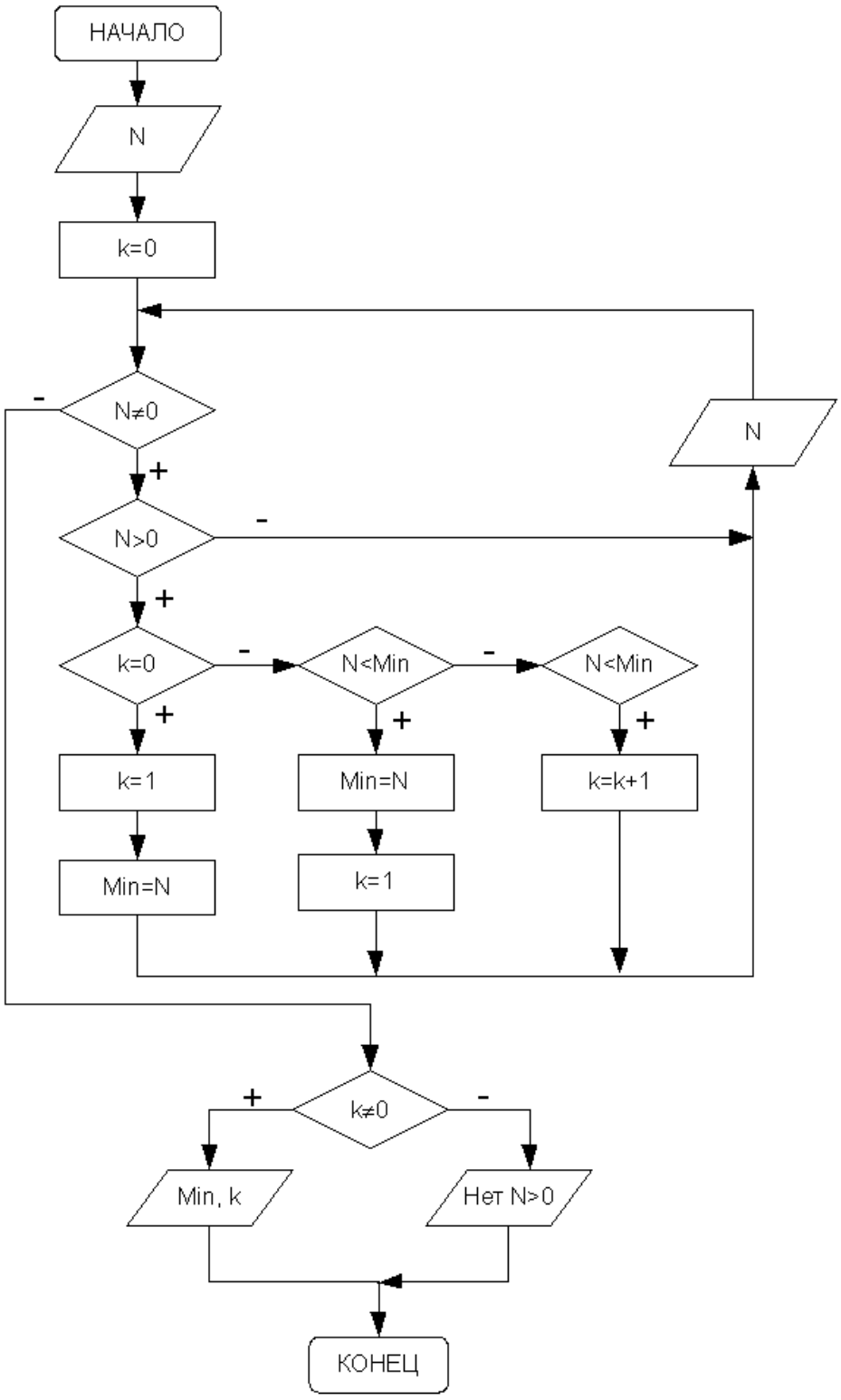

Рисунок 3.43: Алгоритм поиска минимального положительного числа в последовательности

```
//Предположим, что положительных чисел
//нет – К=0. В переменной К будет храниться
//кол-во минимумов среди положительных чисел.
 k:=0;
```

```
    //Пока введенное число не равно нулю,
   //выполнять тело цикла.
    while N<>0 do
    begin
    //Является ли введенное число положительным?
        if N>0 then
        begin
   //Если N>0 и K=0, поступил 1-й
   //положительный элемент, предположим,
   //что он минимальный Min=N, соответственно
   //количество минимумов равно 1.
            if k=0 then
            begin
                k:=1;
                min:=N;
            end
            //Если элемент не первый, сравниваем
            //его с предполагаемым минимумом, если
            //элемент меньше, записываем его в Min.
            //Соответственно количество
            //минимумов равно 1.
            else if N<min then
                begin
                    min:=N;
                    k:=1;
                end
            //Если элемент равен минимуму,
            //увеличиваем количество минимальных
            //элементов на 1.
            else if N=min then k:=k+1;
        end;
        //Ввод следующего элемента.
        S:=InputBox('Ввод элементов последователь-
ности', 'введите число. 0 - конец последовательно-
сти','0');
        N:=StrToInt(S);
    end;        //конец цикла
    if k<>0 then
```

```
        //Если значение счетчика не равно нулю,
        //выводим значение минимального элемента и
        //количество таких элементов,
        MessageDlg('MIN = '+IntToStr(min)+
        ' K='+IntToStr(k),MtInformation,[mbOk],0)
    else
        //в противном случае сообщаем,
        //что положительных чисел нет.
        MessageDlg('Положительных чисел нет',
                    MtInformation,[mbOk],0);
    end;
```

ЗАДАЧА 3.20. Определить, сколько раз последовательность из N произвольных чисел меняет знак.

Чтобы решить задачу, нужно попарно перемножать элементы последовательности. Если результат произведения пары чисел – отрицательное число, значит, эти числа имеют разные знаки.

Пусть `k` — количество смен знака последовательности равно 0. Пусть в переменной `A` храниться текущий элемент последовательности, а в переменой `B` – предыдущий. Введем `N` — количество элементов последовательности. Организуем цикл (переменная `i` меняется от `1` до `N`). В цикле будем делать следующее: вводим очередной элемент последовательности (`A`), если это первый элемент последовательности (`i=1`), то сравнивать его не с чем, и просто переписываем переменную `A` в переменную `B` (`B:=A`). Если это не первый элемент последовательности (`i≠1`), то проверяем знак произведения $A \cdot B$ (текущего и предыдущего элементов последовательности). Если произведение `<0`, то счетчик `k` увеличиваем на `1`. После чего не забываем в переменную `B` записать `A`.

Блок-схема алгоритма приведена на рис. 3.44.

Разметим на форме два объекта типа надпись `Label1` и `Label2`, объект поле ввода `Edit1` и кнопку `Button1` (рис. 3.45). Текст подпрограммы, которая будет выполнена при обращении к кнопке, приведен далее.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    N,A,B,i,k:integer;
    S:string;
```

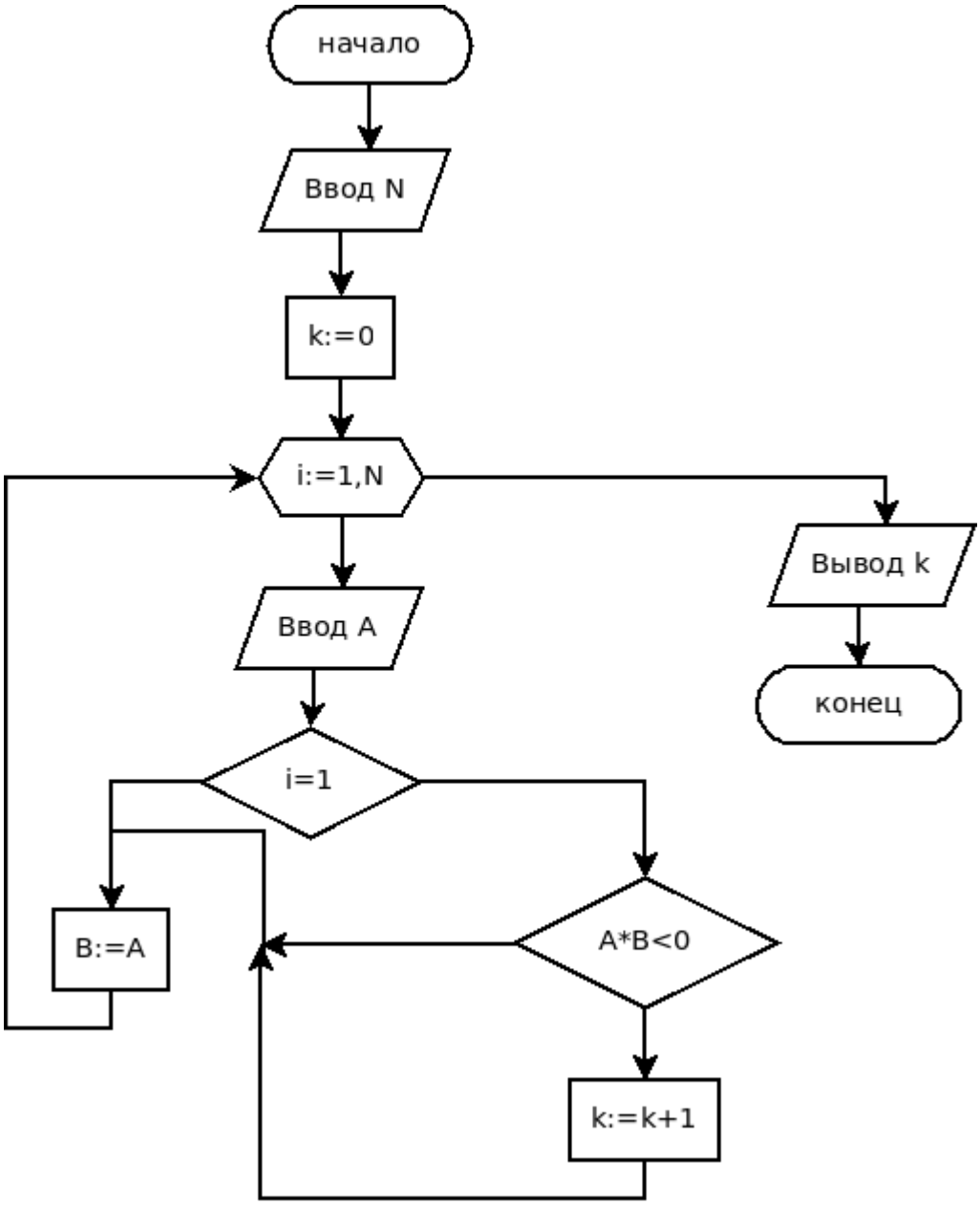

Рисунок 3.44: Алгоритм к задаче 3.20

```
   begin
    N:=StrToInt(Edit1.Text);
    S:=InputBox('Ввод элементов последователь-
ности', 'Введите число. 0 - конец последователь-
ности','0');
    A:=StrToInt(S);
    k:=0;
    for i:=1 to N do
    begin
        S:=InputBox('Ввод элементов последователь-
ности', 'введите число. 0 -конец последователь-
```

```
сти','0');
        A:=StrToInt(S);
        if (i<>1) then
        if A*B<0 then
                k:=k+1;
        B:=A;
    end;
    MessageDlg('K = '+IntToStr(k),
                    MtInformation,[mbOk],0);
end;
```

Результаты работы программы представлены на рис. 3.46-3.47.

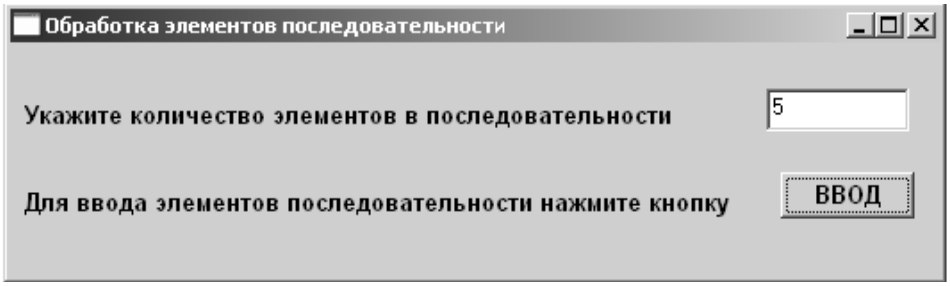

Рисунок 3.45: Первое окно диалога к задаче 3.19

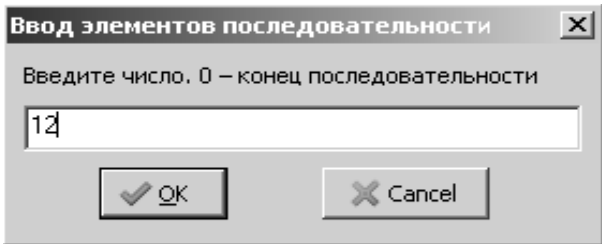

Рисунок 3.46: Второе окно диалога к задаче 3.19

Рисунок 3.47: Результат работы программы к задаче 3.19

3.10 Задачи для самостоятельного решения

Изобразите блок-схему решения задачи и напишите программу.

3.10.1 Разветвляющийся процесс

Дано вещественное число *a*. Для функции *y=f*(*x*), график которой приведен ниже, вычислить *f*(*a*). Варианты заданий представлены на рис. 3.48 -3.55.

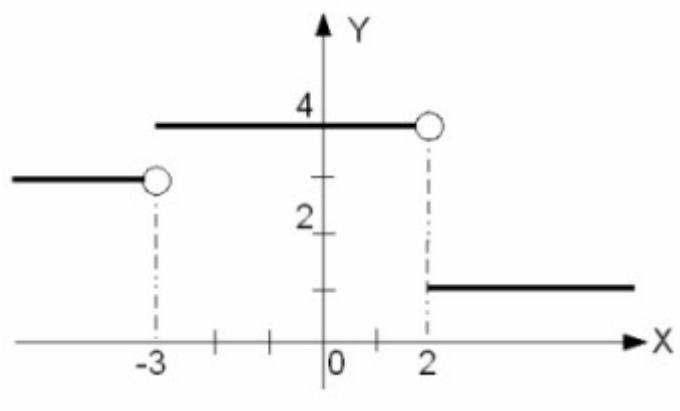

Рисунок 3.48: Задание 1

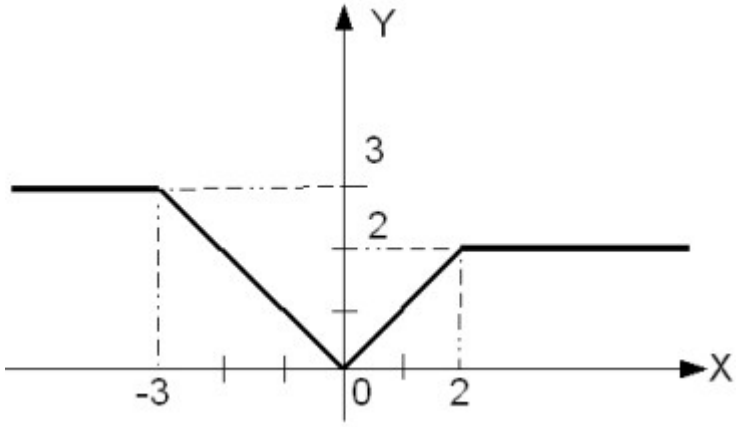

Рисунок 3.49: Задание 2

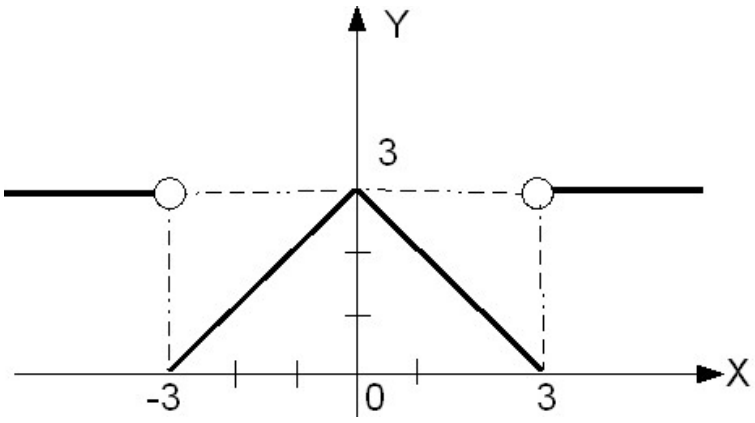

Рисунок 3.50: Задание 3

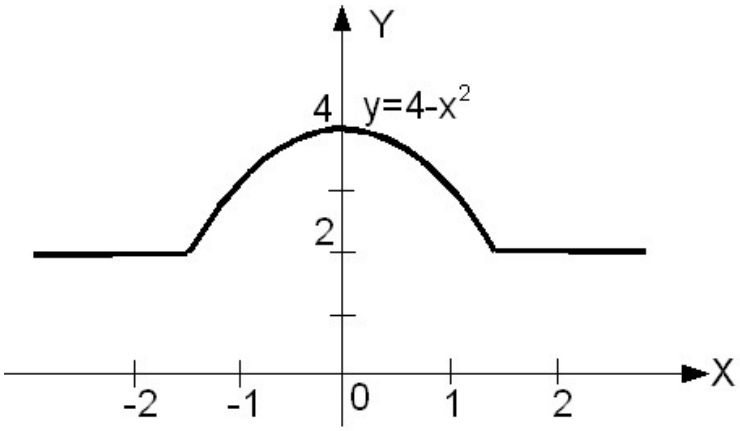

Рисунок 3.51: Задание 4

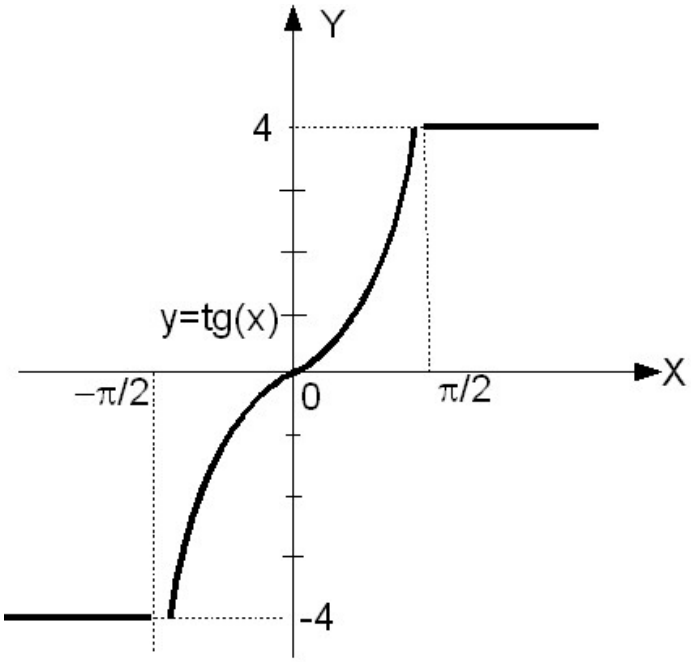

Рисунок 3.52: Задание 5

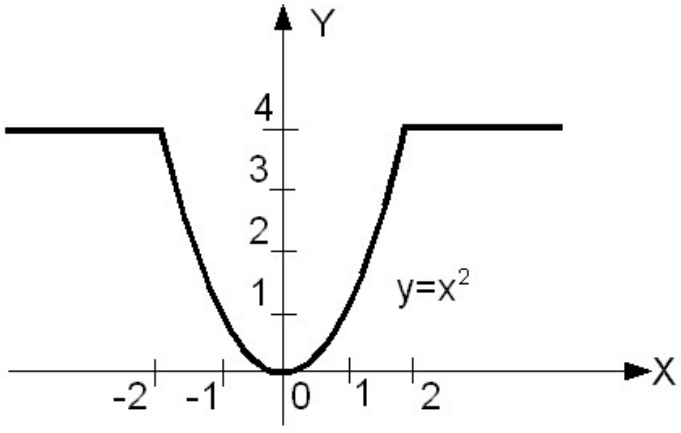

Рисунок 3.53: Задание 6

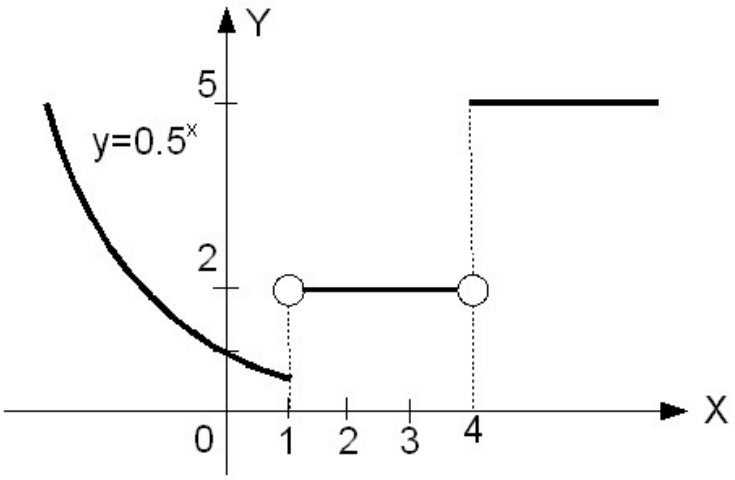

Рисунок 3.54: Задание 7

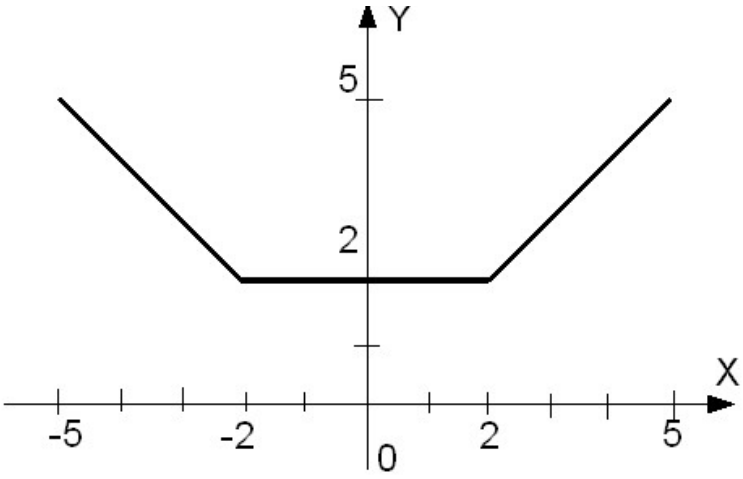

Рисунок 3.55: Задание 8

Даны вещественные числа *x* и *y*. Определить принадлежит ли точка с координатами (*x*; *y*) заштрихованной части плоскости. Варианты заданий представлены на рис. 3.56 - 3.63.

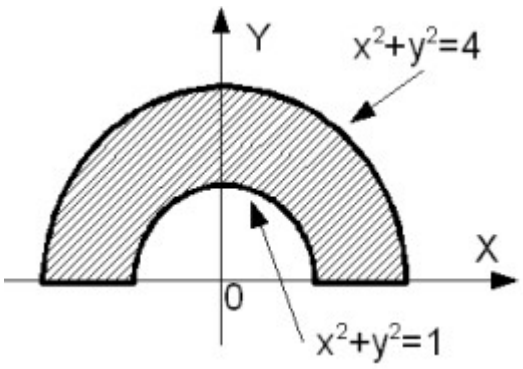

Рисунок 3.56: Задание 9

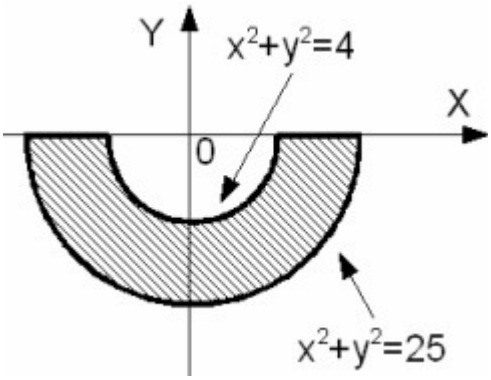

Рисунок 3.57: Задание 10

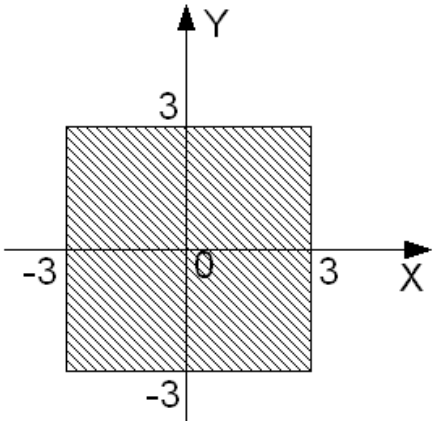

Рисунок 3.58: Задание 11

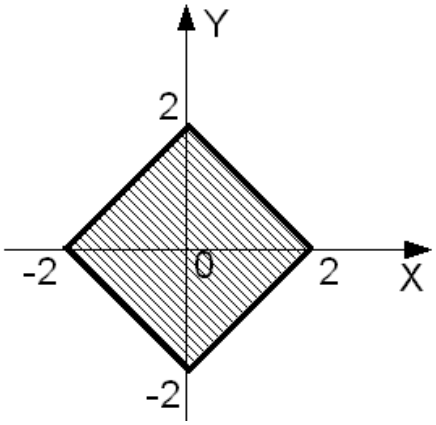

Рисунок 3.59: Задание 12

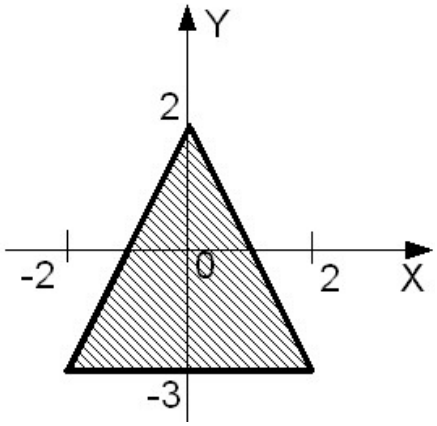

Рисунок 3.60: Задание 13

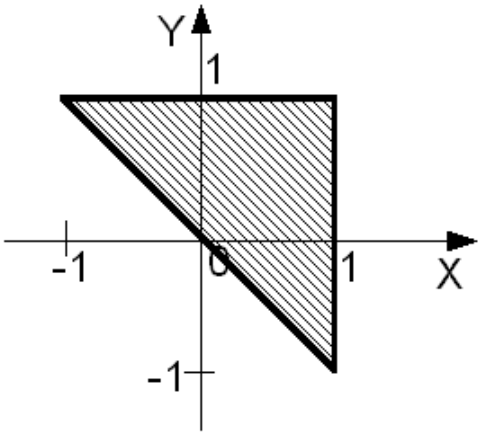

Рисунок 3.61: Задание 14

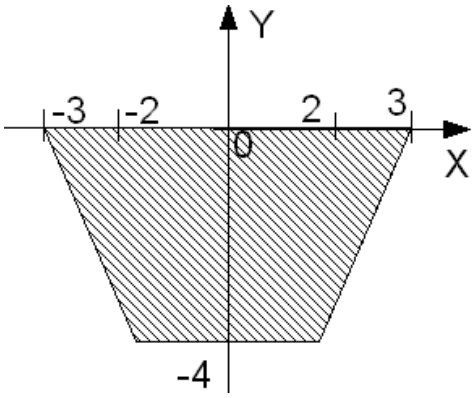

Рисунок 3.62: Задание 15

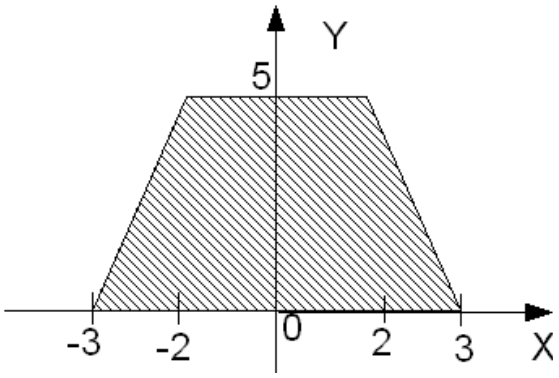

Рисунок 3.63: Задание 16

Решить следующие задачи:

17. Задан круг с центром в точке $O(x_0, y_0)$ и радиусом R_0 и точка $A(x_1, y_1)$. Определить, находится ли точка внутри круга.

18. Определить, пересекаются ли параболы $y=ax^2+bx+c$ и $y=dx^2+mx+n$. Если пересекаются, то найти точку пересечения.

19. Определить, пересекаются ли линии $y=ax^3+bx^2+cx+d$ и $y=kx^3+mx^2+nx+p$. Если пересекаются, найти точку пересечения.

20. Задана окружность с центром в точке $O(x_0, y_0)$ и радиусом R_0, найти точки пересечения линии с осью абсцисс.

21. Задана окружность с центром в точке $O(x_0, y_0)$ и радиусом R_0, найти точки пересечения линии с осью ординат.

22. Определить, пересекаются ли линии $y=bx^2+cx+d$ и $y=kx+m$. Если пересекаются, найти точки пересечения.

23. Задана окружность с центром в точке $O(0,0)$ и радиусом R_0 и прямая $y=ax+b$. Определить, пересекаются ли прямая и окружность. Если пересекаются, найти точку пересечения.

24. Найти точки пересечения линии $y= ax^2+bx+c$ с осью абсцисс.

25. Определить, пересекаются ли линии $y=ax^4+bx^3+cx^2+dx+f$ и $y=bx^3+mx^2+dx+p$. Если пересекаются, найти точку пересечения.

3.10.2 Циклический процесс

1. Вычислить сумму натуральных нечетных чисел, не превышающих N.

2. Вычислить произведение натуральных четных чисел, не превышающих N.

3. Вычислить количество натуральных чисел, кратных трем и не превышающих N.

4. Задано число n. Определить значение выражения:

$$P = \frac{n!}{\sum_{i=1}^{n} i}.$$

5. Вводится последовательность ненулевых чисел, 0 – конец последовательности. Определить сумму положительных элементов последовательности.

6. Вводится последовательность ненулевых чисел, 0 – конец последовательности. Определить, сколько раз последовательность поменяет знак.

7. Вычислить сумму отрицательных элементов последовательности из *N* произвольных чисел.

8. В последовательности из *N* произвольных чисел подсчитать количество нулей.

9. Вводится последовательность ненулевых чисел, 0 – конец последовательности. Определить наибольшее число в последовательности.

10. Дано натуральное число *P*. Определить все простые числа, не превосходящие *P*.

11. Определить, является ли число *L* совершенным. Совершенное число L равно сумме всех своих делителей, не превосходящих *L*. Например, 6=1+2+3 или 28=1+2+4+7+14. В основе решения задачи лежит алгоритм из задачи 3.13.

12. Вводится последовательность ненулевых чисел, 0 – конец последовательности. Определить среднее значение элементов последовательности.

13. Вводится последовательность из *N* произвольных чисел, найти наименьшее положительное число.

14. Вводится последовательность из *N* произвольных чисел, найти среднее значение положительных элементов последовательности.

15. Вводится последовательность ненулевых чисел, 0 – конец последовательности. Подсчитать процент положительных и отрицательных чисел.

16. Вводится последовательность из *N* произвольных чисел. Определить процент положительных, отрицательных и нулевых элементов.

17. Вводится последовательность положительных целых чисел, 0 – конец последовательности. Определить количество совершенных чисел (см. вариант 11).

18. Вводится последовательность из *N* произвольных чисел. Вычислить разность между наименьшим и наибольшим значениями последовательности.

19. Дано натуральное число *P*. Определить все совершенные числа (см. вариант 11), не превосходящие *P*.

20. Вводится последовательность из *N* положительных целых чисел. Найти наименьшее число среди четных элементов последовательности.

21. Вводится последовательность положительных целых чисел, 0 – конец последовательности. Определить, является ли эта последовательность знакочередующейся.

22. Задано число *P*. Если это число простое вычислить *P*!.

23. Вводится последовательность из *N* произвольных чисел. Найти наибольшее число в последовательности. Если таких чисел несколько, определить, сколько их.

24. Задано число *P*. Определить количество его четных и нечетных делителей.

25. Определить, является ли последовательность из *N* произвольных чисел строго возрастающей (то есть каждый следующий элемент больше предыдущего).