Лекция 3. Операторы управления

Для организации в языке С(С++) используются операторы if и switch.

3.1. Условный оператор

Оператор *if* имеет следующую структуру

if (условие) оператор_1; else оператор_2;

где **условие** - логическое выражение, переменная или константа.

Работает условный оператор следующим образом. Если условие оно не равно нулю, т.е. имеет значение истина (true), выполняется оператор_1. В противном случае, когда выражение равно нулю (ложь - false), то оператор _2. Алгоритм , который реализован в условном операторе if, представлен на рис. 3.1.

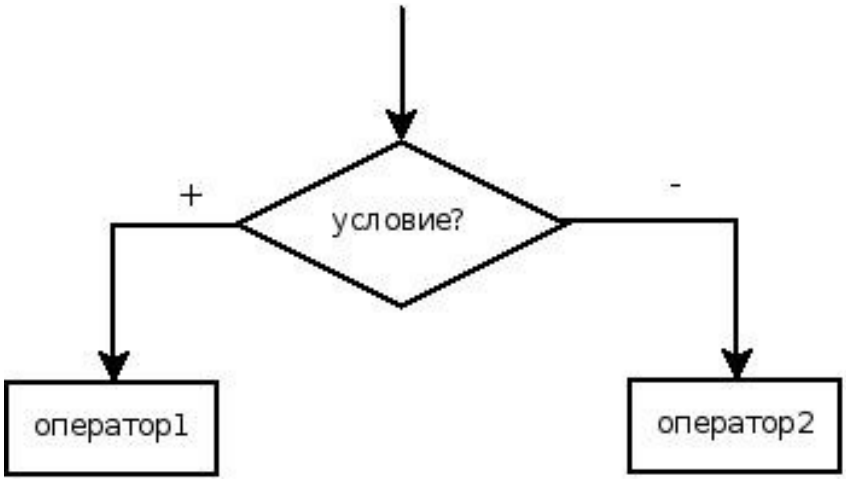


Рисунок 3.1: Алгоритм условного оператора if... else

Составным оператором языка С(С++) называется группа операторов языка, начинающаяся с символа «{» и заканчивающаяся символом «}».

```
{
оператор_1;
...
оператор_n;
}
```

Транслятор воспринимает составной оператор как одно целое.

Таким образом, если в операторе if требуется, чтобы в зависимости от значения условия выполнялся не один оператор, а несколько, то оператор if следует записывать в следующей форме.

```
if (условие)
{
оператор_1;
оператор_2;
...
}
else
{
оператор_1;
оператор_2;
...
}
```

В операторе if могут отсутствовать альтернативная ветвь else, в этом случае оператор if можно записывать так

if (условие) оператор;

или так

if (условие)
{
оператор_1;
оператор_2;
...
}

Использование оператора оператора if рассмотрим на примере решения следующих задач.

ЗАДАЧА 3.1. Написать программу решения квадратного уравнения $ax^2 + bx + c = 0$.

Исходные данные: a, b и c.

Результаты работы программы: x_1 и x_2 корни квадратного уравнения или сообщение о том, что корней нет.

Вспомогательные переменные: вещественная переменная d.

Блок схема решения задач представлена на рис. 3.2.

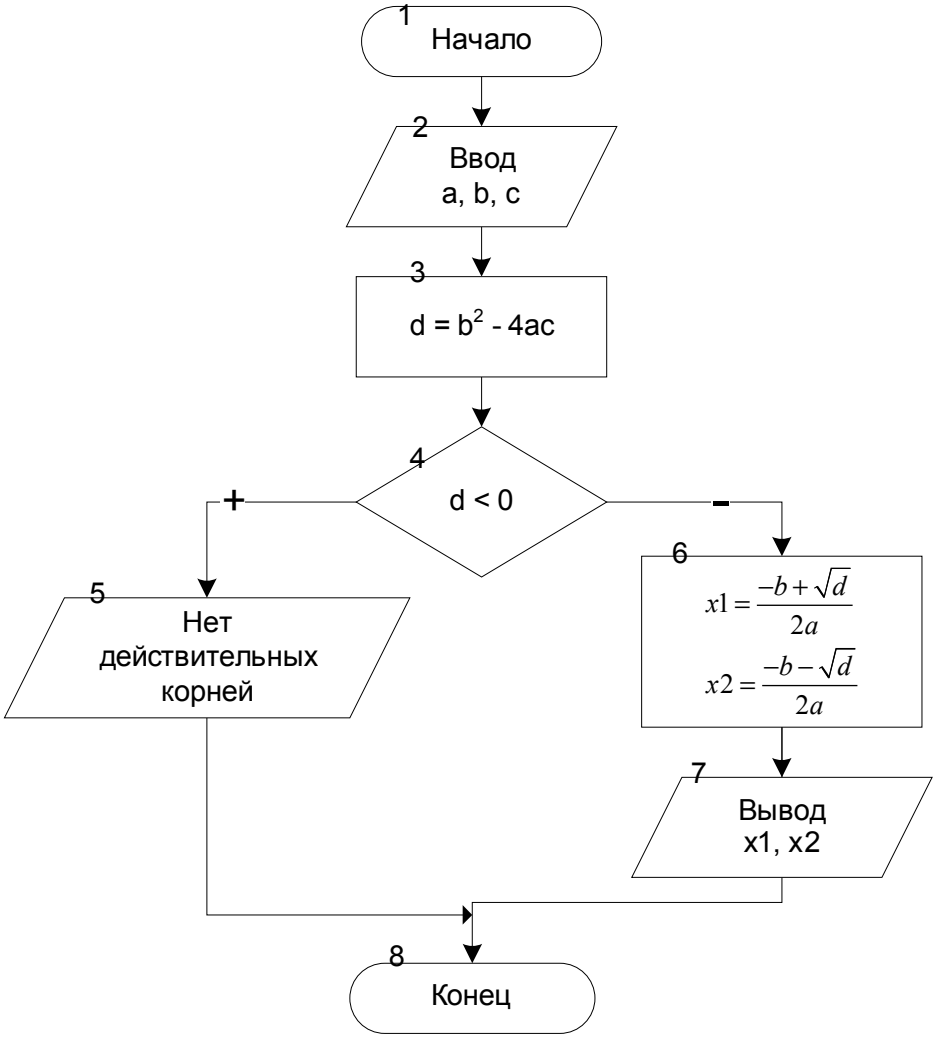


Рисунок 3.2: Блок-схема решения задачи 3.1

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{float a,b,c,d,x1,x2;
//Ввод значений коэффициентов квадратного уравнения
```

```
cout<<"a=";cin>>a;
cout<<"b=";cin>>b;
cout<<"c=";cin>>c;
//Вычисление дискриминанта
d=b*b-4*a*c;
//Если дискриминант отрицателен,
if (d<0)
//то вывод сообщения, что корней нет,
cout<<"Real roots are not present";
else
{//иначе вычисление корней x1, x2
   x1=(-b+sqrt(d))/2/a;
   x2=(-b-sqrt(d))/(2*a);
//и вывод их значений на экран
   cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
}
return 0;
}
```

ЗАДАЧА 3.2. Составить программу нахождения действительных и комплексных корней квадратного уравнения $ax^2 + bx + c = 0$.

Исходные данные: вещественные числа a, b и c.

Результаты работы программы: вещественные числа x_1 и x_2 действительные корни квадратного уравнения, либо x1 и x2 действительная и мнимая части комплексного числа .

Вспомогательные переменные: d.

Блок схема решения задачи представлена на рис. 3.3.

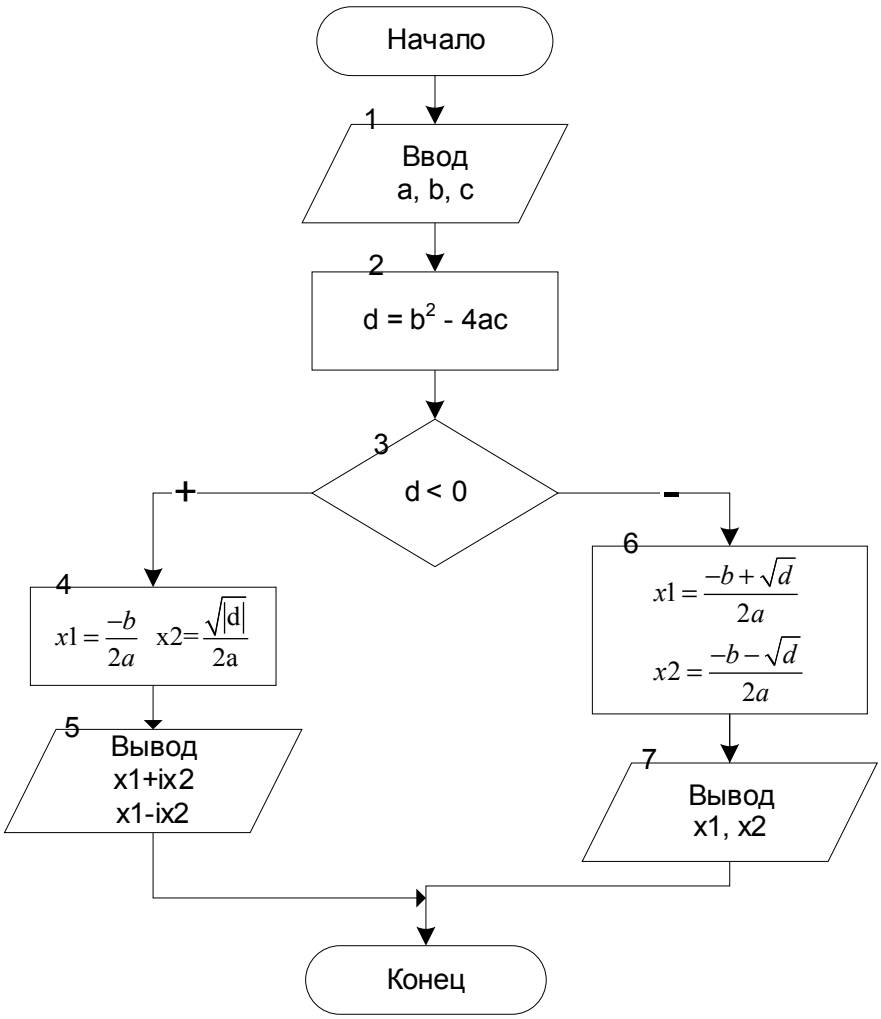


Рисунок 3.3: Блок-схема решения задачи 3.2

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
float a,b,c,d,x1,x2;
cout<<"a=";cin>>a;
cout<<"b=";cin>>b;
cout<<"c=";cin>>c;
d=b*b-4*a*c;
//Проверка знака дискриминанта
if (d<0)
{
//Если дискриминант отрицателен, то вывод сообщения
cout<<"Real roots are not present \n";
//Вычисление действительной части комплексных корней
x1=-b/(2*a);
//Вычисление модуля мнимой части комплексных корней
x2=sqrt(fabs(d))/(2*a);
//Сообщение о комплексных корнях уравнения вида ax^2+bx+c=0
cout<<"Complex roots of equalization \n";
cout<<a<<"x^2+"<<b<<"x+"<<c<<"=0 \n";
//Вывод значений комплексных корней в
// виде x1±ix2
cout<<x1<<"+"<<x2<<"i\t";
cout<<x1<<"-"<<x2<<"i\n";
}
else
{
//иначе вычисление действительных корней x1, x2
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/(2*a);
//и вывод их на экран
cout<<"Real roots of equalization \n";
cout<<a<<"x^2+"<<b<<"x+"<<c<<"=0 \n";
cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
}
return 0;
}
```

ЗАДАЧА 3.3. Составить программу для решения кубического уравнения $ax^3 + bx^2 + cx + d = 0$.

Кубическое уравнение имеет вид

$$ax^3 + bx^2 + cx + d = 0 \qquad (3.1)$$

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

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

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 (3.3)

$$y^3 + py + q = 0, \qquad (3.3)$$

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

Число действительных корней приведенного уравнения (3.3) зависит от знака

дискриминанта $D=\frac{p^3}{3}+\frac{q^2}{2}$ (см. табл. 3.1).

Таблица 3.1. Количество корней кубического уравнения

Дискриминант	Количество действительных корней	Количество комплексных корней
D≥0	1	2
D<0	3	-

Корни приведенного уравнения могут быть рассчитаны по формулам Кардано (3.4).

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

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 следующими формулами (3.5):

$$\begin{aligned} y_1 &= 2\sqrt[3]{\rho}\cos\left(\frac{\varphi}{3}\right) \\ y_2 &= 2\sqrt[3]{\rho}\cos\left(\frac{\varphi}{3}\right)+\frac{2\pi}{3} \\ y_4 &= 2\sqrt[3]{\rho}\cos\left(\frac{\varphi}{3}\right)+\frac{4\pi}{3} \end{aligned} \quad \text{, где } \rho=\sqrt{\frac{-p^3}{27}},\ \cos(\varphi)=-\frac{q}{2\rho}\ . \qquad (3.5)$$

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. 3.12.

```
#include <iostream>
#include <math.h>
#define pi 3.14159
using namespace std;
int main()
{
float a,b,c,d,D,r,s,t,p,q,ro,fi,x1,x2,x3,u,v,h,g;
cout<<"a="; cin>>a;
cout<<"b="; cin>>b;
cout<<"c="; cin>>c;
cout<<"d="; cin>>d;
//Расчет коэффициентов канонического уравнения по формуле (3.2)
r=b/a; s=c/a; t=d/a;
//Вычисление коэффициентов приведенного уравнения по формуле (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)*(p/3)*(p/3)+(q/2)*(q/2);
```

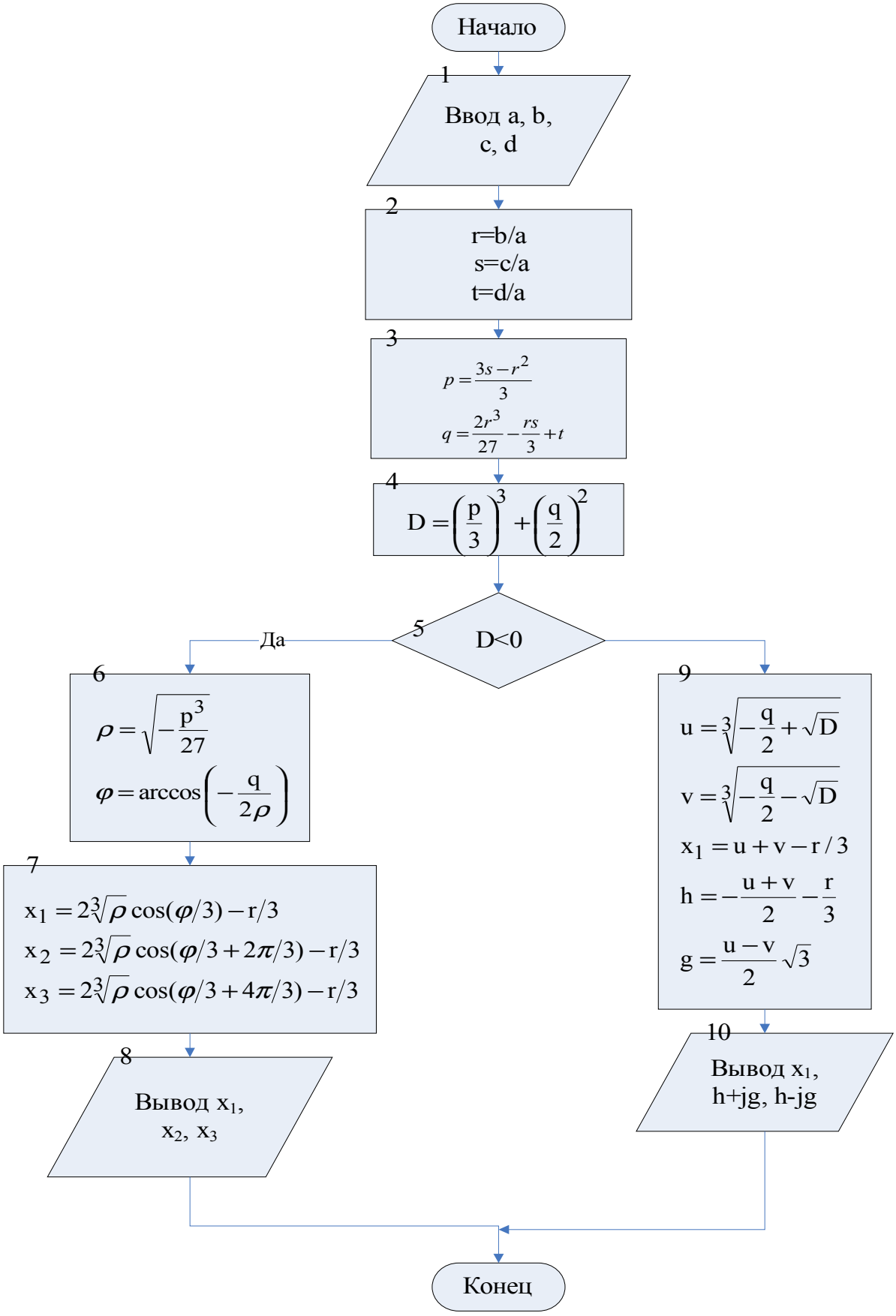


Рисунок 3.4: Блок-схема решения задачи 3.3

```
if (D<0)
//Формулы (3.5)
{
ro=sqrt((float)(-p*p*p/27));
```

```
  fi=-q/(2*ro);
  fi=pi/2-atan(fi/sqrt(1-fi*fi));
  x1=2*pow(ro,(float)1/3)*cos(fi/3)-r/3;
  x2=2*pow(ro,(float)1/3)*
  cos(fi/3+2*pi/3)-r/3;
  x3=2*pow(ro,(float)1/3)*
  cos(fi/3+4*pi/3)-r/3;
  cout<<"\n x1="<<x1<<"\t x2="<<x2<<"\t x3="<<x3<<"\n";
  }
  else
  //Формулы (3.4)
  {
  //К выражению 1/3 необходимо применить операцию преобразования типа
  //(float)1/3,иначе будет выполнена операция целочисленного деления,
  //результат которой равен 0.
  if (-q/2+sqrt(D)>0)
  u=pow((-q/2+sqrt(D)),(float)1/3);
  else
  if (-q/2+sqrt(D)<0)
  u=-pow(fabs(-q/2+sqrt(D)),(float)1/3);
  else u=0;
  if (-q/2-sqrt(D)>0)
  v=pow((-q/2-sqrt(D)),(float)1/3);
  else
  if (-q/2-sqrt(D)<0)
  v=-pow(fabs(-q/2-sqrt(D)),(float)1/3);
  else v=0;
  //Вычисление действительного корня кубического уравнения
  x1=u+v-r/3;
  //Вычисление действительной и мнимой части комплексных
корней
  h=-(u+v)/2-r/3;
  g=(u-v)/2*sqrt((float)3);
  cout<<"\n x1="<<x1;
  cout<<"\t x2="<<h<<"+"<<g<<"i \t x3="<<h<<"-"<<g<<"i \n";
  }
  return 0;
  }
```

ЗАДАЧА 3.4. 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: x_1, x_2, x_3, x_4.

Блок-схема решения задачи приведена на рис. 3.5.

```
  #include <iostream.h>
  #include <math.h>
  int main()
  {
  float a,b,c,d,x1,x2,x3,x4,y1,y2;
  //Ввод коэффициентов уравнения
  cout<<"a="; cin>>a;
  cout<<"b="; cin>>b;
  cout<<"c="; cin>>c;
  //Вычисление дискриминанта
  d=b*b-4*a*c;
  //Если дискриминант <0
  if (d<0)
```

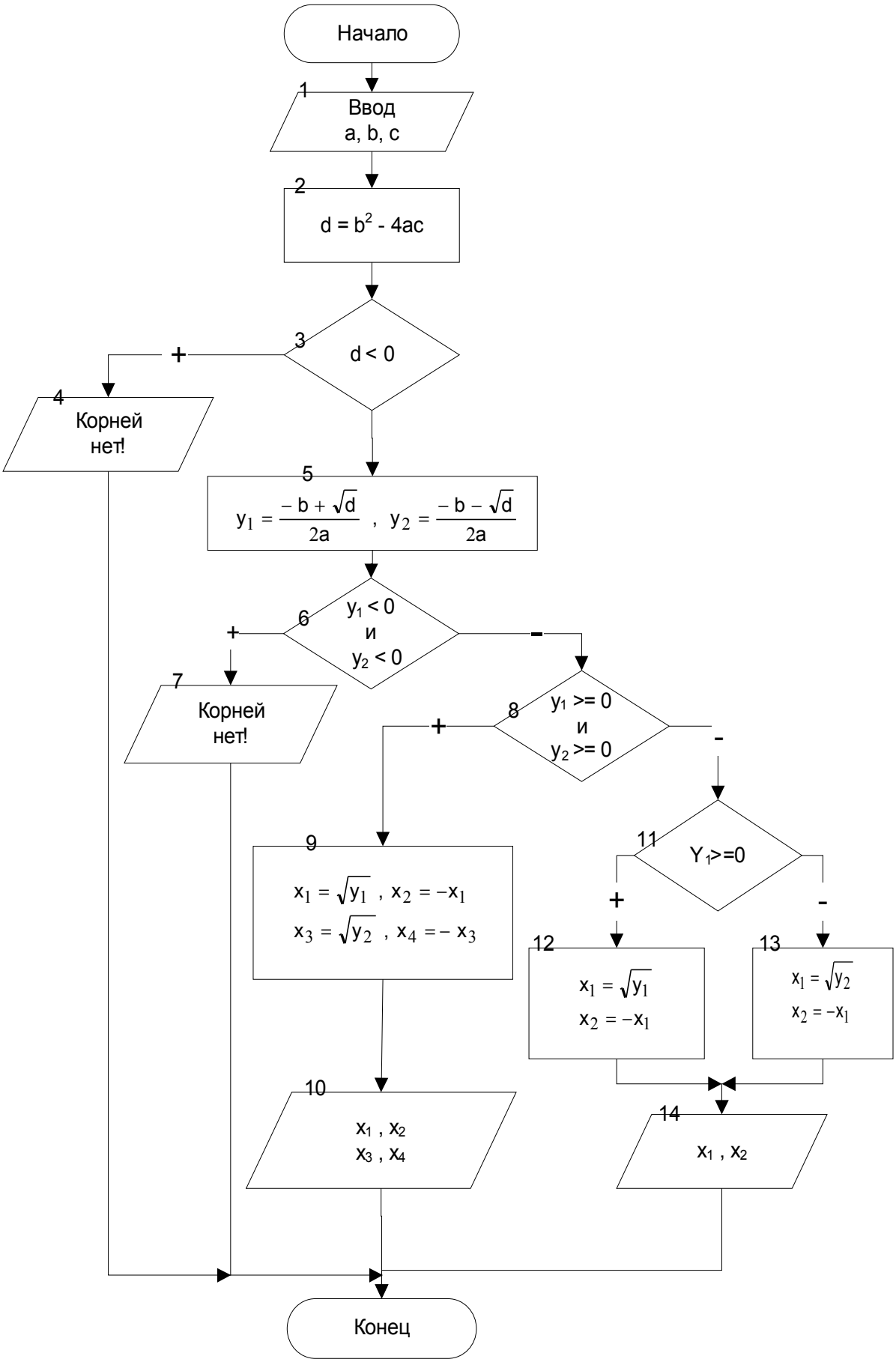


Рисунок 3.5: Блок-схема решения задачи 3.4

```
//Вывод сообщения «Корней нет»
cout<<"Real roots are not present \n";
```

```
//Если дискриминант ≥0
else
{
//Вычисление корней соответствующего квадратного уравнения
y1=(-b+sqrt(d))/2/a;
y2=(-b-sqrt(d))/(2*a);
//Если оба корня квадратного уравнения <0
if (y1<0 && y2<0)
//Вывод сообщения «Корней нет»
cout<<"Real roots are not present \n";
//Если оба корня квадратного уравнения ≥0
else if (y1>=0 && y2>=0)
{
//Вычисление четырех корней биквадратного уравнения
x1=sqrt(y1);
x2=-x1;
x3=sqrt(y2);
x4=-sqrt(y2);
//Вывод корней биквадратного уравнения на экран
cout<<"X1="<<x1<<"\t X2="<<x2;
cout<<"X3="<<x3<<"\t X4="<<x4<<"\n";
}
//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)
//Если оно истинно
{
//для вычисления корней биквадратного уравнения, извлекаем корни из y1
x1=sqrt(y1);
x2=-x1;
cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
}
else
//Если условие y1>=0 ложно, то
{
//для вычисления корней биквадратного уравнения извлекаем корни из y2
x1=sqrt(y2);
x2=-x1;
cout<<"X1="<<x1<<"\t X2="<<x2<<"\n";
}
}
return 0;
}
```

3.2. Оператор варианта

Оператор варианта **switch** необходим в тех случаях, когда в зависимости от значений какой-либо переменной надо выполнить те или иные операторы:

```
switch (выражение)
{
case значение_1: Операторы_1; break;
case значение_2: Операторы_2; break;
case значение_3: Операторы_3; break;
...
case значение_n: Операторы_n; break;
```

default: Операторы; break;

}

Оператор break необходим для того, чтобы осуществить выход из оператора switch. Если оператор break не указан, то будут выполняться следующие операторы из списка, не смотря на то, что значение, которым они помечены, не совпадает со значением выражения. Его использование рассмотрим на примере решения следующей задачи.

ЗАДАЧА 3.6. По заданному номеру месяца m вывести на печать его название.

```
#include <iostream>
using namespace std;
int main()
{
unsigned int m; cout<<"m="; cin>>m;
switch (m)
{
//В зависимости от значения m выводится название месяца
case 1: cout<<"Январь \n"; break;
case 2: cout<<"Февраль \n"; break;
case 3: cout<<"Март \n"; break;
case 4: cout<<"Апрель \n"; break;
case 5: cout<<"Май \n"; break;
case 6: cout<<"Июнь \n"; break;
case 7: cout<<"Июль \n"; break;
case 8: cout<<"Август \n"; break;
case 9: cout<<"Сентябрь \n"; break;
case 10:cout<<"Октябрь \n"; break;
case 11:cout<<"Ноябрь \n"; break;
case 12:cout<<"Декабрь\n"; break;
//Если значение переменной m выходит за пределы области допустимых
// значений, то выдается сообщение об ошибке
default: cout<<"Ошибка! \n"; break;
}
return 0;
}
```

3.3. Операторы цикла

Циклический процесс (*цикл*) повторение одних и тех же действий. *Тело цикла* последовательность действий, выполняемых в цикле. Переменные, изменяющиеся внутри цикла и влияющие на его окончание, называются *параметрами цикла.*

При написании циклических алгоритмов следует помнить следующее. Для того, чтобы цикл закончился, тела цикла должно обязательно влиять на условие входа в цикл.

В C++ для удобства пользователя предусмотрены три оператора, реализующих циклический процесс: while, do& while и for.

3.3.1. Оператор цикла с предусловием

На рис. 3.5 изображена блок-схема алгоритма *цикла с предусловием.* Оператор, реализующий этот алгоритм в C++, имеет вид

while (выражение) оператор;

или

while условие
{
оператор 1;
оператор 2;
...
оператор n;
}

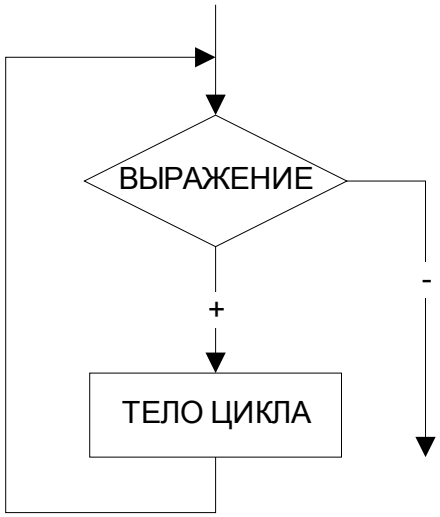
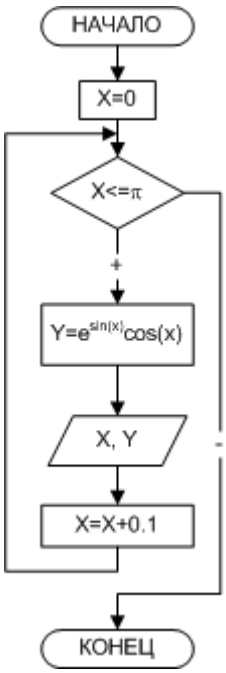


Рисунок 3.6: Блок-схема цикла с предусловием

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



С использование цикла с предусловием программа будет такой:

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159
int main()
{
float x, y;
x=0;
//Цикл с предусловием
while (x<=PI)
//Пока параметр цикла не превышает конечное значение, выполнять тело
// цикла
{
y=exp(sin(x))*cos(x);
printf("x=%f \t y=%f \n",x,y);
x+=0.1; } //Конец цикла
```

```
}
```

3.3.2. Оператор цикла с постусловием

Цикл с постусловием представлен на рис. 3.7

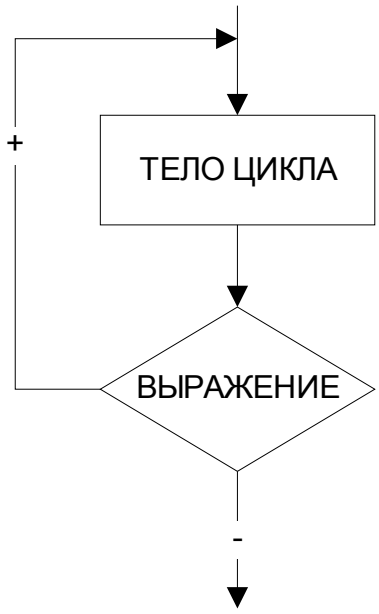


Рисунок 3.7: Цикл с постусловием

Оператор, реализующий этот алгоритм в C++, имеет вид

do оператор while (выражение);

или

do
{
оператор_1;
оператор_2;
...
оператор_n;
}
while (выражение);

Программа, которая выводит таблицу значений функции $y = e^{\sin(x)}\cos(x)$ на отрезке $[0;\pi]$ с шагом 0.1, имеет вид:

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159
int main()
{
float x, y; //Описание переменных
x=0;
do //Цикл с постусловием
{//Выполнять тело цикла
y=exp(sin(x))*cos(x);
printf("x=%f \t y=%f \n",x,y);
x+=0.1; }
while(x<=PI);
return 0;}
```

3.3.3. Оператор цикла с параметром

В C++ существует оператор цикла с параметром следующей структуры

for (начальные_присваивания; выражение; приращение) оператор;

или

for (начальные_присваивания; выражение; приращение)
{
оператор1;
оператор2;
...
}

Алгоритм работы цикла for следующий (см. рис. 3.8)

1. Выполняются начальные_присваивания.
2. Вычисляется значение выражения, если оно не равно 0 (true), то выполняется переход к п. 3. В противном случае выполнение цикла завершается.
3. Выполняется оператор.
4. Управление передается оператору приращение, после чего осуществляется переход к п. 2, то есть опять вычисляется значение выражения и т.д.

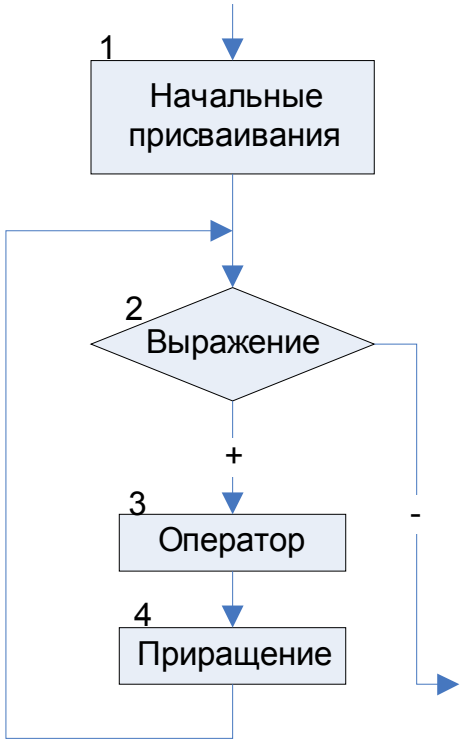


Рисунок 3.8: Алгоритм оператора for

Блок-схему оператора for принято изображать та как показано на рис. 3.9.

Решение задачи вывода таблицы значений функции $y = e^{\sin(x)}\cos(x)$ на отрезке $[0;\pi]$ с шагом 0.1 с помощью оператора for представлено ниже.

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159
int main()
{
float x, y;
//Параметру цикла присваивается начальное значение, если оно не
// превышает конечное значение,то выполняются операторы тела
// цикла,и значение параметра изменяется,в противном случае цикл
// заканчивается
```

Рисунок 3.9: Блок-схема оператора for

```
for (x=0;x<=PI;x+=0.1)
{
y=exp(sin(x))*cos(x);
printf("x=%f \t y=%f \n",x,y);
}
return 0;
}
```

3.3.4. Операторы передачи управления

В C++ есть четыре *оператора передачи управления*, которые принудительно изменяют порядок выполнения команд: goto, break, continue и return.

Оператор

goto метка;

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

метка: оператор;

Оператор **break** осуществляет немедленный выход из циклов while, do& while и for, а также из оператора выбора switch.

Оператор **continue** прерывает выполнение данного шага цикла.

Оператор **return выражение** завершает выполнение функции и передает управление в точку ее вызова. Если функция возвращает значение типа void, то выражение в записи оператора отсутствует. В противном случае выражение должно иметь скалярный тип.

3.4. Решение задач с использованием циклов

ЗАДАЧА 3.7. Найти наибольший общий делитель (НОД) 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 табл. 3.2.

Таблица 3.2. Поиск НОД для чисел A=25 и B=15.

Исходные данные	**Первый шаг**	**Второй шаг**	**Третий шаг**
A=25	A=10	A=10	A=5
B=15	B=15	B=5	B=5

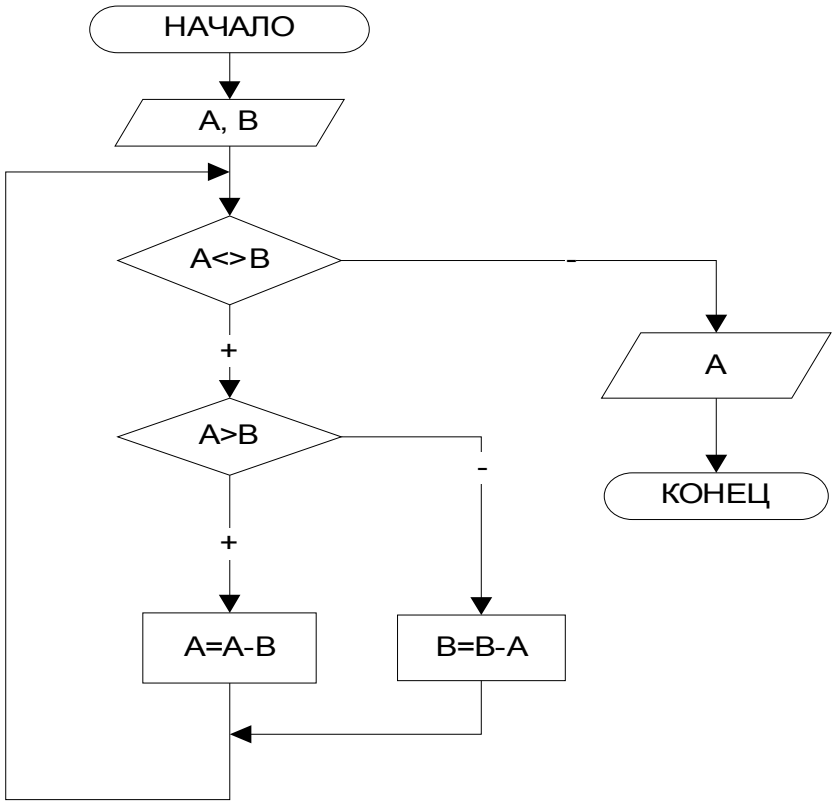


Рисунок 3.10: Поиск наибольшего общего делителя двух чисел

```
#include <iostream>
using namespace std;
int main()
{
unsigned int a,b;
cout<<"A="; cin>>a;
cout<<"B="; cin>>b;
//Если числа не равны, выполнять тело цикла
while (a!=b)
//Если число A больше, чем B, то уменьшить его значение на B,
if (a>b) a=a-b;
//иначе уменьшить значение числа B на A
else b=b-a;
cout<<"NOD="<<a<<"\n";
return 0;
}
```

Результат работы программы не изменится, если для ее решения воспользоваться циклом с постусловием do& while:

```
#include <iostream>
using namespace std;
int main()
{ unsigned int a,b;
cout<<"A="; cin>>a;
cout<<"B="; cin>>b;
do
if (a>b) a-=b; else b-=a;
while (a!=b);
cout<<"NOD="<<a<<"\n";
return 0;}
```

ЗАДАЧА 3.8. 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 1до N, целое число.

Промежуточные переменные: i параметр цикла , целочисленная переменная , последовательно принимающая значения 2, 3, 4 и т.д. до N.

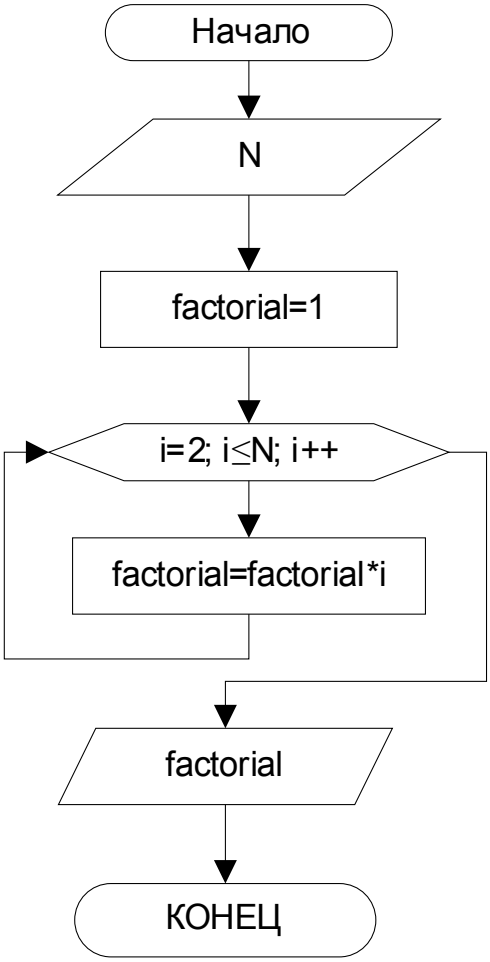


Рисунок 3.11: Вычисление факториала

```
#include "stdafx.h"
#include <iostream.h>
using namespace std;
int main()
{
unsigned int factorial, N, i;
for (cout<<"N=",cin>>N,factorial=1,i=2;i<=N;factorial*=i,i++);
cout<<"factorial="<<factorial<<"\n";
return 0;
}
```

ЗАДАЧА 3.9. 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 т.д., также имеет целочисленное значение.

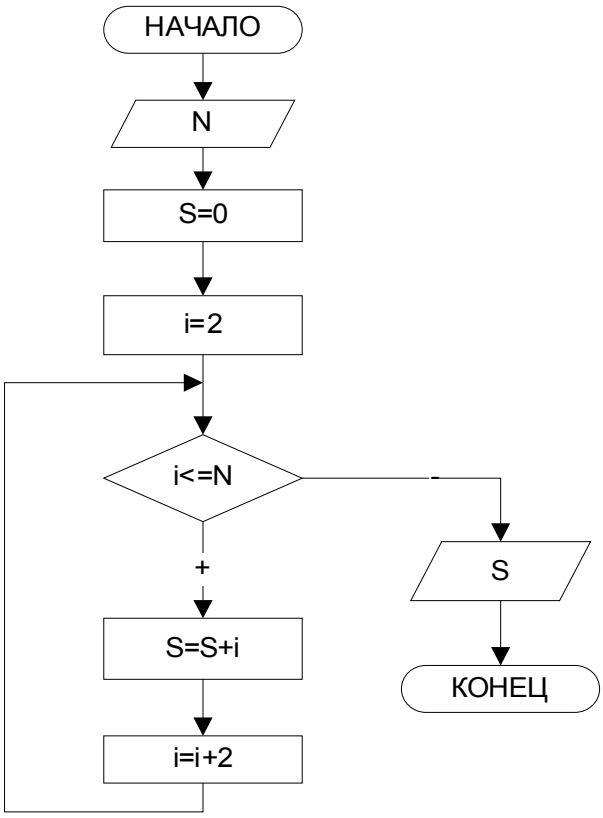


Рисунок 3.12: Вычисление суммы четных, натуральных чисел

Решение задачи с помощью цикла while

```
#include <iostream>
using namespace std;
int main()
{
unsigned int N,i,S;
cout<<"N="; cin>>N;
S=0;
i=2;
while (i<=N)
{
S+=i;
i+=2;
}
cout<<"S="<<S<<"\n";
return 0;
}
```

Решение задачи с помощью цикла for

```
#include <iostream.h>
using namespace std;
int main()
{
unsigned int N,i,S;
for (cout<<"N=",cin>>N,S=0,i=2;
i<=N;S+=i,i+=2);
cout<<"S="<<S<<"\n";
return 0;
```

}

ЗАДАЧА 3.10. Дано натуральное число *N*. Определить *K* количество делителей этого числа, не превышающих его (Например, для *N*=12 делители 1, 2, 3, 4, 6. Количество *K*=5).

Входные данные: N целое число.

Выходные данные: целое число K количество делителей N.

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

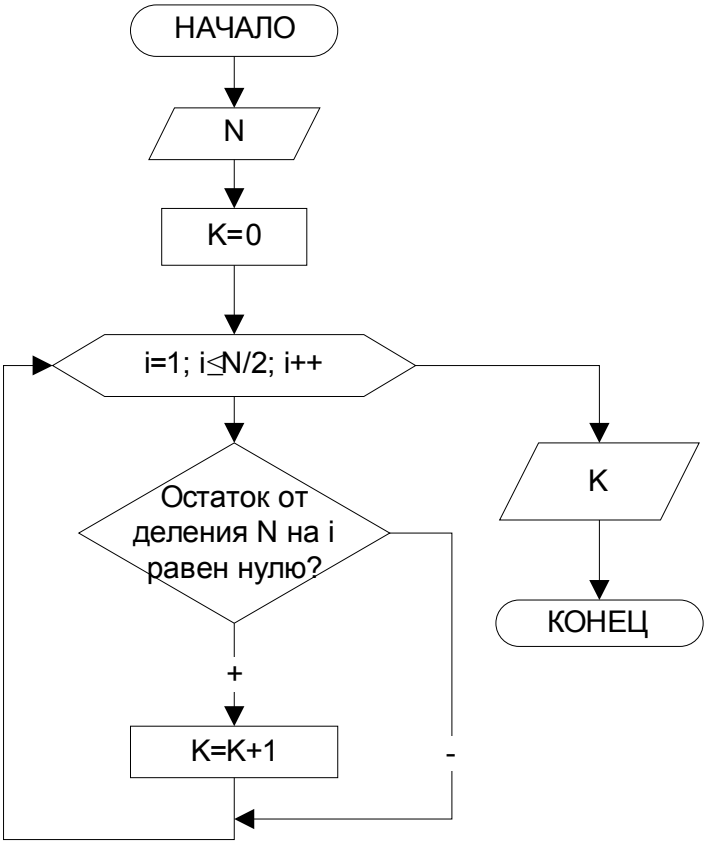


Рисунок 3.13: Определение делителей натурального числа

```
#include "stdafx.h"
#include <iostream.h>
using namespace std;
int main()
{
unsigned int N,i,K;
cout<<"N="; cin>>N;
for (K=0,i=1;i<=N/2;i++)
   if (N%i==0) K=K+1;
cout<<"K="<<K<<"\n";
return 0;
}
```

ЗАДАЧА 3.11. Дано натуральное число *N*. Определить, является ли оно простым. Натуральное число N называется простым, если оно делится без остатка только на единицу и на само себя. Число 13 простое ,так как делится только на 1и 13,а число 12таковым не является, так как делится на 1, 2, 3, 4, 6 и 12.

Входные данные: N целое число.

Выходные данные: сообщение.

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

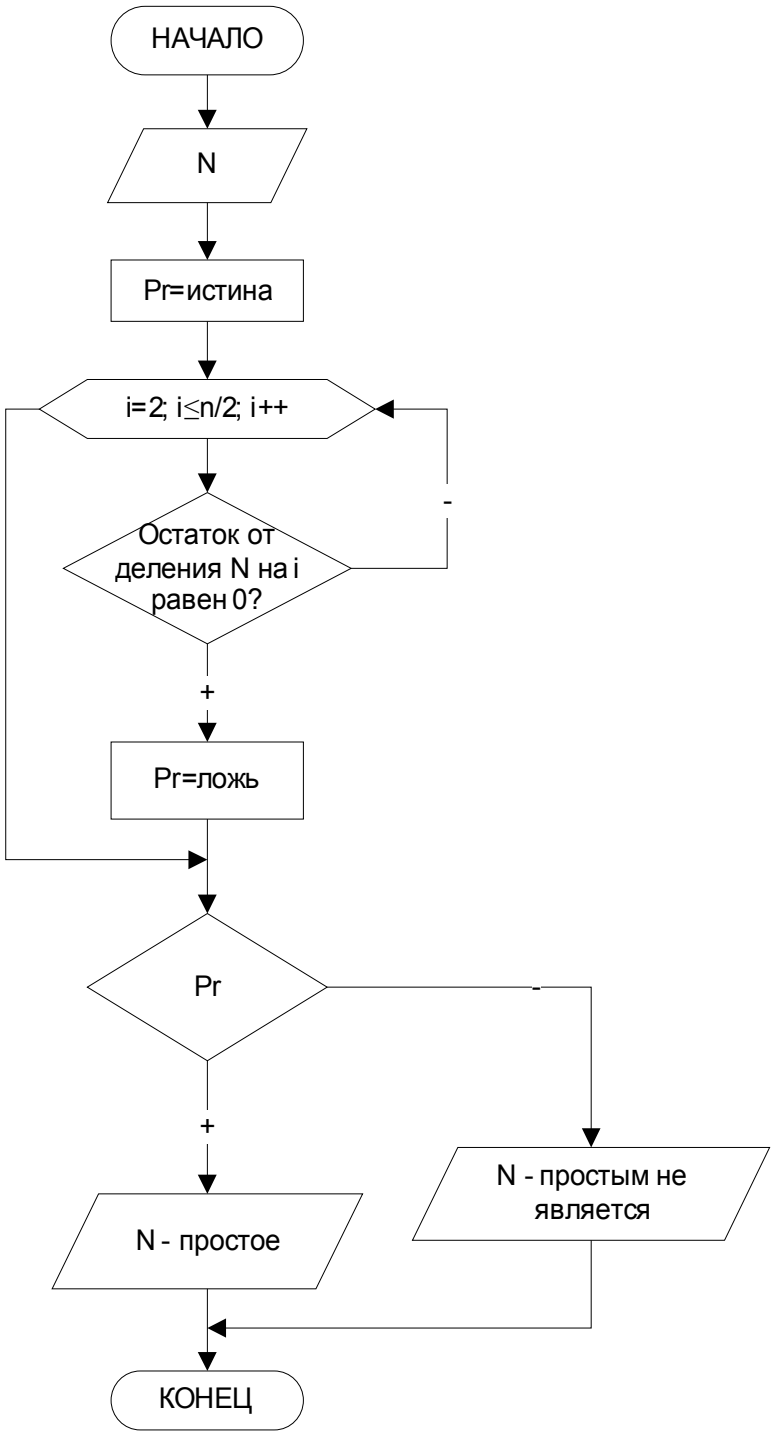


Рисунок 3.14: Определение простого числа

```
#include <iostream.h>
using namespace std;
int main()
{
unsigned int N,i;
bool Pr;
cout<<"N="; cin>>N;
Pr=true;
//Предположим, что число простое
for (i=2;i<=N/2;i++)
if (N%i==0)
{Pr=false;
```

```
break;}
if (Pr)
cout<<"N - prostoe \n";
else
cout<<"N - ne prostoe \n";
return 0;
}
```

ЗАДАЧА 3.12. 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.

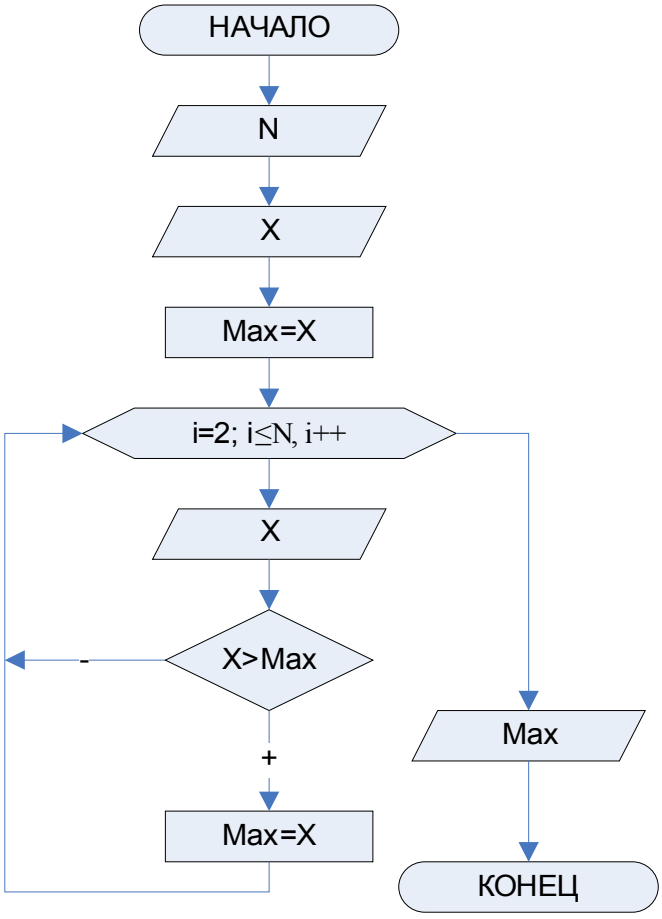


Рисунок 3.15: Поиск наибольшего числа в последовательности

```
#include <iostream>
using namespace std;
int main()
{
unsigned int i,N;
float X,Max;
cout<<"N="; cin>>N;
cout<<"X="; cin>>X;
for (i=2,Max=X;i<=N;i++)
{
```

```
cout<<"X="; cin>>X;
if (X>Max) Max=X;
}
cout<<"Max="<<Max<<"\n";
return 0;
}
```

ЗАДАЧА 3.13. Поступает последовательность целых положительных чисел. 0 конец последовательности . Определить количество простых чисел в последовательности.

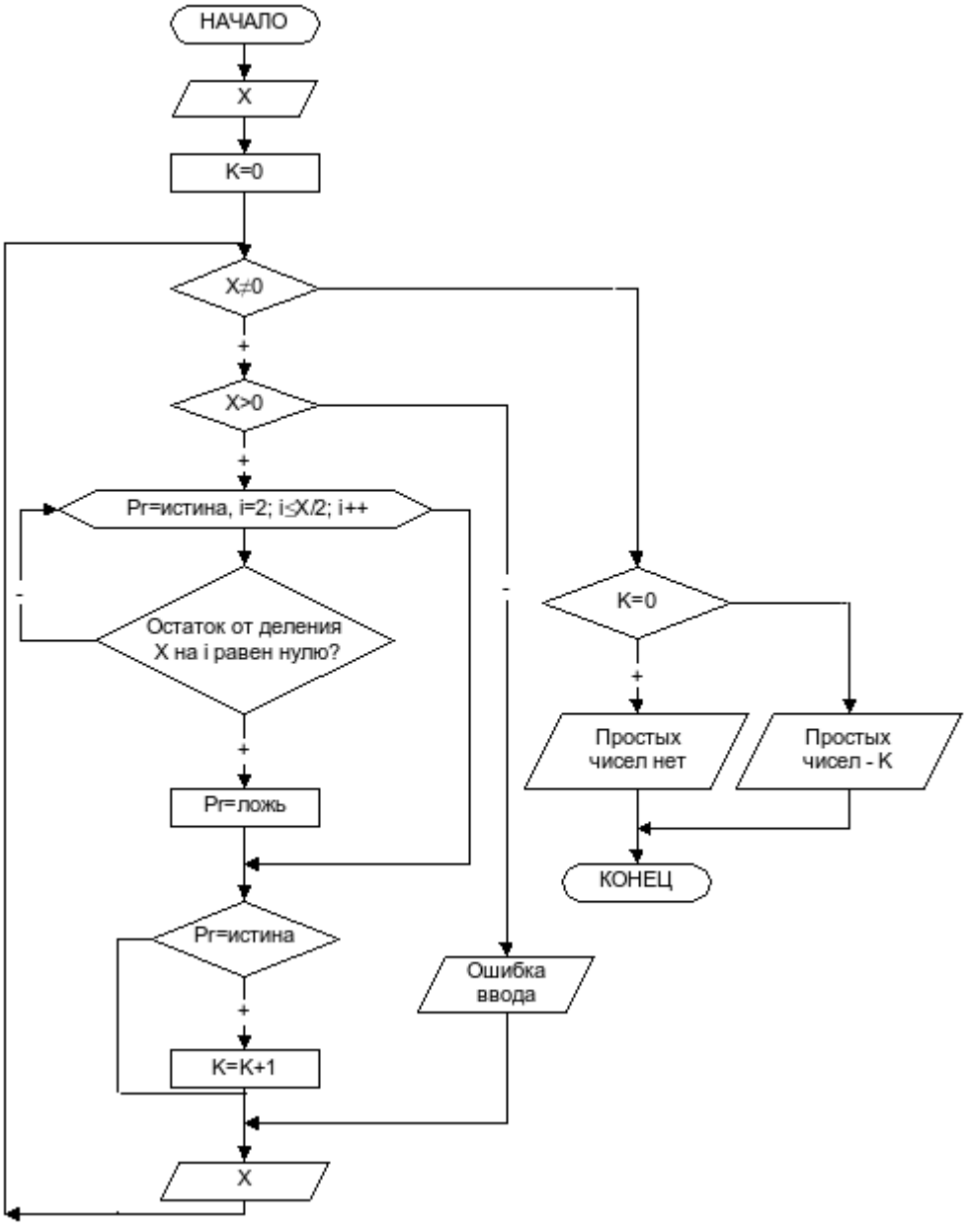


Рисунок 3.16: Поиск простых чисел в последовательности

```
#include <iostream>
using namespace std;
int main()
{unsigned int i,k;
int X;
bool Pr;
cout<<"X="; cin>>X;
k=0;
```

```
while (X!=0)
{
if (X>0)
        {
             for (Pr=true,i=2;i<=X/2;i++)
             if (X%i==0)
              {Pr=false; break;}
             if (Pr) k++;

        }

        else cout<<"Input error"<<"\n";
   cout<<"X="; cin>>X;
}
if (k==0)
cout<<"Prime numbers are not \n";
else

cout<<"Prime numbers k="<<k<<"\n";
return 0;
}
```

ЗАДАЧА 3.14. Вводится последовательность целых чисел, 0 конец последовательности . Найти наименьшее число (см. задачу 3.12) среди положительных, если таких значений несколько, определить, сколько их.

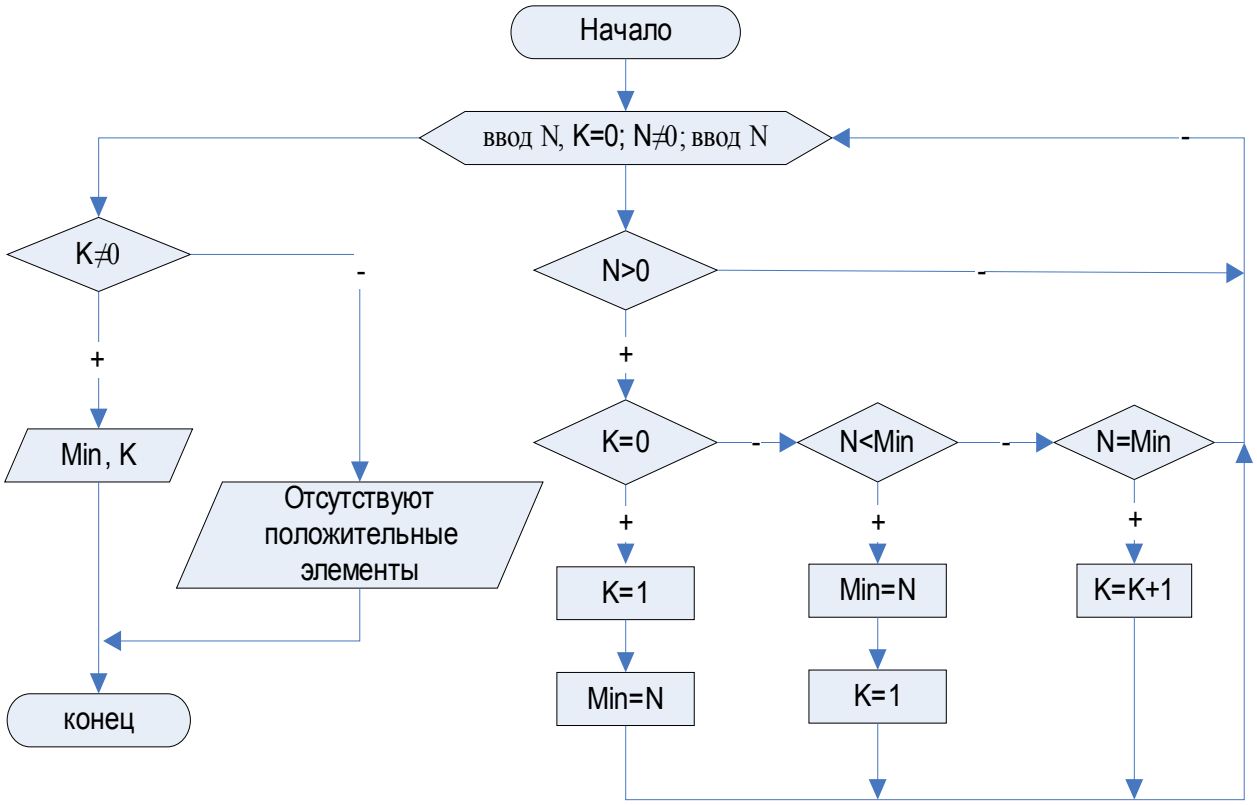


Рисунок 3.17: Поиск минимального положительного числа в последовательности

```
#include <iostream>
using namespace std;
int main()
{float N,Min; int K;
for (cout<<"N=",cin>>N,K=0;N!=0;cout<<"N=",cin>>N)
```

```
if (N>0)
   if (K==0) {K=1;Min=N;}
   else if (N<Min) {Min=N;K=1;}
   else if (N==Min) K++;
if (K!=0)
cout<<"Min="<<Min<<"\n"<<"K="<<K<<"\n";
else
cout<<"Positive elements are absent \n";
return 0;
}
```

ЗАДАЧА 3.15. Определить сколько раз последовательность из *N* произвольных чисел меняет знак.

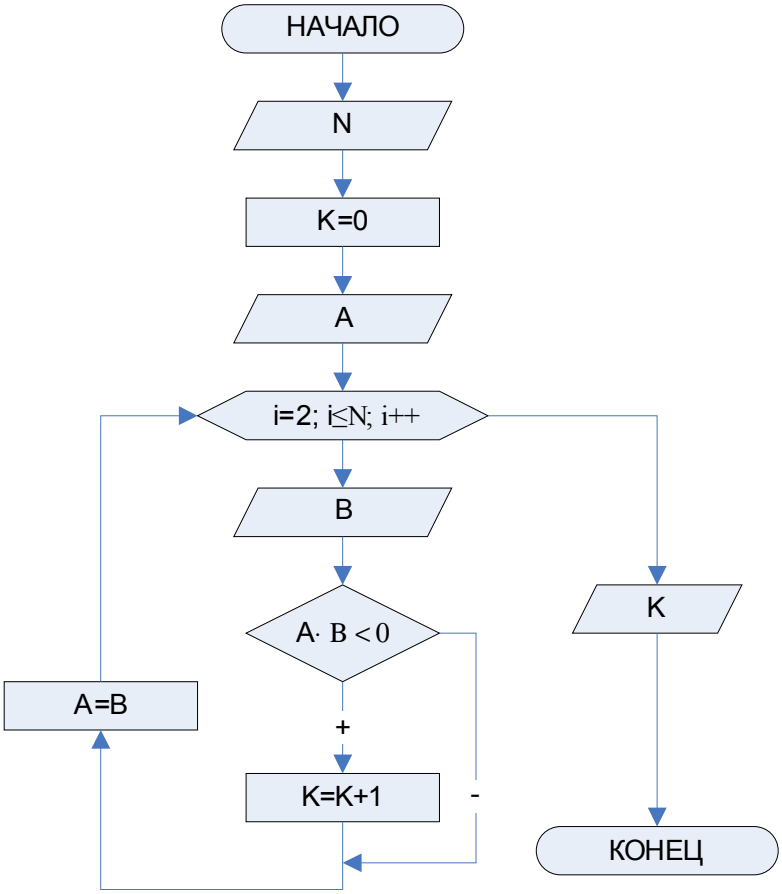


Рисунок 3.18: Блок-схема к задаче 3.15

```
#include <iostream>
using namespace std;
int main()
{float A,B; int i,K,N;
cout<<"N=";cin>>N;
K=0;
cout<<"A=";cin>>A;
for (i=2;i<=N;i++)
{  cout<<"B=";cin>>B;
   if (A*B<0) K++;
   A=B;
}
cout<<"K="<<K<<"\n";
   return 0;}
```

ЗАДАЧА 3.15**.** Дано натуральное число N. Определить самую большую цифру и ее позицию в числе (N=573863, наибольшей является цифра 8, ее позиция четвертая слева).В связи с тем, что при делении на 10, мы будем получать цифры справа налево, то для определения позиции в числе сначала надо найти количество разрядов (см. рис. 3.19), а затем определять цифру и ее позицию (см. рис. 3.20).

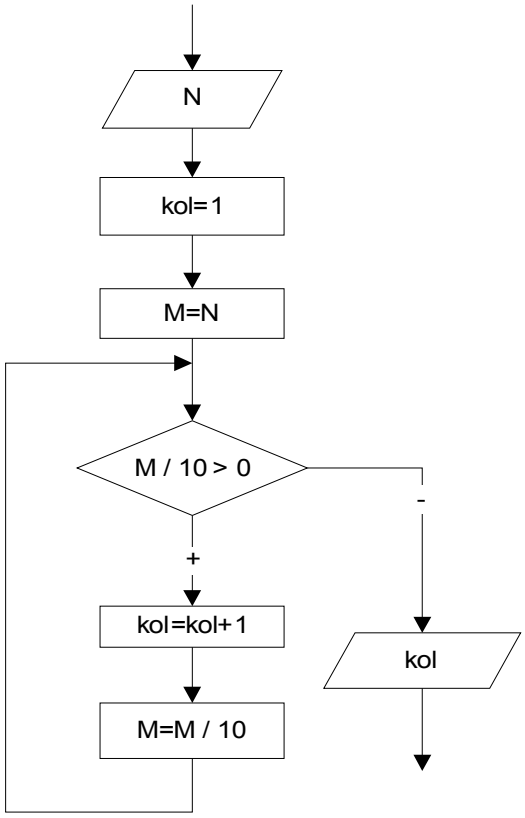


Рисунок 3.19: Определение количества разрядов в числе

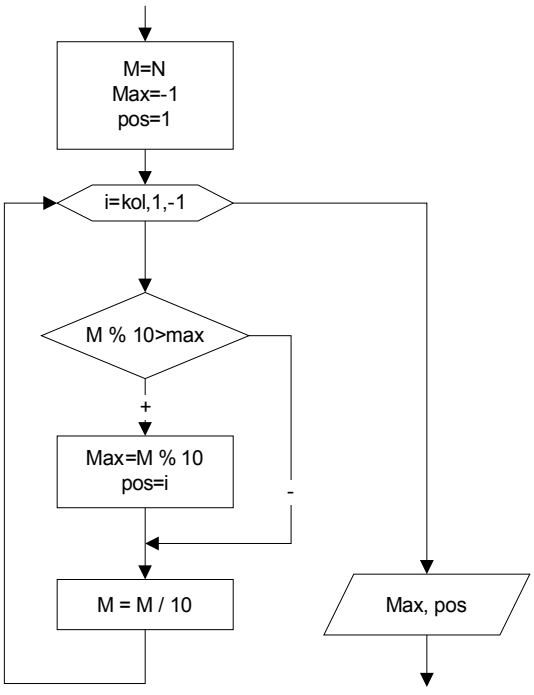


Рисунок 20: Алгоритм решения задачи

```
#include <stdio.h>
```

```
#include <math.h>
int main()
{
long int N,M,kol=1;
int max,pos,i;
printf("\n Введите N>0\n");
scanf("%ld",&N);
M=N;
while(M/10>0)
{
kol++;
M/=10;
}
printf("В числе %ld %ld разрядов\n",
N,kol);
for(M=N,max=-1,pos=1,i=kol;i>1;i--)
{
if (M%10>max)
{
max=M%10;
pos=i;
}
M/=10;
}
printf("В числе %ld максимальная цифра %d, ее номер %d\n", N,max,pos);
}
```

ЗАДАЧА 3.16. Определить количество простых чисел в интервале от N до M, где N и M натуральные числа.

```
#include <stdio.h>
#include <math.h>
int main()
{
unsigned int N,M,i,j,pr,k;
do
{
printf("\n Введите N и M\n");
scanf("%u%u",&N,&M);
}while(M<N);
for(k=0,i=N;i<=M;i++)
{
for (pr=1,j=2;j<i/2;j++)
if (i%j==0)
{
pr=0;
break;
}
if (pr==1)
{
printf("\nЧисло %u - простое\n",i);
k++;
}}
if (k)
printf("В интервале от %u до %u - %u простых чисел",
N,M,k);
else
printf("В интервале от %u до %u - нет простых чисел",
N,M);
  }
```