КЕМЕРОВСКИЙ ИНСТИТУТ (филиал)
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ТОРГОВО-ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Информатика и программирование

Лебедева Т.Ф.

Программирование разветвляющегося вычислительного процесса (р.в.п.)

Участок алгоритма, состоящий из блока выбора решения и связанных с ним блоков по направлениям «да» и «нет», будем называть **разветвленным** (структурой ветвления).

Вычислительный процесс, в котором естественный порядок выполнения действий нарушается в результате проверки некоторого условия и образования двух ветвей дальнейших действий, называется ***разветвляющимся.***

Различают два вида условий – простые и составные.

Простым условием (выражением сравнения) называется выражение, составленное из двух арифметических выражений или двух строковых выражений (иначе их еще называют операндами), связанных одной из операций отношения:

< - меньше, чем...

> - больше, чем...

<= - меньше, чем... или равно

>= - больше, чем... или равно

<> - не равно

= - равно

Например, простыми отношениями являются следующие:

x-y>10; k<=sqr(c)+abs(a+b); 9<>11; ‘мама’<>‘папа’.

Составные логические выражения образуются из простых выражений с помощью логических операций **not, and, or , xor.**

Логические операции, операции отношения и арифметические операции часто встречаются в одном выражении. При этом отношения, стоящие слева и справа от знака логической операции, должны быть заключены в скобки, поскольку логические операции имеют *более высокий приоритет*. Вообще принят следующий приоритет операций:

1. **not**
2. **and, *, /, div, mod**
3. **or, +, -**
4. **операции отношения.**

Примечание. Логическую операцию and еще называют *логическим умножением*, а логическую операцию or - *логическим сложением*.

Кроме того, порядок выполнения операций может изменяться скобками. Например, в логическом выражении расставим порядок действий

4 3 2 1

A or B and not (A or B)

Сначала выполняется заключенная в скобки операция or, а затем операции not, and, or. Если подставить вместо переменных A и B значения True и False, то, используя уже рассмотренный порядок действий, получим значение всего выражения равное True.

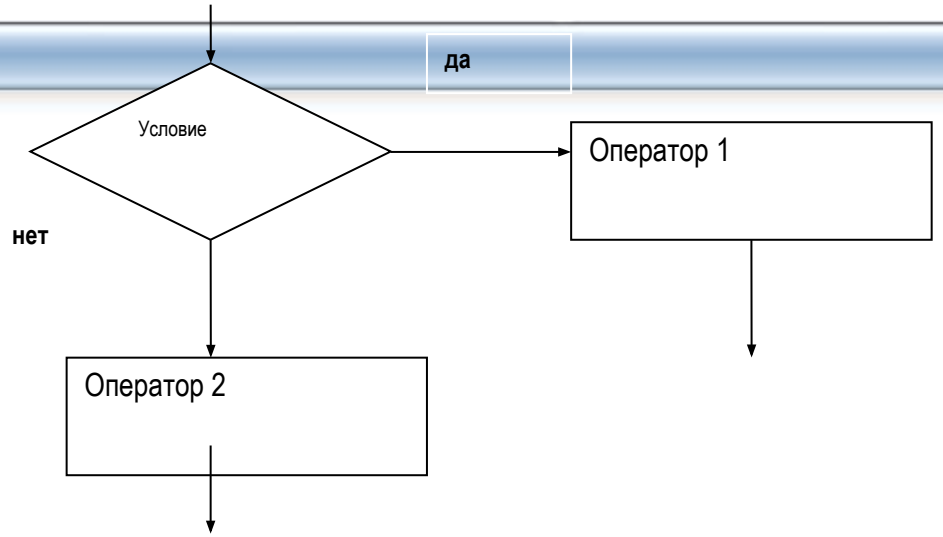
Определите порядок выполнения операций в выражении **(a + b > c) and (b + c < a)**

В смешанных выражениях тип результата определяется типом последней операции

Таблица истинности логических операций

a, b – переменные логического типа

a	b	Not a	a and b	a or b	a xor b
False	False	True	False	False	False
False	True	True	False	True	True
True	False	False	False	True	True
True	True	False	True	True	False



В общем случае полная форма конструкции условного оператора имеет вид:

if **<логическое выражение>**

then

<оператор 1>

else

<оператор 2>;

Условный оператор работает по следующему алгоритму.

Сначала вычисляется значение логического выражения, расположенного за служебным словом IF. Если его результат истина, выполняется <оператор 1>, расположенный после слова THEN, а действия после ELSE пропускаются; если результат ложь, то, наоборот, действия после слова THEN пропускаются, а после ELSE выполняется <оператор 2>.

Если в качестве оператора должна выполниться серия операторов, то они заключаются в операторные скобки begin-end. Конструкция *Begin ... End* называется *составным* оператором.

```
if  <логическое выражение>
  then
     begin
        оператор 1;
        оператор 2;
        ...
     end
  else
     begin
       оператор 1;
       оператор 2;
       ...
     end;
```

Составной оператор **-** объединение нескольких операторов в одну группу. Группа операторов внутри составного оператора заключается в операторные скобки (begin-end).

```
begin
   <оператор 1>;
   <оператор 2>;
end;
```

Можно также использовать и *сокращенную (неполную)* форму записи условного оператора. Эта форма используется тогда, когда в случае невыполнения условия ничего делать не надо.

Неполная форма условного оператора имеет следующий вид:

if <логическое выражение>
then <оператор> ;

Тогда если выражение, расположенное за служебным словом IF , в результате дает *истину,* выполняются действия после слова THEN, в противном случае эти действия пропускаются.

Пример 3. Составить программу, которая, если введенное число отрицательное меняет его на противоположное.

```
Program Chisla;
Var    x : integer; {вводимое число}
Begin
   writeln('Введите число '); {вводим целое число}    readln(x);
   if x<0      then          x:=-x;
    writeln (x);
    readln;
End.
```

В этом примере использована сокращенная форма оператора ***IF.***

Пример 4. Вывести на экран большее из трех данных чисел.

```
Program Example4;
Var
   a, b, c, m : integer; {вводимые числа}
Begin
   writeln('Введите 3 числа '); {вводим два целых числа через пробел}
   readln( a, b, c );
   M:=a;          {назначили  а большим числом}
  if b > M  then  M : = b;
  if  c > M then M: = c;
   writeln (m);  {выводим m}
    readln;
End.
```

Пример 5. Предусмотреть исключение всех случаев возникновения неопределенности при вычислении функции по заданной формуле:

$$y = \frac{\ln(ax - b) + c}{2{,}5bx}.$$

Функция не определена при $ax - b \le 0$ и $2{,}5\,bx = 0$

Для исключения неопределенностей *нужно перед вычислением значения функции организовать проверку на выход аргумента за пределы области существования функции, вывести на печать величины, приводящие к неопределенности, и не выполнять счет.*

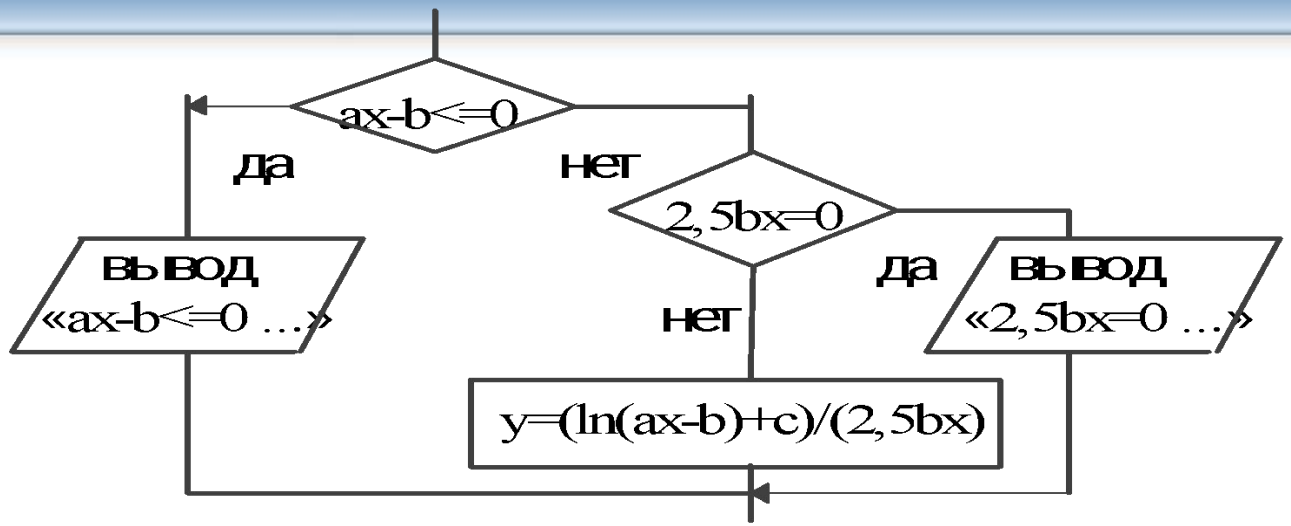


Рис. 3. Фрагмент блок-схемы устранения неопределенностей из примера 3

В общем случае неопределенности могут возникнуть при вычислении функций:

$\frac{1}{x}$	**при x = 0 , где x – любое выражение**
$\sqrt{x}$	**при x < 0**
Ln x	**при x ≤ 0**

Фрагмент программы с исключением неопределенности (рис.3)

```
……….
IF a*x-b<=0 THEN WriteLn (‘ a*x-b<=0 ‘)  ELSE
  Begin
     IF   2.5*b*x=0  THEN  WriteLn (‘2.5*b*x=0 ‘ )
                           ELSE
                                Begin
                                   Y: =(ln(a*x-b)+c)/( 2.5*b*x);
                                    WriteLn (‘y=‘; y:8:3);
                               End;
   End;
……….
```

Вложенные условные операторы

При решении задач часто приходится рассматривать не два, а большее количество вариантов. Это можно реализовать, используя несколько условных операторов. В этом случае после служебных слов Then и Else записывается новый условный оператор. Рассмотрим пример.

Пример 6. Вычислить значение функции:

$$y=\begin{cases} x-12, x>0 \\ 5, x=0 \\ x^2, x<0 \end{cases}$$

Тогда фрагмент программы для решения этой задачи будет выглядеть так:

```
……….
        if x>0
           then
             y := x-12
           else
              if x=0
                    then
                       y := 5
                    else
                      y := sqr(x);
```

Итак, когда оператор if появляется внутри другого оператора if, они считаются *вложенными*. Такое *вложение* используется для уменьшения числа необходимых проверок. Этот метод часто обеспечивает большую эффективность, однако одновременно он уменьшает наглядность программы. Не рекомендуется использовать более одного-двух уровней вложения if. За вторым уровнем вложения становится трудно восстановить последовательность проверки условий каждым условным оператором.

Если часть else используется во вложенных if, то каждое else соответствует тому if, которое ему непосредственно предшествует. Таким образом, при определении последовательности выполнения фрагментов нет двусмысленности.

Оператор выбора *case*

Оператор If позволяет программе выполнять переходы на ту или иную ветвь по значению логического условия. Используя несколько операторов If, можно производить ветвление по последовательности условий. В приведенном фрагменте показано, как при помощи ряда операторов If можно преобразовать целое число (в диапазоне 0-9) к его словесному представлению:

```
if Ziphra = 0
   then
      write (‘Нуль‘);
if Ziphra = 1
   then
      write (‘Единица‘);
if Ziphra = 2
   then
      write (‘Два‘);
и т.д.
```

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы. Параметром, по которому осуществляется выбор, служит так называемый ключ выбора (или селектор) – выражение *порядкового* типа.

Общая форма записи следующая:

case *селектор* ***of***

<значение1> : <оператор1>;

<значение2> : <оператор2>;

. . . . . .

<значениеN> : <операторN>

*[**else** <оператор(N+1)>;]*

end;.

Здесь значение1, значение2,…….- константа или список констант того же порядкового типа, что и селектор;

<оператор1>, <оператор2>,……..- операторы, чаще всего составные.

Оператор выбора работает следующим образом. Сначала вычисляется значение выражения-селектора, стоящее после зарезервированного слова ***case***, а затем выполняется оператор (или составной оператор), соответствующий результату вычисления выражения.

Может случиться, что в списке выбора не окажется константы равной вычисленному значению ключа.

В этом случае управление передается оператору, стоящему за словом ***ELSE***. Если слово ***ELSE*** отсутствует, то управление передается следующему за оператором **case** оператору.

Например,

```
……….
        case NUMBER mod 2 of
           0 : writeln (NUMBER, ‘- число четное‘)
           else : writeln (NUMBER, ‘- число нечетное‘);
         end;
```

Если один оператор выполняется при нескольких значениях, то их можно перечислить в списке через запятую.

```
Var MONTH : byte;
……..
        case MONTH of
           1, 2, 3 : writeln (‘Первый квартал‘);
           4, 5, 6 : writeln (‘Второй квартал‘);
           7, 8, 9 : writeln (‘Третий квартал‘);
           10, 11, 12 : writeln (‘Четвёртый квартал‘);
        end;
```

Оператором может являться не только простой оператор, но также составной и пустой операторы.

```
……….
        case CODE of
           1 :    for i := 1 to 5 do
            writeln (‘*******‘);
           2 :    begin                {составной оператор}
                    x:=sqr(y-1);
                     writeln (x);
                  end;
           3 :                         {пустой оператор}
        end;
```

Если оператор должен выполняться при нескольких значениях селектора следующих друг за другом, образуя некоторый промежуток, то это можно записать в более сжатой форме. Например,

```
var  symvol: char;
.....................
      case symvol of
         '0'..'9' : writeln ('Это цифра');
         'A'..'Z': writeln ('Это прописная буква');
         'a'..'z': writeln ('Это строчная буква');
       else writeln ('Это другой символ');
       end;
```

Пример Написать программу преобразования цифр в слова.

```
Program Number1;
Var
   a, b, c : integer;
Begin
   writeln('Введите цифру ');
   readln(a);
    if (a<0) or (a>9)
       then
          writeln ('Это число не является цифрой')
      else
         case a of
            0 : writeln ('ноль');
            1 : writeln ('один');
            2 : writeln ('два');
            3 : writeln ('три');
            4 : writeln ('четыре');
            5 : writeln ('пять');
            6 : writeln ('шесть');
            7 : writeln ('семь');
            8 : writeln ('восемь');
            9 : writeln ('девять');
         end;
   readln;
End.
```

Программирование циклических вычислительных процессов

Очень многие алгоритмы, выполнение которых поручается компьютеру, по своей природе являются циклическими. И это не случайно, потому что человек обычно поручает машине рутинную работу, где нужно много считать, и счет производится по некоторым одинаковым правилам.

Цикл – это последовательность операторов, которая может выполняться более одного раза.

Циклический алгоритм – это алгоритм, содержащий один или несколько циклов.

Параметр цикла – это простая переменная, которая изменяется при каждом повторении цикла по некоторому закону и управляет работой цикла.

Цикл предусматривает 4 основных действия:

1. Присваивание начального значения параметру цикла;
2. Рабочий участок или ***тело цикла***, содержащий действия, повторяющиеся несколько раз;
3. Изменение параметра цикла;
4. Проверка условия окончания цикла и при невыполнении его переход к началу тела цикла.

Различают циклы с *известным числом повторений* (цикл со счетчиком) и *итерационные* (с пред- и постусловием).

Цикл со счетчиком

Телом цикла называется группа операторов, которая многократно повторяется и ради повторения которых организован цикл.

Для организации цикла с известным числом повторений в Паскале используется оператор **for**.

Структура цикла, организованного с помощью этого оператора, имеет вид:

```
For I := i1 To  i2  Do
Begin
  тело цикла
End;
```

```
For I := i1  DownTo   i2  Do
Begin
  тело цикла
End;
```

Здесь **i**— параметр цикла, изменяющийся в цикле; **i1, i2** — выражения, обозначающие начальное, конечное значение параметра цикла. Шаг изменения параметра цикла равен 1, если в заголовке цикла стоит *To*; и –1 — при *DownTo* (*здесь рассматривается случай, когда **I, I1, i2** переменные и выражения целого типа).* Если тело цикла состоит из одного оператора, то составной оператор begin…..end не нужен.

В общем случае **i, i1, i2** – переменные порядкового типа

Порядок выполнения цикла с шагом 1 следующий:

1) вычисляются значения начального и конечного значений параметра цикла **i1, i2**;
2) параметр **i** принимает начальное значение **i1**;
3) если **i** меньше или равно конечному значению, исполняется **тело** цикла;
4) к значению параметра цикла **i** прибавляется 1, т.е. **i := i + 1**; проверяется условие **i ≤ i2** (для отрицательного шага условие **i ≥ i2**) и при его выполнении цикл повторяется.
5) Выход из цикла осуществляется, если **i >i2** (**i< i2** для H=–1), и выполняется оператор, следующий за оператором цикла. Если **i1 > i2** (или **i1 < i2** для H=–1), ***то тело цикла не выполняется ни разу.***

Если в операторе цикла с параметром начальное или конечное значение параметра заданы переменными или выражениями, то значения этих переменных должны быть определены в программе до оператора цикла.

Рассмотрим примеры использования оператора цикла **for.**

Пр.1 Пусть требуется вывести на экран таблицу квадратов N целых чисел:

```
Var   i, n : integer;
Begin
readln( n );
for i : = 1 to n do  writeln (I : 4,  i *  i : 8) ;
```

Пр.2 Подсчитать сумму 20 вводимых чисел:

```
S: = 0;
for i : = 1 to 20 do
        begin  readln(x);  S: = S + x; end; write (S:6:2);
```

В некоторых случаях бывает удобно, чтобы <параметр цикла> принимал последовательные, но не возрастающие, а убывающие значения. В этом случае надо использовать оператор цикла с параметром следующего вида:

for <параметр цикла> := <N> downto <K> do
<оператор>;

Пр.3 Пусть требуется вывести на экран последовательность букв от M до A и их номера:

```
Var Ch : char;
begin
for Ch := 'M' downto 'A' do
        writeln (Ch:2,  ' номер-',    ord( ch) :4);
```

Рекомендации по программированию цикла for :

1) Не следует внутри цикла изменять параметр цикла;
2) Не следует внутри цикла изменять начальное и конечное значения параметра цикла с помощью операторов присваивания или ввода;
3) Значение параметра цикла не определено после выхода из цикла;
4) Нельзя перейти к оператору тела цикла по оператору goto <метка>, минуя заголовок цикла;
5) Допускается досрочный выход из цикла с использованием процедур:

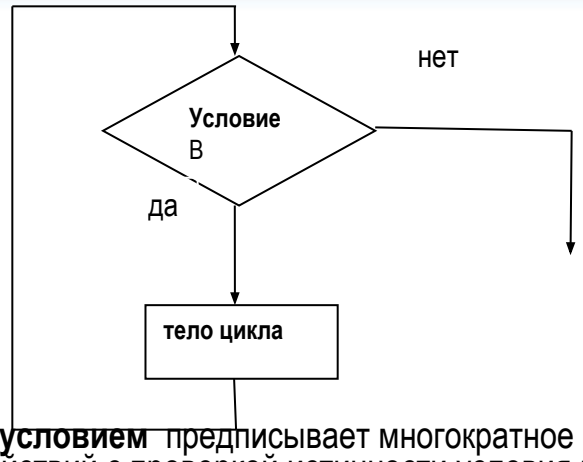
Break - реализует выход из цикла любого типа;

Continue - осуществляет переход на следующую итерацию цикла, игнорируя оставшиеся до конца тела цикла операторы;

Halt (<код, завершения>) - осуществляет выход из программы, возвращая операционной системе заданный код завершения. Считается, что программа завершилась нормально, если код завершения равен нулю. Возвращение кода завершения, отличного от нуля, обычно означает, что программа завершена по обнаружении каких-либо ошибок. Коды завершения назначаются программистом, а информация о них помещается в программную документацию;

Exit- осуществляет выход из подпрограммы. Если процедура использована в основной программе, то выполняется выход из программы

Цикл с предусловием




```
while  B Do
Begin
тело цикла
End;
```

Цикл с предусловием предписывает многократное выполнение одной и той же последовательности действий с проверкой истинности условия перед телом цикла.

Выполнение оператора заключается в многократном повторении двух действий:

- вычислении логического выражения B (условия), записанного в заголовке цикла;
- выполнении оператора, являющегося телом цикла.

Эти действия повторяются, пока выражение в заголовке не станет ложным. Проверка истинности выражения производится до первого выполнения оператора, то есть, если выражение сразу ложно, то тело цикла не выполнится ни разу.

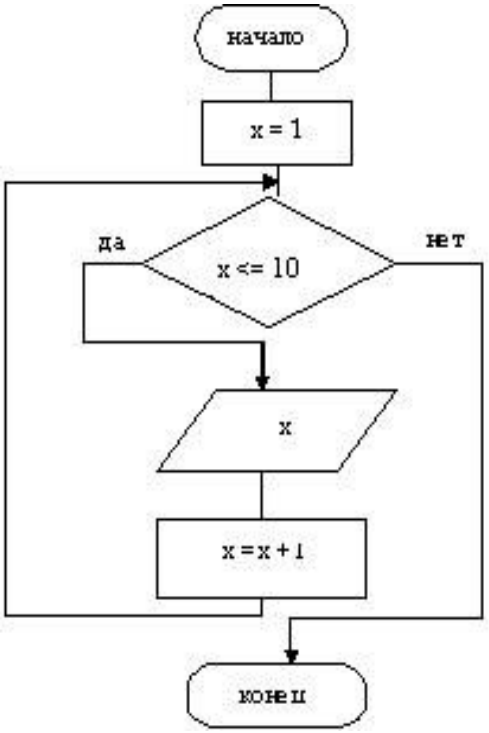
В качестве примера рассмотрим фрагмент программы, который выводит на экран натуральные числа от 1 до некоторого N.

```
Write ('Введите число > ');
        readln (n);
        i := 0;
        while (i <= n) do
          begin
        i := i + 1;
        writeln (i)
        end;
```

Пример5: Вывести на экран в одну строку через один пробел десять значений целого типа, начиная с единицы, в возрастающем порядке.
Алгоритм решения представлен на рисунке.

Текст программы на Паскале:

```
Const k = 10; Var x: integer;
begin x : = 1;
while x <= k do
  begin
    write(x,' '); x: = x + 1;
  end;
readln;
end.
```



Цикл с постусловием

Оператор цикла **repeat** в Паскале также используется в циклах с неизвестным числом повторений и с известным числом повторений. Вид оператора:

repeat s until b;

Здесь s – тело цикла,

b - логическое выражение.

При выполнении оператора

сначала выполняется тело цикла,

а затем проверяется логическое условие.

Таким образом, обеспечивается,

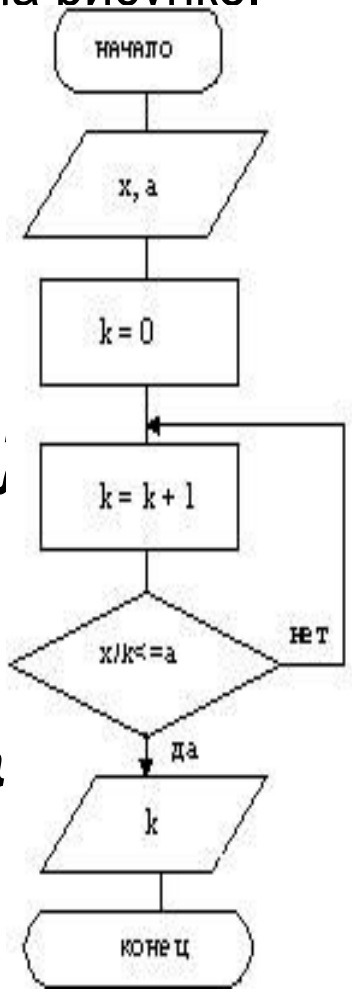
по меньшей мере, одно выполнение тела цикла.

Цикл повторяется, пока логическое выражение **ложно**. Когда оно становится истинным, происходит выход из цикла.

Пример 6: Составить программу определения k, при котором x/k становится меньше или равен a, где k = 1, 2, 3… . Алгоритм решения представлен на рисунке.

Текст программы на Паскале:

```
Var x, a : real; k : integer;
begin
write ('x='); readln (x); {ввод значения x}
write ('a='); readln (a); {ввод значения a}
k : = 0; {начальное значение k перед циклом}
 repeat {начало цикла}
k : =k+1; {увеличение k в цикле}
until x/k <= a; {проверка логического условия}
writeln('k=',k); {вывод найденного значения}
readln;
end.
```



Выполним построение математической модели и алгоритма решения задачи табулирования функции.

Пример 7. Получить таблицу значений функции **y = f (x)** для аргумента **x**, изменяющегося от **a** до **b** с шагом **h** (если **a>b,** то **h** должен быть меньше нуля, т.е. отрицательным).

а) Обозначение переменных:

x – аргумент функции; y – значение функции;

a – начальное значение интервала изменения аргумента;

b – конечное значение интервала изменения аргумента;

h – шаг изменения аргумента на интервале;

i – счётчик цикла; **n** – число повторений цикла.

б) Тип переменных:

i, n – простые переменные целого типа;

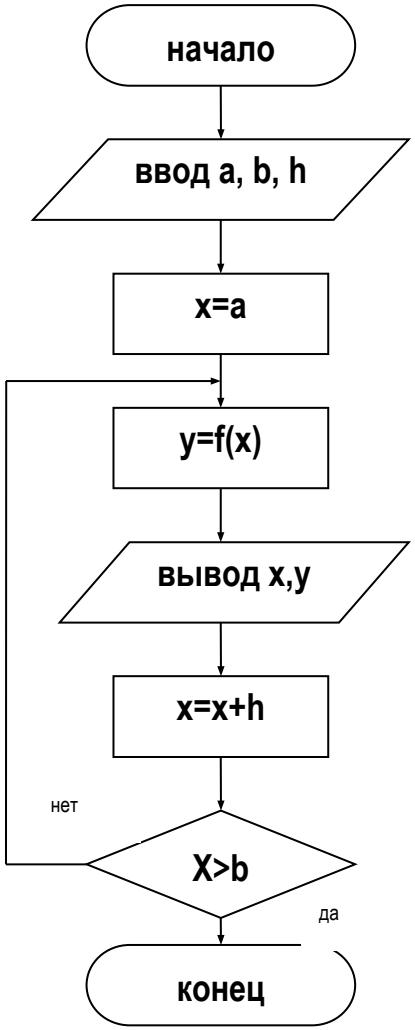
x, y, a, b, h – простые переменные вещественного типа.

в) Классификация по группам:

исходные данные: **a, b, h;**

промежуточные результаты**: i, n**; результаты: **x, y**.

Блок-схема цикла с постусловием к примеру 7



Присваивание начального значения параметру цикла

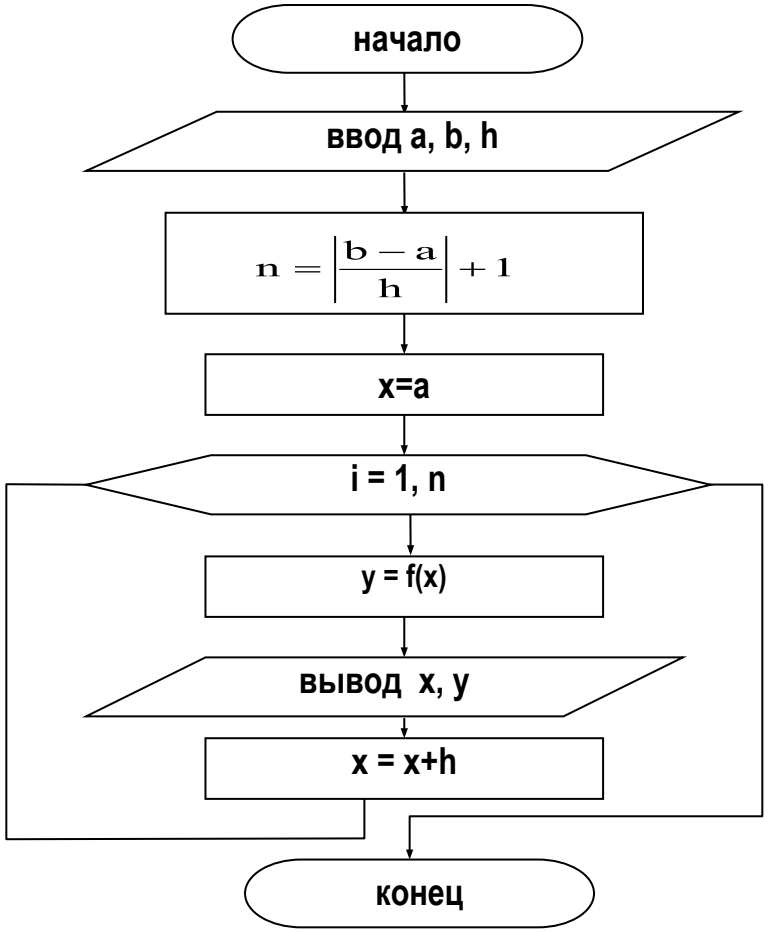
Рабочий участок или тело цикла

Изменение параметра цикла

Проверка условия на выход из цикла

```
Program primer_7_1;
Uses CRT;
Var x, y, a, b, h: real;
begin
write (‘a=’); readln (a); {ввод значения a}
write (‘b=’); readln (b); {ввод значения b}
write (‘h=’); readln (h); {ввод значения h}
Clrscr; {очистка экрана}
writeln( ‘                Таблица’);
writeln( ‘     x                                  y’); {вывод заголовка таблицы}
writeln(‘-------------------------------------------------’); {вывод горизонтальной черты}
x : = a; {начальное значение x перед циклом}
 repeat {начало цикла}
    y : =x * sin( x ); {вычисление функции в цикле}
    Writeln( x :8:3,   y:14:3); {вывод строки таблицы на экран}
    x:= x +h; {изменение параметра x}
until x > b; {проверка логического условия}
writeln(‘-------------------------------------------------’); {вывод горизонтальной черты}
readln;
end.
```

Блок-схема цикла со счетчиком i к примеру 7



```
Program primer_7_2;
Uses CRT;
Var x, y, a, b, h: real;
i, n : integer;
begin
write (‘a =’); readln (a); {ввод значения a}
write (‘b =’); readln (b); {ввод значения b}
write (‘h =’); readln (h); {ввод значения h}
Clrscr; {очистка экрана}
writeln( ‘                Таблица’);
writeln( ‘     x                                  y’); {вывод заголовка таблицы}
writeln(‘-----------------------------------------------------’); {вывод горизонтальной черты}
x : = a; {начальное значение x перед циклом}
N : = trunc( (b –a) / h ) + 1; {вычисление количества строк таблицы (повторений цикла)}
 for i : = 1 to n do         {начало цикла}
   begin
     y : =x * sin( x ); {вычисление функции в цикле}
     Writeln( x :8:3,   y:14:3); {вывод строки таблицы на экран}
     x:= x +h; {изменение параметра x}
   end;
writeln(‘-----------------------------------------------------’); {вывод горизонтальной черты}
readln;
end.
```

Пример 8. Первоначальная стоимость оборудования производственного цеха составляет R0 руб. Ежегодно на сумму D руб. закупали новое оборудование. Ежегодная амортизация (уменьшение стоимости) имеющегося оборудования составляет P% от его стоимости. Составить алгоритм для вычисления стоимости оборудования цеха RN через N лет после ввода его в эксплуатацию согласно формуле ,

$$R_N = R_{N-1}(1 - \frac{P}{100}) + D$$

где **RN** - стоимость оборудования в **N**-й год; где **RN-1** - стоимость оборудования в предыдущем году. Получить таблицу результатов для N=1, 2,…10.

а) Обозначение переменных:

R0 – первоначальная стоимость оборудования;

D – стоимость ежегодно закупаемого нового оборудования;

P – процент ежегодной амортизации оборудования;

N – счётчик цикла, количество лет эксплуатации оборудования.

б) Тип переменных:

P, N – простые переменные целого типа;

R0, D – простые переменные вещественного типа.

в) Классификация по группам:

исходные данные: **R0, D, P**; результаты: **N, R0**.

г) расчетные формулы в последовательности их выполнения:

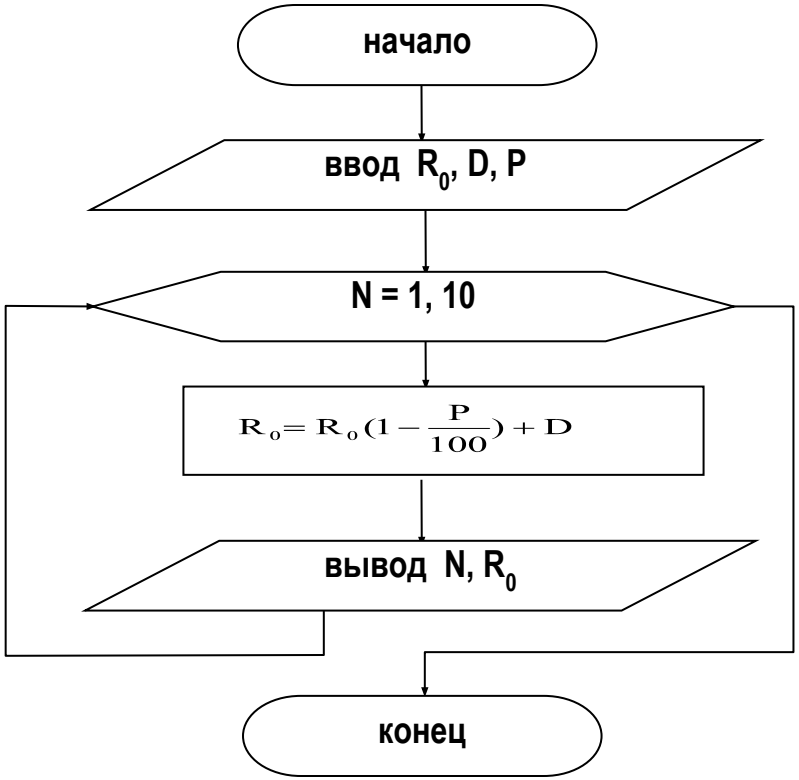
N=1

$R_N = R_0(1 - \frac{P}{100}) + D$

N=N+1

Если N≤10, то повторять действия, иначе выход из цикла

Блок-схема цикла со счетчиком N к примеру 8



Пример 9

Разработать программу, которая определяет первый отрицательный элемент последовательности значений функции sin x при заданных шаге h и диапазоне изменения x [a, Ь].

Вариант с использованием п р о ц е д у р ы break:

```
Program primer_9;
var i, n : integer;   x, y, a, b, h : real;
Begin
Write (‘Введите a, b, h:');
Readln(a, b, h);
n:=round((b-a)/ h + 1.5); {определяем количество элементов}
x := a;
for i : = 1 to n do
   begin
     y: =sin(x);
     If y < 0 then
       begin
         Writeln( ‘y= ', y:8:6, ' при x= ‘,  x:6:3 );
          break; {осуществляем досрочный выход из цикла}
       end;
   end;
{место, куда будет передано управление при выполнении break}
If y> 0 then WriteLn ('Элемент не найден,');
Readln;
End.
```

Для операторов ***repeat*** и ***while*** значения параметра цикла (переменной, входящей в условие) должны изменяться в теле цикла, иначе цикл никогда не завершится (ситуация зацикливания).

Пр.4 ***s : = 0; x: = 0;***

While x < 10 do begin s : = s + x; x : = x – 1; end;

Пр.5 Найти произведение четных чисел, меньших n:

Readln(n);

p: = 1; k : = 2;

Repeat

p: = p * k;

k : = k + 2;

Until k > n;

Циклы с операторами ***repeat*** и ***while*** часто заменяют в Паскале конструкцию

If <условие> then goto <metka> , особенно, если переход по метке должен осуществляться к операторам, расположенным выше. Такие конструкции в Паскале крайне нежелательны.

Пр.6 Реализовать фрагмент контроля правильности ввода исходных данных.

```
Repeat
Write( 'введите a, b');
Readln (a, b); Writeln( 'a=', a :6:2, 'b=', b :6:2);
Readln ('данные введены верно? (1/0' );
Readln (c);
Until c = 1;
```

К циклам с неизвестным числом повторений относятся, в частности, ***итерационные циклы***, в которых происходит последовательное приближение к результату с заданной точностью.

Пример 10.

Вычислить сумму бесконечного ряда

$$x - x^2/2 + x^3/3 - x^4/4 + x^5/5 - \ldots\ldots\ldots\ldots + x^i/i - \ldots\ldots\ldots$$

с заданной точностью **ε** для | **x** | **< 1.** Вычисление суммы заканчивается, когда очередной член ряда становится по модулю меньше точности.

Математическая постановка

а) Обозначение переменных:

x – аргумент; **s** – значение суммы;

e – значение точности (погрешности);

m – член ряда;

p – промежуточная переменная;

i – номер члена ряда;

б) Тип переменных:

i – простая переменная целого типа;

x, s, e, m, p – простые переменные вещественного типа.

в) Классификация по группам:

исходные данные: **x, e**;

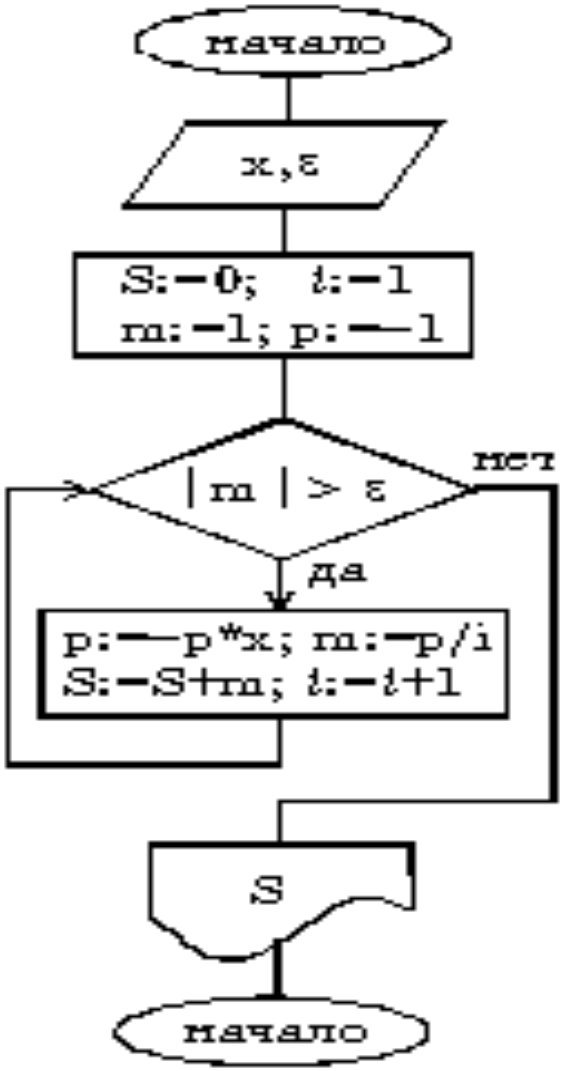
промежуточные результаты**: i, p, m**;

результат: **s**.

г) запись расчетных формул:

```
S = 0
m  = 1
p = -1
i  = 1
если abs (m) >  e выполнить действия
    p = - p * x
    m  = p / i
    s  = s + m
     i = i + 1
```

блок-схема алгоритма примера 9



```
Program primer_10;
Uses CRT;
Var x, s, e, m, p: real;
i : integer;
begin
Clrscr; {очистка экрана}
write (‘x=’); readln (x); {ввод значения x}
write (‘e=’); readln (e); {ввод значения точности}
S : = 0;
m : = 1;
p := -1;
i : = 1; {начальные  значения перед циклом}
 while  abs (m) >  e do    {начало цикла}
   begin
     p := - p * x;  m : = p / i  ; {  вычисление члена ряда}
    s : = s + m;
    i : = i + 1;
   end;
Writeln ( ‘s = ‘, s : 14 :8); {вывод суммы на экран}
Writeln ( ‘количество членов ряда = ’,  i );
readln;
end.
```

Вопросы?