Министерство образования Российской Федерации
Владимирский государственный университет
Кафедра физики и прикладной математики

БАЗОВЫЕ СТРУКТУРЫ АЛГОРИТМОВ И PASCAL 7 ДЛЯ НАЧИНАЮЩИХ

Методические указания
к практическим занятиям по курсу
«Языки программирования»

Составители
Е.В. ХМЕЛЬНИЦКАЯ
С.В. РОЩИН

Владимир 2002

УДК 681.3 (07)

Рецензент:
Кандидат технических наук, доцент, зав. кафедрой «Информатика и защита информации» Владимирского государственного университета
М.Ю. Монахов

Печатается по решению редакционно-издательского совета
Владимирского государственного университета.

Базовые структуры алгоритмов и Pascal 7 для начинающих: Метод. указания к практическим занятиям по курсу «Языки программирования» / Владим. гос. ун-т; Сост.: Е.В. Хмельницкая, С.В. Рощин. Владимир, 2002. 32 с.

Направлены на приобретение практических навыков по алгоритмизации простейших задач и программированию на языке Pascal. Рассматриваются основные алгоритмические структуры и операторы языка, дается краткое описание языка Pascal.

Предназначены для студентов специальности 010200 – прикладная математика и информатика, также могут быть полезны студентам других специальностей, абитуриентам, учащимся старших классов общеобразовательных школ.

Табл. 3. Ил.14. Библиогр. 5 назв.

УДК 681.3 (07)

Предисловие

Решение научных, инженерных, экономических задач сегодня невозможно без эффективного использования ЭВМ и языков программирования высокого уровня. В связи с этим важной задачей общеобразовательной школы и вуза следует считать подготовку учащихся по дисциплине «Информатика».

Брошюра может быть использована в качестве методического пособия при изучении курса по программированию, для самостоятельного овладения основами алгоритмизации и программирования на языке Pascal, что полностью согласуется с современным подходом к обучению.

В методических указаниях нашла свое отражение методика преподавания информатики, успешно используемая авторами во Владимирском государственном университете. Ценность данной методики заключается в том, что базовые алгоритмические структуры изучаются одновременно с операторами языка, им соответствующими. Такой подход способствует быстрому установлению в сознании обучающегося связей между блок-схемой алгоритма решаемой задачи и написанием программы на языке программирования. Кроме этого построение всех алгоритмических структур изучается на конкретных простых примерах, что делает излагаемый материал понятным и доступным даже для тех, кто сталкивается с алгоритмическими языками впервые.

Данное издание будет полезно школьникам, студентам, преподавателям, начинающим пользователям.

1. Основные алгоритмические структуры

1.1. Алгоритмы

Решение задач на ЭВМ состоит из нескольких этапов:

1. Постановка задачи.
2. Математическая формулировка задачи.
3. Создание алгоритма решения задачи.
4. Написание программы на языке программирования.
5. Ввод входных данных.
6. Отладка программы.
7. Получение и обработка результатов.

Под **алгоритмом** мы будем понимать четкое описание последовательности действий, которые необходимо выполнить для решения поставленной задачи. Разрабатываемый алгоритм при этом должен быть универсальным (т.е. результат должен получаться при различных исходных данных), результативным (т.е. результат должен получаться через конечное число шагов), определенным (т.е. каждое предписание должно восприниматься однозначно), дискретным (т.е. алгоритм должен состоять из определенного набора отдельных шагов).

Алгоритм решения задачи можно записать в словесной форме, в виде блок-схемы или на алгоритмическом языке. Познакомимся с основными блоками, которые мы будем использовать в **блок-схемах** алгоритмов (рис.1).

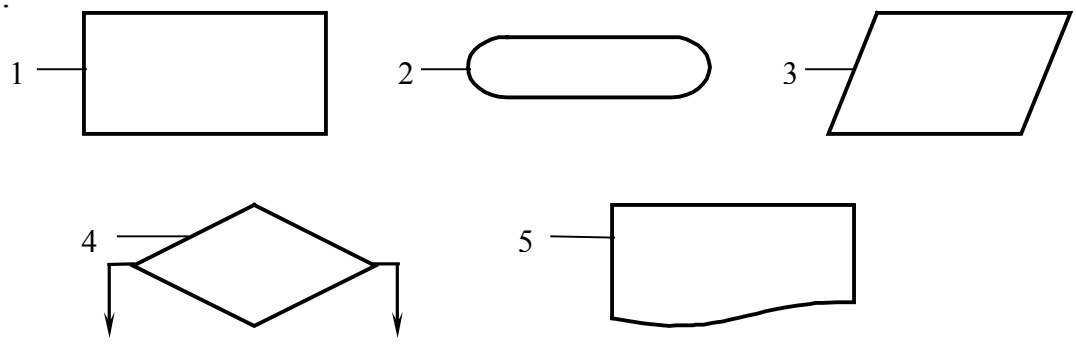


Рис. 1

Прямоугольник (1), соотношение ширины и высоты 2:1, используют для изображения **этапа вычислений** (обработки); внутри прямоугольника записывается содержание этого этапа. Прямоугольник (2) со скругленными

краями, соотношение ширины и высоты 4:1, используется для изображения **начала и конца** алгоритма. Внутри блока пишут слова «начало» или «конец». Параллелограмм (3), соотношение ширины и высоты 2:1, так изображается **ввод данных**. Внутри блока перечисляют имена вводимых переменных. Прямоугольник (5), одна сторона которого – линия обрыва, соотношение ширины и высоты 2:1, используется для изображения **вывода данных**. Внутри блока указывают имена выводимых переменных. Ромб (4), соотношение диагоналей 2:1, так изображается **проверка условия**. Внутри блока указывают проверяемое условие, в результате проверки осуществляется выбор одного из двух возможных путей вычисления.

Чтобы записать алгоритм решения задачи на алгоритмическом языке, необходимо знать его простейшие конструкции, правила записи программы, структуру ее текста.

1.2. Простейшие конструкции языка Pascal

Программа на языке Pascal состоит из описательной и операторной (выполняемой) частей. Текст программы начинается с заголовка, где используют ключевое слово **Program**:

Program <Имя программы>; .

Далее идут разделы объявлений, начало каждого раздела обозначают одним из зарезервированных слов: **Const** (константы), **Var** (переменные), **Label** (метки), **Type** (типы), **Function** (функция), **Procedure** (процедура). Объявление переменных строится по схеме:

Var <Список имен переменных> : <Тип>; .

Объявим, что в программе будут использоваться переменные *x*, *y*, *z* целого типа:

Var *x,y,z*: Integer; .

Пока мы будем работать лишь с двумя типами:

Integer – целый (числа в диапазоне от –32768 до 32767);

Real – вещественный (числа в диапазоне от $2{,}9\cdot10^{-39}$ до $1{,}7\cdot10^{38}$).

После разделов объявлений записывают выполняемую часть программы, которая представляет собой составной оператор. Начало и

конец этой части обозначают операторными скобками **Begin – End**. В конце текста программы (после End) ставится точка. Выполняемую часть программы называют еще операторной. **Оператор** – это запись действия, его заканчивают точкой с запятой.

Оператор присваивания служит для вычисления значения выражения и присваивания его имени результата. Общий вид оператора:

<Имя переменной> **:=** <Выражение соответствующего типа>; .

Запишем на языке Pascal фразу «переменной *z* присвоить значение выражения *r*3,14»:

z:=*r*3.14; .

Запись **арифметических операций**:

Сложение – «+»; Умножение – «*»;

Вычитание – «-»; Деление – «/»;

Деление нацело, с отбрасыванием остатка – «**div**»;

Нахождение остатка от деления нацело –«**mod**» .

Сложение и вычитание имеют низший приоритет, т.е. выполняются в последнюю очередь. Умножение, деление и вычисление функций будут выполняться в порядке их записи.

Основные математические функции и ключевые слова языка Pascal приведены в табл. 1, 2.

Таблица 1

Математические функции в Pascal

Функция	Математический смысл
Abs(*x*)	Абсолютная величина *x*
Sqr(*x*)	Квадрат числа *x*
Sqrt(*x*)	Квадратный корень из числа *x*
Exp(*x*)	Экспонента e^x
Ln(*x*)	Натуральный логарифм *x*
Sin(*x*)	Синус *x*
Cos(*x*)	Косинус *x*
Arctan(*x*)	Арктангенс *x*

Для возведения модуля вещественного числа x, отличного от нуля, в вещественную степень q используют выражение: x^q=**Exp(*a*Ln(Abs(*x*)))**.

$$\log_b a = \frac{\ln a}{\ln b}; \qquad \lg x = \frac{\ln x}{\ln 10}.$$

Таблица 2

Ключевые слова в Pascal

Слово	Перевод
And	И
Array	Массив
Begin	Начало
Do	Выполнять
Downto	Уменьшать
End	Конец
File	Файл
For	Для
Goto	Перейти на
If	Если
In	Включение
Not	Не
Of	Из
Or	Или
Repeat	Повторять
Then	То
To	До
Until	До
While	Пока
With	С

Знаки **операций отношения**:

> (больше); < (меньше); <= (меньше или равно);

>= (больше или равно); = (равно); < > (не равно).

Другие **символы**, используемые в Pascal:

1. Буквы: 26 латинских букв.
2. Цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
3. Скобки: «(», «)», «[», «]», «{», «}».

1.3. Линейные алгоритмы

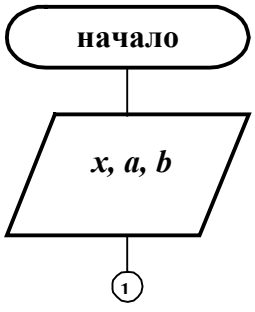
Линейные алгоритмы – это такая вычислительная структура, при которой все предписания выполняются последовательно друг за другом, в порядке, заданном схемой. Такой порядок выполнения называется естественным. В чистом виде линейные алгоритмы встречаются редко, но как фрагменты они присутствуют во всех программах.

Пример1. Составить линейный алгоритм и программу для вычисления выражения $y = \dfrac{b^2 - e^x}{\sqrt{14a}}$ при x = -2,4; a = 3,05; b = 3,02 .

Запишем **алгоритм решения в словесной форме**:

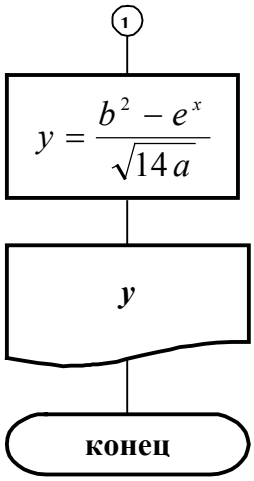
1. Ввести исходные данные для x, a, b.
2. Вычислить y по предложенной формуле.
3. Записать результат (y =).

Блок-схема алгоритма (рис.2):



Программа:

```
Program primer1;
Var x,a,b,y: Real;
Begin
    Writeln ('введите a,b,x');
    Readln (a,b,x);
```




```
y:=(sqr(b)-exp(x))/sqrt(14*a);

Writeln ('y=',y);
Readln;
End.
```

Рис. 2

Общий вид **операторов ввода** с клавиатуры:

Read (<Список объектов ввода>);

Readln (<Список объектов ввода>).

Оператор ввода вызывает приостановку программы, переключение на экран пользователя, ожидание ввода значения. После набора необходимых значений и нажатия клавиши Enter происходит переключение на текст программы, продолжается ее работа. Оператор Readln после набора значения последней переменной обеспечивает переход к началу новой строки файла. В нашем примере для ввода значений переменных служит оператор Readln (*a*,*b*,*x*), вводимые значения для *a*, *b*, *x* можно набирать в одной строке, разделяя пробелами, затем нажать Enter либо нажимать Enter после набора каждого значения.

Общий вид **операторов вывода** на дисплей:

Write (<список объектов вывода>);

Writeln (<список объектов вывода>).

Оператор Write размещает выводимые значения в одной строке, оператор Writeln после вывода последнего значения осуществляет переход к новой строке. В нашем примере первый оператор Writeln (‘введите *a*,*b*,*x*’) выведет на экран фразу, заключенную в апострофы, второй оператор Writeln (‘*y*=’,*y*) сначала выведет фразу в апострофах ‘*y*=’, затем само значение переменной *y*, которое ей было присвоено при выполнении программы.

Записанный в конце программы оператор Readln задерживает экран пользователя, не дает ему переключаться на текст программы, чтобы можно было сразу увидеть результаты. Для возвращения к тексту программы нужно нажать Enter. Чтобы увидеть результат еще раз, нажимают одновременно клавиши Alt и F5.

1.4. Разветвленные алгоритмы

Разветвленные алгоритмы – это такая вычислительная структура, которая содержит не одну, а несколько возможных ветвей решения, в структуре алгоритма есть хотя бы одна операция сравнения, т.е. в зависимости от выполнения некоторого логического условия вычислительный процесс осуществляется по одной или другой ветви.

Реализовать такую структуру можно с помощью **условного оператора**. Он имеет два вида записи:

If <Логическое выражение> **Then** <Оператор 1>;

If <Логическое выражение> **Then** <Оператор 1> **Else** <Оператор 2>; .

Если значение <Логическое выражение> – истина, выполняется <Оператор 1>, в противном случае (Else) – <Оператор 2>.

Пример 2. Составить разветвленный алгоритм и программу для вычисления функции

$$y = \begin{cases} 4\sin x - 1 & \text{при } x<1; \\ \ln x + 5 & \text{при } x \geq 1. \end{cases}$$

Запишем **алгоритм решения в словесной форме**:

1. Ввести x.
2. Сравнить x с единицей.
3. Вычислить y по одной из предложенных формул, в зависимости от результата сравнения.
4. Записать результат.

Блок-схема алгоритма (рис.3):

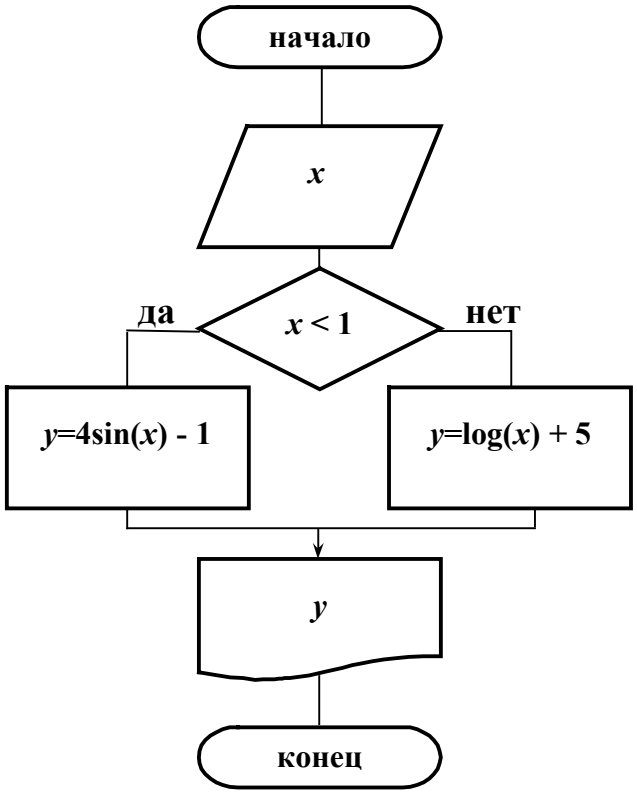


Рис.3

Программа:

```
Program primer2;
Var x,y: Real;
 Begin
    Writeln ('введите x');
     Readln (x);

 If x<1 Then y:=4*sin(x)-1
        Else y:=ln(x)+5;

    Writeln  ('y=',y);
 End.
```

В структуре алгоритма может присутствовать несколько операций сравнения, разветвления на разное количество ветвей. Рассмотрим такой пример.

Пример 3. Составить разветвленный алгоритм и программу для вычисления функции

$$y = \begin{cases} t^2 - 21t + 63 & \text{при } t \le -23; \\ t^3 - 3t^2 + 2\operatorname{tg} t & -23 < t < -3; \\ 2{,}13 & -3 < t \le 0; \\ 12{,}65t - t^2 & t > 0. \end{cases}$$

Алгоритм в словесной форме:

1. Ввести t.
2. Последовательно сравнить t с величинами -23, -3, 0.
3. В зависимости от результата сравнения вычислить y по одной из предложенных формул, учесть, что в точке -3 функция не определена.
4. Записать результат.

Блок-схема алгоритма (рис.4):

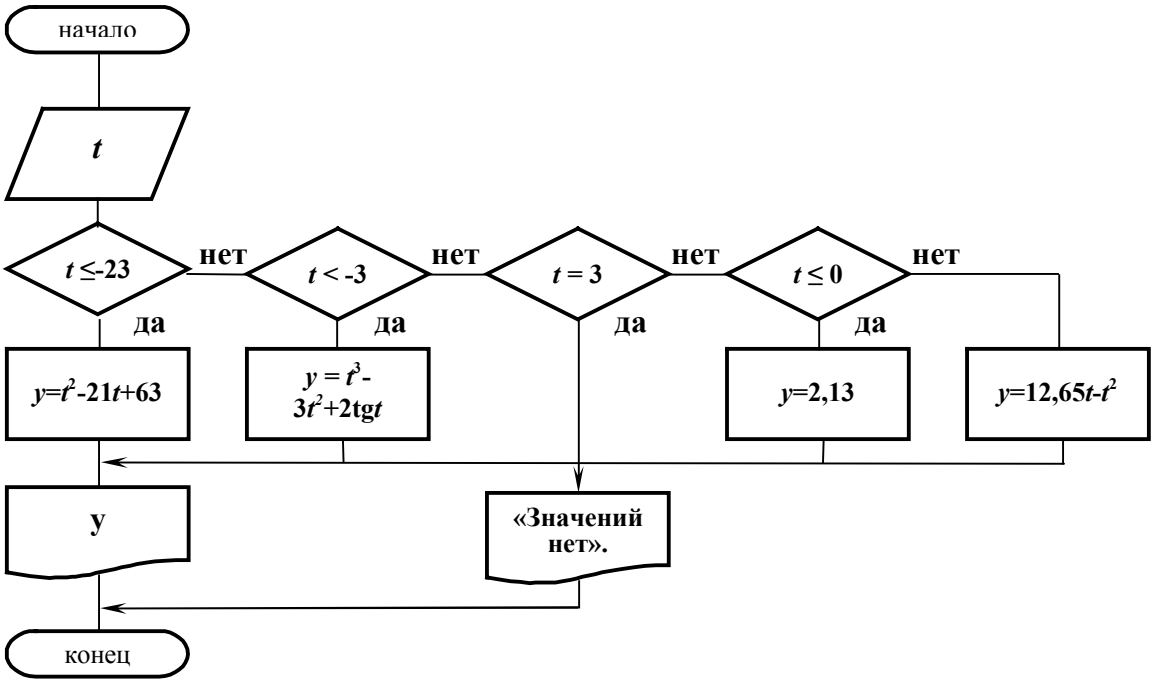


Рис.4

Программа:

```
Program primer3;
Var t,y: Real;
Begin
        Writeln (‘Введите t’);
         Readln (t);
         If  t = 3  Then  Writeln (‘Значений нет’)  Else
         Begin
                If  t <= -23  Then  y:=Sqr(t)-21*t+63  Else
                If  t < -3  Then  y:=Sgr(t)*t-3*Sqr(t)+2*Sin(t)/Cos(t)  Else
                If  t <= 0  Then  y:=2.13  Else  y:=12.65*t- Sqr(t);
               Writeln (‘y=’,y);
          End;
End.
```

1.5. Простые циклы

Простые циклы – это такая вычислительная схема разветвленной структуры, в которой одна ветвь операции сравнения является обратной связью на предыдущую часть алгоритма, т.е. идет назад. Таким образом, некоторая последовательность операций алгоритма будет выполняться многократно (в цикле), образуя тело цикла. Так можно многократно вычислять значения выражений по одним и тем же математическим зависимостям для различных значений входящих в них величин. Использование циклов позволяет сократить объем схемы алгоритма и длину программы.

Циклы могут быть с заданным и с неизвестным числом повторений. Переменная, которая управляет циклом, называется **параметром цикла**.

При построении любого цикла необходимо предусмотреть три момента:

1) организовать данные для первого цикла;
2) после выполнения цикла надо обновлять данные для очередного цикла;
3) необходимо управлять циклом, т.е. должно быть условие автоматического прекращения циклических вычислений.

Циклические структуры реализуют с помощью **операторов цикла**.

Оператор цикла, управляемого параметром

Используется для организации циклов с известным числом повторений. Общая форма записи при увеличении значения параметра цикла:

For <Параметр цикла> **:=** <Граница1> **to** <Граница2> **do** <Оператор>;

При уменьшении значения параметра цикла:

For <Параметр цикла> **:=** <Граница1> **downto** <Граница2> **do** <Оператор>;

<Параметр цикла> – простая переменная, обычно целого типа. <Граница1>, <Граница2> – это выражения того же типа, что и параметр. При выполнении оператора цикла параметр цикла изменяется от начального своего значения (граница 1) до конечного (граница 2).

Поочередно для всех значений параметра цикла в интервале между границами выполняется тело цикла (<Оператор>). <Оператор> может состоять из нескольких действий (т.е. быть составным), тогда тело цикла необходимо оформить операторными скобками Begin-End, иначе повторяться будет только первое действие.

Оператор, реализующий цикл с предусловием

Используется для организации циклов с неизвестным числом повторений. Общая форма записи оператора:

While <Логическое выражение> **do** <Тело цикла>;

Данный оператор содержит условие входа в цикл, потому возможны случаи, когда тело цикла не выполнится ни разу. Значения переменных, входящих в условие должны изменяться в теле цикла. Если тело цикла представлено составным оператором, тогда его оформляют операторными скобками Begin-End.

Оператор, реализующий цикл с постусловием

Используется для организации циклов с неизвестным числом повторений. В отличие от оператора While проверка условия производится в конце оператора, поэтому тело цикла в любом случае выполнится хотя бы раз. Общая форма записи оператора:

Repeat <Тело цикла> **Unil** <Логическое выражение>;

Данный оператор описывает повторяемые действия (<Тело цикла>) и содержит условие прекращения повторений (<Логическое выражение>). Слова «Repeat-Until» выполняют одновременно роль операторных скобок Begin-End.

Разберем применение операторов цикла на конкретных примерах.

Пример 4. Составить простой цикл для вычисления заданной функции

$$y = 4\sin^2 x + 1 \ ,$$

если $x_0 = 1{,}5$; $x_k = 4$; $\Delta x = 0{,}5$ (шаг изменения переменной 0,5).

При выполнении задания мы должны многократно вычислять y по предложенной формуле, каждый раз с новым значением переменной x. Переменная x будет меняться от своего начального значения (x_0 = 1,5) до конечного (x_k = 4), при каждом вычислении ее значение должно возрастать на 0,5. Условием прекращения вычислений будет достижение переменной x своего конечного значения, при этом не важно, сколько раз мы вычислим функцию y (т.е. количество циклов нам не известно).

Запишем **алгоритм в словесной форме**:

1. Присвоить переменной x значение x_0.
2. Вычислить y.
3. Вывести результаты (x =, y =).
4. Изменить текущее значение переменной x на величину 0,5.
5. Сравнить текущее значение переменной x с x_k.
6. В зависимости от результата сравнения снова вычислять y и выводить результаты, либо закончить процесс вычислений, если x = 4.

Блок-схема алгоритма (рис.5):

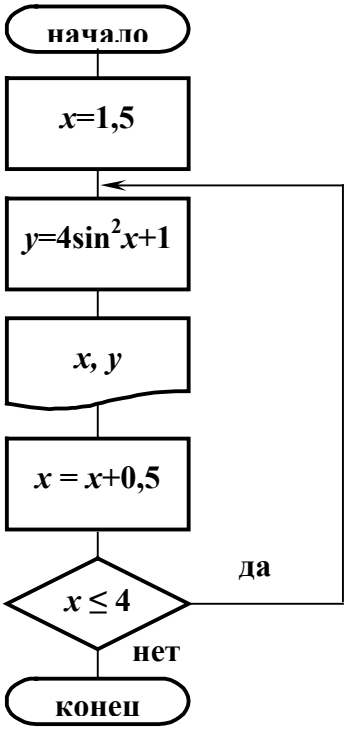


Рис. 5

Программа:

```
Program primer4;
Var x,y: Real;

Begin
   x:=1.5;
   Repeat

        y:=4*sqr(sin(x))+1;

        Writeln ('x=',x,'  y=',y);

        x:=x+0.5;

   Until x > 4;

End.
```

Фразу «Repeat … Until $x > 4$» можно перевести так: повторяй до тех пор, пока x не станет больше, чем 4. В данном примере переменная x является **параметром цикла**, она им управляет.

В блок-схемах алгоритмов все подготовительные блоки цикла (присвоение параметру начального значения, изменение параметра, сравнение его с конечным значением) можно изображать в виде одного блока, предшествующего телу цикла. Этот блок имеет специальную структуру в виде шестиугольника (рис.6):

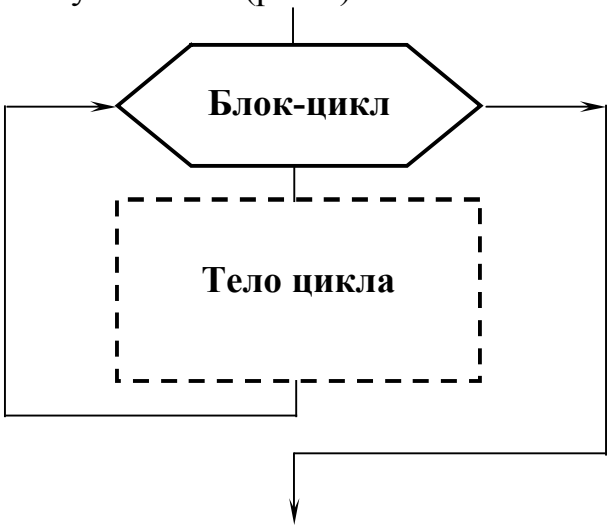


Рис. 6

Внутри **блока-цикла** указывается параметр цикла, его начальное, конечное значение и шаг изменения. Построим блок-схему алгоритма для примера 4, используя блок-цикл (рис.7):

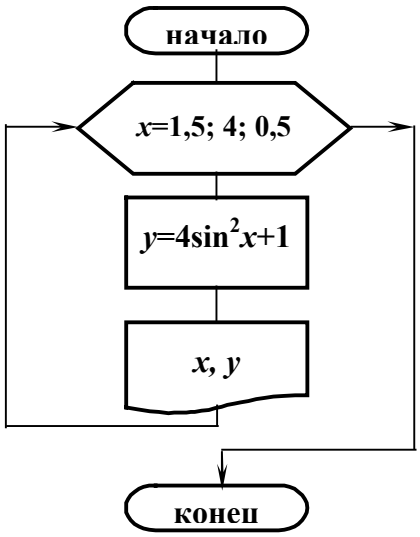


Рис. 7

Запись внутри блока-цикла звучит так: сделай все операции, которые записаны ниже (в теле цикла), много раз, пока параметр x меняется от 1,5 до 4 с шагом 0,5.

Пример 5. Составить простой цикл для вычисления заданной функции

$$y = 4\sin^2 x + 1,$$

если $x_0 = 1{,}5$; $\Delta x = 0{,}5$ (шаг изменения переменной 0,5); $n = 6$ (количество расчетных точек 6).

При выполнении задания мы должны 6 раз вычислить y по предложенной формуле, каждый раз с новым значением переменной x. Переменная x будет изменяться от своего начального значения ($x_0 = 1{,}5$) с шагом 0,5. Но управлять циклом переменная x не может, так как мы не знаем ее конечного значения. Зато мы знаем, что должны провести вычисления 6 раз, т.е. после шести циклов их необходимо закончить. Здесь удобно ввести дополнительную переменную (например, i), которую мы будем называть **счетчиком циклов** (это параметр для нашего цикла) и которая будет менять свое значение от 1 до 6 с шагом 1.

Блок-схема алгоритма (рис. 8):

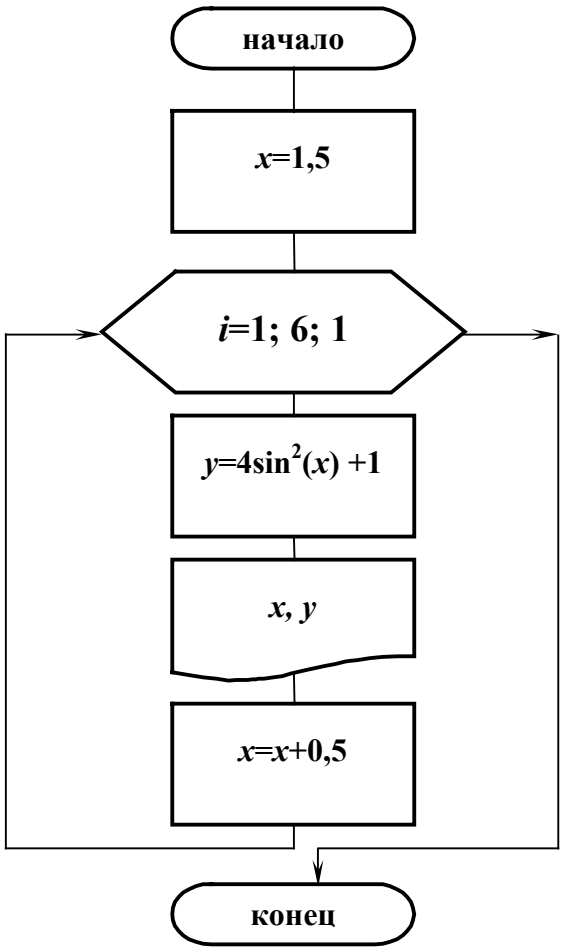


Рис.8

Программа:

```
Program primer5;
Var x,y: Real;
      i: Integer;
Begin
x:=1.5;
For i:=1 to 6 do
    Begin
       y:=4*Sqr(Sin(x))+1;
      Writeln ('x=',x,'  y=',y);
       x:=x+0.5;
    End;
End.
```

1.6. Вложенные циклы

Вложенные циклы – это такие циклические алгоритмы, которые имеют несколько параметров (более одного), каждый из которых меняется по своему закону. Цикл, который содержит другие циклы (**внутренние**), называют **внешним**. Количество внешних циклов, которые охватывают данный цикл, называют **глубиной вложения** цикла. Глубина вложения практически не ограничена. Организация внешних и внутренних циклов осуществляется по тем же правилам, которые мы рассматривали для простых циклов. Параметры внешнего и внутреннего циклов должны быть разными и изменяться не одновременно. Таким образом, при одном значении параметра внешнего цикла параметр внутреннего цикла принимает поочередно все значения.

Пример 6. Составить алгоритм с вложенными циклами и программу для вычисления функции

$$y = \frac{\ln|x+2|}{x+a^2},$$

если $a_0 = -1{,}2;$ $x_0 = -8;$ $\Delta a = 0{,}8;$ $\Delta x = 0{,}5;$ $a_k = 1{,}2;$ $x_k = -4.$

Таким образом, мы видим, что y – функция двух аргументов, она зависит от переменных x и a. Наша задача последовательно проследить изменение переменных x и a, не пропустить ни одной точки, где должна вычисляться функция y. Пусть a меняется в первую очередь, т.е. будет внутренним аргументом и будет управлять внутренним циклом. Тогда переменная x будет образовывать внешний цикл.

Сначала аргументу x мы дадим значение x_0 и вычислим y несколько раз, пока a меняется от a_0 до a_k. Затем изменяем аргумент x на Δx и снова вычисляем y при условии, что a меняется от a_0 до a_k, и так далее, пока аргумент x не перерастет свое конечное значение. Фактически мы должны заполнить таблицу (табл. 3) из значений аргументов и соответствующих им значений функции. Такой процесс называют еще **табулированием** функции.

Таблица 3

x	-8	-8	-8	-8	-7,5	-7,5	-7,5	-7,5	-7	-7	-7	-7
a	-1,2	-0,4	0,4	1,2	-1,2	-0,4	0,4	1,2	-1,2	-0,4	0,4	1,2
y	…	…	…	…	…	…	…	…	…	…	…	…

Запишем **алгоритм в словесной форме**:

1. Присвоить переменной *x* значение x_0 .
2. Присвоить переменной *a* значение a_0 .
3. Вычислить *y*.
4. Вывести результаты (*x*=, *a*=, *y*=).
5. Изменить текущее значение *a* на величину Δ *a*.
6. Сравнить *a* с a_k. Если значение *a* еще не стало больше a_k , то продолжать внутренний цикл по *a*.
7. Если значение *a* стало больше a_k, то изменить текущее значение *x* на величину Δ *x*.
8. Сравнить *x* с x_k. В зависимости от результата сравнения продолжать циклические вычисления либо закончить их.

Предложенный алгоритм можно записать короче и реализовать его в виде блок-схемы, используя блок-цикл (рис.9):

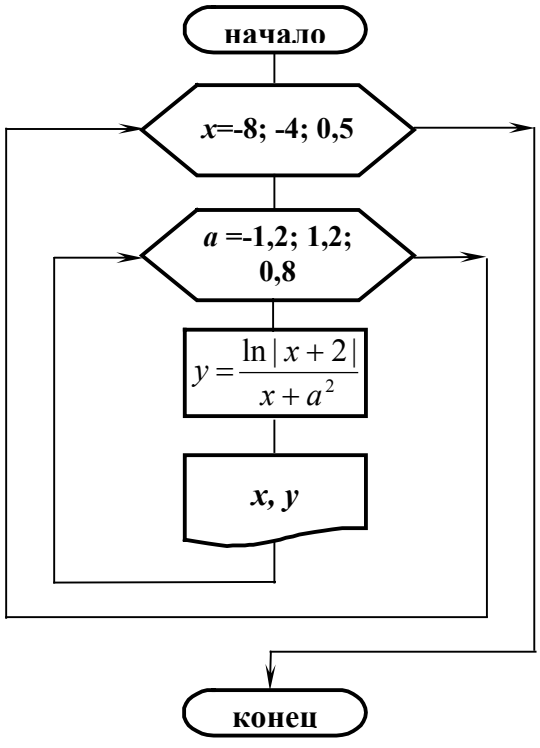


Рис. 9

Данная запись звучит так: сделай все операции, которые записаны ниже (в теле цикла), много раз, пока параметр *x* меняется от -8 до -4 с шагом 0,5; а также пока параметр *a* меняется от -1,2 до 1,2 с шагом 0,8.

Программа:

```
Program primer6;
Var x,a,y:Real;
Begin
        x:=-8;
        Repeat
                a:=-1.2;
                Repeat
                        y:=Ln(Abs(x+2))/(x+Sqr(a));
                        Writeln ('x=',x,'  a=',a,'  y=',y);
                        a:=a+0.8;
                Until a > 1.2;
        x:=x+0.5;
        Until x > -4;
End.
```

2. Работа с массивами в Turbo Pascal 7.0

2.1. Понятие массива

Данные, обрабатываемые при решении задач с применением ЭВМ, в большинстве случаев взаимосвязаны и образуют структуры различной сложности, которые в зависимости от типа элементов можно отнести к тому или иному структурированному типу данных. К наиболее часто используемым структурам данных относятся массивы и записи.

Массив элементов представляет собой совокупность значений одинакового типа.

Каждый элемент массива имеет свой уникальный номер, который может состоять из одного или нескольких индексов в зависимости от **размерности** массива.

Математическим эквивалентом массивов является математический объект – **матрица**. Как известно из математики, матрицей называется система чисел, образующих прямоугольную таблицу из m строк и n столбцов:

$$\|A\| = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

Приведённая матрица является примером двумерного массива, каждый элемент которого a_{ij} имеет два индекса: номер строки i и номер столбца j. Таким образом, для того чтобы обратиться к элементу массива, который расположен во второй строке и в первом столбце (a_{21}), необходимо задать значения индексов $i = 2,\ j = 1$.

Примером одномерного массива может служить **вектор** – матрица, состоящая из одной строки и n элементов:

$$\|A\| = [a_1, a_2, \cdots, a_n].$$

В этом случае каждый элемент массива имеет только один индекс, например, a_2, a_4.

2.2. Синтаксис описания массивов в языке Pascal

Для описания массивов в языке Pascal используется следующая конструкция:

```
array [<index-type>] of <element-type>;
```

Здесь <index-type> - описание индексов массива. Допускается описывать более одного индекса, при этом они должны разделяться запятой. При описании индексов в Turbo Pascal 7 допускается использовать только простые типы данных (ordinal). При описании пределов индексов можно использовать нетипизированные константы. Так, следующие примеры показывают допустимые варианты описания индексов массивов:

```
1..100     { целочисленные границы индекса }
'a'..'z'   { литерные (символьные) границы индекса }
False..True  { использование диапазона значений типа
Boolean }
1..MaxInt  { использование константы в качестве
верхней границы индекса }
MinIndex..MaxIndex   { использование констант для
описания границ индекса }
```

Конструкция <element-type> при описании массивов определяет тип элементов массива. Допускается использовать любые типы данных, поддерживаемые Turbo Pascal 7.

Приведём несколько примеров описания массивов:

- одномерные массивы:

```
{ массив из десяти элементов типа Integer }
A: array [1..10] of Integer;
{ массив из 26-ти элементов типа Char }
B: array ['a'..'z'] of Char;
{ массив из двух элементов типа String }
C: array [False..True] of String;
```

- двумерные массивы:

```
{ массив 10x15 элементов типа Integer }
A: array [1..10, 1..15] of Integer;
{ массив 10x15 элементов типа Integer }
B: array [1..10] of array [1..15] of Integer;
{ массив 5x26 элементов типа Byte }
C: array [1..5, 'a'..'z'] of Byte;
```

- массивы с размерностью три и выше:

```
{ трёхмерный массив из 100 элементов типа String }
A: array [1..2, 1..5, 1..10] of String;
{ чётырёхмерный массив из элементов типа Word }
B: array [False..True, 'a'..'z'] of array [1..10,
1..5] of Word;
```

2.3. Ввод/вывод элементов массивов

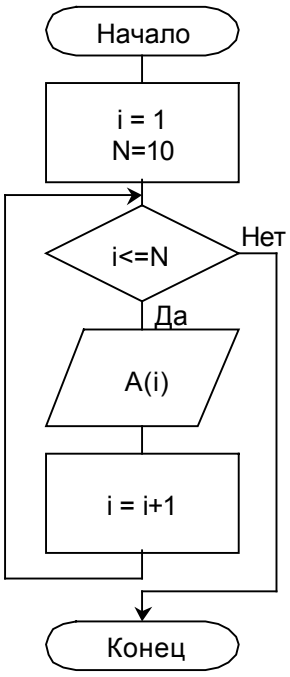
При работе с массивами часто необходимо организовывать ввод или вывод значений элементов. Ввиду того, что, например, при вводе значений элементов массива многократно выполняется одна и та же операция, то удобно алгоритмически реализовать весь процесс в виде цикла (обычно это цикл с предусловием).

Приведём примеры алгоритмов, демонстрирующих ввод/вывод значений элементов массивов. Далее при описании других алгоритмов обработки массивов соответствующие фрагменты блок-схем будут оформляться с использованием блока «подпрограмма», которому будет соответствовать один из описанных далее алгоритмов.

Ввод/вывод значений элементов одномерных массивов

Имеется одномерный массив *A*, состоящий из *N* элементов (пусть, например, *N*=10). Необходимо организовать ввод и вывод значений элементов массива.

Приведённая на рис. 9 блок-схема и программа на языке Pascal реализуют алгоритм ввода/вывода значений элементов одномерного массива. При этом стоит отметить, что блок-схемы алгоритмов ввода и вывода абсолютно идентичны, а приведённая программа демонстрирует одновременно и ввод и вывод значений элементов.




```
program ArrayIO;
const
  N = 10;
var
  A: array [1..N] of Integer;
  I: Integer;
begin
  for I := 1 to N do
  begin
    { Ввод элементов массива }
    Write('A[',I,']=');
    Readln(A[I]);

    { Вывод элементов массива }
    Writeln('A[',I,']=',A[I])
  end;
end.
```

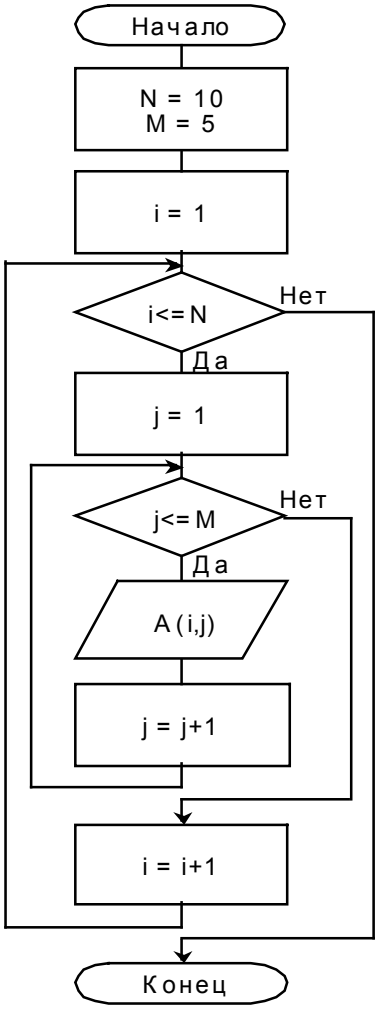
Рис. 9

Ввод/вывод значений элементов двумерных массивов

Имеется двумерный массив *A*, состоящий из *N* строк и *M* столбцов (пусть, например, *N*=10, *M*=5). Необходимо организовать ввод и вывод значений элементов массива.

Как видно из блок-схемы алгоритма, приведённой на рис. 10, для организации ввода/вывода значений элементов двумерного массива необходимо использование вложенных циклов. Нетрудно заметить, что количество вложенных циклов при этом равно размерности массива, т.е.

для трёхмерного массива потребуется три вложенных цикла, четырёхмерного – четыре и т.д.




```
program ArrayIO;
const
  N = 10;
  M = 5;
var
  A:array[1..N,1..M] ofInteger;
  I, J: Integer;
begin
  for I := 1 to N do
    for J := 1 to M do
    begin
      {Ввод элементов массива}
      Write('A[',I,',',J,']=');
      Readln(A[I,J]);

      {Вывод элементов массива}

Writeln('A[',I,',',J,']=',A[I,J])
    end;
end.
```

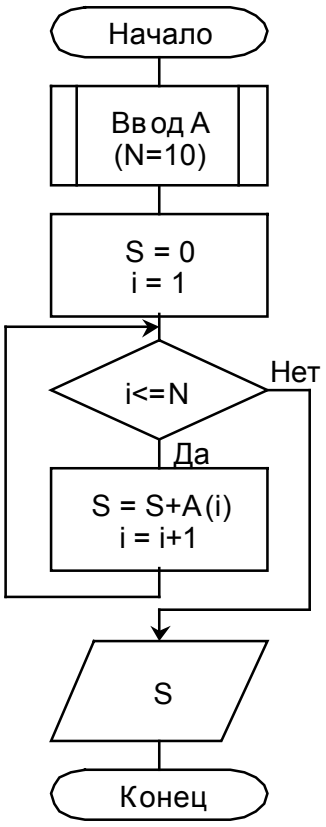
Рис. 10

2.4. Суммирование элементов массива

Задача суммирования элементов массива подразумевает нахождение арифметической суммы значений всех элементов массива *A*.

Пусть необходимо найти сумму элементов массива *A*, состоящего из *N* элементов целочисленного типа (пусть, например, *N*=10).

Далее представлены блок-схема алгоритма (рис. 11) и программа на языке Pascal, реализующие решение задачи.




```
program ArraySum;
const
  N = 10;
var
  A: array [1..N] of Integer;
  I, S: Integer;
begin
  { Ввод элементов массива }
  for I := 1 to N do
  begin
    Write('A[',I,']=');
    Readln(A[I])
  end;

  { Суммирование элементов }
  S := 0;
  for I := 1 to N do
    S := S + A[I];

  { Вывод результата }
  Writeln('S=', S)
end.
```

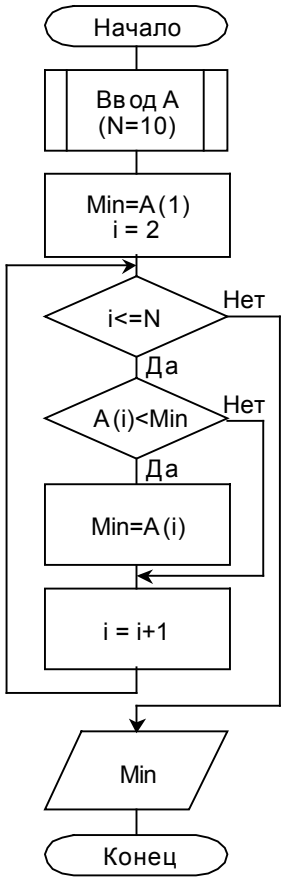
Рис. 11

2.5. Поиск минимального элемента массива

Поиск минимального (максимального) элемента – одна из задач, наиболее часто встречающаяся на практике. Алгоритм поиска минимального элемента применяется, например, при сортировке элементов массивов по возрастанию.

Пусть необходимо найти элемент массива *A* с минимальным значением. Массив состоит из *N* элементов целочисленного типа (пусть, например, *N*=10).

Далее представлены блок-схема алгоритма (рис. 12) и программа на языке Pascal, реализующие решение задачи.




```
program ArrayMin;
const
  N = 10;
var
  A: array [1..N] of Integer;
  I, Min: Integer;
Begin
  { Ввод элементов массива }
  for I := 1 to N do
  begin
    Write('A[',I,']=');
    Readln(A[I])
  end;

  {Поиск минимального элемента}
  Min := A[1];
  for I := 2 to N do
if A[I] < Min then Min := A[I];

  { Вывод результата }
  Writeln('Min =', Min)
end.
```

Рис. 12

2.6. Сортировка массивов

Задача сортировки массивов подразумевает упорядочивание элементов массива в порядке возрастания или убывания их значений.

Предположим, что необходимо упорядочить по возрастанию элементы массива *A*. Массив состоит из *N* элементов целочисленного типа (пусть, например, *N*=10).

Приведённый далее алгоритм сортировки (рис. 13) базируется на алгоритме поиска минимального элемента массива с последующей перестановкой значений.

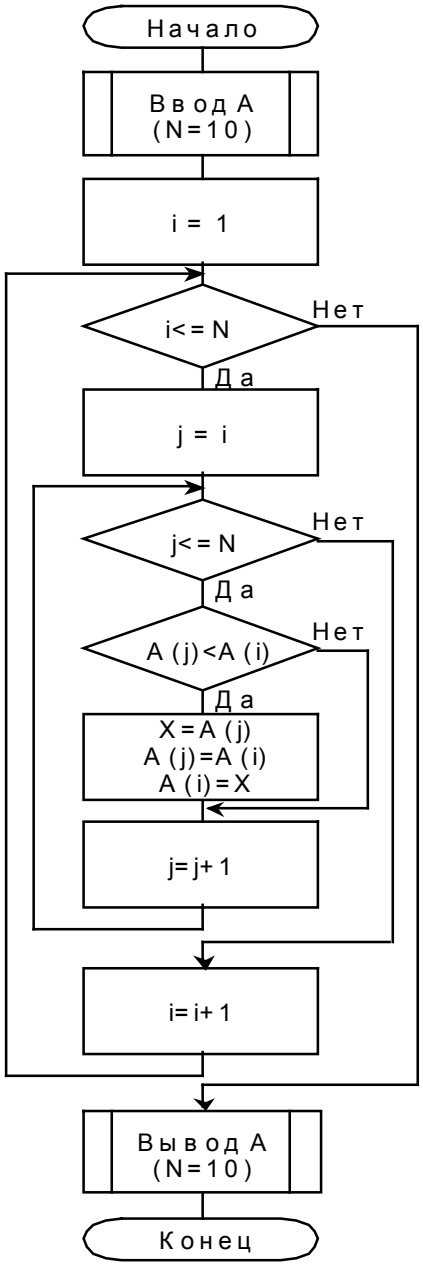


Рис. 13

```
program ArraySort;
const
  N = 10;
var
  A: array [1..N] of Integer;
  I, J, X: Integer;
begin
  { Ввод элементов массива }
  for I := 1 to N do
  begin
    Write('A[',I,']=');
    Readln(A[I])
  end;

{Сортировка элементов массива}
  for I := 1 to N do
    for J := I to N do
      if A[J] < A[I] then
      begin
        X := A[I];
        A[I] := A[J];
        A[J] := X
      end;

  { Вывод элементов массива }
  for I := 1 to N do
    Writeln('A[',I,']=',A[I]);
end.
```

2.7 Сортировка методом «пузырька»

Ещё один алгоритм сортировки элементов массива – метод «пузырька» - также достаточно часто используется на практике при решении задач сортировки (рис 14).

Суть метода заключается в том, что элементы массива попарно сравниваются, и если это необходимо, то их значения меняются местами. При этом происходит своеобразное «вытеснение» элементов с меньшими значениями элементами с большими значениями (в случае сортировки по возрастанию), отсюда и название метода.

Этот процесс ведется до тех пор, пока при «просмотре» всего массива не найдется ни одной пары элементов, значения которых необходимо переставить местами. Это означает, что массив отсортирован.

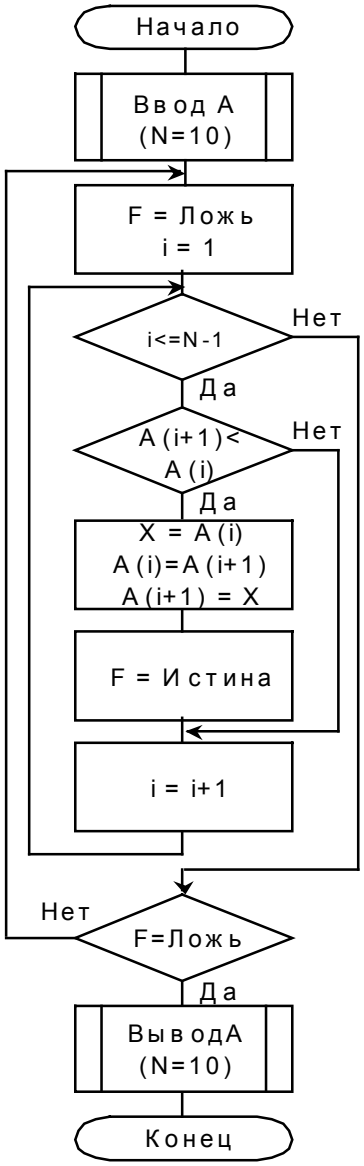


Рис. 14

```
program ArraySort;
const
  N = 10;
var
  A: array [1..N] of Integer;
  I, X: Integer;
  F: Boolean;
begin
  { Ввод элементов массива }
  for I := 1 to N do
  begin
    Write('A[',I,']=');
    Readln(A[I])
  end;

 {Сортировка элементов массива}
  repeat
    F := False;
    for I := 1 to N-1 do
      if A[I+1] < A[I] then
      begin
        F := True;
        X := A[I];
        A[I] := A[I+1];
        A[I+1] := X
      end;
  until F = False;

  { Вывод элементов массива }
  for I := 1 to N do
    Writeln('A[',I,']=',A[I]);
end.
```

Библиографический список

1. *Гусева А.И.* Учимся программировать: PASCAL 7.0 Задачи и методы их решения. – 2-е изд., перераб. и доп. - М.: «Диалог-МИФИ», 1999. – 256 с.
2. *Гусева А.И., Детинин О.О., Детинина О.Н*. Компьютерный учебник «Учимся программировать: Pascal 7.0». – М.: Российский фонд компьютерных программ ИНИНФО МО РФ, N 1725, 1997. – 250 с.
3. *Вирт Н*. Алгоритмы и структуры данных. – М.: Мир, 1989. – 360 с.
4. Вычислительная техника и программирование. Практикум по программированию: Практ. пособие/ В.Е. Алексеев, А.С. Ваулин, Г.Б. Петрова; Под ред. А.В. Петрова. – М.: Высш. шк., 1991. – 400 с.
5. *Зубов В.С.* Программирование на языке TURBO PASCAL. – М.: Информационно-издательский дом «Филинъ», 1997. – 320 с.

Оглавление

БАЗОВЫЕ СТРУКТУРЫ АЛГОРИТМОВ И PASCAL 7 ДЛЯ НАЧИНАЮЩИХ

Методические указания к практическим занятиям по курсу
«Языки программирования»

Составители
ХМЕЛЬНИЦКАЯ Елена Валерьевна
РОЩИН Сергей Васильевич
Ответственный за выпуск – зав. кафедрой профессор С.М. Аракелян

Редактор Р.С. Кузина

ЛР №020275. Подписано в печать 11.02.02.
Формат 60x84/16. Бумага для множит. техники. Гарнитура Таймс.
Печать офсетная. Усл. печ..л. 1,86. Уч.-изд. л. 2,06. Тираж 50 экз.
Заказ
Владимирский государственный университет.
Подразделение оперативной полиграфии
Владимирского государственного университета.
Адрес университета и подразделения оперативной полиграфии:
600000, Владимир, ул. Горького, 87.
E-mail: rio-m2@vpti.vladimir.su