



32.373-01
Г 70
НАРОДНЫЙ УНИВЕРСИТЕТ
естественнонаучный факультет

А.Б.ГОРСТКО, С.В.КОЧКОВАЯ

АЗБУКА ПРОГРАММИРОВАНИЯ

ИНФОРМАТИКА ДЛЯ ВСЕХ

567890

3456789

153

МОСКВА
1988

ЗНАНИЕ

НАРОДНЫЙ УНИВЕРСИТЕТ
естественнонаучный факультет

Издается с 1961 г.

А.Б.ГОРСТКО,

доктор физико-математических наук

С.В.КОЧКОВАЯ,

инженер-программист

АЗБУКА ПРОГРАММИРОВАНИЯ

(Информатика для всех)

Издательство «Знание»

Москва 1988

ББК 32.973-01

Г 70

ГОРСТКО Александр Борисович — лауреат Государственной премии СССР, доктор физико-математических наук, профессор, заведующий кафедрой прикладной математики и программирования Ростовского государственного университета. Является автором более 100 научных и научно-популярных работ в области математического моделирования экономических и экологических систем.

КОЧКОВАЯ Светлана Викторовна — научный сотрудник той же кафедры Ростовского государственного университета, специалист в области системного и прикладного программирования.

Рецензенты: Суворов Б. П. — профессор, Базыкин А. Д. — кандидат физико-математических наук,

Горстко А. Б., Кочкова С. В.

Г 70 Азбука программирования (Информатика для всех). — М.: Знание, 1988. — 144 с. — (Нар. ун-т. Естественнонаучный фак.).

45 к.

200 000 экз.

В век научно-технического прогресса вычислительная техника проникает буквально во все сферы человеческой деятельности. Сегодня ЭВМ стали насущной необходимостью для всех — для рабочего и инженера, научного работника и экономиста.

Книга представляет собой руководство, которое поможет каждому освоить простейшие приемы и навыки программирования, методы решения задач.

Книга иллюстрирована многочисленными примерами из практики и может служить пособием для слушателей народных университетов естественнонаучных знаний.

Г $\frac{2404010000-010}{073 (02)-88}$ 12-88

ББК 32.973-01

ПРЕДИСЛОВИЕ

Современная научно-техническая революция стремительно вовлекает в число пользователей электронно-вычислительных машин (ЭВМ) представителей все новых и новых профессий. Если 15—20 лет назад по преимуществу это были программисты, то сейчас в непосредственном общении с ЭВМ участвует множество экономистов, физиков, химиков, биологов, специалистов многих других областей народного хозяйства.

Появление в программе средней школы курса информатики, проводимая там широкая компьютеризация дополнили число пользователей ЭВМ также целой армией школьников.

Так как работа на ЭВМ требует обязательно владения хотя бы одним алгоритмическим языком, резко возросла потребность в учебной литературе по программированию, причем литературе, ориентированной на различные группы пользователей в зависимости от их подготовки.

Предлагаемая вниманию читателей книга А. Б. Горстко и С. В. Кочковой «Азбука программирования», как явствует из названия, рассчитана на самый широкий круг читателей—будущих пользователей ЭВМ. Избранный авторами стиль изложения материала (от конкретной задачи—к правилу, от русского языка—к языку Паскаль и др.) позволяет совершенно неподготовленному читателю в короткий срок овладеть основами программирования на языке Паскаль и, что, может быть, даже важнее, безбоязненно одолеть психологический барьер, незримо разделяющий посвященных и не посвященных в «секреты ЭВМ».

Не сомневаюсь, что эта книга окажется полезной многим читателям.

Член-корреспондент АН СССР В. Л. Макаров

ОТ АВТОРОВ

С каждым днем все шире используются электронные вычислительные машины (ЭВМ) в самых разнообразных сферах жизни и деятельности человека, даже термин появился—электронизация. Однако сама по себе ЭВМ ничего сделать не может. Она должна работать под руководством квалифицированного человека, который должен уметь общаться с ней.

Общение человека и ЭВМ на естественном человеческом языке—дело более или менее отдаленного будущего. А в настоящее время задания машине должны формулироваться на понятном для нее языке—на языке программирования.

Хотя сейчас насчитывается уже несколько тысяч языков программирования, популярность их неодинакова. Эта книга посвящена универсальному языку программирования Паскаль, разработанному швейцарским ученым Виртом. Присущие **языку** особенности—простота и вместе с тем возможность писать достаточно сложные программы—сделали его **языком** практического программирования—почти все современные ЭВМ «понимают» Паскаль.

Существует достаточное количество руководств по языку Паскаль, в которых последовательно в виде строгих определений излагаются конструкции языка. Поэтому авторы не ставили перед собой задачу сделать еще одно описание всех возможностей языка. Мы хотели создать руководство для широкого круга людей, не имеющих никакого опыта программирования и не знакомых ни с одним из языков программирования. Полагаем, что Азбука поможет и школьникам, и людям почтенного возраста освоить простейшие приемы и приобрести некоторые навыки программирования на Паскале. Она познакомит также со способами постро-

ения алгоритмов для решения различных задач, с другими вопросами, входящими в современную информатику. Все необходимые сведения приведены в совершенно доступной форме.

Существует два основных подхода к изложению материала: «от правила к задаче» и «от задачи к правилу». Первый способ, наиболее распространенный, заключается в том, что сначала описываются основные понятия, конструкции языка, а затем приводятся примеры, демонстрирующие их использование. Во втором случае сначала рассматривается пример, в котором необходимо использовать ту или иную конструкцию, конструкция используется и только затем приводится ее описание. Мы избрали второй способ, так как считаем, что знания неискушенным читателем воспримутся лучше, если будет понятно, для чего они нужны.

Предусмотрена и возможность самостоятельной работы читателя—в книге имеется довольно много задач и упражнений.

Для того чтобы облегчить работу читателя с книгой, мы ввели специальные условные знаки, характеризующие авторские пожелания. Заметьте себе, что

□—правило, которое нужно понять и знать;

|| —совет;

○—упражнение очень желательное;

□—упражнение просто желательное;

||? —вопрос.

Умение общаться с ЭВМ сейчас требуется от все большего и большего числа людей. Мы будем рады, если сможем помочь хотя бы малому их числу в этом непростом, но таком интересном деле.

Нам очень приятно выразить здесь благодарность И. М. Сливняку и Л. А. Чикину, прочитавшим рукопись книги и сделавшим ряд полезных замечаний.

Глава I

ЗАЙМЕМСЯ АРИФМЕТИКОЙ

Чтобы облегчить себе физический труд, люди придумывают и создают различные механизмы и машины. Скажем, ни у кого не возникает сомнений в том, что вырыть, например, котлован гораздо легче с помощью экскаватора, а не лопаты. Но экскаватор сам по себе работать не может, нужно научиться им управлять. Подобным же образом дело обстоит и в сфере интеллектуального труда.

Очевидно, взяв карандаш и бумагу, вы смогли бы найти значение выражения:

$$\frac{102,835 \cdot (-7,1327)}{293,439 + 0,4371} - \frac{243,119 + 0,39 \cdot 427,11}{112 \cdot 12}. \quad (1)$$

|| Вычислите значение выражения. При этом за- || □
метьте, сколько времени займут у вас вычисления.
|| Уверены ли вы в точности полученного результа- ||
та с точностью до 2-й значащей цифры? ||

Представьте себе, что вам нужно решить не один, а сто или тысячу подобных примеров. Это уже будет очень тоскливо и утомительно, примерно как рыть котлован с помощью лопаты. Задача несколько облегчится, если под рукой окажется калькулятор, с помощью которого можно производить **арифметические действия: сложение, вычитание, умножение, деление.**

В этом случае вы, по-видимому, организуете свою работу так: станете в определенном порядке набирать на клавиатуре числа, знаки арифметических действий и записывать на бумаге некоторые промежуточные результаты. Для решения ста, а тем более тысячи примеров эта работа все равно окажется достаточно утомительной.

Вот тут-то и нужна помощь электронно-вычислительной машины (ЭВМ), которая выполняет не только отдельные арифметические операции, но может в автоматическом режиме произвести всю цепочку вычислений, необходимых для решения задачи в целом (вспомните: экскаватор и лопата).

Но ЭВМ нужно дать задание, и сделать это можно только на специальном языке программирования, понятном ей. Таких языков много, но в этой книге мы будем говорить только об одном из них, особенно распространенном, — о языке Паскаль.

Составить задание — это значит написать программу работы ЭВМ, короче — программу для ЭВМ. Когда вы закончите читать эту книгу, окажется, что вы уже умеете писать довольно сложные программы. Сейчас же нам нужно познакомиться с тем, как средствами языка Паскаль сообщить ЭВМ условия примеров. Вернемся к выражению (1). На языке Паскаль оно записывается так:

$$102.835 * (-7.1327) / (293.439 + 0.4371) - \\ (243.119 + 0.39 * 427.11) / (112 * 12).$$

Запишите, не читая дальше, по аналогии с (1) следующие выражения: ○

$$\begin{array}{ll} 12,5 + 31; & \frac{312,18 - 39,01 \cdot 6,1}{124,12 \cdot 3,1}; \\ 7,086 + 2,31 \cdot 81; & \\ \frac{1}{2} + \frac{1}{3}; & \frac{1,01 - 0,25}{1,01 + 0,25}. \end{array}$$

Если при записи некоторая часть выражения переносится на другую строку, то переносимый знак при этом не повторяется.

А теперь познакомимся с правилами записи чисел, порядком выполнения арифметических действий в языке Паскаль.

ЧИСЛА. В языке Паскаль числа записываются в основном так же, как и при обычной математической записи. Приведем несколько примеров (1-я строка — обычная форма, 2-я — Паскаль):

1; -5; 0,25; $1,7 \cdot 10^{-4}$; +49,22; $-2,035 \cdot 10^5$.
1; -5; 0.25; 1.7E-4; +49.22; -2.035E5.

На языке допустимы числа положительные (1; 0.25; $1,7E-4$; +49.22), нуль (0) и отрицательные (-5; $-2,035E5$). Их удобно подразделять на целые (1; -5) и вещественные (иначе, действительные) (0.25; $1,7E-4$; $-2,035E5$; +49.22).

Целые числа состоят из нескольких цифр и, возможно, знака. Если знак не указан и число не равно нулю, то оно положительно.

Вещественные числа—это десятичные дроби и, в частности, целые числа, записанные в виде десятичных дробей.

Пример. Число 0—целое, а 0.0—вещественное.

Цифру нуль следует, как вы уже, наверное, заметили, изображать так: 0.

Целая и дробная части вещественного числа разделяются десятичной точкой, а не запятой: 0.39; +27.1.

Число не может начинаться с точки и не может ею заканчиваться (записи 0. и .6 недопустимы). В том случае, когда нужно записать вещественное число со степенью 10 (например, $2,315 \cdot 10^{-3}$), используется буква E, справа от которой находится показатель степени ($2,315E-3$). Букву E, входящую в состав числа, следует читать, как «умножить на 10 в степени». Например, $-3,5E-2$ означает $-3,5 \cdot 10^{-2}$ или $-0,035$.

|| Запишите числа: || ○
-390; 5; 27,14; -32,07; $2,18 \cdot 10^5$; $3,49 \cdot 10^{-6}$;
-13,24 · 10⁸; -1,3481 · 10⁻¹⁷.

При записи чисел 1,25; 0,17; 2,0; 0,4; $2,102 \cdot 10^{-2}$; $3,15 \cdot 10^4$ средствами языка Паскаль ученик написал: 1,25; 0.17; 2.; .4; 2.102E-10; 3.15-4. Он сделал много ошибок. Например, в числе .4 пропущена целая часть, число начинается с десятичной точки.

|| Найдите остальные ошибки. || □

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ, ВЫРАЖЕНИЯ.

С помощью чисел и арифметических операций можно построить выражения.

Знаки арифметических операций:

сложение	+
вычитание	—
умножение	*
деление	/
деление нацело	DIV
остаток от деления нацело	MOD

Например:

$5 + 2.7;$ $3 - 4.5;$ $2 * 24;$
 $1/5;$ $-25/5.$

Их значения равны соответственно $7.7;$ $-1.5;$ $48;$ $0.2;$ $-5.0.$

Заметим, что при делении $-25/5$ получается не целое, а вещественное число $-5.0.$

Чтобы найти значение частного в целых числах, следует использовать специальную операцию деления нацело: $-25 \text{ DIV } 5$. Между знаком DIV и числами, участвующими в делении, должно находиться хотя бы по одному пробелу.

Пробел—незаполненная позиция в тексте. Можно сказать, что это невидимый, но воспринимаемый ЭВМ символ, занимающий в тексте одну позицию. Обозначается он так: $_$.

Вычисление остатка от деления нацело обозначается знаком MOD: $35 \text{ MOD } 6;$ $-1 \text{ MOD } 2$. Значения приведенных выражений равны соответственно 5 и -1 . Обе эти операции применяются только к целым числам.

При вычислении арифметического выражения действуют обычные правила старшинства операций: сначала выполняются умножение, деление, деление нацело и нахождение остатка от деления нацело в том порядке, в каком они входят в выражение, а затем сложение и вычитание.

Несколько примеров:

$$5 + 2 * 10 = 25;$$

$$10/2 * 5 - 7 + 8.6/2 = 22.3;$$

$$(5 + 105) \text{ DIV } 7 = 15.$$

Проверьте!

Круглые скобки служат для изменения обычного порядка выполнения операций.

Вычислите значения выражений:

а) $(5 + 7 * 2) * 7;$

г) $5 + 2/10;$

б) $3 - 8 + 21 \text{ DIV } 3;$

д) $(5 + 2)/10;$

в) $(2 + 3) \text{ MOD } 7 - 2;$

е) $7/7.$

Пример. Выражение

$$\frac{3,18-4,98}{1,1221-0,24} \cdot (0,7-0,24) + 4,98$$

на языке Паскаль записывается так:

$$(3.18-4.98)/(1.1221-0.24) * (0.7-0.24) + 4.98.$$

Как вы думаете, почему неправильно будет записать его так: $(3.18-4.98)/((1.1221-0.24) * (0.7-0.24)) + 4.98$?

|| Имеют ли смысл выражения:
2.5 DIV 7.1; 7 MOD 1.2.

|| □

Глава 2 ОБРАТИМСЯ ПО ИМЕНИ

Представим себе, что нам нужно вычислить значения выражений:

$$\begin{aligned} & \frac{3,18-4,98}{1,1-0,2} \cdot (0,7-0,24) + 4,98, \\ & \frac{3,18-4,98}{1,1-0,2} \cdot (0,72-0,24) + 4,98, \\ & \frac{3,18-4,98}{1,1-0,2} \cdot (0,89-0,24) + 4,98. \end{aligned} \quad (2)$$

Целесообразно не торопиться с вычислениями "в лоб", особенно если выражений не три, а много больше. Найдем сначала значение выражения $\frac{3,18-4,98}{1,1-0,2}$ и обозначим его переменной А.

Каждая переменная имеет две характеристики: имя и значение. Именем переменной А является буква А, а значением — вещественное число — 2,0.

Переменная — объект, которому дано имя и который может принимать различные значения.

Переменные, так же как и числа, могут быть вещественными (их значения — вещественные числа) и целыми (значения — целые).

Запишем выражения (2), используя переменную А:

$$A \cdot (0,7-0,24) + 4,98;$$

$$A \cdot (0,72 - 0,24) + 4,98;$$

$$A \cdot (0,89 - 0,24) + 4,98.$$

Еще раз преобразуем полученные выражения:

$$A \cdot 0,7 + (4,98 - A \cdot 0,24);$$

$$A \cdot 0,72 + (4,98 - A \cdot 0,24);$$

$$A \cdot 0,89 + (4,98 - A \cdot 0,24).$$

Выражение $4,98 - A \cdot 0,24$ обозначим переменной B .
Значением переменной B является число 5,46.

Тогда получим:

$$A \cdot 0,7 + B = 4,06;$$

$$A \cdot 0,72 + B = 4,02;$$

$$A \cdot 0,89 + B = 3,68.$$

Выражение $A \cdot 0,72 + B$ на языке Паскаль выглядит так: $A * 0.72 + B$.

В состав арифметического выражения, кроме чисел, могут входить и переменные. В этом случае при вычислении значения выражения вместо имени переменной следует подставить ее значение.

|| Запишите выражения: ||

$$\frac{a}{b}; \quad ax^2 + bx + c; \quad 0,5 \cdot (x-1) \cdot (x-2) \cdot (x-3)$$

Под словосочетанием "имя переменной" можно понимать более привычное—"обозначение переменной". Обычно в математике переменную обозначают одним символом и используют при этом латинские или греческие буквы. В языке Паскаль имя может быть "длинным". Оно может состоять из нескольких латинских букв и цифр: AB ; AS ; $K24$; $MASSA$. Имя обязательно начинается с буквы. Записи $1A$, $5CC$ нельзя использовать в качестве имен.

Имя—конечная последовательность букв и цифр, начинающаяся с буквы.

Кроме букв и цифр, в состав имени не могут входить никакие другие символы. Например, запись $A.A$ не является именем, так как в ее состав входит символ ".".

|| Какие из следующих записей являются именами: || ?

- | | | |
|-------------|--------------------|--------------|
| а) $X3$; | г) $X \square 3$; | ж) OX ; |
| б) $A1B2$; | д) MAX ; | з) $0X$; |
| в) BEC ; | е) $\beta 1$; | и) $x + 3$. |

|| Запишите все допустимые трехсимвольные имена, которые можно построить из символов А, В и 1. Из них же образуйте двухсимвольные записи, которые не являются именами. || □

Имена переменных IMJAPERE и IMJAPEREE допустимы, но для ЭВМ они будут одинаковыми, так как принято условие, что имена отличаются друг от друга только первыми восемью символами.

Именем переменной может быть и записанное латинскими буквами ваше собственное имя, имя вашего друга или кличка собаки, однако в качестве имени лучше выбирать такое, которое отражает смысл используемой переменной. Скажем, переменная, физическим смыслом которой является длина, может быть обозначена DLINA, или DLIN, или DLN, или DL.

В языке Паскаль имена используются не только для обозначения переменных, но об этом вы узнаете позднее.

И переменные, и числа являются частным случаем выражений.

|| Какие имена используются в выражениях: || ?
а) $(A4 + XC)/OS$; в) 0.25;
б) PERIMETR; г) S/D.

Глава 3 УЧИМСЯ ПРИКАЗЫВАТЬ

Мы уже говорили, что в сфере интеллектуального труда используется ЭВМ. При этом ей отводится роль исполнителя приказов человека, записанных по определенным правилам языка программирования. Нарушение этих правил может привести к тому, что ЭВМ не воспримет (или воспримет неправильно) приказы, в результате чего требуемые действия не будут выполнены. Следовательно, до решения задачи человек (программист) должен продумать и правильно записать последовательность приказов, выполнение которых он хочет поручить ЭВМ.

Прежде чем учиться приказывать ЭВМ, освежим в памяти предыдущую главу.

Запишите выражения средствами языка Паскаль:

а) $\frac{i+j}{k+n} + m$; в) $\frac{a+b}{c+\frac{d}{f}}$; д) $(a+b)^2$;

б) $\frac{a}{b} + \frac{c \cdot d}{e \cdot f \cdot g}$; г) $\frac{1}{x^2}$; е) $\frac{m+3}{i} + 1$.

Для того чтобы ЭВМ могла вычислить значение выражения $(A+B)/(A*B)$, она должна "знать" значения переменных А и В. Задать эти значения можно, сделав, например, такие приказы:

А — положить равным значению выражения 3.18;

В — положить равным значению выражения $3.2 * A - 1$.
(Напоминаем: число есть частный случай выражения.)
Для слов "положить равным значению выражения" имеется более короткий синоним — "присвоить значение". Таким образом, приведенные выше приказы можно переписать так:

А — присвоить значение 3.18;

В — присвоить значение $3.2 * A - 1$.

Если же воспользоваться специальным символом присваивания $:=$, заменяющим слова "присвоить значение", то получим приказы, записанные на языке Паскаль:

A := 3.18;

B := 3.2 * A - 1.

Приказ — это побуждение к некоторому действию. В языке программирования приказы принято называть операторами.

Приказ "присвоить значение" называется оператором присваивания.

Арифметическое выражение не является оператором, а представляет собой правило (формулу), в соответствии с которым может быть вычислено некоторое значение. Оно может использоваться как составная часть различных операторов, в частности оператора присваивания.

Пример. В операторе присваивания $Y := (A * X + B) * X + C$ справа от знака $:=$ записано выражение $(A * X + B) * X + C$, а слева — переменная Y.

Для каких целей служит оператор присваивания?

Пример. Запись $X = 2.5 + W$ не является оператором

присваивания, так как знак $:=$ заменен здесь знаком равенства.

Запишите операторы присваивания, определяющие значение переменной X по следующим правилам:

$$\text{а) } X = 0; \quad \text{в) } X = a + \frac{b}{a}; \quad \text{д) } X = \frac{a}{b \cdot \frac{c}{d \cdot \frac{e}{f \cdot g}}}.$$

$$\text{б) } X = a^2 + b^2; \quad \text{г) } X = y;$$

Каждый оператор точно определяет соответствующие ему действия.

Оператор присваивания определяет такие действия:

1. Вычислить значение выражения, записанного справа от знака присваивания.
2. Вычисленное значение запомнить как значение переменной, находящейся в левой части оператора присваивания.

Пример. Если дана последовательность операторов (в этом случае операторы отделяются друг от друга точкой с запятой):

$$Y := 1;$$

$X := 2 * Y$, то значением переменной X является 2.

$$\left\| \begin{array}{l} \text{Чему равно значение переменной } X, \text{ если за-} \\ \text{дана последовательность операторов} \\ \text{а) } A := 10; \quad \text{б) } A := 4; \\ \quad B := 3 * A; \quad \quad B := 2; \\ \quad X := (A + B) / A * B - A; \quad X := A / B \sqcup \text{MOD} \sqcup B. \end{array} \right\| \bigcirc$$

Одним из очень важных свойств ЭВМ является умение не только считать, но и запоминать. ЭВМ имеет память, которая поделена на отдельные участки (блоки, ячейки). Запомнить значение переменной — это значит занести его в ячейку памяти с тем, чтобы потом в случае необходимости им можно было воспользоваться.

Пример. Дана последовательность операторов:

$$A := 10; \quad B := 5; \quad C := A * A + B * B.$$

Опишем процесс ее выполнения.

В результате выполнения оператора $A := 10$ в ячейку памяти, отведенную переменной A , будет записано число 10, $B := 5$ — в ячейку памяти, отведенную переменной B , будет записано число 5. Чтобы вычислить значение выражения $A * A + B * B$, вместо переменных A и B подставляются их значения, которые хранятся в ячейках памяти, отведенных этим переменным. Значение $10 * 10 +$

$+5*5$, равное 125, будет записано в ячейку памяти, отведенную переменной С.

Объясните результат последовательного выполнения операторов: $PI:=3.14;$ $R:=5;$ $C:=2*PI*R.$	□
--	---

Пример. Дана последовательность операторов:

$A:=5;$

$B:=A*A-2*A;$

$A:=(B+2)*(B-3).$

В результате выполнения первых двух операторов в ячейки памяти, отведенные переменным А и В, будут записаны значения 5 и 15. При выполнении оператора $A:=(B+2)*(B-3)$ будет вычислено значение выражения $(B+2)*(B-3)$, и полученное число 204 будет записано в ячейку переменной А. Прежнее ее значение, 5, при этом будет утрачено.

Значения переменных могут меняться в ходе выполнения операторов. Правило выполнения оператора присваивания гласит: «Каким бы ни было значение переменной в левой части, оно должно быть заменено вновь вычисленным значением выражения в правой части».

Может ли левая часть оператора присваивания быть арифметическим выражением?	?
---	---

Найдите ошибки в каждом из следующих операторов присваивания: а) $X=I+4;$ в) $3*X:=L;$ б) $X:=2,76*A;$ г) $-V:=A+B;$ д) $X:=A/-B;$ е) $1:=1;$ ж) $P:=2.5 \sqcup MOD \sqcup 2.$	□
---	---

Глава 4

ДОЛОЖИТЕ ОБСТАНОВКУ

Покажем сейчас, как использовать операторы присваивания для вычисления значения переменной Y , заданной формулой $Y=(a^2+2a+0,59) \cdot (a^2+2a-0,25)$ при $a=1,27$.

Вычисления можно провести разными способами, мы же рассмотрим здесь только два из них.

I. $A := 1.27$;

$Y := (A * A + 2 * A + 0.59) * (A * A + 2 * A - 0.25)$.

II. Выражение $a^2 + 2a$ обозначим через b , тогда

$Y = (b + 0.59) \cdot (b - 0.25)$ и последовательность операторов следующая:

$A := 1.27$;

$B := (A + 2) * A$;

$Y := (B + 0.59) * (B - 0.25)$.

Интересно отметить, что в первом варианте используются две ячейки памяти и выполняются девять арифметических действий, во втором же — три ячейки и пять действий. Выигрывая в одном, проигрываем в другом!

|| Запишите свой вариант вычисления Y .

|| □

Оба варианта обладают существенным недостатком. В результате выполнения любой из последовательности операторов в ячейку, отведенную переменной Y , будет записан результат. Но как узнаем об этом значении мы? Едва ли нас утешит, что ЭВМ знает результат. Нам самим нужно его знать!

К счастью, ЭВМ умеет сообщать (выводить) значения переменных, выражений.

Чтобы выяснить обстановку, сложившуюся в результате выполнения операторов, узнать интересующие нас значения переменных, выражений, ЭВМ нужно дать соответствующий приказ:

выдать значение.

В языке Паскаль такой приказ называется оператором (процедурой) вывода. В данном случае, когда нас интересует значение переменной Y , он записывается так: `WRITE(Y)`.

WRITE — пиши

В результате выполнения этого приказа на печать будет выдано значение переменной Y . А что же останется в ячейке, отведенной для Y ? Запомните, что содержимое этой ячейки в результате выполнения оператора `WRITE` не изменится, там по-прежнему будет храниться значение переменной Y .

|| Запишите операторы для вывода переменной: || ○
а) A ; б) `SUMMA`; в) $P1$.

С помощью одной процедуры WRITE можно выдавать на печать значения нескольких выражений.

Пример. Если второй вариант дополнить приказом WRITE (A,B,Y), то на печать будут выданы значения переменных A, B, Y в том порядке, в котором они указаны в инструкции WRITE.

Переменные A, B, Y называются параметрами вывода.

Параметры вывода отделяются друг от друга запятой и заключаются в круглые скобки.

Число параметров вывода может быть произвольным.

|| Запишите процедуры для вывода нескольких || ☐
переменных: ||
а) A, A2, A3, A4; б) X и Y.

Пример. Параметрами вывода процедуры WRITE (3,A+B) являются число 3 и выражение A+B. В результате ее выполнения на печать будут выданы число 3 и значение выражения A+B.

Запишите приказ выдать значения:

- а) 1.5; в) $A+B/2$ и 3;
б) X, Y; г) X1, Y1, 1.5 и Z.

Процедура WRITE приводит к тому, что вычисленные значения параметров вывода (они могут быть и выражениями) одно за другим печатаются на выходе.

Пример. Процедура WRITE(A B Y) содержит ошибку: параметры вывода должны разделяться запятыми, а не пробелами!

|| Укажите ошибки, допущенные в приведенных || ☐
ниже записях: ||
а) WRITE(Y1;Y2); г) WRITE[A,B];
б) WRITE(A B); д) WRITE 1.25;
в) WRITE A,B; е) WRITE(A,B/F;).

Запишите последовательность операторов для вычисления и вывода переменных:

а) $y = bx + c$ при $x = 0,5$; $b = 2,1$; $c = 6,2$;

б) $f = (0,05x^2 + 0,39x - 0,08) \cdot (0,05x^2 - 0,08)$ при $x = 0,97$.

Глава 5

БОЛЬШУЮ ПОЛОЧКУ ИЛИ МАЛЕНЬКУЮ?

Если для вычисления переменной $y = 27,3x^2 + 0,9x + 6,1$ при $x = 2,344$ мы сообщаем ЭВМ такую последовательность приказов: $X := 2.344$;

$Y := (27.3 * X + 0.9) * X + 6.1$;

WRITE(Y),

то, несмотря на то что каждая инструкция записана верно, ЭВМ нас не поймет. Чтобы ЭВМ могла выполнить задание, ему следует придать вид программы.

Программа—задание для ЭВМ, написанное на понятном для нее языке в установленной форме.

Программа на языке Паскаль обязательно начинается с заголовка и заканчивается точкой.

В качестве иллюстрации приведем программу, вычисляющую значение выражения $\frac{2,25 + 7,1 \cdot 2,49}{4,38 - 3,21 \cdot 2,13}$.

```
PROGRAM ZNACHEN(OUTPUT);  
BEGIN  
WRITE((2.25+7.1*2.49)/(4.38-3.21*2.13))  
END.
```

Заголовок программы: PROGRAM ZNACHEN (OUTPUT); он состоит из ключевого слова PROGRAM—программа, имени программы—ZNACHEN. В скобках записано имя OUTPUT, которое указывает, что в программе содержатся инструкции вывода на печать результатов счета. В конце заголовка обязательно ставится точка с запятой.

Последовательность операторов, образующих программу (в данном случае оператор WRITE), заключена в операторные скобки: BEGIN, END. Оба эти слова также являются ключевыми словами. Их смысл: BEGIN

(начало) — открывающая и END (конец) — закрывающая операторные скобки.

Ключевые (служебные) слова особенные: они не могут использоваться в качестве имен.

Ошибки в написании этих слов недопустимы. Ошибочное написание хотя бы одного ключевого слова приводит к тому, что ЭВМ откажется выполнять вашу программу.

|| Чтобы лучше запомнить ключевые слова, выпишите их в специальный словарь, поясняя значения. ||

Те, кто знаком с английским языком, без труда установят происхождение слов. Это не должно удивлять, так как именно этот язык близок создателю Паскаля Н. Вирту.

PROGRAM — программа

BEGIN — начало

END — конец

Имя программы выбирается произвольно с учетом правил написания имени в языке Паскаль. В частности, рассматриваемой программе можно было бы дать и такие имена:

PROGR, A1, ZN, SSS и др.

|| Однако лучше выбирать имя, отражающее назначение данной программы, действия, которые в ней производятся. ||

Попробуем теперь составить программу с именем YX для решения задачи, приведенной в начале этой главы.

```
PROGRAM YX(OUTPUT);
```

```
BEGIN
```

```
X:=2.344;
```

```
Y:=(27.3*X+0.9)*X+6.1;
```

```
WRITE(Y)
```

```
END.
```

Программа YX, записанная по аналогии с ZNACHEN, окажется невыполнимой, и вот почему. YX в отличие от ZNACHEN содержит переменные X и Y. Этим переменным должны быть выделены ячейки памяти: такие “полочки в электронном мозге” ЭВМ, на которые она будет “ставить” значения переменных.

Вам известны уже два типа переменных: целые и вещественные. Переменным различных типов выделяются

ячейки памяти различной длины и структуры. Поэтому в начале программы следует сообщить информацию о том, каким переменным должна быть выделена память, а также указать их тип, чтобы ЭВМ “знала”, сколько нужно выделить ячеек для переменных, закрепила за каждой переменной свою ячейку, т. е. “полочку” определенных размеров:

```
PROGRAM YX(OUTPUT);  
VAR X,Y:REAL;  
BEGIN  
X:=2.344;  
Y:=(27.3*X+0.9)*X+6.1;  
WRITE(Y)  
END.
```

Это делается в разделе переменных
VAR X,Y:REAL;

Раздел переменных начинается с ключевого слова VAR—переменные (сокращение английского слова VARIABLES), за которым следует описание переменных, состоящее из описательных предложений. В описательном предложении указываются имена переменных и их тип.

VAR — переменные

В программе YX используются две переменные вещественного типа X и Y. Вещественный тип обозначается ключевым словом REAL, целый—INTEGER.

REAL — вещественный, INTEGER — целый.
--

Имена переменных отделяются друг от друга запятыми, а от указателя типа—двоеточием. В конце предложения ставится точка с запятой.

Пример. В разделе переменных

```
VAR X,Y:REAL;  
I,J,K:INTEGER;  
Z:REAL;
```

описаны три вещественные переменные: X, Y, Z и три переменные целого типа: I, J, K.

Постройте раздел переменных программы, в котором описываются:

- а) A — вещественная; в) X, Y — вещественные
б) J1, J2, J3 — целые; и I, J — целые

Пример. Приведем пример ошибочного описания переменных:

```
VAR A,X,SLAGAEM,P,S:REAL;  
    SUMC,NOMC,A INTEGER;
```

В описании отсутствует двоеточие перед указателем типа INTEGER, кроме того, переменная A описана дважды.

Все переменные, используемые в программе, должны быть описаны. Каждая переменная описывается только один раз.

Укажите ошибки, допущенные в разделе переменных:

- а) VAR A,B,C,D:REAL;
б) VAR A,B,C;D,E:REAL;
 I,J,K:INTEGER;
в) VAR A,B,C REAL;
г) VAR A,B,C,D:REAL;
д) VAR I,J,MAX:INTEGER;
 A,MAX,MIN:REAL;

Пример. Составим программу для вычисления выражения

$$y = \frac{12400}{-(x^2 + 2x)^2} \quad \text{при } x = 5.$$

Прежде всего обозначим знаменатель дроби $(x^2 + 2x)^2$ через a^2 . Тогда $y = -\frac{12400}{a^2}$, где $a = (x + 2) \cdot x$ и $x = 5$.

Программа состоит из заголовка, раздела переменных, раздела операторов и точки.

Раздел операторов — последовательность всех операторов программы вместе с операторными скобками.

Раздел операторов описываемой программы имеет вид:

```

BEGIN
X:=5;
A:=(X+2)*X;
Y:=-12400/(A*A);
WRITE(Y)
END

```

В программе используются две переменные целого типа—X и A и одна действительная переменная Y, так как результатом операции деления является вещественное число. Раздел переменных программы, в котором описываются эти переменные, имеет вид:

```

VAR A,X:INTEGER;
    Y: REAL;

```

Порядок расположения имен в списке переменных не играет роли. Однако

лучше описать вначале переменные целого типа, затем вещественного, упорядочивая имена в списке по алфавиту. Это облегчит поиск имени при чтении программы. Следует помнить, что программы читаются не только ЭВМ, но и людьми. Поэтому необходимо заботиться о наглядной структуре программы.

Наконец, дав программе имя PRIMER, можем привести ее текст полностью:

```

PROGRAM PRIMER(OUTPUT);
VAR A,X:INTEGER;
    Y:REAL;
BEGIN
X:=5;
A:=(X+2)*X;
Y:=-12400/(A*A);
WRITE(Y)
END.

```

В конце программы не забудьте поставить точку.

Структуру программы можно представить в виде схемы:

```

      <заголовок>
    [<раздел переменных>]
      <раздел операторов>.

```

Квадратные скобки означают, что раздел переменных может отсутствовать, если переменные отсутствуют в программе.

|| Составьте программы вычисления переменной y : || \square
 а) $y = ((x - 3)^2 + 8)^2$ при $x = 10$;
 б) $y = (t^2 + 1) - 3t(t^2 + 1)$ при $t = 4$. ||

Глава 6

ПОИСКИ КЛАДА, ИЛИ ЧТО ТАКОЕ АЛГОРИТМ?

Найти клад — большая удача. Во все времена находились энтузиасты — кладоискатели. Однако если ничего неизвестно о возможном местоположении клада, то попытки его найти, скорее всего, окажутся безрезультатными. Другое дело, если в ваши руки попадет старинный манускрипт с таким, например, описанием.

В лесу есть поляна, на которой расположен холм. У подножия холма протекает ручей. Встав у истока ручья, иди по такому маршруту: сделай два шага на юго-восток, затем три — на восток, еще восемь шагов — на север. Комай здесь! (рис. 1).

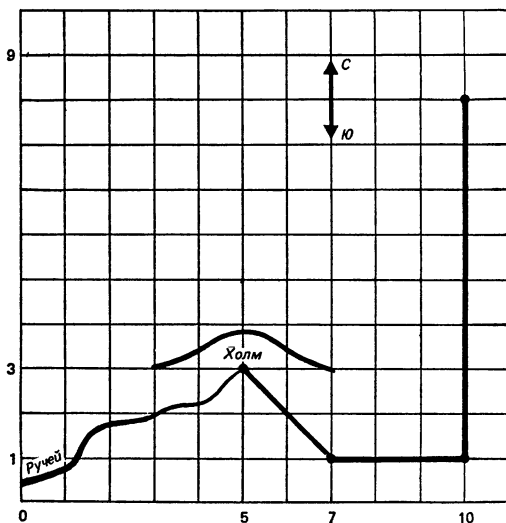


Рис. 1

На наш взгляд, имея такое описание, стоит поискать клад. Этим мы сейчас и займемся, правда, лишь в теоретическом плане.

Найдем координаты точки, в которой спрятан клад. Исходная позиция, исток ручья,—это точка с координатами (5,3). Теперь точно выполним каждое действие, указанное в описании.

Совершив два шага на юго-восток, придем в точку (7,1). Три шага на восток приведут в точку (10,1). Последние восемь шагов на север заканчиваются в точке (10,9). Это и есть координаты точки, в которой спрятан клад.

Итак, выполняя каждый шаг описания (алгоритма) из точки (5,3), мы переместились в точку (10,9).

Алгоритм—это правило, указывающее действия, в результате выполнения которых осуществляется переход от исходных данных к искомому результату. Цепочка действий называется алгоритмическим процессом, а каждое действие—шагом алгоритма.

В рассматриваемом примере исходными данными является точка (5,3), результатом—(10,9).

В повседневной жизни мы практически ежедневно многократно сталкиваемся с алгоритмами. Например, делая звонок по телефону, приготавливая кофе, сажая деревья и т. д., мы выполняем некоторые алгоритмы. Алгоритм можно рассматривать как последовательность приказов, а программы представляют собой алгоритмы, записанные по определенным правилам языка программирования. Поэтому языки программирования называют иначе алгоритмическими языками.

Выполняя программу, ЭВМ является исполнителем алгоритма. Поэтому ясно, что каждый шаг алгоритма должен быть понятен его исполнителю, известны правила выполнения каждого шага.

А может ли ЭВМ выполнить алгоритм, описывающий маршрут движения к кладу? Конечно, она не может выйти на поляну и отправиться на север или восток, но... с ее помощью можно многое сделать. В частности, покажем, как поручить ЭВМ поиск координат точки, в которой находится клад.

Пусть X и Y —координаты местоположения кладоискателя: X —абсцисса, Y —ордината. Вначале $X=5$; $Y=3$. Условимся, что для ЭВМ шаг в северном направлении означает увеличение ординаты на 1, в восточном—увеличение абсциссы на 1. Шаг в промежуточном

направлении, например на юго-запад, равносителен одному шагу на юг и одному на запад.

|| Что означает движение на один шаг на запад? ||?
на юг? на северо-восток?

Опишем теперь алгоритм поиска клада средствами языка Паскаль:

```
PROGRAM KLAD(OUTPUT); (*программа движе-
ния к кладу*)
VAR X,Y:REAL; (*координаты кладоискателя*)
BEGIN
  X:=5; Y:=3; (*исходное положение*)
  X:=X+2; Y:=Y-2; (*2 шага на юго-восток*)
  X:=X+3; (*3 шага на восток*)
  Y:=Y+8; (*8 шагов на север*)
  WRITE(X,Y)
  END.
```

|| Считаете ли вы оператор $X:=X+2$ неправиль- ||?
ным?

В математике утверждение $X=X+2$ всегда неверно, но здесь запись $X:=X+2$ является не утверждением, а приказом, в соответствии с которым ЭВМ:

- 1) вычислит значение выражения в правой части, для чего вместо X подставит его значение -5 , т. е. $X+2=5+2=7$;
- 2) полученное значение 7 запишет в ячейку, отведенную переменной X . Теперь значением переменной X является 7, а ее старое значение, 5, утрачено.

Значения переменных в ходе выполнения программы могут подвергаться многократным изменениям.

|| Выполните программу KLAD. || ○

Для улучшения наглядности программа KLAD снабжена комментариями. Комментарий—это текст, заключенный между сочетаниями символов (* и *). (В некоторых реализациях языка текст заключается в фигурные скобки.)

ЭВМ не читает текст комментария.

Однако мы уже говорили, что программы читаются не только машинами, но и людьми. Комментарии помогают ориентироваться в программе, выявлять содержащиеся в ней ошибки, которые не сможет обнаружить ЭВМ.

Помните, ЭВМ не умеет распознавать ошибки, содержащиеся в алгоритме.

Например, если в программе KLAD вместо оператора $X:=X+3$ ошибочно будет записан оператор $X:=X-3$, то ЭВМ не укажет на наличие ошибки, а выдаст неправильные координаты местоположения клада.

Программа — это запись алгоритма на понятном для ЭВМ языке. Прежде чем писать задание, выполнение которого вы хотите поручить ЭВМ, необходимо самому разобраться в том, как это задание следует выполнять. Для этого нужно сначала выяснить, что дано (исходные данные), что требуется получить (искомые результаты), а затем описать алгоритмы получения искомых результатов из исходных данных.

Алгоритм может быть описан различными способами. Приведем некоторые примеры.

Пример 1. Словесно-формульный способ описания алгоритма вычисления переменных $Y1$, $Y2$, $Y3$:

$$Y1 = \frac{3,18 - 4,98}{1,1 - 0,2} \cdot (0,7 - 0,24) + 4,98;$$

$$Y2 = \frac{3,18 - 4,98}{1,1 - 0,2} (0,72 - 0,24) + 4,98;$$

$$Y3 = \frac{3,18 - 4,98}{1,1 - 0,2} \cdot (0,89 - 0,24) + 4,98;$$

1. начало;

2. $A = \frac{3,18 - 4,98}{1,1 - 0,2};$

3. $B = 4,98 - A \cdot 0,24;$

4. $Y1 = A \cdot 0,7 + B;$

5. $Y2 = A \cdot 0,72 + B;$

6. $Y3 = A \cdot 0,89 + B;$

7. печать ($Y1$, $Y2$, $Y3$);

8. конец.

Пример 2. Опишем этот же алгоритм с помощью блок-схемы. Для этого каждому шагу алгоритма типа „вычислить“ поставим в соответствие прямоугольный блок

(рис. 2), а шагам типа „выдать“, „печатать“ — блок в виде параллелограмма (рис. 3).

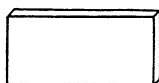


Рис. 2

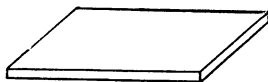


Рис. 3

Вот какой вид имеет блок-схема алгоритма (рис. 4): Стрелка, выходящая из блока, показывает, к какому шагу алгоритма нужно перейти по окончании данного блока.

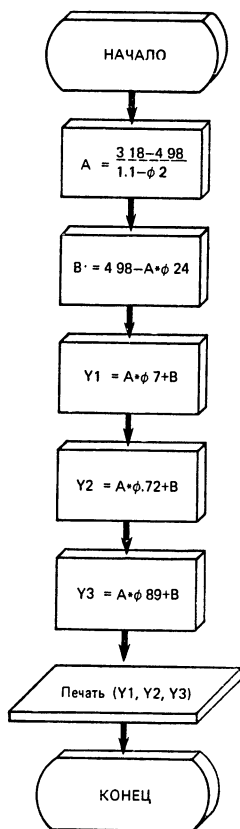


Рис. 4

Опишите алгоритмы вычисления функций:

а) $f = (0,05x^2 + 0,39x - 0,08) \cdot (0,05x^2 - 0,08)$ при $x = 0,97$;

б) $y = \frac{x^2 - 2ax + 1}{x^2 - 1}$ при $x = 0,8$; $a = 2,43$.

Предположим, что между моментами, когда клад был спрятан и когда его ищут, произошло землетрясение, в результате которого поляна исчезла и превратилась в овраг. Тогда, двигаясь по описанному маршруту, вы натолкнетесь на непреодолимые препятствия. Этот алгоритм вам исполнить не удастся.

В случае, когда алгоритмический процесс безвозвратно обрывается, говорят о неприменимости заданного алгоритма.

Укажите значения переменной x , для которых алгоритм

1. начало;

2. задать x (*считайте, что значение x некоторым образом задано*);

3. $y = \frac{1}{x^2 - 1}$;

4. печать (y);

5. конец

неприменим.

Для каждого алгоритма существует некоторое множество объектов, которые он может использовать в качестве допустимых исходных данных. Это свойство называется массовостью алгоритмов.

Для приведенного в упражнении алгоритма допустимыми исходными данными являются все вещественные значения x , кроме 1 и -1 . А исходной точкой для поиска клада может быть лишь исток ручья. Если задать другую исходную точку, то алгоритм изменится.

Каждый алгоритм, выполнение которого поручается ЭВМ, должен оканчиваться после конечного числа шагов.

Все рассмотренные нами алгоритмы обладают этим свойством.

|| Придумайте какой-нибудь алгоритм движения на плоскости из точки (0,0) в точку (3,4) при условии, что через точки (0,2), (0,3), (0,4), (3,0) проходить нельзя. Опишите этот алгоритм словесно, с помощью блок-схемы и на языке Паскаль. || □

Попробуйте подробно записать алгоритм поиска клада, который в беллетризованном виде имеется в рассказе Эдгара По «Золотой жук».

Глава 7

ЭВМ ПЕЧАТАЕТ ТАБЛИЦЫ, А ЧИТАТЕЛЬ ВПЕРВЫЕ ВСТРЕЧАЕТСЯ С ОПЕРАТОРОМ ЦИКЛА

Ученик получил задание: вычислить квадраты и квадратные корни следующих двадцати чисел: 0,001; 0,007; 0,009; 0,013; 0,019; 0,27; 0,31; 0,35; 0,37; 0,41; 0,44; 0,45; 0,55; 0,61; 0,64; 0,69; 0,71; 0,77; 0,79; 0,8.

Вычисления трудоемки, поэтому понятно его желание поручить работу ЭВМ. С чего начать? Естественно, с изучения содержания задачи, как говорят, с постановки задачи. Выяснив, что исходными данными является набор из двадцати чисел, ученик выписал эти числа на отдельный лист бумаги, пометив его: «Исходные данные».

Результатом должны быть квадраты и квадратные корни соответствующих чисел. Играет ли роль форма, в которой результаты будут выданы? Еще бы! Если бы вам показали результаты в такой, например, форме:

1.0000000E—06	3.1622776E—02	4.9000000E—05
8.3666002E—02	8.1000000E—05	9.4868329E—02
1.6300000E—04	1.1401754E—01	3.6100000E—04
1.3784048E—01	...	

то вы, безусловно, сказали бы, что ученик плохой и труд его был напрасен. Действительно, непонятно, где здесь квадрат, где квадратный корень и к какому числу они относятся. Другое дело, если в каждой строке будут помещены число, его квадрат и квадратный корень.

Например, так:

0,001 1.0000000E—06 3.1622776E—02

0,007 4.9000000E—05 8.3666002E—02.

Заботьтесь о форме выводимых результатов.

Итак, решено—результаты будут выдаваться в три столбца: в первом—число, во втором—квадрат этого числа, в третьем—квадратный корень из него. Ученик выделил отдельный лист бумаги для результатов и пометил его: „Результаты: число, квадрат, квадратный корень“.

Чтобы построить алгоритм, по которому смогла бы работать ЭВМ, ученик представил себе, как следует действовать без ЭВМ. Очевидно, так:

- 1) прочитать число из набора исходных данных;
- 2) записать его в первый столбец;
- 3) вычислить квадрат этого числа;
- 4) записать его во второй столбец;
- 5) вычислить квадратный корень из числа;
- 6) записать его в третий столбец.

Затем нужно прочитать следующее число и вновь выполнить действия 2—6. Таким образом, последовательность действий 1—6 нужно повторить 20 раз для каждого числа.

Чтобы поручить решение задачи ЭВМ, используя только известные уже читателю средства языка Паскаль: операторы присваивания и вывода, нужно написать программу длиной в несколько сот операторов. Дело это скучное и долгое. Нужен другой путь.

Мы вместе с учеником воспользуемся тем свойством задачи, что каждое число обрабатывается по одному и тому же алгоритму.

Пусть K —номер очередного прочитанного числа. Это переменная величина, которая последовательно принимает значения 1, 2, 3, ..., 20. Через X обозначим число из исходного набора, X^2 —квадрат числа, \sqrt{X} —его квадратный корень. Тогда процедуру обработки чисел можно описать так:

$K:=1$;

читать (X);

$X^2:=X^2$;

$\sqrt{X}:=\sqrt{X}$;

печать (X , X^2 , \sqrt{X});

$K:=2$;

читать (X);

$X2 := X^2;$
 $X3 := \sqrt{X};$
 печать (X, X2, X3)
 и т. д. для $K=3, 4, \dots, 20$.

(Вы, наверное, заметили, что описание сделано на „смеси“ языков — на русском и на Паскале. На данном этапе так удобнее. Ведь вы без труда поняли смысл?)

Приведенное выше описание процедуры обработки чисел можно сделать более кратким:

для K , изменяющегося от 1 до 20, повторять серию операторов: читать (X);

$X2 := X^2;$

$X3 := \sqrt{X};$

печать (X, X2, X3).

Условимся причастный оборот „изменяющегося от 1 до 20“ записывать „:=1 до 20“, а серию операторов заключаем в фигурные скобки. Тогда возникает конструкция вида

для $K:=1$ до 20 повторять

{читать (X);

$X2 := X^2;$

$X3 := \sqrt{X};$

печать (X, X2, X3)},

называемая циклической конструкцией.

Циклическая конструкция является приказом многократно повторить серию операторов.

В данном случае число повторений нам заранее известно и определяется начальным и конечным значениями переменной K . Переменная K является указателем числа повторений. Ее называют обычно счетчиком, или параметром цикла.

Пример 1. Рассмотрим циклическую конструкцию с параметром i :

для $i:=2$ до 4 повторять

печать (i).

Заметим, что если серия состоит только из одного оператора, то скобки можно опустить.

|| Укажите начальное и конечное значения параметра цикла. || □

Серия операторов, выполняемая с каждым значением параметра, называется телом цикла.

В нашем примере телом цикла является конструкция “печать“, которая будет выполняться при $i=2$, $i=3$ и $i=4$. В результате будут напечатаны числа 2, 3, 4.

Пример 2. Найдём сумму первых n -натуральных чисел: $S=1+2+3+\dots+n$ при $n=25$. Для этого воспользуемся следующим часто употребляемым приемом. Положив вначале результат суммирования S равным 0, будем добавлять к нему поочередно слагаемые:

$n:=25$;

$S:=0$;

для $i:=1$ до n повторять

$S:=S+i$;

печать (S).

Определите, что получится в результате выполнения следующего алгоритма:

1. начало; $N:=5$; $X:=2$;

2. $S:=1$;

3. для $i:=1$ до N повторять;

3.1. $S:=S*X$;

4. печать (S)

5. конец.

Итак, общий вид циклической конструкции таков:
для $P:=NZ$ до KZ повторять
 S , где

P —параметр цикла;

NZ —начальное значение параметра;

KZ —конечное значение параметра;

S —тело цикла, а конструкция «для $P:=NZ$ до KZ повторять» называется заголовком цикла.

Для наглядности тело цикла лучше записывать с некоторым сдвигом вправо относительно заголовка.

Описанной циклической конструкции в языке Паскаль соответствует оператор цикла с параметром. Чтобы перейти к его записи, слова “для“, “до“, “повторять“ следует заменить ключевыми словами FOR, TO, DO,

а фигурные скобки нужно заменить операторными скобками BEGIN и END.

FOR—для

TO—до

DO—делать

BEGIN—начало

END—конец

Пример 3. Запишем оператор цикла для циклической конструкции в примере 1. Вот его вид:

```
FOR I:=2 TO 4 DO  
  WRITE(I).
```

В результате выполнения этого оператора будут напечатаны значения 2, 3, 4.

Параметр цикла является переменной целого типа. Он не может быть вещественным.

Пример 4. Запишем программу суммирования первых $N=25$ натуральных чисел. (Алгоритм приведен в примере 2.):

```
PROGRAM SUMMA (OUTPUT);  
(* сумма первых N натуральных чисел *)  
VAR I,N,S:INTEGER;  
BEGIN  
  N:=25;  
  S:=0;  
  FOR I:=1 TO N DO  
    S:=S+I;  
  WRITE(S)  
END.
```

В качестве начального и конечного значений параметра цикла можно использовать не только числа, но и выражения, принимающие целые значения.

Пример 5.

```
S:=0;  
FOR K:=5 TO 1 DO  
  S:=S+K;  
WRITE(S).
```

Что является начальным и конечным значениями пара-

метра цикла? Вы верно заметили, что в отличие от предыдущих примеров начальное значение счетчика превышает его конечное значение. Ошибки здесь нет. Просто в этом случае тело цикла не будет выполнено ни разу, и в результате будет напечатано число 0.

Действия, определяемые оператором цикла, таковы:

- 1) вычислить начальное и конечное значения параметра цикла;
- 2) если начальное значение превосходит конечное, то тело цикла не выполнять, а перейти к следующему за циклом оператору. В противном случае, изменяя параметр цикла, вместе с каждым его значением выполнить тело цикла.

Вернемся, однако, к нашему ученику и его проблеме. Оператор цикла—замечательное средство, которое позволяет экономить силы и время программиста, поэтому, естественно, ученик использует его при записи алгоритма:

0. начало;

1. печатать ('результаты: число, квадрат числа, квадратный корень из числа');
2. для $K:=1$ до 20 повторять
 - 2.1 {перейти на новую строку;
 - 2.2 читать (X);
 - 2.3 $X2:=X^2$;
 - 2.4 $X3:=\sqrt{X}$;
 - 2.5 печать (X, X2, X3);
3. конец.

Теперь нужно представить этот алгоритм в виде программы на языке Паскаль. Сразу это не получится, так как нам придется сначала познакомиться с некоторыми новыми конструкциями, которые понадобятся в программе.

Для первого действия, аналогичного тому, как ученик надписывал лист бумаги с результатами, имеется подходящая процедура

WRITE ('РЕЗУЛЬТАТЫ: ЧИСЛО, КВАДРАТ, КВАДРАТНЫЙ КОРЕНЬ').

Параметром вывода в этом случае является текст. Он заключен в одиночные кавычки или апострофы.

Последовательность символов, заключенная в апострофы, называется строкой.

Пример 6. Правильные строки:
'СТРОКА', '+', 'A+B', 'СОЛНЦЕ'.

Пример 7. Неправильные строки:
ДЕРЕВО, 'СПИСОК', 'ТУЧА', 'ОБ'ЕМ', „ОСЕНЬ“.

Если в состав строки нужно включить апостроф или кавычки, то их изображают так: „. Например, 'ОБ"ЕМ' или 'КИНОТЕАТР „РОССИЯ“'.

Укажите, какие ошибки содержат строки в примере 7.

Запишите строку, которая является предложением с прямой речью.

Пример 8. В результате выполнения последовательности операторов

```
N:=5;  
WRITE ('N РАВНО', N)
```

ЭВМ напечатает:

N РАВНО 5.

Для перехода на новую строку при печатании имеется инструкция WRITELN. Она может использоваться без параметров, сама по себе, при этом результатом ее действия является только переход на новую строку. Если же ее использовать с параметрами, например, WRITELN ('РЕЗУЛЬТАТЫ'), то сначала будет напечатано слово РЕЗУЛЬТАТЫ и лишь потом осуществлен переход на новую строку.

Пример 9. Следствием выполнения операторов

```
X:=0;  
WRITELN ('ИСХОДНЫЕ ДАННЫЕ:');  
WRITE (X);  
WRITELN ('РЕЗУЛЬТАТЫ:');  
WRITE(X+1)
```

является такой текст:

```
ИСХОДНЫЕ ДАННЫЕ:  
0 РЕЗУЛЬТАТЫ:  
1
```

Для того чтобы ЭВМ могла прочесть значение переменной X, имеется специальная инструкция ввода:

READ(X).

READ—читай

Но ведь читать надо откуда-то! Придется снабдить программу специальным приложением, называемым входным файлом, в котором через пробел перечисляется вся последовательность обрабатываемых чисел.

При выполнении инструкции READ(X) параметру ввода—переменной X будет присвоено очередное значение из приложения, т. е. число, прочитанное из входного файла, будет записано в ячейку, отведенную переменной X. Старое содержимое ячейки при этом утратится. Таким образом, инструкция READ—это второй способ «заполнить полочку», отведенную переменной. А первый способ, оператор присваивания, вы не забыли? Пример 10. Переменным A, B, C, D следует придать значения 2, 7, 3, 9. Это можно сделать так: в программе записать READ (A, B, C, D), а во входной файл поместить числа 2 7 3 9.

Инструкция READ может иметь произвольное количество параметров ввода.

Значения во входном файле могут разделяться произвольным числом пробелов, но должны следовать друг за другом в том же порядке, в каком соответствующие параметры ввода перечислены в инструкции READ. Пример 11. В программе содержатся две инструкции ввода:

```
READ(A, B);  
READ(C, D).
```

Во входном файле содержатся числа:

3.2 —12.1 6.35 —4.2E—5.

В результате выполнения этих инструкций переменным A, B, C, D будут присвоены следующие значения:

```
A = 3.2;  
B = —12.1;  
C = 6.35;  
D = —4.2E—5.
```

В том случае, если среди переменных A, B, C, D

окажутся переменные целого типа, произойдет аварийное прекращение выполнения программы.

Переменным целого типа должны соответствовать целые значения.

Входной файл имеет имя INPUT, и оно должно быть указано в заголовке программы: PROGRAM NAME (INPUT, OUTPUT);

OUTPUT—это имя выходного файла—совокупности всех выводимых на печать результатов.

Чтобы справиться с заданием, ученику (надеемся, что вы не забыли о нем!) требуется еще совсем немного—узнать, как вычисляются квадраты и квадратные корни из чисел. К счастью, в языке Паскаль имеются стандартные функции, позволяющие легко решать подобные задачи. В частности, SQR(X) вычисляет X^2 , а SQRT(X)—арифметический корень из X.

Пример 12. Запишем на языке Паскаль выражения: $\frac{x^2}{2}$, $4\sqrt{x}$, $\sqrt{x+y} + \sqrt{x-y}$, $\sqrt{x^2+y^2}$. Вот какой вид они будут иметь:

SQR(X)/2, 4*SQRT(X), SQRT(X + Y) + SQRT(X - Y), SQRT(X*X + Y*Y).

SQR и SQRT—имена функций возведения в квадрат и извлечения корня.

|| Так как обращение к стандартной функции занимает у ЭВМ больше времени, чем умножение, ||
лучше вместо SQR(X) использовать X*X.

В языке имеются и другие стандартные функции, например ABS(X), SIN(X), COS(X), ARCTAN(X), LN(X), EXP(X).

Традиционная математическая запись этих функций такова: $|X|$, $\sin X$, $\cos X$, $\arctg X$, $\ln X$, e^X .

Если назначение некоторых из перечисленных функций вам неизвестно, не огорчайтесь. Очевидно, у вас еще не возникли задачи, в которых эти функции используются.

Пример 13. Аргументами стандартных функций могут быть:

числа — ABS(—5); SQR(0.0001);
переменные — COS(Y), SQRT(Y1), SIN(A2B);
арифметические выражения — SQR(X + Y);
SQR(X12), SQR(SQRT(X) + SQRT(Y)).

Аргументы стандартных функций заключаются в круглые скобки.

|| Назовите аргументы стандартных функций, приведенных в примере 13. || ☐

|| Запишите выражения, которым соответствуют $\text{SQR}(\text{COS}(X))$ и $\text{COS}(\text{SQR}(X))$. || ☐

|| При записи выражений $\sin \ln X$ и e^{X+Y} допущены ошибки:

$\text{SINLN}(X)$;

$\text{EPS}(X + Y)$. Исправьте их. || ☐

Теперь у нас достаточно знаний, чтобы разобраться в алгоритме, который ученик написал для своей задачи, пользуясь средствами языка Паскаль:

```
PROGRAM XX2X3(INPUT,OUTPUT);
VAR K:INTEGER;
    X,X2,X3:REAL;
BEGIN
  WRITE ('РЕЗУЛЬТАТЫ:ЧИСЛО,КВАДРАТ,
        КВАДРАТНЫЙ КОРЕНЬ');
  FOR K:=1 TO 20 DO
    BEGIN
      WRITELN;
      READ(X);
      X2:=SQR(X);
      X3:=SQRT(X);
      WRITE(X,X2,X3)
    END
  END.
```

В файл INPUT следует поместить набор исходных данных.

|| Напишите программу для вычисления синусов и косинусов 30 чисел, задаваемых во входном файле. || ☐

|| Напишите программы для вычисления переменной S :

а) $S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$ при $n = 30$;

б) $S = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \dots \cdot 10$. ||

Глава 8

ЕСЛИ... ТО...

Программа ХХ2Х3, описанная в предыдущей главе, имеет свою допустимую область исходных данных: неотрицательные числа. Если из входного файла будет прочитано отрицательное число, то выполнение программы закончится аварийно и оставшиеся значения не будут обработаны.

Каждая программа, насколько это возможно, должна осуществлять контроль за тем, чтобы величины, участвующие в процессе вычислений, были допустимыми. Например, в программе ХХ2Х3, прочитав очередное значение X , нужно проверить, является ли оно неотрицательным:

если $X \geq 0$,

то обработать (X),

иначе выдать ('ошибочное данное: ', X).

Под действием "обработать (X)" понимаем вычисление квадрата, квадратного корня числа и вывод результатов. Обработка будет производиться лишь при выполнении условия: $X \geq 0$. В случае отрицательного X будет выдано сообщение: 'ошибочное данное' — вместе со значением X , к которому это сообщение относится.

Конструкция вида

если B

то $S1$

иначе $S2$

где B — логическое условие,

$S1, S2$ — операторы,

называется условным оператором.

Пример 1. Условный оператор используется при вычислении значения функции $y = \frac{1}{x}$:

если $x \neq 0$

то выдать ($1/x$)

иначе выдать (' \perp не входит в область определения функции $y = 1/x$ ').

В качестве логического условия здесь взято отношение $x \neq 0$.

Отношение — два выражения, соединенных операций отношения.

Пример 2. Отношениями являются равенства и неравенства:

$$y < 0, a > b, a^2 = b^2 + c^2, x \leq a - b, x \neq y.$$

Операции отношения: $>$ (больше), $<$ (меньше), \geq (не меньше), \leq (не больше), $=$ (равно), \neq (не равно) — на языке Паскаль записываются соответственно: $>$, $<$, $>=$, $<=$, $=$, $<>$ и имеют более низкий приоритет по сравнению с арифметическими операциями. Иными словами, сначала выполняются арифметические операции, а потом операции отношения.

$>$ больше
 $<$ меньше
 $>=$ не меньше
 $<=$ не больше
 $=$ равно
 $<>$ не равно

Пример 3. Условие $a + b \neq c + d$ на языке Паскаль записывается так:

$$A + B <> C + D.$$

Запишите средствами языка Паскаль отношения, приведенные в примере 2. □

В зависимости от значений переменных, входящих в состав отношения, условия могут выполняться или не выполняться. В первом случае говорят, что условие истинно, во втором — ложно.

Пример 4.

а) условие $y < 0$ при $y = 1$ ложно, а если $y = -2$, то истинно;

б) $a^2 = b^2 + c^2$ при $a = 5$, $b = 4$, и $c = 3$ истинно;

в) $a^2 < 0$ заведомо ложно.

Истинны или ложны условия:

а) $a + b < a - b$ при $a = 2$, $b = -3$;

б) $\frac{d}{f} \leq \frac{c}{e}$ при $d = 7$, $c = -12$, $f = 3$, $e = -2$?

Пример 5. Условие принадлежности точки с координатами (x, y) кругу с центром в точке $(0, 0)$ и радиусом r :

$$x^2 + y^2 \leq r^2.$$

Запишите условие того, что точка (x, y) лежит вне круга, центр которого точка $(1, 0)$, а радиус равен 5. Проверьте, удовлетворяет ли этому условию точка $(4, 4)$. ○

Всякое условие может быть либо истинным, либо ложным. Истина и ложь — логические значения — на языке Паскаль обозначаются с помощью ключевых слов TRUE и FALSE.

TRUE — истина

FALSE — ложь

Эти значения упорядочены и удовлетворяют соотношению:
 $FALSE < TRUE$.

Выражение, служащее для вычисления логического значения, называется логическим выражением или логическим условием.

Отношения являются частным случаем логических выражений.

Пример 6. Значением логического выражения $1 * 1 = 2 - 1 > FALSE$ является TRUE.

Покажем это.

Арифметические выражения имеют больший приоритет, т. е. они выполняются раньше, поэтому выражение эквивалентно такому: $1 = 1 > FALSE$.

Операции отношения выполняются слева направо в порядке их следования в выражении: $TRUE > FALSE$. В силу упорядоченности логических значений это выражение является истинным.

|| Вычислите: || □
 $FALSE > (X - 2 > 0)$ при $X = -3$.

Пример 7. Известно, что $25/5$ равно 5, однако, проверяя условие $25/5 = 5$, машина не всегда выдаст значение TRUE. Это связано с тем, что вещественные числа представляются в памяти ЭВМ приближенно. Поэтому вещественные значения в силу ограниченности их представления сравниваются с некоторой погрешностью. Например, два вещественных числа будем считать равными, если они отличаются не более чем на 10^{-7} . Соответствующее условие запишется так: $|25/5 - 5| < 10^{-7}$ или на языке Паскаль:

$ABS(25/5 - 5) < 1E-7$.

|| Как следует проверить равенство значений двух || ?
вещественных переменных A и B?

Одним из важных примеров использования логиче-

ских условий является условный оператор. Условный оператор позволяет ЭВМ выбрать тот или иной способ действий, тот или иной оператор в зависимости от значения логического условия.

Пример 8. Переменной М присвоить значение большего из двух чисел а и б. В зависимости от результатов сравнения значений а и б переменную М следует положить равной либо а, либо б. Опишем алгоритм словесно:

1. начало;
2. читать (а, б);
3. если $a > b$
 - 3.1 то $M := a$
 - 3.2 иначе $M := b$ (* в этом случае а не больше б *);
4. печать (М)
5. конец.

При описании алгоритма с помощью блок-схемы проверка логического условия В изображается блоком (рис. 5), имеющим два выхода.

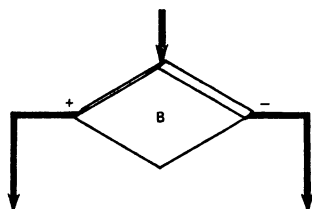


Рис. 5

Знаком "+" помечается ветвь, соответствующая выполнению условия В (истинному его значению), знаком "—" — соответствующая невыполнению условия В (ложному его значению). Изображая алгоритм с помощью блок-схемы, условный оператор будем представлять с помощью базовой конструкции: (рис. 6)

Правила выполнения условного оператора предусматривают такую последовательность действий: вычислить значение логического выражения В, при его истинности выполнять оператор S1, в противном случае — оператор S2.

По каждой ветви выполняется только один оператор.
Пример 9. Блок-схема нахождения большего из двух

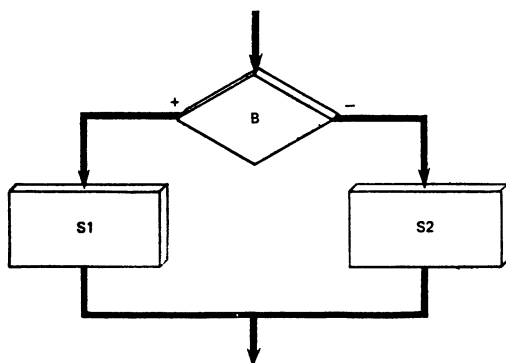


Рис. 6

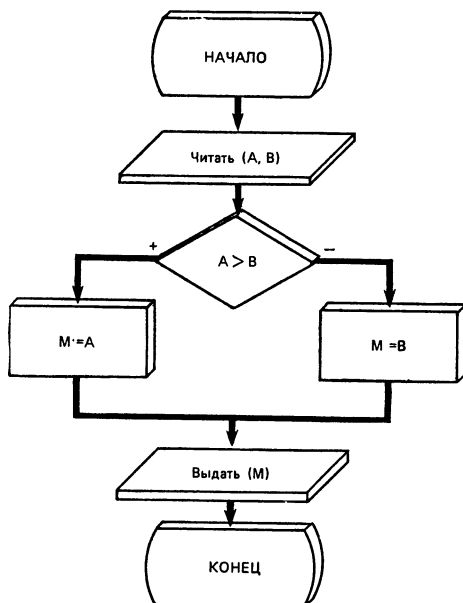


Рис. 7

чисел (рис. 7). Здесь действие «читать» изображается с помощью параллелограмма, так же как и действие «выдать».

Опишите словесно и с помощью блок-схемы алгоритм вычисления:

а) функции

$$y = \begin{cases} x^2 & \text{при } x \leq 0 \\ \sqrt{x} & \text{при } x > 0 \end{cases}$$

для одного прочитанного значения x ;

б) функции, задаваемой графиком (рис. 8),
для одного прочитанного значения x .

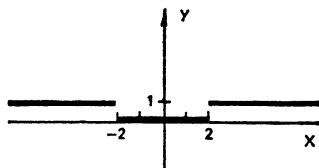


Рис. 8

При записи условного оператора на языке Паскаль:

```
IF <условие>  
THEN <оператор 1>  
ELSE <оператор 2>
```

используются следующие ключевые слова:

IF — если

THEN — то

ELSE — иначе

Пример 10. Программа MAX2 находит большее из двух значений:

```
PROGRAM MAX2(INPUT,OUTPUT);  
VAR A,B,M:REAL;  
BEGIN  
  READ(A,B); (* читать значения A и B *)  
  IF A > B  
    THEN M:=A (* A больше B *)  
    ELSE M:=B; (* A не больше B *)  
  WRITE(M)  
END.
```

Составьте программы вычисления функций, описанных в предыдущем упражнении.

Чтобы улучшить программу XX2X3 (см. предыдущую главу), воспользуемся условным оператором:

если $x \geq 0$,
то обработать (X),
иначе выдать (X, 'ошибочное данные').

Изобразим этот оператор в виде элемента блок-схемы (рис. 9) По ветви, соответствующей истинности условия, ЭВМ должна выполнить последовательность приказов. Напомним, что условный оператор может управлять лишь одним оператором. Как быть? Как разрешить противоречие? Для решения этой проблемы прибегнем к приему, описанному в известной сказке Шарля Перро: чтобы справиться с людоедом, кот попросил его превратиться в маленькую мышку.

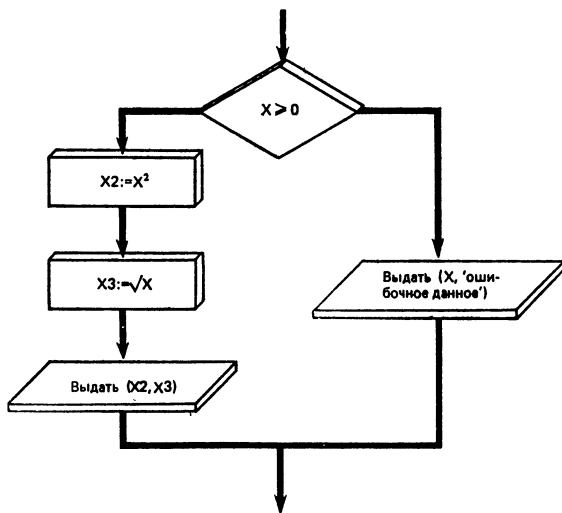


Рис. 9

Для превращения последовательности операторов в один оператор не требуется волшебства: достаточно последовательность операторов заключить в операторные скобки: BEGIN, END — и вы получите один составной оператор:

```
BEGIN
X2 := X * X;
X3 := SQRT(X);
WRITE(X, X2, X3)
END.
```

Составной оператор — это последовательность, состоящая из одного или нескольких операторов, разделенных ";", заключенная в операторные скобки BEGIN, END. Символ ";" служит как разделитель между операторами.

Пример 11. Составным оператором является

BEGIN

S:=0;

END.

Символ ";" в данном случае разделяет оператор присваивания S:=0 и пустой оператор.

Пустой оператор не влечет никаких действий и в записи программы никак не обозначается.

Пример 12. Составной оператор

BEGIN END

включает лишь один пустой оператор.

Сколько операторов входит в следующий составной:

а) BEGIN б) BEGIN;

S:=0; END

END

|| Запишите улучшенный вариант программы XX2X3. Сколько составных операторов она содержит? || □

Глава 9 СНОВА ПОГОВОРИМ О ЦИКЛАХ

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

Попробуем изобразить с помощью блок-схемы некоторый циклический алгоритм, например алгоритм вычисления суммы чисел, читаемых из входного файла.

Исходными данными в этом случае являются переменная N —количество чисел и набор N чисел. Значение, читаемое из входного файла, обозначим переменной X .

Результатом работы алгоритма станет сумма этих чисел, которую обозначим переменной S .

Допустимые значения переменной N должны удовлетворять условию $N > 0$, так как неразумно говорить о сумме не положительного количества слагаемых.

Опишем алгоритм с помощью блок-схемы (рис. 10.)

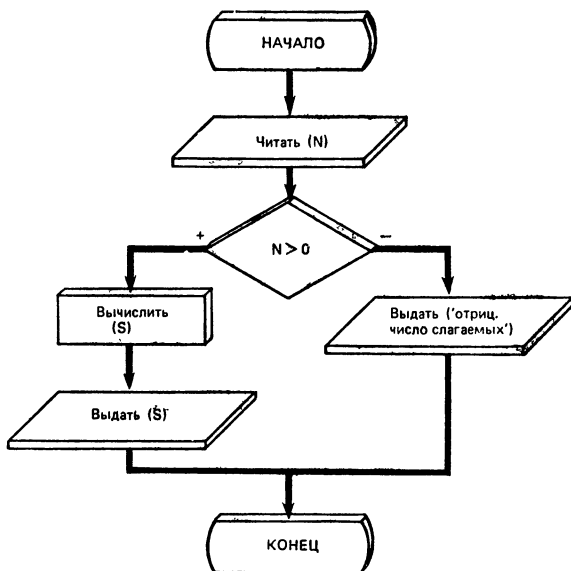


Рис. 10

Теперь расшифруем блок:
вычислить (S).

При вычислении суммы пользуются следующим приемом: вначале, когда еще не прочитано ни одно слагаемое, сумму полагают равной нулю, а затем, получая очередное слагаемое, прибавляют его к сумме:

$S := 0$;

для $K := 1$ до N повторять

{ читать (X);

$S = S + X$ }.

Блок-схема этой части алгоритма имеет следующий вид (рис. 11).

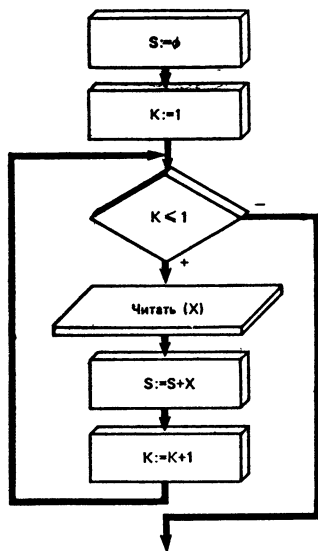


Рис. 11

Параметр цикла K принимает свое начальное значение, после чего проверяется условие: не превосходит ли текущее значение параметра цикла его конечного значения.

В случае истинности этого условия выполняется тело цикла, состоящее в данном примере из приказа "читать" и оператора присваивания. Затем параметр цикла принимает свое следующее значение, и вновь проверяется условие $K \leq N$. Если текущее значение параметра больше его конечного значения, то оператор цикла заканчивает свою работу.

Итак, оператору цикла с параметром соответствует следующая базовая конструкция (рис. 12), где S — тело цикла — один оператор, при необходимости составной.

|| Изобразите блок-схему алгоритма вычисления про- ||
 || изведения N чисел, читаемых из входного файла ||
 Проверим блок вычисления суммы на простейшем примере. Пусть уже прочитано значение N , равное 3, а во входном файле находятся числа: 5, 7, —3.

Будем имитировать работу машины, а результаты отразим в виде следующей таблицы (табл. 1).

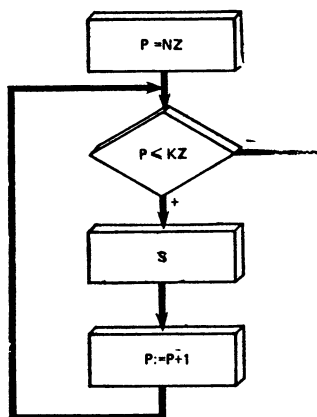


Рис. 12

Таблица 1

Шаг	N	K	X	S	Содержание действия	
0	3				Прочитано N.	
1				0	Сумма получает начальное значение.	
2		1			Начинает работать оператор цикла, $K:=1$.	
3					Проверяется условие $K \leq N$ ($1 \leq 3$), оно истинно.	
4				5	Очередное слагаемое читается из входного файла.	
5				5	Вычисляется переменная $S:S:=S+X$, $S=0+5$, $S=5$.	
6		2			Изменяется значение параметра цикла: $K:=K+1$, $K=1+1$, $K=2$.	
7					Условие $K \leq N$ ($2 \leq 3$) истинно.	
8				7	Чтение очередного слагаемого.	
9				12	Выполнение оператора $S:=S+X$, $S=5+7$, $S=12$.	
10		3			Изменение параметра цикла.	
11					Условие $K \leq N$ ($3 \leq 3$) истинно.	
12					Чтение очередного слагаемого.	
13				-3	9	Вычисление S : $S:=S+X$, $S=12-3$, $S=9$.
14		4				Изменение значения параметра цикла.
15						Условие $K \leq N$ ($4 \leq 3$) ложно, выход из цикла.

В результате работы алгоритма переменные N, X, S примут значения соответственно 3, —3, 9. А значение переменной K будет не определено.

После окончания цикла значение параметра цикла не определено.

Выполняя функции исполнителя алгоритма, программист проверяет программу в работе. Это дает ему более наглядное представление о процессе, заложенном в программе, и позволяет выявить ошибки алгоритма еще до передачи задания на ЭВМ.

|| Проверьте вручную, как будет работать алгоритм суммирования, если удалить проверку условия $N > 0$, и прочитано отрицательное значение N, например $N = -1$. || ○
Проверили?

Итак, аварийного прекращения выполнения программы (как говорят, аварийного останова) не произошло, и в качестве значения переменной S выдан нуль. Однако для нас останется непонятным, является нуль результатом сложения чисел входного файла или это следствие неправильно введенного значения переменной N. Поэтому проверку условия $N > 0$ лучше сохранить.

Если начальное значение параметра цикла превосходит его конечное значение в операторе:

FOR P: = NZ TO KZ DO
S,

то тело цикла не выполнится ни разу.

|| Чему равно значение P, определяемое в ходе выполнения операторов: || ○

- а) $P := 1;$
FOR K: = —1 TO —5 DO
P: = P*K
- б) $X := 2;$
P: = 1;
FOR K: = 0 TO 2 DO
P: = P*X.

Составьте алгоритм и программу вычисления суммы:

- а) $2^2 + 4^2 + 6^2 + \dots + 100^2$,
б) $1/1^2 + 1/3^2 + 1/5^2 + \dots + 1/99$ и
произведения: а) $1 \cdot 2 \cdot 3 \cdot \dots \cdot n$;
б) x^n ,
в) $\cos(1) \cdot \cos(2) \cdot \dots \cdot \cos(n)$.

□

Как бы ни была тщательно проверена программа вручную, установить ее пригодность для работы можно только по результатам выполнения ее на ЭВМ. Для этого специальным образом подбирают различные варианты исходных данных, и с этими данными реализуют программу на ЭВМ. Затем задачу решают вручную и сравнивают полученные результаты.

Пример 1. Вычислить значение многочлена:

$$a_0 + a_1X + a_2X^2 + \dots + a_{N-1}X^{N-1} + a_NX^N.$$

Исходными данными являются:

N —степень многочлена ($N \geq 0$), переменная X и коэффициенты $a_0, a_1, a_2, \dots, a_N$.

Значения этих величин читаются из входного файла. Результат—значение многочлена, обозначим переменной Y .

Решение задачи будем производить по следующему алгоритму:

1. начало;
2. читать (N);
3. если $N \geq 0$
 - 3.1. то { вычислить (Y);
 - 3.2. выдать (Y)
 - 3.3. иначе выдать (' N отрицательно');
4. конец.

Блок-схема алгоритма (рис. 13).

Переменную Y можно представить как сумму $N + 1$ слагаемого. При этом первое слагаемое состоит только из коэффициента a_0 , а все остальные—из произведения коэффициента и степени переменной X .

Пусть Z —степень переменной X , входящая в состав очередного слагаемого, A —прочитанное из входного файла значение коэффициента, тогда блок "вычислить (Y)" уточним так:

- читать (X);
- читать (A);
- здать начальное значение Z ;

```

Y := A;
для K := 1 до N повторить
    { читать (A);
      вычислить (Z);
      Y := Y + A * Z }.

```

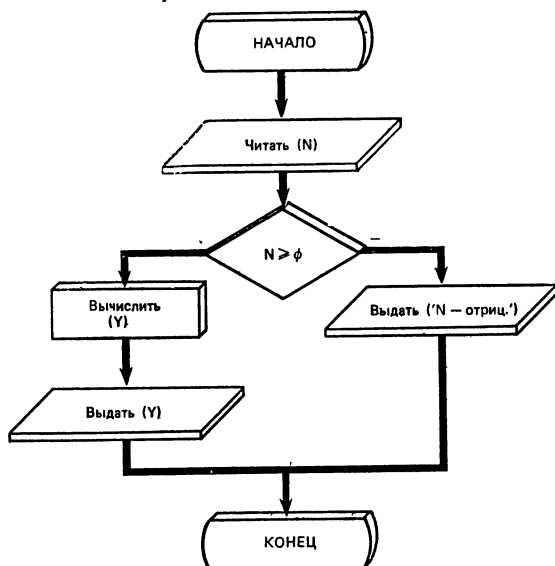


Рис. 13

В данном случае, вычисляя очередное значение Z , целесообразно использовать результаты предыдущего шага. Текущее значение Z определяется оператором $Z := Z * X$, а начальным значением Z является 1.

Итак, опишем алгоритм вычисления значения многочлена N -й степени с заданными коэффициентами:

начало;

читать (N) ;

если $N \geq 0$

то { читать (X) ;

читать (A) ;

$Z := 1$;

$Y := A$;

для $K := 1$ до N повторять

{ читать (A) ;

$Z := Z * X$;

$Y := Y + A * Z$ }

выдать (Y) }

иначе выдать ('N—отрицательно');

конец.

|| Составьте блок-схему этого алгоритма. || □

А вот как выглядит программа POLINOM—записанный средствами языка Паскаль алгоритм вычисления значения многочлена:

```
PROGRAM POLINOM(INPUT, OUTPUT);
(* ВЫЧИСЛЕНИЕ МНОГОЧЛЕНА N-СТЕПЕНИ *)
(* С ЗАДААННЫМИ КОЭФФИЦИЕНТАМИ *)
VAR K,N:INTEGER;
    A,X,Y,Z:REAL;
BEGIN
  READ(N); (* СТЕПЕНЬ МНОГОЧЛЕНА *)
  IF N >= 0
    THEN BEGIN (* ВЫЧИСЛИТЬ Y *)
      READ(X);
      READ(A);
      Z:=1;
      Y:=A;
      FOR K:=1 TO N DO
        BEGIN READ(A);
          Z:=Z*X;
          Y:=Y+A*Z
        END;
      WRITE(Y)
    END
    ELSE WRITE('N—ОТРИЦАТЕЛЬНО:', N)
  END.
```

|| Произведение $1 \cdot 2 \cdot 3 \cdot \dots \cdot N$ обозначим $N!$ (N — факториал). Например, $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$. || □

|| По определению $0!$ положим равным 1. Запишите алгоритм и программу вычисления переменной p : || □

$$\text{а) } p = 1! + 2! + 3! + \dots + n!; \quad \text{б) } p = \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!}$$

$$\text{в) } p = \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}.$$

Программу POLINOM выполните со следующими вариантами исходных данных:

з) $N=2$, $a_0=1$, $a_1=1$, $a_2=1$.

В этом случае многочлен выглядит так:

$$1 + X + X^2$$

Его значением при $X=2$ является 7,

б) $N=1$, $a_0=1$, $a_1=1$, $X=2$;

в) $N=0$, $a_0=1$, $X=2$;

г) $N=-1$.

|| Как выглядит многочлен и чему равно его значение в случаях б), в), г)? ||

|| Программу следует проверять не только на правильных, но и при недопустимых исходных данных. ||

Различные варианты исходных данных, на которых проверяется работоспособность программы, вместе с эталонными результатами называются тестами, а этап испытания программы с целью выявления содержащихся в ней ошибок — тестированием.

Существуют специальные правила подбора тестовых примеров. Однако главным нужно считать следующее: тесты следует подбирать таким образом, чтобы они не подтверждали правильность программы, а выявляли имеющиеся в ней ошибки.

Тестирование предполагает сравнение результатов работы ЭВМ с эталонными результатами. Если решение задачи вручную невозможно или требует больших затрат, то как проверить работу программы? В этом случае для сравнения можно взять результаты решения, полученные по другому алгоритму.

Пример. Решение задачи, рассматриваемой в предыдущем примере, можно выполнить другим способом.

Многочлен нулевой степени имеет вид a_0 , первой — $a_1X + a_0$, второй — $a_2X^2 + a_1X + a_0 = (a_2X + a_1) \cdot X + a_0$, третьей — $a_3X^3 + a_2X^2 + a_1X + a_0 = (a_3X^2 + a_2X + a_1)X + a_0 = ((a_3X + a_2)X + a_1)X + a_0$.

|| Как выглядит многочлен четвертой степени? ||

Если первоначальным значением многочлена считать нуль, то алгоритм будет состоять из $N+1$ шагов, заключающихся в умножении текущего значения многочлена на X и прибавлении очередного коэффициента:

начало;

читать (N) ;

если $N \geq 0$,

```

то { читать (X);
    Y:=0;
    для K:=0 до N повторять
    { читать (A);
      Y:=Y*X+A }
    выдать (Y) }
иначе выдать ('N отрицательно', N);
конец.

```

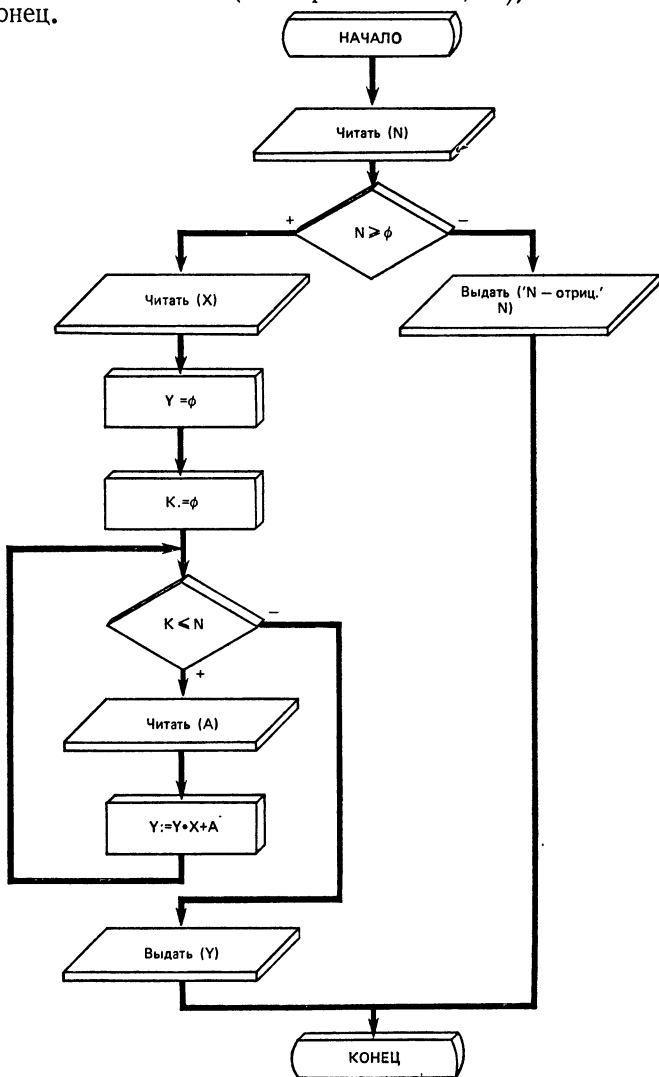


Рис. 14

Блок-схема решения задачи (рис. 14).

Запишем алгоритм на языке Паскаль:

```
PROGRAM POLINOM2(INPUT, OUTRUT);
(* ВЫЧИСЛЕНИЕ ЗНАЧЕНИЯ МНОГОЧЛЕНА *)
VAR K,N:INTEGER;
    A,X,Y:REAL;
BEGIN
  READ(N);
  IF N >= 0
  THEN BEGIN
    READ(X);
    Y:=0;
    FOR K:=0 TO N DO
      BEGIN
        READ(A);
        Y:=Y*X + A
      END;
    WRITE(Y)
  END
  ELSE WRITE ('N ОТРИЦАТЕЛЬНО', N);
END.
```

Коэффициенты во входном файле должны располагаться в следующем порядке: a_N, a_{N-1}, \dots, a_0 .

|| Проверьте алгоритм вручную, имитируя работу || ○
машины.

Рассмотренный метод вычисления значения многочлена называется схемой Горнера.

|| Какой из двух приведенных алгоритмов вы- || ?
числения значения многочлена является более
эффективным по количеству выполняемых опе-
раций? ||

Глава 10

ИЩЕМ БРАКОВАННЫЕ СТЕРЖНИ, А НАХОДИМ... ЛОГИЧЕСКИЕ ПЕРЕМЕННЫЕ И ЛОГИЧЕСКИЕ ОПЕРАЦИИ

Некий цех выпускает металлические стержни. Для проверки качества отобраны пятьдесят экземпляров. Стержень считается стандартным, если его длина отличается от заданной длины L не более чем на величину A , а диаметр — от заданного диаметра не более чем на B . В противном случае изделие является браком.

Требуется подсчитать количество стандартных стержней и массу всех бракованных изделий.

Чтобы решить эту задачу, контролеру должны быть известны: стандартные длина (L) и диаметр (D), максимальные значения возможных отклонений (A, B), плотность металла, из которого изготовлены стержни (RO), а также длина (DL) и диаметр (DM) каждого стержня.

В результате решения задачи будут определены количество стандартных стержней (KOL) и масса бракованных изделий (M). Опишем последовательность действий контролера:

1. задать величины (L, D, A, B, RO);
2. задать начальные значения KOL и M;
3. просмотреть 50 стержней и для каждого выполнить действия:

3.1. { определить DL, DM (* длина и диаметр *);

3.2. если стержень стандартный,
то увеличить на 1 KOL (* количество стандартных стержней *),
иначе изменить величину M };

4. выдать (KOL, M).

Уточним некоторые этапы алгоритма. Вначале, когда ни один стержень еще не просмотрен, переменные KOL и M разумно положить равными нулю. В результате проверки может оказаться, что ни один стержень не является стандартным—в этом случае KOL=0. Если все стержни удовлетворяют заданным требованиям, то M=0.

Действие "увеличить на 1 количество стандартных стержней" реализуется оператором: KOL:=KOL+1.

Изменение переменной M означает ее увеличение на массу очередного стандартного стержня. Известно, что масса стержня длины l, диаметра d, изготовленного из металла плотности ρ, определяется формулой:

$$m = \frac{\pi d^2 l \rho}{4}.$$

Поэтому переменную M следует увеличить на

$$3.1415 * DM * DM * DL * RO / 4.$$

Вясним, как в математической форме записать условие: стержень стандартный. Это условие означает, что отклонение длины стержня от заданной не превосходит величины A, т. е. $|L - DL| \leq A$. Диаметр же отличается от заданного не более чем на B, т. е.

$|D - DM| \leq B$. Причем оба требования должны выполняться одновременно. Условие стандартности стержня сформулируем так:

$$(|L - DL| \leq A) \text{ И } (|D - DM| \leq B).$$

Союзу И поставим в соответствие ключевое слово AND языка Паскаль, которое является знаком операции логического умножения или конъюнкции.

AND—И

Операция И, будучи примененной к двум логическим выражениям, дает значение TRUE, если оба выражения являются истинными.

Логические переменные — переменные, которые могут принимать одно из двух логических значений: истину или ложь.

Пусть X и Y — логические переменные.

Тогда $X \text{ AND } Y = \begin{cases} \text{TRUE, если } X = \text{TRUE и} \\ Y = \text{TRUE} \\ \text{FALSE в противном случае} \end{cases}$

Вычислите: а) $X \text{ AND } Y \text{ AND } Z$ при $X = \text{TRUE}$
 $Y = \text{TRUE}$
 $Z = \text{TRUE}$

б) $X \text{ AND } Y \text{ AND FALSE}$ при
 $X = \text{TRUE}$
 $Y = \text{TRUE}$

Вернемся к алгоритму решения задачи:

Блок-схема алгоритма (рис. 15).

1. начало;
2. читать (L, D, A, B, RO);
3. KOL := 0; M := 0;
4. для i := 1 до 50 повторять
 - 4.1 {читать (DL, DM);
 - 4.2. если $(|L - DL| \leq A)$ и $(|D - DM| \leq B)$
 - 4.2.1 то KOL := KOL + 1
 - 4.2.2 иначе M := M + 3.1415 * DM * DM * DL * RO / 4;
5. выдать (KOL, M);
6. конец.

Условие C — $(|L - DL| \leq A)$ и $(|D - DM| \leq B)$.

Оператор S — $M := M + 3.1415 * DM * DM * DL * RO / 4$.

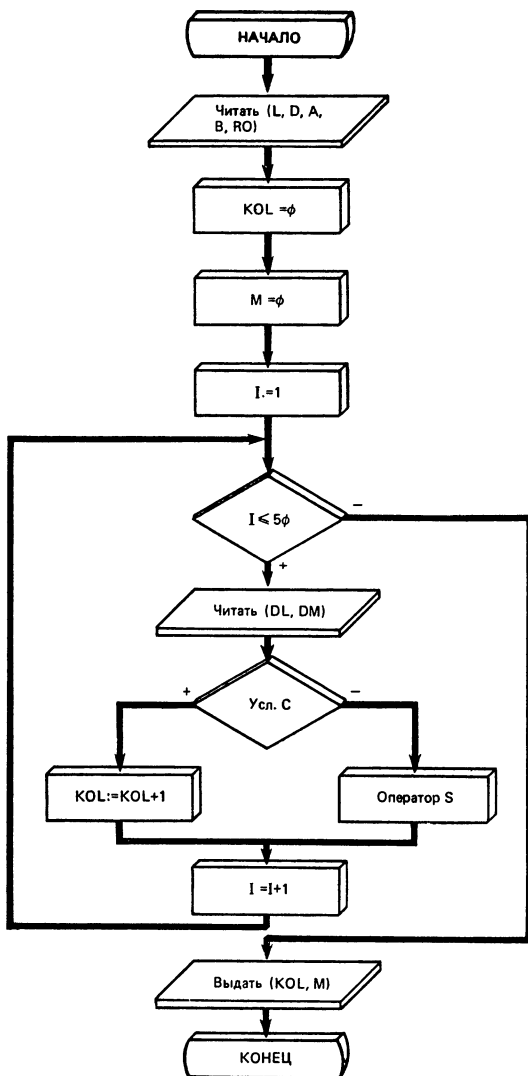


Рис. 15

Если содержимое отдельных блоков велико, то оно может выписываться отдельно, а не на блок-схеме.

А вот как выглядит программа, реализующая этот алгоритм на языке Паскаль:

```
PROGRAM KONTROL (INPUT, OUTPUT);
VAR I, KOL:INTEGER;
    A, B, D, DL, DM, L, M, RO:REAL;
BEGIN
  READ(L, D, A, B, RO);
  KOL:=0; M:=0;
  FOR I:=1 TO 50 DO
    BEGIN
      READ (DL, DM);
      IF (ABS(L—DL) <= A)AND(ABS(D—DM)
        <= B)
        THEN KOL:=KOL+1
        ELSE M:=M+3.1415*DM*DM*DL*RO/4
      END;
    WRITE (KOL, M)
  END.
```

Помимо конъюнкции в языке Паскаль используются логические операции "не" и "или".

Пример. Значение переменной Y зависит от значения X следующим образом:

$$Y = \begin{cases} 1, & \text{если } X \in [A, B]; \\ 0, & \text{если } X \notin [A, B]. \end{cases}$$

Напомним, X принадлежит отрезку [A, B], если выполняется условие: $A \leq X \leq B$.

Как записать это условие на языке Паскаль? Легко сообразить, что запись $A \leq X \leq B$ будет неверной. Объясним, почему. Операции отношения выполняются слева направо в порядке их следования в логическом выражении. В результате выполнения операции $A \leq X$ будет получено логическое значение (TRUE или FALSE). Далее его следует сравнить с некоторым числовым значением B. Разумеется, такое действие не выполнимо.

Условие $A \leq X \leq B$ равносильно тому, что условия $A \leq X$ и $X \leq B$ выполняются одновременно. На языке Паскаль, как мы знаем уже, это может быть записано с использованием конъюнкции: $(A \leq X) \text{AND} (X \leq B)$.

|| Запишите алгоритм вычисления переменной Y. || □

Сформулируем условие: X не принадлежит отрезку [A, B]. В этом случае истинным является не

$(A \leq X) \text{ AND } (X \leq B)$, а противоположное ему условие.

Логическая операция "не" в языке Паскаль обозначается ключевым словом NOT, а ее выполнение определяется правилом: пусть X — логическая переменная, тогда

$$\text{NOT } X = \begin{cases} \text{TRUE,} & \text{если } X = \text{FALSE;} \\ \text{FALSE,} & \text{если } X = \text{TRUE.} \end{cases}$$
 .NOT — не.

Используя эту операцию, условие $X \notin [A, B]$ можно записать так: $\text{NOT}((A \leq X) \text{ AND } (X \leq B))$.

|| Упростите NOT NOT A || ○

С другой стороны, если X расположен вне отрезка $[A, B]$, то справедливо либо утверждение $X > B$, либо утверждение $X < A$, либо то и другое одновременно, например, при $A = 10$, $B = 51$, $X = 71$. Поэтому условие $X \notin [A, B]$ эквивалентно такому: $(A > X)$ или $(X > B)$.

Операция "или" — логическое сложение (дизъюнкция) обозначается ключевым словом OR.

OR — или

Результат выполнения дизъюнкции может быть описан с помощью таблицы (табл. 2).

Т а б л и ц а 2

X	Y	X OR Y
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

где X и Y — логические переменные.

|| Упростите выражения

- а) $X \text{ OR } \text{FALSE}$; в) $X \text{ OR } X$;
 б) $X \text{ OR } \text{TRUE}$; г) $X \text{ OR } \text{NOT } X$.

|| ○

Пример. Вычислим значение логического выражения:

$X \text{ OR } Y = \text{NOT } X \text{ AND } \text{NOT } Y$

при $X = \text{TRUE}$ и $Y = \text{FALSE}$.

Нам потребуется знание порядка выполнения операций.

Самый высокий приоритет имеет операция отрицания NOT. За ней следуют мультипликативные операции: *, / , AND. На третьем уровне — аддитивные операции +, —, OR, и самый низкий приоритет у операций отношения. В соответствии с этими приоритетами выполняются все операции.

Подставим вместо X и Y в заданном выражении их значения:

TRUE OR FALSE = NOT TRUE AND NOT FALSE.

Выполним операцию отрицания:

TRUE OR FALSE = FALSE AND TRUE.

Это эквивалентно выражению

TRUE OR FALSE = FALSE, или
TRUE = FALSE, или FALSE.

|| Вычислите при A = TRUE, B = TRUE: || □
 а) NOT A OR B в) NOT(NOT A AND NOT B)
 б) NOT (A OR B) г) A AND NOT A ||

Пример: сравним записи NOT A и NOTA.

Первая представляет собой отрицание переменной A, вторая — имя переменной NOTA.

Знаки логических операций должны отделяться от операндов (выражений, к которым они относятся) скобками или пробелами.

|| Запишите на языке Паскаль выражения: || □
 а) не A в) A или не A
 б) A или A г) A или не A и A. ||

Глава II

САМЫЙ БОЛЬШОЙ И САМЫЙ МАЛЕНЬКИЙ

На краю поля лежат 100 арбузов. Вы хотите выбрать арбуз с самым большим весом. Как бы вы осуществили свое намерение? Разумеется, в вашем распоряжении имеются весы. Возможно, что вы выберете такой путь.

1. Выбираете произвольно какой-либо арбуз, взвешиваете

ваете его и, так как ничего лучшего пока нет, полагаете:

пусть это самый тяжелый арбуз. (Пока!)

2. Далее вы взвешиваете следующий арбуз и сравниваете с тем, который пока считался наибольшим.

Если очередной вес оказался больше прежнего, считавшегося максимальным, то наибольшим называете только что взвешенный арбуз, а прежний откладываете в сторону. В противном случае в сторону откладываете только что взвешенный арбуз.

Так будете поступать, пока не сравните вес всех ста арбузов.

Теперь опишем алгоритм, которым могла бы пользоваться ЭВМ при выборе наибольшего арбуза. Впрочем, ЭВМ не сможет взвешивать арбузы, а будет искать наибольшее из ста чисел.

Пусть X — вес очередного арбуза,

I — номер взвешиваемого арбуза (или номер шага процесса поиска),

MAX — максимальный вес, определяемый на каждом шаге.

Взвесить арбуз означает узнать вес арбуза. Этому действию мы поставим в соответствие приказ: читать (X). Тогда алгоритм будет выглядеть так:

начало;

$I := 1$; (* первый шаг *)

читать (X); (* взвесить первый арбуз *)

$MAX := X$; (* считать его наибольшим *)

для $I := 2$ до 100 повторять (* сравнить с остальными *)

 читать (X); (* взвесить очередной арбуз *)

 если $X > MAX$

 то $MAX := X$ (* вес очередного арбуза оказался бóльшим *)

 иначе ; (* действий нет *)

печать (MAX);

конец.

В условном операторе

если $X > MAX$

 то $MAX := X$

иначе

по ветви "иначе" находится пустой оператор. Это означает, что в случае невыполнения условия $X > MAX$ никаких специальных действий не предусмотрено. В таком случае условный оператор может быть записан

более кратко:

если $X > \text{MAX}$
то $\text{MAX} := X$

Конструкция

если B

то S ,

где B — логическое условие,

S — оператор,

представляет собой неполную форму условного оператора (или неполный условный оператор).

На языке Паскаль неполный условный оператор записывается так:

IF B

THEN S .

Его работа описывается следующей базовой конструкцией (рис. 16).

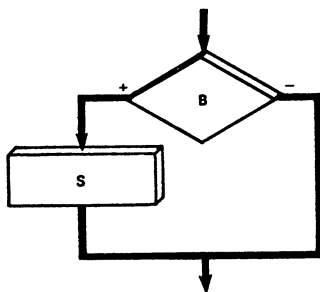


Рис. 16

Порядок действий здесь следующий:

1. Вычисляется значение логического выражения B .
2. Если оно истинно, то выполняется оператор S , а затем оператор, следующий за условным. Иначе, в случае ложности условия, сразу выполняется оператор, следующий за условным.

Опишем решение задачи с помощью блок-схемы (рис. 17).

Замечание: оператор $I := 1$ используется только для наглядности, чтобы выделить первый шаг. При желании он может быть включен в цикл.

Вес арбуза — некоторое вещественное число, поэтому

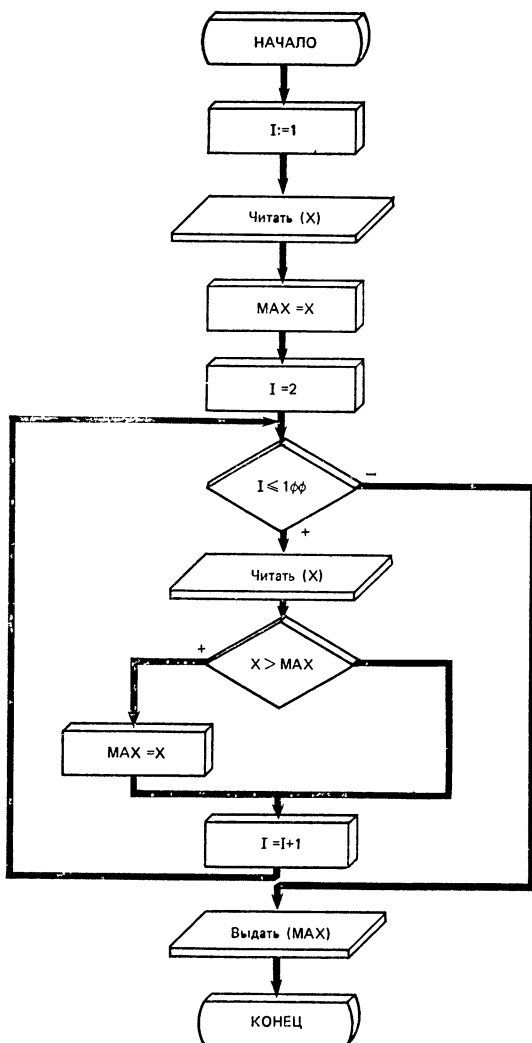


Рис. 17

переменные X и MAX — вещественные. Номер шага I — переменная целого типа.

Рассматриваемый алгоритм может быть реализован следующей программой на языке Паскаль:

```

PROGRAM BEC (INPUT, OUTPUT);
VAR I: INTEGER;

```

```

      MAX, X:REAL;
BEGIN
  READ(X);
  MAX:=X;
  FOR I:=2 TO 100 DO
    BEGIN
      READ(X);
      IF X > MAX
        THEN MAX:=X
      END;
    WRITE(MAX)
  END.

```

И наконец, к этой программе нужно подготовить входной файл, в котором следует задать 100 чисел (перечислить вес всех ста арбузов).

|| Во входном файле заданы 30 чисел: x_1, x_2, \dots || □
 \dots, x_{30} .
 Опишите алгоритмы решения следующих задач:
 1) вычислить сумму положительных чисел;
 2) вычислить количество отрицательных чисел;
 3) выдать содержащиеся во входном файле значения, заменяя нулями отрицательные. ||

Предположим, вам интересно узнать не только максимальный вес, но и какой по счету арбуз оказался наибольшим.

Обозначим через K номер наибольшего арбуза. В ходе решения задачи нужно определить значения двух переменных: MAX и K . Очевидно, что эти значения изменяются одновременно:

1. начало;
2. читать (X);
3. $MAX:=X$; $K:=1$; (* пусть первый арбуз наибольший *);
4. для $I:=2$ до 100 повторять:
 - 4.1 {читать (X);
 - 4.2 если $X > MAX$
 - 4.2.1 то { $MAX:=X$; (* считать новое значение наибольшим *)
 - 4.2. $K:=I$ (* и запомнить его номер *);
5. выдать (K, MAX);
6. конец.

|| Опишите этот алгоритм с помощью блок-схемы || ○
 и средствами языка Паскаль.

Заметим, что в случае истинности условия $X > \text{MAX}$ следует выполнить два действия: переопределить значение максимального элемента и запомнить его номер. Условный оператор управляет только одним оператором, поэтому по ветви "то" должен находиться составной оператор.

|| Как будет работать написанная вами программа, || ?
|| если в условном операторе опустить скобки ||
|| BEGIN, END? ||

Усложним обстановку. Представим, что на краю поля находятся арбузы, причем их количество неизвестно. Задача та же: выбрать наибольший.

Вот как выглядит последовательность действий, которая приведет нас к цели:

1. взвесить первый арбуз и считать его наибольшим;
2. пока куча арбузов не пуста, повторять действия;
 - 2.1. {взвесить очередной арбуз;
 - 2.2. сравнить его с тем, который прежде считался наибольшим};
3. сообщить наибольший вес арбуза.

Если операции взвешивания арбуза мы поставили в соответствие чтение из входного файла, то условие: куча арбузов не пуста — очевидно, означает, что не все числа из входного файла еще прочитаны.

Ситуация, при которой во входном файле нет непрочитанных еще данных, называется концом файла.

Выполнять чтение данных и сравнение нужно только в случае, если данные во входном файле еще есть, т. е. выполняется условие: не конец файла.

Опишем алгоритм в более формализованном виде: начало;

читать (X);

MAX := X;

пока не конец файла повторять

{читать (X);

если $X > \text{MAX}$

то MAX := X}

выдать (MAX)

конец.

В этом алгоритме мы используем новую циклическую конструкцию.

Конструкция вида
пока В повторять

 S называется оператором цикла с предусловием

Здесь В — логическое выражение, S — оператор. Оператору цикла с предусловием на языке блок-схем соответствует следующая базовая конструкция (рис. 18).

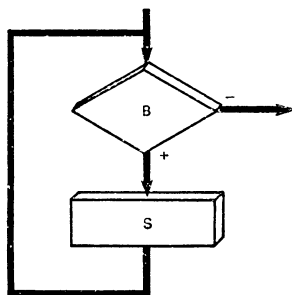


Рис. 18

Выполнение этого оператора определяется правилом: вычислить логическое условие В.

Если оно истинно, то выполнить оператор S и снова перейти к проверке условия В.

Процесс будет повторяться до тех пор, пока В не станет ложным. Тогда выполнение оператора цикла будет закончено.

Условие В является условием повторения цикла, оператор S — телом цикла, а конструкция пока В повторять — заголовком цикла с предусловием.

В нашем примере в случае, когда входной файл еще не пуст, нужно выполнить два действия: прочитать новое число и сравнить его с максимальным. Но телом цикла может быть только один оператор, поэтому он будет составным.

В отличие от оператора цикла с параметром в цикле "пока" число повторений оператора S заранее неизвестно. В частности, тело цикла может быть не выполнено ни разу.

Пример 1. В последовательности операторов:

$X := 0; Y := 0;$
пока $X > 0$ повторять
 $\{Y := Y + X * X;$
 $X := X - 1\}.$

оператор цикла закончит свою работу после первой проверки условия $X > 0$. Так как условие $0 > 0$ ложно, тело цикла выполняться не будет.

При определенных условиях тело цикла может выполняться бесконечное число раз, например, в такой последовательности операторов:

$S := 5;$
пока $S > 0.1E-7$ повторять
 $S := 10/S.$

S будет попеременно принимать значения 5 и 2 и никогда не станет меньше или равной 10^{-7} .

Означает ли это, что если ЭВМ получит соответствующую программу, то она будет выполняться бесконечно долго?

Вместе с каждым заданием ЭВМ получает информацию о том, какое наибольшее время она может потратить на его выполнение. По истечении этого времени независимо от того, выполнен алгоритм полностью или нет, выполнение задания прекращается.

В каком из примеров тело цикла будет выполняться:

- а) конечное число раз;
- б) бесконечно;
- в) не выполнится ни разу:

1) пока $S < 1E-7$ повторять

$S := S/10;$

2) пока $S > 1E-7$ повторять

$S := S * 10;$

3) пока $S > 1E-7$ повторять

$S := S/10,$

если первоначально значением S является число 0.01.

Проверим, правильно ли работает рассмотренный алгоритм отыскания наибольшего значения (см. стр. 67), если учесть, что ситуация „конец файла“ обнаруживается только после попытки чтения из пустого файла.

Пусть в файле INPUT записано единственное число: 5. Выполним алгоритм вручную (табл. 3).

Таблица 3

	X	MAX	"Конец файла"	Действие
5 пусто	5	5	нет	читать (X) MAX := 5 условие "не конец файла" истинно
?	?	выполняем тело цикла		читать (X)

Делается попытка прочитать очередное значение, но во входном файле ничего нет. Следовательно, значение X не определено, и мы не можем выполнить условный оператор, который подлежит исполнению прежде, чем будет проверено условие, записанное в заголовке цикла. Попробуем расположить операторы в теле цикла таким образом, чтобы за оператором чтения следовала проверка условия "не конец файла":

начало;

читать (X);

MAX := X;

пока не конец файла повторять

{если $X > MAX$

то MAX := X;

читать (X)};

выдать (X);

конец.

В этом случае максимальное значение сравнивается с числом, прочитанным на предыдущем шаге, а алгоритм имеет один "холостой" ход.

|| Какой? ||?

Этот недостаток — не ошибка. Худшее заключается в том, что алгоритм не сможет работать, если входной файл не содержит ни одного значения.

Исправим алгоритм:

начало;

читать (X);

если не конец файла

то {MAX := X;

пока не конец файла повторять

{если $X > MAX$

то MAX := X;

читать (X)};

выдать (MAX));
 иначе выдать ('входной файл пуст');
 конец.

Блок-схема этого алгоритма выглядит так (рис. 19):

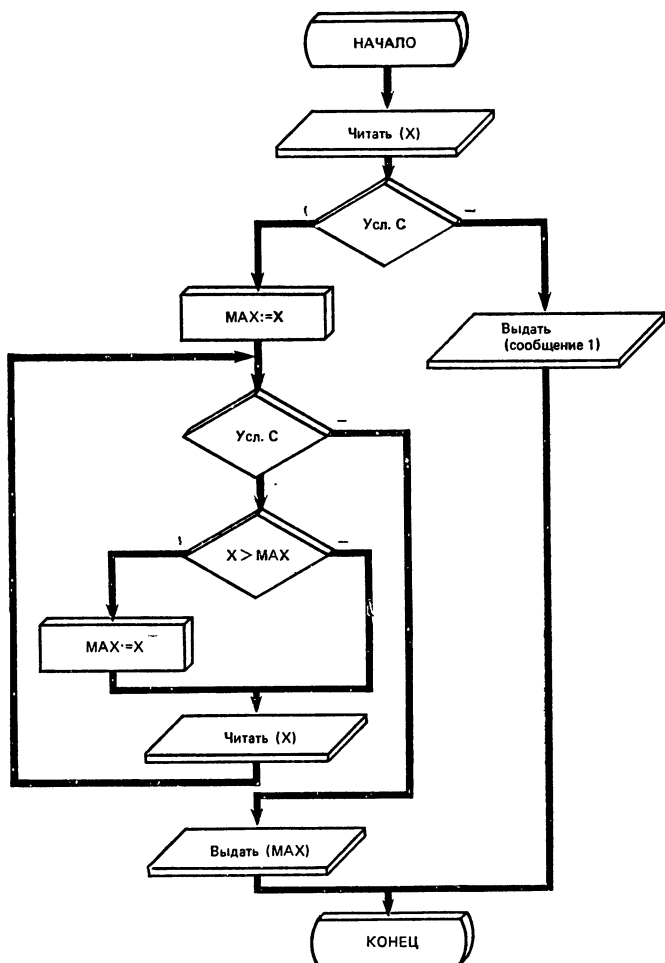


Рис. 19

Запишем этот алгоритм средствами языка Паскаль.
 Для этого нужно знать правила написания оператора
 цикла с предусловием и уметь сформулировать условие
 "не конец файла".

Конструкции

пока <условие> повторять
<оператор>
на языке Паскаль соответствует оператор
WHILE <условие> DO
<оператор>

WHILE — пока

Логическое выражение, управляющее повторением тела цикла, вычисляется при каждой операции (на каждом шаге), поэтому его следует делать как можно более простым.

Определить наличие ситуации "конец файла" можно с помощью стандартной функции EOF (END OF FILE) (конец файла). Аргументом у этой функции является имя файла, в нашем случае—INPUT : EOF(INPUT). Значение функции определяется в результате выполнения приказа "читать". Если при чтении возникает ситуация "конец файла", то функция принимает истинное значение, в противном случае—ложное значение.

Чтобы записать условие "не конец файла", используем логическую операцию NOT (отрицание): NOT EOF (INPUT). Итак, программа BECNN:

```
PROGRAM BECNN (INPUT, OUTPUT);
VAR MAX, X: REAL;
BEGIN
  READ (X);
  IF NOT EOF (INPUT)
    THEN BEGIN
      MAX:=X;
      WHILE NOT EOF (INPUT) DO
        BEGIN
          IF X > MAX
            THEN MAX:=X;
          READ(X)
        END;
      WRITE (MAX)
    END
  ELSE
    WRITE ('входной файл пуст')
END.
```

|| Найдите самый маленький арбуз из 100, лежа- || □
щих на краю поля.

Глава 12

АЛГОРИТМ, КОТОРЫЙ ПРИДУМАЛ ЕВКЛИД

В предыдущих главах мы рассмотрели различные примеры алгоритмов. Теперь остановимся на одном из самых древних и известных математических алгоритмов.

В третьем веке до нашей эры греческий математик Евклид изложил правило получения наибольшего общего делителя (НОД) двух целых чисел. Напомним, что НОД двух целых чисел называется наибольшее целое число, на которое оба заданных числа делятся без остатка.

Это правило основывается на следующих свойствах НОД.

1. Пусть a и b — целые положительные числа, причем $a \geq b$. Тогда

$\text{НОД}(a, b) = \text{НОД}(b, r)$, где r — остаток от деления нацело a на b .

Пример 1: Пусть $a = 45$, $b = 39$.

$$\text{НОД}(45, 39) = 3.$$

Вычислим остаток от деления 45 на 39:

$$r = \text{остаток}(45 : 39) = 6.$$

$$\text{НОД}(39, 6) = 3.$$

Следовательно, $\text{НОД}(45, 39) = \text{НОД}(39, 6) = 3$.

2. $\text{НОД}(a, 0) = a$.

Пример 2. $\text{НОД}(3, 0) = 3$.

Алгоритм Евклида состоит в последовательном выполнении серии шагов. Каждый шаг начинается с проверки равенства нулю числа b . Если $b = 0$, то, согласно свойству 2, наибольшим общим делителем является число a и алгоритм заканчивается. В противном случае вычисляется r — остаток от деления a на b , и, воспользовавшись свойством 1, число a заменяется на b , а b полагается равным r . Далее осуществляется переход к следующему шагу.

Рассмотрим алгоритм Евклида на примере.

Пример 3. Найти $\text{НОД}(49, 126)$.

Числа 49, 126 неотрицательны и одновременно не равны 0.

$$* \text{НОД}(126, 49) = ?$$

— Число b ($b = 49$) не равно нулю?

— Да.

— Вычислим $r = \text{остаток}(126 : 49) = 28$.

$$* \text{НОД}(126, 49) = \text{НОД}(49, 28) \text{ (по свойству 1).}$$

— Число b ($b = 28$) не равно нулю?

— Да.

— Вычислим $r = \text{остаток } (49, 28) = 21$.

* $\text{НОД}(49, 28) = \text{НОД}(28, 21)$ (по свойству 1).

— Число b ($b = 21$) не равно нулю?

— Да.

— Вычислим $r = \text{остаток } (28, 21) = 7$.

* $\text{НОД}(28, 21) = \text{НОД}(21, 7)$ (по свойству 1).

— Число b ($b = 7$) не равно нулю?

— Да.

— Вычислим $r = \text{остаток } (21, 7) = 0$.

* $\text{НОД}(21, 7) = \text{НОД}(7, 0)$.

Согласно свойству 2 $\text{НОД}(7, 0) = 7$.

Если выписать цепочку равенств, отмеченных знаком *, то сразу видно, что

$$\text{НОД}(126, 49) = 7.$$

|| Используя алгоритм Евклида, вычислите || \square
 $\text{НОД}(121, 209)$.

Опишем этот алгоритм более формально, постепенно детализируя каждый его шаг.

Пусть X — большее из двух чисел A и B , Y — меньшее.

$$\text{НОД}(A, B) = \text{НОД}(X, Y).$$

Через НОД обозначим наибольший общий делитель чисел X и Y , через R — остаток от деления X на Y . Тогда алгоритм предполагает выполнение такой последовательности действий:

1. читать (A, B) ;
2. вычислить (X, Y) ;
3. найти $\text{НОД}(X, Y)$.

Уточним некоторые этапы алгоритма.

Вычислить (X, Y) , согласно определению переменных X и Y , означает выполнение условного оператора:

если $A > B$,
то $\{X := A;$
 $Y := B\}$
иначе $\{X := B;$
 $Y := A\}$.

Чтобы применить алгоритм Евклида для отыскания НОД чисел X и Y , таких, что $X \geq Y$, они должны удовлетворять условию: $X > 0$ и $Y \geq 0$, иначе исходные данные являются ошибочными:

если $X > 0$ и $Y \geq 0$,

то вычислить $\text{НОД} = \text{НОД}(X, Y)$,
иначе выдать ('ошибка в данных').

Действие "вычислить $\text{НОД} = \text{НОД}(X, Y)$ " заключается в применении алгоритма Евклида к числам X и Y :
пока $Y \neq 0$ повторять

```
{R := остаток (X:Y);  
  X := Y;  
  Y := R}  
НОД := X;  
выдать ('Н.О.Д. чисел', A,  
'и', B, 'равен', НОД).
```

Рассматриваемый алгоритм будет завершен на некотором шаге, когда очередной остаток от деления будет равным нулю.

Для любых ли чисел A и B такая ситуация наступит, или, иными словами, конечен ли алгоритм?

Остаток всегда меньше делителя, а убывающая последовательность неотрицательных целых чисел не может быть бесконечной, поэтому на некотором шаге очередной остаток обратится в нуль, и алгоритм закончится.

Построим блок-схему алгоритма (рис. 20).

Вспомним некоторые уже известные вам сведения.

Вы знаете, что деление в языке Паскаль обозначается символом $/$. При этом результатом деления A/B является вещественное число. В рассматриваемом алгоритме R является остатком от деления нацело переменных A и B . Деление нацело в Паскале обозначается ключевым словом **DIV**.

DIV — деление нацело

Пример 4. Значением выражения $5 \text{ DIV } 2$ является 2.

Пример 5. Выражение **B DIV A** представляет собой переменную с именем **B DIV A**, но не деление нацело переменной B на A , так как отсутствуют необходимые пробелы между операторами и знаком операции **DIV**.

Деление нацело B на A правильно было бы записать так:

$B \text{ DIV } A$ или $(B) \text{ DIV } (A)$.

Деление нацело применимо только к операндам целого типа.

Пример 6. Неправильным является выражение $\text{SQRT}(X) \text{ DIV } (X + Y)/Y$, так как стандартная функция **SQRT** и деление $/$ дают вещественный результат.

Нам нужно вычислить остаток от деления нацело переменных X и Y . Если для этого использовать функ-

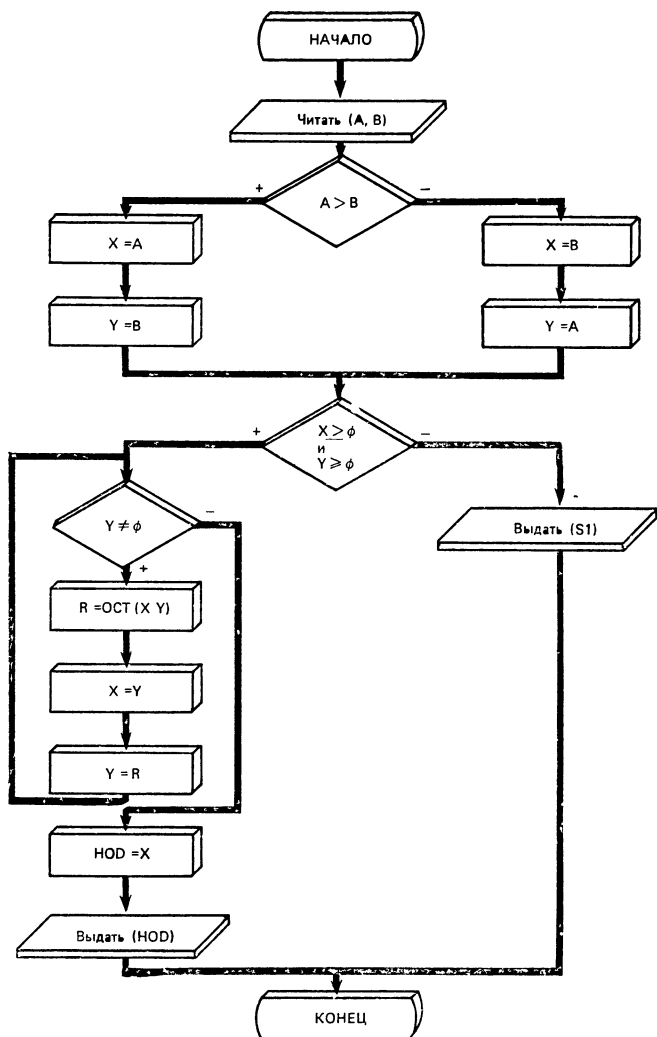


Рис. 20

цию DIV, то остаток R может быть найден с помощью оператора: $R := X - (X \text{ DIV } Y) * Y$.

Однако в языке Паскаль существует специальная операция вычисления остатка от деления нацело: MOD.

MOD — остаток от деления нацело

Пример 7. Значением выражения $A \text{ MOD } B$ при $A = 5$ и $B = 2$ является 1.

Правила использования операции MOD такие же, как и операции DIV.

Опишем алгоритм отыскания НОД средствами языка Паскаль:

```
PROGRAM HOD (INPUT, OUTPUT);
VAR A, B, NOD, R, X, Y: INTEGER;
BEGIN
  READ (A, B);
  IF A > B
    THEN BEGIN X:=A; Y:=B END
    ELSE BEGIN X:=B; Y:=A END;
  IF (X > 0) AND (Y >= 0)
    THEN BEGIN
      WHILE Y < > 0 DO
        BEGIN
          R:=X MOD Y;
          X:=Y;
          Y:=R
        END;
        NOD:=X;
        WRITE ('НОД чисел', A, 'и', B,
              'равен', NOD)
      END
    ELSE
      WRITE ('ошибка в данных')
    END.
```

ГЛАВА 13

ЕСЛИ ИСЧЕЗНУТ СТАНДАРТНЫЕ ФУНКЦИИ

Для того чтобы вычислить квадратный корень с помощью ЭВМ, на языке Паскаль можно обратиться к стандартной функции SQRT. Получив приказ вычислить значение функции SQRT, машина выполняет его по некоторому уже известному ей алгоритму.

Представим себе, что в языке нет стандартных функций. Это означает, что ЭВМ «не знает» соответствующего алгоритма. И следовательно, такой алгоритм мы должны запрограммировать самостоятельно.

Прежде чем этим заняться, заметим, что арифметические операции и вычисление стандартных функций над вещественными числами выполняются приближенно,

т. е. с некоторой погрешностью. Это объясняется тем, что количество разрядов для хранения вещественного числа в памяти ЭВМ ограничено.

Рассмотрим метод приближенного вычисления квадратного корня с некоторой заданной точностью (погрешностью) ε .

Будем говорить, что y — приближенное значение \sqrt{x} с точностью ε , если выполняется условие $|\sqrt{x} - y| \leq \varepsilon$.

Метод заключается в последовательном выполнении ряда шагов, на каждом из которых вычисляется очередное приближение к \sqrt{x} .

Пусть y_1 — приближение \sqrt{x} на первом шаге, y_2 — на втором, y_3 — на третьем, ..., y_i — на некотором i -м шаге.

На первом шаге зададим произвольное начальное приближение $y_1 > 0$, например $y_1 = 1$, и построим последовательность $y_1, y_2, y_3, \dots, y_i, y_{i+1}$ по следующему закону:

$$y_2 = \frac{1}{2} \left(y_1 + \frac{x}{y_1} \right);$$

$$y_3 = \frac{1}{2} \left(y_2 + \frac{x}{y_2} \right);$$

...

$$y_i = \frac{1}{2} \left(y_{i-1} + \frac{x}{y_{i-1}} \right);$$

$$y_{i+1} = \frac{1}{2} \left(y_i + \frac{x}{y_i} \right).$$

Методами математического анализа доказывается, что такая последовательность приближается к \sqrt{x} , и если на некотором шаге два последовательных приближения y_i и y_{i+1} отличаются на величину, не превосходящую ε : $|y_{i+1} - y_i| < \varepsilon$, то y_{i+1} отличается от точного значения \sqrt{x} также не более чем на ε , т. е. $|y_{i+1} - \sqrt{x}| < \varepsilon$. Следовательно, приближение, полученное на $i+1$ -шаге, и будет приближением к \sqrt{x} с заданной точностью.

Рассмотрим этот метод на примере.

Вычислим $\sqrt{4}$ с точностью $\varepsilon = 10^{-3}$. Нам известно точное значение $\sqrt{4} = 2$. Им мы воспользуемся только для проверки полученного решения, а пока о нем забудем.

Пусть $y_1 = 1$.

$$\text{Тогда } y_2 = \frac{1}{2} \left(y_1 + \frac{x}{y_1} \right) = \frac{1}{2} \left(1 + \frac{4}{1} \right) = 2,5;$$

$$|y_2 - y_1| = |2,5 - 1| = 1,5; y_3 = \frac{1}{2} \left(y_2 + \frac{x}{y_2} \right) = \frac{1}{2} \cdot \left(2,5 + \frac{4}{2,5} \right) = 2,05; |y_3 - y_2| = |2,05 - 2,5| = 0,45; y_4 = \frac{1}{2} \times \times \left(y_3 + \frac{x}{y_3} \right) = \frac{1}{2} \left(2,05 + \frac{4}{2,05} \right) \approx 2,000647; |y_4 - y_3| = = 0,049353;$$

Аналогичным образом вычислив y_5 с точностью до 6 знаков после запятой, получим $y_5 = 2,000000$. $|y_5 - y_4| = = 0,000647 < 10^{-3}$ и $|y_5 - \sqrt{4}| = |2,000000 - 2| = 0 < 10^{-3}$.

|| Воспользовавшись этим методом, вычислите || □
|| значение $\sqrt{2}$.

Опишем алгоритм вычисления квадратного корня на ЭВМ.

Приближенное значение на каждом шаге вычисляется с помощью результатов предыдущего шага, причем число шагов заранее неизвестно. Процесс вычисления продолжается до тех пор, пока разность между двумя соседними приближениями не станет меньше ϵ .

Пусть ХН — приближение, вычисляемое на некотором шаге, ХС — приближенное значение, полученное на предыдущем шаге. Тогда, прочитав значение Х, из которого следует извлечь корень, и проверив, является ли оно неотрицательным, нужно задать начальное приближение ХС и точность ϵ . Положим, например, $\epsilon = 10^{-6}$.

Вычислим следующее приближение ХН с помощью оператора $\text{ХН} := (\text{ХС} + \text{Х}/\text{ХС})/2$, а пока разность между двумя соседними приближениями ХС и ХН по абсолютной величине больше ϵ , будем повторять такую последовательность действий:

- 1) ХС положить равным значению, вычисленному на предыдущем шаге: $\text{ХС} := \text{ХН}$;
 - 2) вычислить новое приближение: $\text{ХН} := (\text{ХС} + \text{Х}/\text{ХС})/2$.
- По окончании выдать полученные результаты: ХН и $|\text{ХН} - \text{ХС}|$.

Опишем алгоритм словесно:

```

начало;
читать (X);
если  $X > 0$ ;
    то { $\epsilon := 1.0 \times 10^{-6}$ ;
         $\text{ХС} := 1$ ;

```



```

    ХН:=(ХС + Х/ХС)/2;
    пока |ХС—ХН| > Е повторять:
        {ХС:=ХН;
        ХН:=(ХС + Х/ХС)/2};
    выдать (ХН, |ХН—ХС|);
    иначе выдать ('попытка извлечения квадратного
    корня из отрицательного числа');

```

конец.

|| Постройте блок-схему этого алгоритма. || ○

Программа SQRTX реализует записанный алгоритм на языке Паскаль:

```

PROGRAM SQRTX (INPUT, OUTPUT);
VAR E, X, XC, XH:REAL;
BEGIN
    READ (X);
    IF X > 0
    THEN BEGIN
        E:=1.0E-6;
        XC:=1;
        XH:=(XC + X/XC)/2;
        WHILE ABS (XC—XH) > E DO
            BEGIN
                XC:=XH;
                XH:=(XC + X/XC)/2
            END;
        WRITE (XH, ABS (XH—XC))
    END
    ELSE
        WRITE ('попытка извлечения
        квадратного корня из отрица-
        тельного числа')
    END.

```

Изменим алгоритм следующим образом. На первом шаге зададим приближение $XH:=1$, а на каждом следующем шаге будем вычислять очередное приближение и сравнивать его с предыдущим. Процесс продолжим до тех пор, пока не выполнится условие: $|XC - XH| \leq E$.

Опишем измененный алгоритм с помощью блок-схемы (рис. 21)

Здесь S1—сообщение: попытка извлечения квадратного корня из отрицательного числа.

И словесно:

```

    начало;
    читать (X);

```

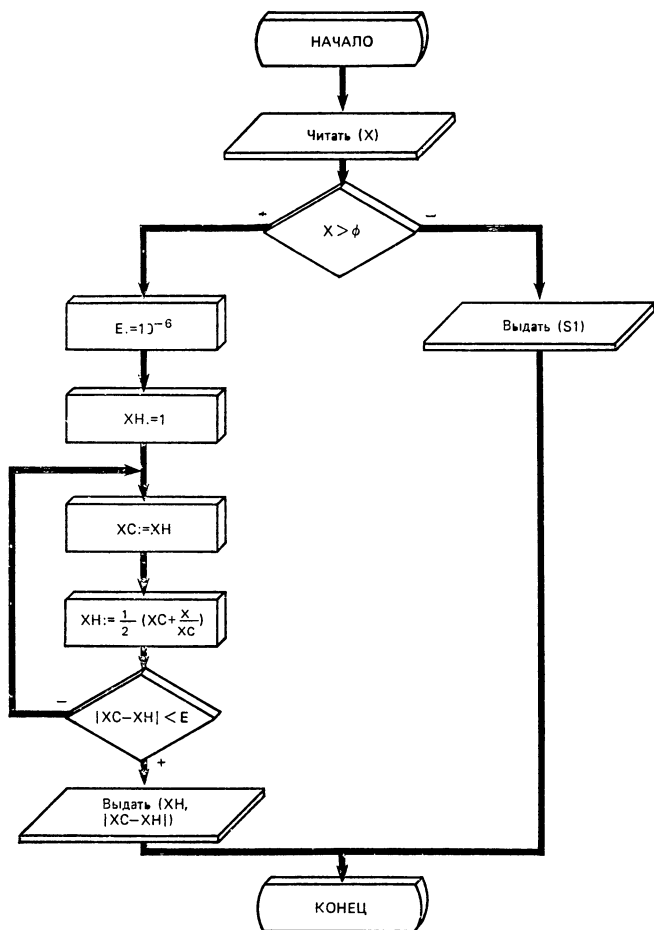


Рис. 21

если $X > 0$
 то {читать (E);
 $XH := 1$;
 выполнять
 $XC := XH$;
 $XH := (XC + X/XC)/2$
 до $|XC - XH| \leq E$;
 выдать (X, $|XC - XH|$);
 иначе выдать ('попытка извлечения квадрат-

ного корня из отрицательного числа');

конец.

При записи алгоритма мы воспользовались новой циклической конструкцией:

Выполнять

S1;

S2;

...

SN

до В, где S1, S2, ..., SN—операторы,

В—логическое условие.

Ее работа может быть описана следующим базовым элементом на языке блок-схем:

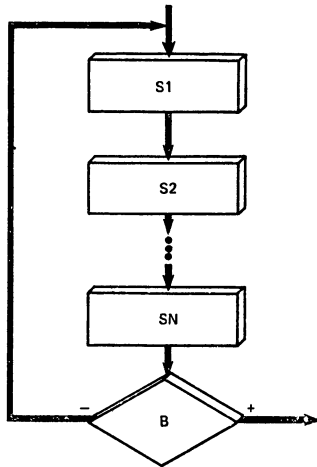


Рис. 22

Выполняется последовательность операторов S1, S2, ..., SN.

Вычисляется логическое условие В. Если оно ложно, то снова повторяются S1, S2, ..., SN. Так происходит до тех пор, пока выражение В не примет истинное значение. В этом случае оператор цикла заканчивает свою работу.

Оператор цикла "выполнять... до" называется оператором цикла с постусловием и на языке Паскаль записывается с помощью ключевых слов:

REPEAT и UNTIL:

REPEAT

S1;

S2;

...

SN

UNTIL B

Пример: REPEAT XC:=XH;

XH:=(XC+X/XC)/2

UNTIL ABS(XC-XH) < E.

REPEAT	—повторять
UNTIL	—до

Запишите программу вычисления квадратного корня, используя оператор REPEAT. || □

Выделим основные отличия операторов цикла с пред- и постусловиями. Их работа описывается соответственно следующими элементами блок-схемы (рис. 23, 24).

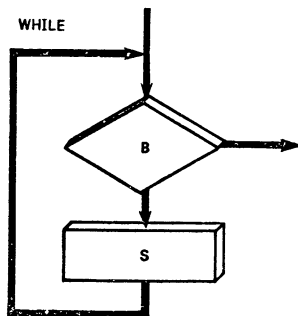


Рис. 23

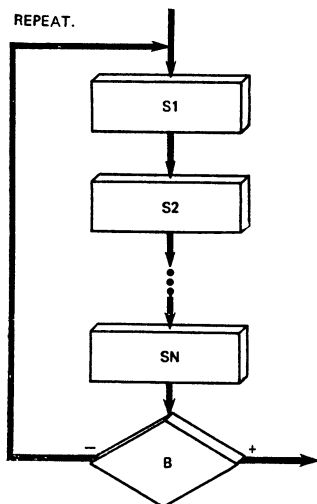


Рис. 24

Дальнейшие результаты сравнения описываются с помощью таблицы, правую часть которой вам предстоит заполнить (табл. 4).

Таблица 4

	Оператор WHILE	Оператор REPEAT
Тело цикла	Телом цикла может быть только один оператор	
Последовательность выполнения тела цикла и проверки условия	Сначала проверяется условие, а затем выполняется тело цикла	
Условие повторения	Тело цикла выполняется, если условие истинно	
Условие выхода	Выход из цикла осуществляется, если условие ложно	
Минимальное число повторений	Тело цикла может не выполниться ни разу, если условие В сразу принимает значение FALSE	

Общим для обоих операторов является возможность «изобрести» вечные циклы.

Пример 1:

```
S:=2;
REPEAT S:=5/S;
UNTIL S < 0.01.
```

Пример 2:

```
S:=1; N:=20; I:=1;
REPEAT S:=S*I;
UNTIL I > N.
```

|| Почему циклы, приведенные в примерах 1 и 2, || ?
|| будут бесконечными?

Глава 14

ДЛЯ ТЕХ, КТО РЕШИЛ РАЗВОДИТЬ КРОЛИКОВ

Тем, кто решил разводить кроликов или хочет познакомиться с программированием рекуррентных процессов, мы предлагаем задачу, сформулированную в 1228 году итальянским математиком Леонардо, известным под именем Фибоначчи: «Некто поместил пару кроликов в некоем месте, огороженном со всех сторон стеной, чтобы узнать, сколько пар кроликов родится при этом в течение года, если природа кроликов такова, что через месяц пара кроликов производит на свет другую пару, а рождают кролики со второго месяца после своего рождения».

Итак, сколько же пар кроликов будет насчитано в конце года?

Начнем решение этой задачи.

Первая пара даст потомство в первом месяце, следовательно, в первом месяце окажется две пары. Из них первая пара рождает и в следующем месяце, так что во втором месяце оказывается уже три пары. В третьем месяце дадут потомство первые две пары, и количество пар станет равным пяти. В четвертом месяце родится еще три пары, и их число достигнет восьми. Из них пять пар произведут другие пять пар, которые, будучи сложенными с 8 парами, дадут 13.

|| Продолжите решение. || □

Вы, очевидно, заметили, что при подсчете количества пар образуется последовательность: 1, 1, 2, 3, 5, 8, 13, ..., каждый член которой, за исключением первого и второго, равен сумме двух предыдущих. Члены последовательности обозначим переменными F_1, F_2, F_3, \dots , $\dots, F_{i-2}, F_{i-1}, F_i, \dots$

Тогда $F_3 = F_2 + F_1$;

$$F_4 = F_3 + F_2;$$

$$\vdots$$
$$F_i = F_{i-1} + F_{i-2};$$

$$F_{i+1} = F_i + F_{i-1}.$$

Последовательности, в которых каждый член представляет собой функцию одного или нескольких предыдущих элементов, называются рекуррентными, а процесс получения элементов рекуррентных последовательностей — рекуррентным процессом.

Чтобы однозначно задать рекуррентный процесс, нужно указать не только закон получения очередных элементов последовательности, но и задать значения нескольких первых членов. Например, для последовательности F_i нужно указать значения F_1 и F_2 .

Положив F_1 и F_2 равными 1 и вычисляя остальные элементы по закону $F_i = F_{i-1} + F_{i-2}$, получим последовательность чисел Фибоначчи:

1, 1, 2, 3, 5, 8, 13, 21, 35, 56, ...

|| Выпишите несколько последующих чисел. || □

Получение рекуррентных последовательностей может быть поручено ЭВМ. Опишем алгоритм вычисления N — первых членов последовательности Фибоначчи.

Пусть FN — число, вычисляемое на некотором шаге, FC — элемент, предшествующий FN , а FCC — предшествующий FC , K — номер получаемого числа. На первом шаге FC совпадает с F_1 , а FN — с F_2 ; их значения равны 1.

Каждый новый член последовательности будем вычислять как сумму двух предыдущих: $FN = FC + FCC$, переопределив предварительно значения FC и FCC :

начало;

читать (N); если $N > 1$,

то {(*зададим два начальных числа*)

$FC := 1$;

$FN := 1$;

выдать (FC, FN);

для $K := 3$ до N повторять

{(* переопределить числа, предшествующие вычисляемому *)

$FCC := FC$;

$FC := FN$;

(* вычислить новое число *)

$FN := FC + FCC$;

выдать (FN)}}

иначе выдать ('ошибка в данных');

конец.

|| Можно ли поменять местами операторы $FCC := FC$ и $FC := FN$? || ?
 || Какая последовательность чисел при этом будет получена? ||

Построим блок-схему (рис. 25)

Здесь C — сообщение: «ошибка в данных».

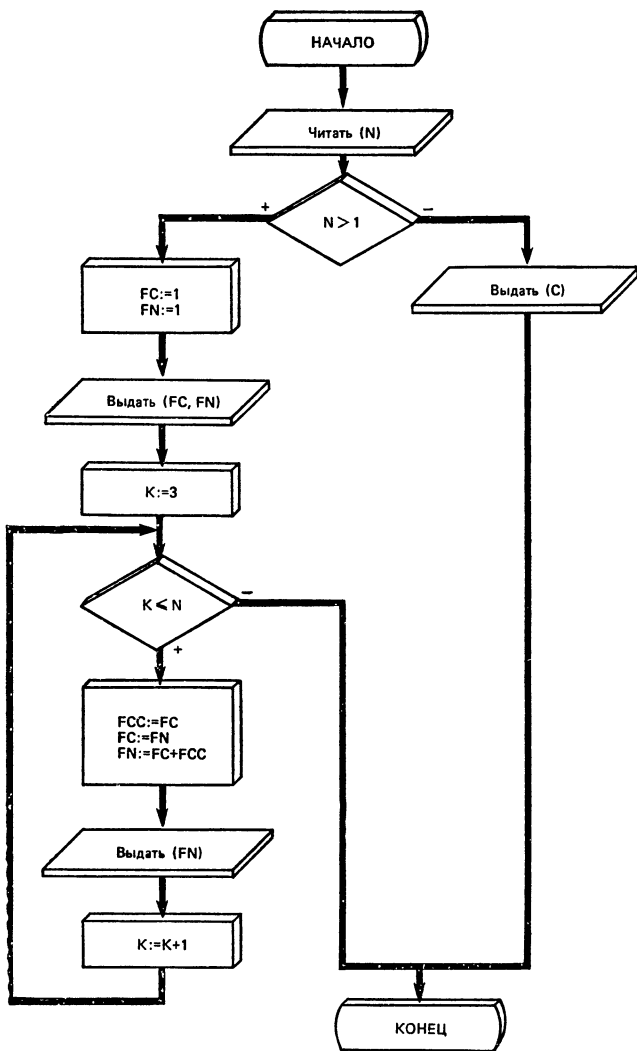


Рис. 25

Запишите средствами языка Паскаль программу получения суммы N первых чисел Фибоначчи. || ○

Составьте алгоритм и программу вычисления первых N членов последовательности: p_1, p_2, p_3, \dots
 $\dots, p_i, p_{i+1}, \dots, p_n$, такой, что $p_1 = 1$; $p_2 = p_1 + 1$;
 $p_{i+1} = p_i + \frac{1}{p_i - 1}$. □

Глава 15

НАЙДИТЕ КРАСНЫЙ ШАР

В непрозрачном мешке находится N цветных шаров. Из мешка вынимаются без возврата по одному шару. Интересно, какой по счету шар окажется красным?

Попробуйте произвести такой опыт, если, конечно, у вас найдутся мешок и шары. Если их нет, то опыт можно провести мысленно. Вот так.

Вынимаем первый шар. Если он не является красным, то, отложив его в сторону и выбрав очередной шар, следует проверить, является ли красным вновь выбранный шар. Как только вынут красный шар, опыт можно завершить, не просматривая, какие шары еще остались. Возможно, что в мешке не окажется красных шаров, но об этом сможем узнать лишь после того, как будут вынуты все N шаров и мешок окажется пустым. Таким образом, ответ на поставленный вопрос можно будет получить, выполнив не более чем N шагов.

Пусть K — номер очередного шара.

Опишем алгоритм поиска:

1. начало;
2. узнать, какое количество шаров находится в мешке (* значение переменной N *);
3. для $K := 1$ до N повторять
 - 3.1. {вынуть шар;
 - 3.2. если шар красный
 - 3.2.1 то {сообщить (K);
перейти к п. 5}}
4. (* все шары просмотрены, красного нет, поэтому к п. 5 мы еще не перешли *);
выдать ('красного шара нет');
5. конец.

Пусть в мешке могут находиться красные, синие, зеленые и белые шары. Каждому цвету поставим в соответствие некоторое число: красному — 1, синему — 2, зеленому — 3, белому — 4. Нашим мешком будем считать

входной файл. Расположим числа во входном файле некоторым образом, например, так:

4, 4, 3, 2, 2, 1, 4, 3, 1.

В этом случае наша цель: найти номер первого числа, равного единице. При этом во входной файл нужно поместить значение переменной N —количество шаров в мешке (или чисел во входном файле), в нашем примере—9.

Тогда узнать количество шаров можно посредством команды: читать (N).

Пусть переменная A означает цвет вынутого шара. Всякий раз A может принимать одно из значений: 1, 2, 3, 4. Под действием «вынуть шар и определить его цвет» будем понимать чтение очередного значения A . Шар является красным, если $A = 1$ (красный цвет мы условились обозначать 1). Действие: перейти к п. 5 зададим так: перейти к 5. Тогда алгоритм может быть описан следующим образом:

начало;

читать (N);

если $N > 0$

то {для $K := 1$ до N повторять

{читать (A);

если $A = 1$

то {выдать (K);

перейти к 5}};

выдать ('красного шара нет'));

иначе выдать (ошибка в данных);

5. конец.

Построим блок-схему (рис. 26).

Действие перейти к п. 5 на блок-схеме изображается стрелкой, указывающей на тот блок, куда должен быть сделан переход. В нашем примере—это помеченный цифрой 5 блок конца.

Для выхода из цикла в случае, если найден красный шар, мы воспользуемся оператором перехода: перейти к P , где P —метка. На языке Паскаль он записывается с помощью ключевого слова $GOTO : GOTO P$, где P —метка, и означает, что дальнейшее выполнение алгоритма нужно продолжить с оператора, помеченного меткой.

$GOTO$ —перейти к.

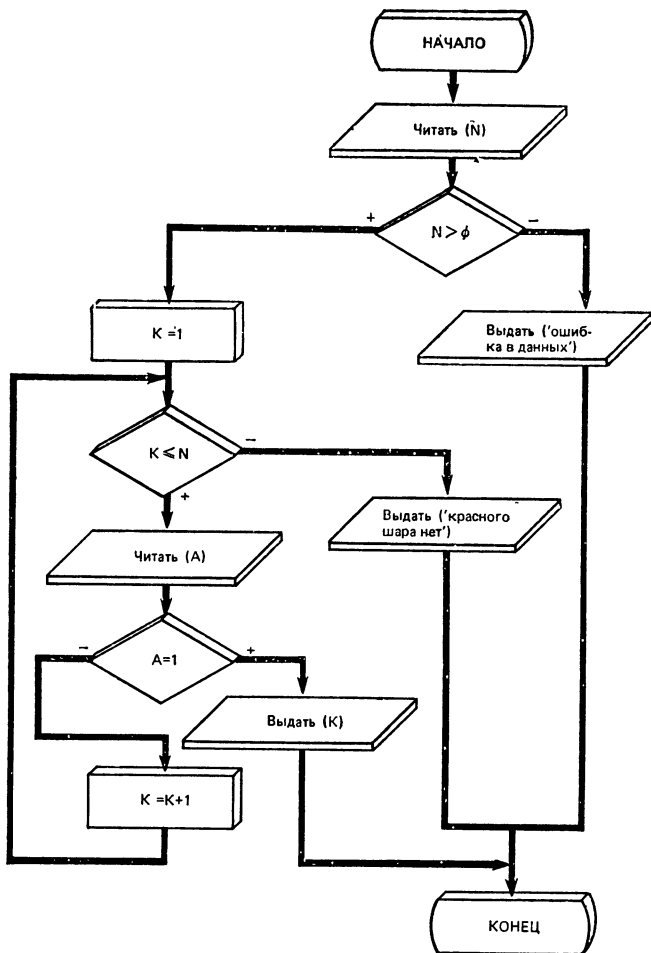


Рис. 26

Что такое метка?

Метка—целое число без знака, состоящее не более чем из четырех цифр. Используется для того, чтобы пометить операторы.

В нашем примере—это 5, а оператор перехода должен иметь вид: GOTO 5.

Вот некоторые примеры операторов перехода: GOTO 1425; GOTO 35.

Пример 1. Операторы перехода GOTO 12039 и GOTO +49 неверны: в первом из них метка состоит более чем из четырех цифр, во втором она является числом со знаком.

|| Какие ошибки содержат следующие операторы: || ?
	а) GOTO A11;	
	б) GOTO—1;	
	в) GOTO 3,5;	

О том, что в программе используются метки (операторы перехода), ЭВМ нужно сообщить. До сих пор в описательную часть программы мы включали лишь раздел описания переменных. При использовании меток описательная часть программы расширяется следующим образом:

[<раздел меток>
[<раздел переменных>].

Раздел описания меток начинается с ключевого слова LABEL (метка), за которым следует список всех меток, используемых в программе, например LABEL 7;

LABEL—метка .

Метки в списке разделяются запятыми:

LABEL 12,3504,99,1;

В рассматриваемом примере будет использоваться метка 5, следовательно, описательная часть программы должна содержать раздел меток: LABEL 5;

Оператор, на который осуществляется переход с помощью GOTO, должен быть помечен соответствующей меткой

<метка> : <оператор>

Метка отделяется от оператора двоеточием.

Пример 2. Меткой 249 помечен оператор присваивания:
249: X:=A+B.

Пример перехода на условный оператор:

GOTO 1

i: IF A > B
THEN X:=A
ELSE X:=B

Пример 3. Пустой оператор также может быть помечен:

```
GOTO 2;
```

```
  . . .  
2: ;  
  . . .
```

Запишем алгоритм решения задачи о шарах на языке Паскаль.

```
PROGRAM КРАСН (INPUT, OUTPUT);  
  (*РАЗДЕЛ МЕТОК*)  
  LABEL 5;  
  (*РАЗДЕЛ ПЕРЕМЕННЫХ*)  
  VAR A,K,N:INTEGER;  
  (*РАЗДЕЛ ОПЕРАТОРОВ*)  
  BEGIN  
    READ(N);  
    IF N > 0  
      THEN BEGIN  
        FOR K:=1 TO N DO  
          BEGIN  
            READ(A);  
            IF A = 1  
              THEN BEGIN  
                WRITE(K);  
                GOTO 5  
              END  
            END;  
          WRITE ('└КРАСНОГО  
                ШАРА НЕТ.')        END  
      ELSE WRITE ('└ОШИБКА  
                В ДАННЫХ.');
```

5: END.

Может ли в программе содержаться два оператора, помеченных одинаковой меткой? Конечно, нет, так как в этом случае будет непонятно, откуда следует продолжить выполнение программы. Как говорят, неясно, куда передать управление.

Пример 4. Рассмотрим, как будет выполняться такая последовательность операторов:

```
A:=1; B:=1;  
GOTO 1212;
```

```
  . . .  
FOR K:=1 TO N DO  
  BEGIN
```

```
1212: P:=A + K*B*(K—1);  
      WRITE(P)  
      END
```

. . .

Меткой 1212 помечен оператор присваивания, входящий в тело цикла. Тело цикла—оператор, выполняемый с каждым значением параметра цикла. Но поскольку мы пытаемся войти внутрь оператора цикла, минуя его заголовок, то неизвестно, при каком значении параметра K должно вычисляться значение переменной P . Следует запомнить важное правило использования операторов перехода.

Передача управления внутрь оператора цикла извне его недопустима.

Вообще оператор перехода следует применять осторожно, в исключительных ситуациях. Например, для досрочного выхода из цикла (как в нашей задаче) или для досрочного завершения программы, когда на ее вход поступают неправильные исходные данные. Применение оператора перехода в каждом конкретном случае должно быть обоснованным.

Глава 16

НУЖНО ЛИ ИЗОБРЕТАТЬ ВЕЛОСИПЕД?

Изобретение велосипеда позволило человеку, затрачивая такую же энергию, как и при ходьбе, существенно увеличить свою скорость. В «довелосипедный» период существовал лишь один способ обзавестись велосипедом—создать его. Но после того, как это было сделано, появились другие альтернативы: приобрести велосипед в магазине или взять его напрокат. Этими возможностями и пользуется большинство любителей велосипедной езды.

Программист, причем не только начинающий, зачастую сталкивается с такой ситуацией. Сегодня он сочинил программу, которая решает, скажем, некоторую задачу A . Завтра, решая задачу B , он обнаруживает, что задача A является частью этой задачи и, стало быть, ему снова, кроме всего прочего, потребуется писать программу решения задачи A . Хорошо ли это,

снова изобретать велосипед? Мы рассмотрим сейчас способы создания таких программ, которые можно в дальнейшем использовать при решении разных задач. Не будем изобретать велосипед, а будем брать его напрокат!

Пусть, к примеру, требуется найти действительные корни уравнений вида $ax^2 + bx + c = 0$. Уравнения такого вида отличаются друг от друга коэффициентами a , b , c и соответственно количеством и значениями корней.

Обозначим через P признак количества действительных корней уравнения и будем считать, что P принимает значение 0, если уравнение не имеет действительных решений; $P=1$, если уравнение имеет один действительный корень; $P=2$ в случае двух действительных корней и, наконец, $P=3$, если решений бесконечно много.

При решении любого уравнения рассматриваемого вида используется один из двух алгоритмов: алгоритм решения квадратного уравнения, если $a \neq 0$, и алгоритм решения линейного уравнения, если $a=0$ (уравнение в этом случае имеет вид $bx + c = 0$).

Допустим теперь, что N «троек» коэффициентов (a , b , c) находится во входном файле. Опишем алгоритм решения нашей задачи. Словесно в самом общем виде он будет выглядеть так:

начало;

читать (N);

для $K:=1$ до N повторять

 {читать (a , b , c);

 если $a=0$

 то решить линейное уравнение с коэффициентами b и c ;

 иначе решить квадратное уравнение с коэффициентами a , b и c ;

 выдать решение};

конец.

|| Постройте блок-схему рассматриваемого алгоритма. || ○

Рассмотрим решение квадратного уравнения с коэффициентами e , f , g : $ex^2 + fx + g = 0$.

Пусть D — дискриминант этого уравнения, вычисляемый согласно формуле $D = f^2 - 4eg$.

Число и вид действительных корней уравнения определяются значением дискриминанта.

Если $D < 0$, то уравнение не имеет действительных корней: $P=0$. При $D > 0$ существуют два различных действительных корня: X_1 и X_2 , значения которых определяются следующими формулами:

$$X_1 = \frac{-f + \sqrt{D}}{2e};$$

$$X_2 = \frac{-f - \sqrt{D}}{2e}, \text{ при этом } P=2.$$

В случае, когда $D=0$, эти корни совпадают, и можно сказать, что уравнение имеет решение $X_1 = -\frac{f}{2e}$, а $P=1$.

Решите уравнения:
 а) $2x^2 - 4x + 3 = 0$; б) $25x^2 - 30x + 9 = 0$;
 в) $3x^2 + 2x - 2 = 0$. □

Итак, конкретизируем действие: решить квадратное уравнение $ex^2 + fx + g = 0$.

Описание алгоритма:

```

начало;
D:=f*f-4*e*g;
если D < 0
  то P:=0 (* нет действительных корней *)
  иначе { (* вычислим знаменатель *)
    e:=2*e;
    если D=0
      то { (* один корень *)
        P:=1;
        X1:=-f/e;
      }
      иначе { (* два корня *)
        P:=2;
        D:=SQRT(D);
        X1:=(-f+D)/e;
        X2:=(-f-D)/e;
      }
  }

```

конец.

Построим блок-схему алгоритма решения квадратного уравнения: (рис. 27).

Линейное уравнение $fx + g = 0$ имеет единственное решение, если $f \neq 0$: $X = -g/f$.

В случае, когда $f=0$, уравнение принимает вид $g=0$. Тогда при g , равном 0, оно имеет бесконечное множество решений ($0 \cdot x + 0 = 0$ справедливо для любого значения x) и не имеет решения при $g \neq 0$. Перемен-

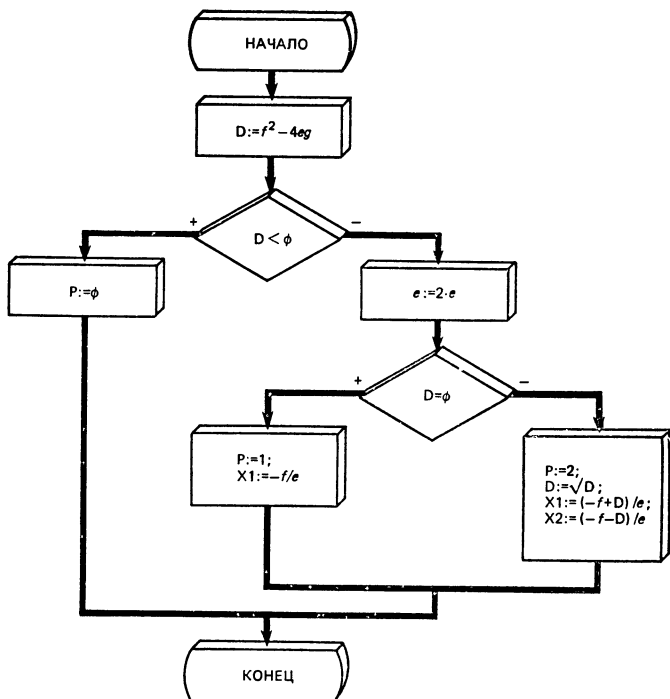


Рис. 27

ная P будет соответственно принимать значения 1, 3 или 0.

|| Опишите словесно и с помощью блок-схемы || ☐
|| алгоритм решения линейного уравнения.

Если вы выполнили предыдущее упражнение, то теперь осталось «научить» машину решать линейные и квадратные уравнения по описанным алгоритмам.

При этом хотелось бы, чтобы ЭВМ "умела" решать такие уравнения независимо от того, какие имена мы используем для обозначения коэффициентов и получаемых решений. Опишем на языке Паскаль алгоритм решения квадратного уравнения так, чтобы он был независим от использующей его программы и мог бы быть использован в качестве составной части разных программ. Для этого будем использовать аппарат процедур.

Процедура — поименованное сложное действие, которое представляет собой совокупность операторов, вычисляющих некоторое число результатов в зависимости от некоторого числа аргументов. По сути, это программа, которую удобно использовать в составе других программ.

Процедура решения квадратного уравнения, назовем ее KWADR, вычисляет три результата: признак количества решений Р и два решения X1, X2 в зависимости от трех аргументов: коэффициентов Е, F, G.

Аргументы и результаты вместе называются параметрами процедуры. Процедура задается своим описанием. Описание процедуры состоит из заголовка и тела процедуры.

В заголовке указываются имя процедуры, которое будет использоваться другими программами для обращения к ней, список параметров, называемых формальными, и их тип. Например: процедура KWADR

(переменные Е, F, G: вещественные;

переменные X1, X2: вещественные;

переменная Р: целая);

Порядок перечисления формальных параметров в заголовке процедуры несуществен, но лучше, если сначала перечисляются входные параметры (аргументы), а затем выходные (результаты).

Тело процедуры описывает процесс получения выходных параметров из исходных данных.

Итак, поставив в соответствие слову "процедура" ключевое слово PROCEDURE, а слову "переменные"—

VAR запишем процедуру

PROCEDURE — процедура

KWADR:

PROCEDURE KWADR (VAR E, F, G: REAL;
VAR X1, X2: REAL;
VAR P: INTEGER);

(*тело процедуры*)

VAR D: REAL; (*дискриминант*)

BEGIN

D:=F*F—4*E*G;

IF D < 0

```

THEN P:=0
ELSE BEGIN
  E:=E*2;
  IF D=0
    THEN BEGIN P:=1;
              X1:=-F/E END
    ELSE BEGIN
              P:=2;
              D:=SQRT (D);
              X1:=(-F + D)/E;
              X2:=(-F - D)/E
            END
  END
END;

```

Как видим, процедура, помимо параметров, содержит промежуточную переменную D — дискриминант, которая необходима для вычислений только в этой процедуре и нигде больше за ее пределами. Эта переменная является локальной для процедуры KWADR.

Если процедуру представить как некоторый "домик", то параметры — "обитатели", которые могут входить в "дом" и выходить из него, а локальные переменные — "обитатели", которые перемещаются только внутри этого "дома".

Все локальные переменные процедуры, а также локальные метки подлежат описанию в этой процедуре.

|| Составьте процедуру LINEAR для решения || □
|| линейного уравнения. ||

Описания процедур, используемых в некоторой программе, помещаются в разделе процедур этой программы, который в описательной части следует сразу за разделом переменных.

В этом случае описательная часть имеет такую структуру:

```

[<раздел меток>]
[<раздел переменных>]
[<раздел процедур>]

```

Описание процедуры не является приказом выполнить действия, а представляет собой лишь описание этих действий. Так, например, процедура KWADR описывает алгоритм решения квадратного уравнения, но не является приказом "решить уравнение".

В математике принято ссылаться на один раз опи-

санные объекты. Например, определив функцию $f(x, y) = \frac{x}{y}$, мы можем обращаться к ней, задавая различные аргументы. Так, значение $f(1, 2)$ равно 0.5, а значением $f(a-b, a+b)$ при $a=3$ и $b=2$ является 0.2.

Аналогичным образом дело обстоит с процедурами.

Чтобы выполнить действия, описанные в процедуре, нужно обратиться к ней. Обращение к процедуре называемое иначе вызовом процедуры, осуществляется посредством оператора процедуры.

Программу, которая обращается к процедуре для выполнения некоторого действия, будем называть вызывающей программой.

В вызывающей программе может встречаться несколько операторов обращения к процедуре (иначе операторов процедуры), например KWADR(A, B, C, X1, X2, P); KWADR(A1, B1, C1, Y1, Y2, P). В первом операторе X1, X2—решения уравнения с коэффициентами A, B, C; P—признак количества решений, во втором Y1, Y2—решения уравнения с коэффициентами A1, B1, C1; P—признак количества решений.

Чтобы вызвать процедуру, в вызывающей программе достаточно написать имя этой процедуры вместе с заключенным в скобки списком фактических параметров, каждому из которых предстоит заменить соответствующий формальный параметр.

Фактические параметры—это переменные, в отдельных случаях и выражения, к которым хотим применить процедуру. Например, при обращении к процедуре KWADR: KWADR(A, B, C, X1, X2, P1) формальным параметрам E, F, G, X1, X2, P соответствуют фактические: A, B, C, X1, X2, P1, которые и подставляются на место формальных. Количество формальных и фактических параметров должно быть одинаковым. Соответствие между ними устанавливается порядком написания в списке: первый фактический параметр соответствует первому формальному, второй—второму и т. д.

|| Запишите оператор вызова описанной вами || □
|| процедуры LINEAR для решения уравнения ||
|| $bx + c = 0$. ||

Встретив в вызывающей программе оператор процедуры, ЭВМ выполнит его по следующему правилу.

Разыскивается описание процедуры с тем же именем, что указано в операторе. Затем каждый формальный параметр связывается с соответствующим ему фактическим (происходит передача параметров). Далее выполняются операторы, составляющие тело процедуры так, если бы они были написаны на том месте, где находится оператор процедуры.

Из этого правила следует, что после выполнения тела процедуры будет выполняться оператор, следующий за оператором процедуры.

Вернемся к программированию алгоритма рассматриваемой задачи. Опишем структуру программы, при этом операторы, реализующие вывод результатов, предлагаем записать читателю. Мы же условно обозначим их словом WIWOD, а на одном из возможных вариантов вывода полученных результатов остановимся позже.

```
PROGRAM LINKWAD(INPUT, OUTPUT);
  (*РАЗДЕЛ ПЕРЕМЕННЫХ*)
  VAR A,B,C,X,X1,X2:REAL;
      K,N,P:INTEGER;
  (*РАЗДЕЛ ПРОЦЕДУР*)
  PROCEDURE KWADR(VAR E,F,G:REAL; VAR
                  X1,X2:REAL;
                  VAR P:INTEGER);
  ...
  (*ПРОЦЕДУРА РЕШЕНИЯ КВАДР.
  УРАВНЕНИЯ*)
  ...
  END;
  PROCEDURE LINEAR(...);
  (*ПРОЦЕДУРА РЕШЕНИЯ ЛИНЕЙНОГО
  УРАВНЕНИЯ*)
  ...
  END;
  (*РАЗДЕЛ ОПЕРАТОРОВ*)
  BEGIN
  READ(N);
  IF N < 0 THEN WRITE ('ошибка в данных')
  ELSE BEGIN FOR K:=1 TO N DO
```

```

BEGIN
  READ(A,B,C);
  IF A=0
    THEN LINEAR (...)
           (*ВЫЗОВ ПРОЦЕДУРЫ
            LINEAR*)
    ELSE KWADR(A,B,C,X1,X2,P);
           (*ВЫВОД РЕЗУЛЬТАТОВ*)
  WRITELN('УР-НИЕ с КОЭФ.', A,B,C);
  WIWOD
END
END

```

END.

В языке Паскаль существует несколько способов передачи параметров. При описании процедуры KWADR мы использовали параметры-переменные.

На тот факт, что параметр передается таким способом, указывает ключевое слово VAR перед соответствующим формальным параметром. При этом фактический и формальный параметры должны быть переменными одного типа.

В чем заключается передача параметров-переменных?

Вы помните, что при объявлении переменной в программе ей отводится ячейка памяти, в которую впоследствии помещается значение соответствующей переменной. Каждая ячейка имеет адрес. Если параметр процедуры является параметром-переменной, то при каждом вызове передается не значение фактического параметра, а адрес ячейки, в которой этот параметр хранится. Таким образом, процедура получает доступ к самому фактическому параметру, и любые действия над формальным параметром непосредственно распространяются на соответствующий ему фактический. Всякое изменение формального параметра в теле процедуры является, по существу, изменением фактического.

Из сказанного следует, что программа LINKWAD содержит ошибку, которую не заметит ЭВМ. В процедуре в случае, когда дискриминант неотрицателен, переменной E мы присваиваем значение знаменателя E*2. Так как E является параметром-переменной, то в результате выполнения этого оператора значение фактического параметра A тоже будет удвоено. Следовательно, оператор WRITELN('УР-НИЕ С КОЭФ.', A,B,C)

выдаст удвоенное значение коэффициента А и коэффициенты В, С.

Приведем несколько примеров неправильного использования параметров-переменных.

Пример 1. Неправильным будет обращение к процедуре KWADR посредством оператора KWADR(5,5, —3, X1, X2, P).

В качестве параметров-переменных можно использовать только переменные. Выражения и их частный случай, константы, недопустимы.

|| Почему? || ?

Пример 2. Пусть с помощью процедуры KWADR решаются уравнения $ax^2 + ax + c = 0$. Оператор процедуры KWADR (A, A, C, X1, X2, P) содержит ошибку.

Фактические параметры, соответствующие формальным параметрам-переменным, должны быть различными.

Часто нужно обрабатывать параметры процедуры как переменные, изменяя их значения. Но при этом желательно, чтобы значения фактических параметров не изменялись. В этом случае следует использовать другой вид параметров: параметры-значения. Передача параметров-значений заключается в том, что при вызове процедуры формальным параметрам выделяются ячейки памяти, в которые записываются начальные значения соответствующих фактических параметров. Затем формальные параметры обрабатываются как локальные переменные. Процедура может изменять значение этих переменных, но на значение соответствующего фактического параметра это не влияет.

Вообще входные параметры, если они не должны изменяться в результате выполнения процедуры, лучше задавать как параметры-значения. Этот способ более безопасен, так как значения фактических параметров не будут случайным образом испорчены. Можно ли выходные параметры передавать как параметры значения? Конечно, нет, поскольку предполагается, что они вычисляются в результате выполнения процедуры.

При описании формального параметра-значения в заголовке процедуры отсутствует ключевое слово VAR.

Опишем коэффициенты E, F, G в процедуре KWADR как параметры-значения. Изменения коснутся лишь заголовка процедуры:

```
PROCEDURE KWADR (E, F, G:REAL; VAR X1,  
                  X2:REAL; VAR P:  
                  INTEGER);
```

Процедуры KWADR и LINEAR, предварительно тщательно проверив, можно включить в библиотеки процедур, чтобы их в дальнейшем можно было использовать в программах.

Чтобы закончить работу с программой LINKWAD, опишем один из возможных алгоритмов вывода полученных результатов.

После того как определены выходные параметры и сообщены коэффициенты решаемого уравнения, должно быть выдано одно из следующих сообщений:

- а) не имеет решения, если $P = 0$;
- б) имеет одно решение, X_1 при $P = 1$;
- в) имеет два действительных решения, X_1, X_2 в случае, когда $P = 2$;
- г) имеет бесконечное множество решений, если P принимает значение 3.

Алгоритм может быть запрограммирован с помощью вложенных условных операторов:

если $P = 0$

 то выдать ('не имеет решения')

 иначе если $P = 1$

 то выдать ('имеет одно решение', X_1)

 иначе если $P = 2$

 то выдать ('имеет 2 корня', X_1, X_2)

 иначе если $P = 3$

 то выдать ('имеет
 бесконечное множество
 решений').

|| Опишите этот алгоритм с помощью блок-схемы || ○
и на языке Паскаль.

Переменная P может принимать одно из возможных значений: 0, 1, 2, 3. Каждому значению соответствует свой оператор вывода. Вывод полученных результатов может быть сделан с помощью конструкции "выбрать", называемой оператором варианта.

Выбрать P из

- 0: выдать (' не имеет решения ');
- 1: выдать ('имеет одно решение', X1);
- 2: выдать ('имеет два действ. корня', X1, X2);
- 3: выдать ('имеет бескон. мн-во решений');

конец.

В операторе перечислены все возможные варианты, из которых в соответствии со значением Р нужно выбрать только один.

При записи оператора варианта на языке Паскаль используются ключевые слова: CASE, OF, END.

CASE —случай, вариант

Оператор имеет вид:

CASE P OF

M1:S1;

M2:S2;

...

MN:SN

END.

Здесь Р—выражение, S1, S2, ..., SN—операторы, M1, M2, ..., MN—метки варианта.

Выполнение оператора определяется следующим правилом.

Вычисляется значение выражения Р, называемого селектором. Выполняется оператор, метка варианта которого равна текущему значению селектора, после чего оператор варианта заканчивает свою работу. Но если селектор примет значение, которого нет среди меток варианта, то действие оператора не определено.

Что такое метка варианта? Это—константа того же типа, что и селектор (выражение Р). Метка варианта и метка программы, используемая в операторах GOTO,—разные понятия. Метки варианта не нужно описывать в разделе меток и в каком бы то ни было другом разделе описаний. На них нельзя ссылаться с помощью операторов перехода. Метка варианта может быть не только целым числом, но и логическим значением.

Пример 3. Оператор CASE A > B
TRUE:M:=A;

```
FALSE:M:=B
END
```

эквивалентен условному оператору:

```
IF A>B
  THEN M:=A
  ELSE M:=B, выбирающему максимум из двух чисел.
```

Пример 4. Неполный условный оператор

```
IF (X>0) AND (Y>0)
  THEN BEGIN Z:=SQRT(X)+SQRT(Y);
            Z:=(Z-1)/(Z+1)
          END
```

может быть записан с помощью оператора варианта следующим образом:

```
CASE (X>0) AND (Y>0) OF
  TRUE: BEGIN Z:=SQRT(X)+SQRT(Y);
          Z:=(Z-1)/(Z+1)
        END;
  FALSE:
```

END.

Очевидно, использование оператора CASE вместо условного не всегда является целесообразным, но в случае нескольких взаимоисключающих действий оператор варианта может быть полезен.

|| Опишите вывод результатов программы || □
|| LINKWAD с помощью оператора CASE. ||

|| Запишите конечный вид программы LINKWAD. || □

В наименьшей степени «велосипедами, берущимися напрокат», являются стандартные функции. Например, чтобы вычислить значение функции

$$f(x, y) = \frac{\sqrt{x^2+1} + \sqrt{x^2+y^2}}{\sqrt{\frac{x^2+y^2}{2}}},$$

мы должны трижды обратиться к стандартной функции SQRT. Представим себе, что мы в очередной раз оказались в ситуации, когда ЭВМ не умеет извлекать квадратные корни. Нам предстоит обучить ее этому, для чего воспользуемся ранее описанным алгоритмом приближенного вычисления корней.

|| Опишите процедуру KWKOR для вычисления || □
	квадратного корня с точностью 10^{-6} степени.	
	Очевидно, она будет иметь два параметра:	
	X — входной,	

|| описываемый как значение параметра
(почему?)
и выходной: Y — значение квадратного корня. ||

Пусть $Y1 = \sqrt{x^2 + 1}$, $Y2 = \sqrt{x^2 + y^2}$, $Y3 = \sqrt{\frac{x^2 + y^2}{2}}$.

Тогда $F = \frac{Y1 + Y2}{Y3}$.

Значение F будет вычислено в результате выполнения такой последовательности операторов:

```

READ(X, Y);
X:=X*X;
KWKOR(X + 1, Y1);
X:=X + Y*Y;
KWKOR(X, Y2);
KWKOR(X/2, Y3);
F:=(Y1 + Y2)/Y3.

```

Чтобы вычислить квадратный корень с помощью процедуры KWKOR, использован оператор процедуры. Мы же привыкли обращаться к вычислению квадратного корня из арифметического выражения. Например, хорошо бы вычислить значение F таким образом:

$X := X * X; Y := Y * Y;$

$F = (\text{SQRTM}(X + 1) + \text{SQRTM}(X + Y)) / \text{SQRTM}((X + Y) / 2).$

В языке Паскаль имеется возможность описать процедуру, к которой можно обращаться как к функции из арифметического или логического выражения. Это — процедуры-функции. Опишем программу вычисления функции $f(x, y)$, воспользовавшись этим средством:

```

PROGRAM FUNC (INPUT, OUTPUT);
(*РАЗДЕЛ ПЕРЕМЕННЫХ*)
VAR X, Y, F:REAL;
(*РАЗДЕЛ ПРОЦЕДУР*)
FUNCTION SQRTM(X:REAL):REAL;
(*ПРОЦЕДУРА — ФУНКЦИЯ ВЫЧИСЛЕНИЯ
КВАДРАТНОГО КОРНЯ*)
VAR XC, XH:REAL;
BEGIN IF X > 0
      THEN BEGIN XH:=1;
                  REPEAT XC:=XH;
                           XH:=
                               (XC + X/XC)/2
                  UNTIL ABS(XC - XH)
                           < 1E-6;

```

```

                                SQRTM: = XH
                                END
                                ELSE WRITE ('АРГУМЕНТ SQRTM
                                ОТРИЦАТЕЛЬНЫЙ', X)

                                END;
                                (*РАЗДЕЛ ОПЕРАТОРОВ*)
                                BEGIN
                                READ (X, Y);
                                Y:=Y*Y; X:=X*X;
                                F:=(SQRTM(X + 1) + SQRTM(X + Y))/
                                SQRTM((X + Y)/2);
                                WRITE(F)
                                END.

```

Выделим основные особенности при описании процедуры функции:

1. В заголовке процедуры-функции указывается слово FUNCTION.
2. Функция имеет единственный выходной параметр (результат), который в списке параметров не указывается.
3. После списка формальных параметров задается тип результата.
4. В теле процедуры-функции должен быть по крайней мере один оператор присваивания, в левой части которого находится имя функции. Такое присваивание выдает результат функции.
5. Функция вызывается по имени из некоторого выражения.

|| Запишите процедуру-функцию для вычисления $F = 1 \cdot 2 \cdot 3 \cdot \dots \cdot N$ и процедуру-функцию вычисления N -го числа Фибоначчи. || ○

Итак, теперь вместо стандартной функции SQRT вы всякий раз можете использовать функцию SQRTM. Но стоит ли? Ведь функция SQRT уже проверена многими программами, с помощью специальной программы-транслятора переведена на внутренний язык машины. Для процедуры-функции SQRTM это только предстоит сделать. В каждой новой процедуре возможны ошибки, которые нужно найти и устранить.

Поэтому, программируя всякий раз свою задачу, прежде чем писать вспомогательные процедуры, убедитесь, нет ли уже готовых средств. Возможно, что они

есть в библиотеке процедур или у ваших товарищей.

Нужно ли тратить время на велосипед, если вы изобретаете автомобиль?

Глава 17

ЕСЛИ БЫ ВЫ БЫЛИ ДИРЕКТОРОМ

Представьте себя в роли директора овощной базы в некотором городе N... Перед вами может возникнуть задача, подобная следующей.

Чтобы обеспечить население города картофелем, его следует закупить в количестве 3000 т в четырех колхозах. Количество картофеля, которое может продать каждый колхоз, и цена перевозки 1 т приведены в таблице:

Таблица 5

Колхоз	1	2	3	4
Количество, продаваемое колхозом, т	1100	900	800	700
Цена перевозки 1 т	3	2,5	2	2,7

Нужно разработать план закупки картофеля и найти затраты на его перевозку.

К решению этой задачи можно было бы подойти различными способами. Рассмотрим одно из возможных решений, не претендующее на то, чтобы быть наилучшим. Согласно этому решению предлагается следующий план действий:

в первом колхозе закупить 1100 т, во втором — 900 т, в третьем — 800 т и, наконец, в четвертом — оставшиеся 200 т. Подсчитаем стоимость перевозки картофеля из первого колхоза — $1100 \times 3 = 3300$ руб., из второго — $900 \times 2,5 = 2250$ руб., из третьего — $800 \times 2 = 1600$ руб. и четвертого — $200 \times 2,7 = 540$ руб. Таким образом, на доставку всего картофеля в город будет затрачено 7690 руб.

Но, будучи рачительным директором, вы, очевидно, не станете действовать такими методами, а попытаетесь найти решение, которое давало бы наименьшую стоимость перевозок.

Итак, ваш план?

|| Сравните затраты на перевозки, получаемые согласно вашему плану, со стоимостью перевозок ранее рассмотренного решения. || □

Опишем последовательность действий, которые мог бы выполнять экономный директор. Во-первых, следует подсчитать количество картофеля, которое могут продать колхозы: $1100 + 900 + 800 + 700$. Если это количество меньше требуемого овощной базе, то нужно констатировать, что город не может быть обеспечен картофелем, а план закупки при этом определяется однозначно: закупить зесь картофель, который могут продать колхозы.

Если колхозы могут продать ровно столько картофеля, сколько нужно, то план закупок такой же, как и в первом случае.

И наконец, в случае, когда предложение превышает спрос: колхозы могут продать картофеля больше, чем нужно овощной базе, у вас появляется возможность выбора.

Последовательность закупки определяется ценой перевозки 1 т.

Сначала целесообразно закупить картофель в колхозе, стоимость перевозки из которого наименьшая. Это — колхоз 3. В колхозе 3 можно закупить 800 т ($800 < 3000$). Далее, среди оставшихся колхозов вновь выбирается тот, где стоимость перевозки самая маленькая. Теперь это колхоз 2. В колхозе 2 закупается 900 т: $800 + 900 = 1700 < 3000$. Теперь картофель можно покупать либо в колхозе 1, либо — в 4. Выбираем колхоз 4, так как цена перевозки там меньше: $1700 + 700 = 2400 < 3000$ т.

Наконец, в колхозе 1 можно купить 1100 т: $2400 + 1100 = 3500$ т. Такое количество овощной базе не требуется. В колхозе 1 достаточно купить $3000 - 2400 = 600$ т.

Подсчитаем затраты на перевозку картофеля:
 $800 \cdot 2 + 900 \cdot 2,5 + 700 \cdot 2,7 + 600 \cdot 3 = 7540$ руб.

Теперь представим себе, что овощная база является достаточно крупной и картофель можно закупать не у

четырёх колхозов, а у сорока. Тогда, если вы действительно хотите искать оптимальное решение, вам может очень пригодиться ЭВМ, а также некоторые сведения о решении подобных задач.

Итак, вам известно количество картофеля, требуемого овощной базой, — S ($S > 0$). Пусть k — номер колхоза, $k = 1, 2, \dots, 40$. Через b_k обозначим количество картофеля, которое можно купить в k -колхозе, через c_k — цену перевозки 1 т картофеля из k -колхоза.

Таким образом, помимо переменной S , нам должны быть известны две последовательности, или два массива чисел:

b_1, b_2, \dots, b_{40} и c_1, c_2, \dots, c_{40} .

Эти сведения могут быть представлены в таблице:

Т а б л и ц а 6

к—№ колхоза	1	2	...	к	...	40
b_k — количество картофеля	b_1	b_2	...	b_k	...	b_{40}
c_k — цена перевозки 1 т	c_1	c_2	...	c_k	...	c_{40}

В результате решения задачи нужно определить, какое количество картофеля следует закупать в каждом колхозе, т. е. получить массив a_1, a_2, \dots, a_{40} и найти затраты на перевозку закупаемого картофеля: Z .

Переменные b_1, b_2, \dots, b_{40} , имеющие один и тот же тип, являются элементами массива b ; c_1, c_2, \dots, c_{40} — элементы массива c ; a_1, a_2, \dots, a_{40} — элементы массива a (они удовлетворяют такому соотношению): $0 \leq a_k \leq b_k$.
 || Какой смысл имеет это соотношение? ||?

Массив — это упорядоченная по номерам совокупность значений, объединенных общим типом и именем.

Массив имеет фиксированное имя, фиксированный тип и количество значений. Например, имя массива b_1, b_2, \dots, b_{40} — b тип — вещественный, количество значений — 40.

Каждый элемент массива определяется именем, совпадающим с именем массива, а также индексом.

Индекс—это величина, характеризующая положение элемента относительно начала массива.

Например, b_{12} —двенадцатый элемент массива b , c_{39} —тридцать девятый элемент массива c . Если массив состоит только из 40 элементов: b_1, b_2, \dots, b_{40} , то нельзя использовать переменную b_{41} , так как такой переменной нет в массиве.

Каждому массиву, используемому в программе, выделяется место в памяти. Но в отличие от простых переменных массиву отводится не одна ячейка, а последовательность расположенных друг за другом ячеек, в каждую из которых записывается значение соответствующего элемента, например массив a , состоящий из сорока элементов, в памяти ЭВМ может реализоваться так: $a: a_1 | a_2 | \dots | a_k | \dots | a_{40}$.

Память ЭВМ состоит из конечного числа ячеек, поэтому машина может работать только с конечными массивами.

Элементы массива—это переменные с индексами. Индексы можно вычислять. Например, b_{i+1} и $b_{(2i+1) \cdot 4}$ при $i=0$ —это соответственно первый и четвертый элементы массива, а при $i=3$ —это—4-й и 28-й элементы массива b .

|| Какие массивы потребуются при решении задачи? ||

Как обычно, начиная решение задачи, полагаем, что сначала закуплено нулевое количество картофеля, т. е. $a_k=0$, при $k=1, 2, \dots, 40$, и Z —затраты соответственно составляют 0 руб.

Пусть P —количество картофеля, которое могут продать все колхозы: $P=b_1+b_2+b_3+\dots+b_{40}$. Сокращенно такая сумма записывается:

$$P = \sum_{k=1}^{40} b_k.$$

Опишем алгоритм решения задачи:

1. начало;
2. читать (S);
3. читать таблицу (b, c);
4. $Z := 0$;

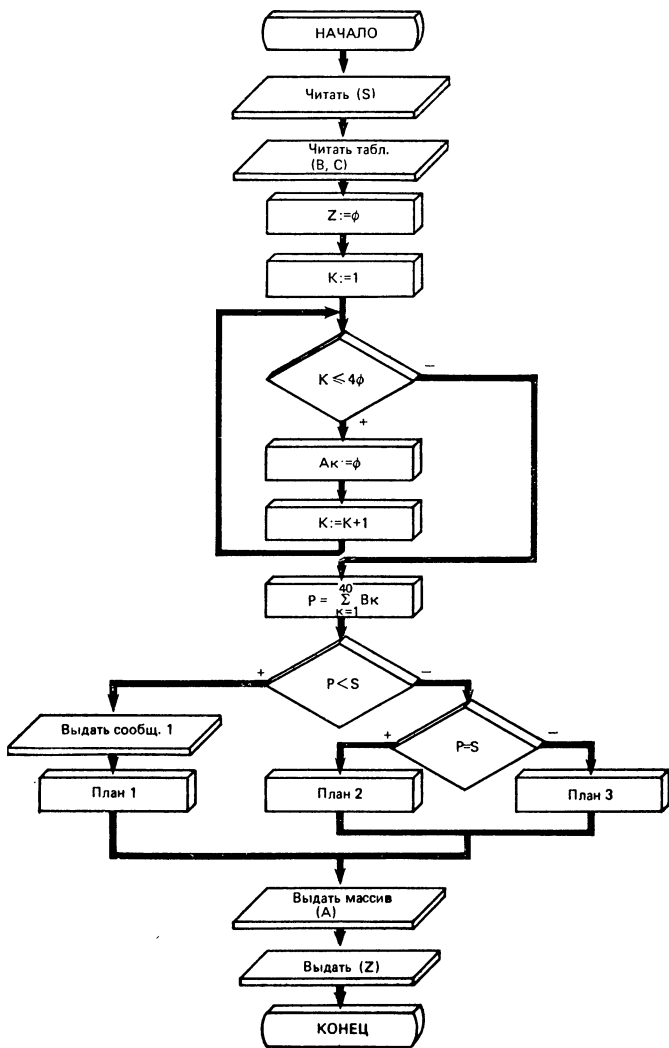


Рис. 28

5. для $k := 1$ до 40 повторять

5.1. $a_k := 0$;

6. вычислить $P = \sum_{k=1}^{40} b_k$;

7. если $P < S$

- 7.1. то {выдать ('мало картофеля');
- 7.2. реализовать план 1}
- 7.3. иначе если $P = S$
- 7.4. то реализовать план 2
- 7.5. иначе реализовать план 3;
8. выдать массив a ;
9. выдать (Z) ;
10. конец.

Уточним некоторые пункты алгоритма.

Чтобы прочитать таблицу (b, c) нужно, отдельно прочитать каждый массив:

читать массив b ;

читать массив c .

Массивы читаются поэлементно: каждый элемент массива вводится отдельно. Например, ввод массива b , состоящего из сорока элементов, может быть выполнен так:

для $k := 1$ до 40 повторять
 читать (b_k) .

|| Запишите словесно и с помощью блок-схемы || □
алгоритм чтения массива c .

Для вычисления переменной $P = \sum_{k=1}^{40} b_k$ воспользуемся

рассмотренным ранее алгоритмом суммирования.

P — сумма сорока слагаемых. Каждое слагаемое — элемент массива b , прочитанного ранее:

начало;

$P := 0$;

для $k := 1$ до 40 повторять

$P := P + b_k$;

конец.

Блок-схема алгоритма имеет вид: (рис. 29).

План 1 реализуется в случае, когда колхозы не могут продать картофель в достаточном количестве ($P < S$). Согласно этому плану у каждого колхоза следует закупить весь имеющийся картофель:

колхоз 1 может продать b_1 т картофеля, следовательно, $a_1 = b_1$ и затраты на его перевозку составляют $a_1 \cdot c_1$ руб. Второй колхоз может продать b_2 т, $a_2 = b_2$, затраты на перевозку из второго колхоза составляют $a_2 \cdot c_2$ руб. Аналогичным образом устанавливается объем закупок у других колхозов:

начало;

(* ранее полагалось $z = 0$ *)

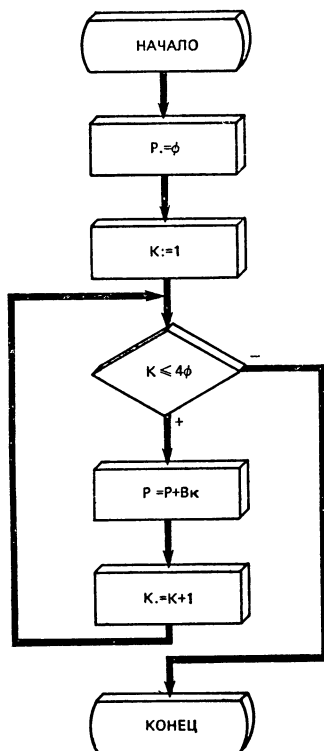


Рис. 29

для $k := 1$ до 40 повторять
 $\{a_k := b_k;$
 $Z := Z + a_k * c_k\}.$

Блок-схема приведена на рис. 30.

Согласно плану 2 следует закупить весь картофель, который могут продать колхозы, так как его количество в точности совпадает с тем, что требуется овощной базе. Следовательно, планы 1 и 2 совпадают.

Рассмотрим план 3 (случай: $P > S$).

Пусть Z_N — количество закупленного у колхозов картофеля. Вначале Z_N и соответственно Z (затраты) равны нулю.

Определим сначала колхоз, цена перевозки из которого наименьшая. Пусть это колхоз i . Предположим, далее, что в i -колхозе закупаем весь имеющийся кар-

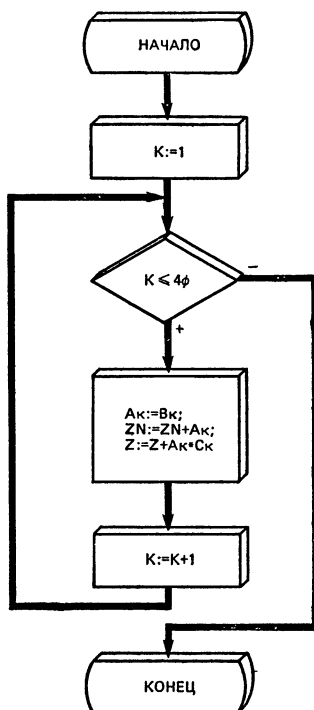


Рис. 30

тофель, тогда количество закупленного картофеля станет равным $ZN + b_i$.

Если оно больше, чем требуется овощной базе, то в i -колхозе следует закупить не весь картофель, а разность между требуемым количеством и действительно закупленным $a_i = S - ZN$, в противном случае $a_i = b_i$.

Если все же картофеля закуплено недостаточно, то, исключив из рассмотрения i -колхоз, эти действия следует повторить: найти новое i и т. д.

Чтобы исключить из рассмотрения колхоз, у которого уже закуплен картофель, соответствующую цену перевозки нужно заменить большим числом, с тем чтобы она не могла быть выбрана как наименьшая. Таким большим числом в языке Паскаль может быть константа MAXINT.

Числа в машине, как вы понимаете, представляются в некотором диапазоне. Значение MAXINT — наибольшее целое положительное число, которое может быть

представлено в ЭВМ. Естественно, что в действительности затраты на перевозку 1 т картофеля заведомо меньше этого числа.

Итак, алгоритм, реализующий план 3:

начало;

(* ранее положили Z равным 0 *)

$ZN := 0$ (* кол-во закупленного картофеля *);

пока $ZN < S$ повторять

{вычислить i (* — номер наименьшего элемента массива c *)};

если $ZN + b_i > S$

то $a_i := S - ZN$

иначе $a_i := b_i$;

$Z := Z + a_i * c_i$;

$ZN := ZN + a_i$;

$c_i := \text{MAXINT}$ }

(* величина была нами ранее прочитана *).

Блок-схема имеет вид: (рис. 31).

Опишем блок: вычислить i .

Этот блок должен реализовывать следующую задачу: найти номер минимального элемента в массиве $c = (c_1, c_2, \dots, c_{40})$.

|| Вспомните алгоритм нахождения наибольшего из 100 арбузов. Возможно, он подскажет решение и этой задачи. ||

Через i обозначим номер наименьшего элемента массива MIN — его значение.

(* пусть первый элемент — наименьший *)

$i := 1$; $\text{MIN} := c_i$;

(* сравним его с остальными элементами массива *)

для $k := 2$ до 40 повторять

{если $c_k < \text{MIN}$

то { $\text{MIN} := c_k$;

$i := k$ } }.

Итак, в результате реализации одного из планов будут вычислены массив a и переменная Z .

Вывод массива осуществляется поэлементно: для $k := 1$ до 40 повторять

выдать (k, a_k) .

|| Запишите конечный вид словесного описания алгоритма. || □

Запишем алгоритм средствами языка Паскаль. Для этого следует выяснить правила использования массивов в Паскаль-программе.

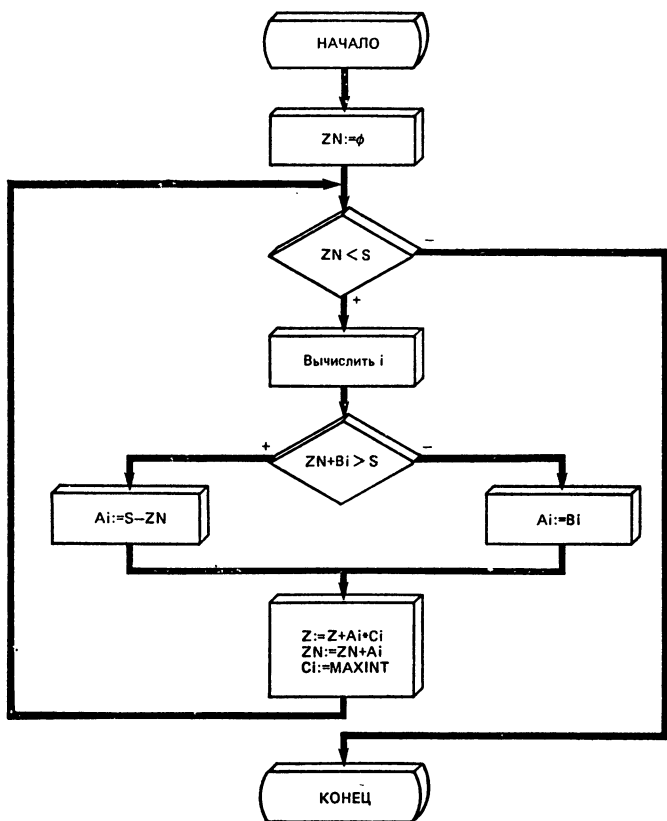


Рис. 31

Массивы, как и простые переменные, подлежат описанию в разделе переменных.

Пример 1: VAR A:ARRAY[1..40] OF REAL;
 B,C:ARRAY[1..N] OF REAL;
 D:ARRAY[5..20] OF INTEGER;
 P:REAL;

В приведенном разделе переменных описаны четыре массива: A, B, C, D и простая переменная P.

Ключевое слово ARRAY означает, что описываемый объект является массивом.

ARRAY — массив

В описании массива содержатся следующие сведения:
1) сведения о типе элементов массива (элементы массивов А, В, С—вещественные, массива D—целые);
2) диапазон изменения индексов, определяемый граничной парой, например: 1..40, 1..N, 5..20.

Нижняя граница отделяется от верхней двумя точками. Нижняя граница показывает наименьшее возможное значение индекса, верхняя—наибольшее.

Так, наименьшим возможным значением индекса у элементов массива А является 1, у элементов массива D—5.

Очевидно, что нижняя граница не может превосходить верхнюю.

Итак, описание массива строится по следующей схеме: имя массива: ARRAY [граничная пара] OF тип элементов.

Пример. В описании массивов DF и CF

```
DF:ARRAY[1..10] OF REAL;  
CF:ARRAY[1..10] OF REAL
```

указаны один и тот же тип элементов и одинаковая граничная пара, поэтому такое описание может быть заменено более коротким:

```
DF, CF:ARRAY [1..10] OF REAL.
```

Если несколько массивов имеют один и тот же тип и одинаковые граничные пары, то их описания можно объединить, разделив имена массивов запятыми.

Каждому из массивов DF и CF выделяется по 10 последовательно расположенных друг за другом ячеек памяти. О том, какое количество ячеек памяти нужно выделить массиву, машина «узнает» из описания.

Пусть массив K описан следующим образом:

```
VAR K:ARRAY[N..M] OF INTEGER.
```

Какое количество ячеек должно быть выделено для хранения элементов массива K? Это зависит от значений N и M.

Выделение памяти для переменных и массивов производится раньше, чем выполняется какой-либо оператор программы. N и M должны получить значение, прежде чем начнется процесс выделения памяти для переменных.

Определить значения N и M можно в разделе констант, который помещается в программе между разделом меток и переменных. Снова уточним последова-

тельность разделов в описательной части программы:

[раздел меток;]

[раздел констант;]

[раздел переменных;]

[раздел процедур и функций.]

Раздел констант имеет следующую структуру:

CONST имя = значение; имя = значение;

Пример. Программа чтения элементов массива и вывода элементов k_i , помноженных на число 3.1415.

```
PROGRAM RWR (INPUT, OUTPUT);
```

```
CONST N = 1; M = 10; PI = 3.1415;
```

```
VAR I: INTEGER;
```

```
    K: ARRAY[N..M] OF INTEGER;
```

```
BEGIN
```

```
  FOR I := N TO M DO
```

```
    BEGIN
```

```
      READ (K[I]);
```

```
      WRITE (K[I] * PI)
```

```
    END
```

```
END.
```

В разделе констант, который начинается с ключевого слова CONST, имя N объявлено синонимом константы 1, M — 10, PI — 3.1415.

CONST — константа

Примеры неправильного определения констант:

- 1) CONST N := 1; (* знак = заменен символом := *);
- 2) CONST N = 1, M = 10; (* если определяется несколько констант, то каждое определение отделяется от другого точкой с запятой *);
- 3) в качестве константы нельзя использовать выражение, например: CONST PI = 3.14 + 0.0015;
- 4) если некоторое имя определено в разделе констант, то оно не может объявляться в разделе переменных, а также использоваться в левой части оператора присваивания.

Пусть в разделе констант объявлено:

```
CONST N = 3.19;
```

тогда оператор присваивания $N := 1$ недопустим, как недопустим и оператор $3.19 := 1$.

|| Почему вместо констант лучше использовать || ?
их имена?

Использование раздела констант позволяет сгруппировать в начале программы величины, характерные для конкретного примера. Здесь их легче изменить. Например, если программу RWR хотим выполнить для массива, состоящего из сорока элементов, то достаточно изменить лишь раздел констант. В случае когда константы 1, 40, 3,1415 были бы записаны явно, изменения пришлось бы вносить в различные части программы, при этом некоторые части, требующие изменения, могли бы остаться не замеченными человеком, исправляющим программу. Особенно часто это бывает, если программа достаточно велика. Таким образом, использование констант не только облегчает процесс изменения программы, но и делает его более надежным.

При указании граничных пар в описании массива лучше использовать имена констант.

Итак, в рассмотренном примере под именем K используется переменная-массив, тип которой

ARRAY [N..M] OF INTEGER.

Обозначим это довольно длинное определение некоторым именем, например AR. Будем считать, что переменная K имеет тип AR, и описывать ее в разделе переменных программы так—K:AR.

Однако машине нужно сообщить, что следует понимать под типом AR. Сделаем это в разделе описания типов, который располагается между разделами описания констант и переменных. В нашем примере запись будет выглядеть так:

TYPE AR=ARRAY[N..M] OF INTEGER.

Теперь наша программа будет выглядеть так:

```
PROGRAM RWR (INPUT, OUTPUT);
CONST N=1; M=40; PI=3.1415;
TYPE AR=ARRAY[N..M] OF INTEGER;
VAR I:INTEGER;
    K:AR;
BEGIN
FOR I:=N TO M DO
    BEGIN
    READ (K[I]);
    WRITE (K[I]*PI)
    END
END.
```

Раздел описания типов начинается с ключевого слова

TYPE TYPE — тип, затем идет имя типа, в нашем случае AR, и наконец, после знака равенства записывается тип, который мы хотим обозначить указанным именем. Описание типа завершается точкой с запятой. Описание массива определяет две категории имен: имя массива в целом и имена, формируемые для обозначения каждого элемента, т. е. переменные с индексом.

Имена элементов массива строятся следующим образом:

имя массива соединяется с индексом, заключенным в квадратные скобки: имя массива [индекс], например:

$C[1], B[10], A[I + 2], B[2 * (I + 1) + 1].$

Опишем алгоритм составления плана закупки картофеля на языке Паскаль:

```
PROGRAM KARTOF (INPUT, OUTPUT);
CONST N=40;
VAR I, K:INTEGER;
    MIN, S, Z, ZN:REAL;
    A, B, C:ARRAY[1..N] OF REAL;
BEGIN
  READ(S);
  (* ВВЕСТИ МАССИВ *)
  FOR K:=1 TO N DO
    READ (B[K]);
  FOR K:=1 TO N DO
    READ (C[K]);
  ZN:=0;      (* ЗАКУПЛЕННЫЙ КАРТОФЕЛЬ *)
  Z:=0;      (* ЗАТРАТЫ *)
  FOR K:=1 TO N DO
    A[K]:=0;
  (* ВЫЧИСЛИТЬ P *)
  P:=0;
  FOR K:=1 TO N DO
    P:=P+B[K];
  IF P<S
    THEN BEGIN
      WRITE ('КАРТОФЕЛЯ НЕДОСТА-
        ТОЧНОЕ КОЛИЧЕСТВО');
      (* ПЛАН I *)
      FOR K:=1 TO N DO
        BEGIN
```

```

    A[K] := B[K];
    ZN := ZN + A[K];
    Z := Z + A[K] * C[K]
END
END
ELSE IF P = S
    THEN (* ПЛАН 2-ПЛАН 1*)
        FOR K := 1 TO N DO
            BEGIN
                A[K] := B[K];
                ZN := ZN + A[K];
                Z := Z + A[K] * C[K]
            END
        ELSE (* ПЛАН 3 *)
            WHILE ZN < S DO
                BEGIN
                    I := 1; MIN := C[I]; -
                    FOR K := 2 TO N DO
                        BEGIN
                            IF MIN > C[K]
                                THEN BEGIN
                                    MIN := C[K];
                                    I := K
                                END
                            END;
                        IF ZN + B[I] > S
                            THEN A[I] := S - ZN
                        ELSE A[I] := B[I];
                        Z := Z + A[I] * C[I];
                        ZN := ZN + A[I];
                        C[I] := MAXINT
                    END;
                    FOR K := 1 TO N DO
                        WRITELN (K, A[K]);
                        WRITELN ('ВСЕГО ЗАКУПЛЕНО', ZN,
                            'ТОНН. ');
                        WRITELN ('ЗАТРАТЫ', Z, 'РУБЛЕЙ')
                    END.
                
```

Программа получилась громоздкой и плохочитаемой. Она содержит две одинаковые последовательности операторов, реализующие планы 1 и 2. Ввод массивов В и С осуществляется по одному и тому же алгоритму. Структура программы будет улучшена, если некоторые ее части оформить в виде процедур.

Запишем в виде процедуры алгоритм решения за-

дачи: ввести элементы вещественного массива A, описанного с граничной парой [M..N]. Естественно, в программе, вызывающей процедуру, M и N определены как константы. Кроме того, там описан тип ARMNR:

```
TYPE ARMNR = ARRAY [M..N] OF REAL.
```

Входными параметрами являются значения M и N — нижняя и верхняя границы граничной пары. Они могут быть определены как параметры значения. Выходными параметрами являются элементы массива A, определяемые в результате чтения из входного файла.

Процедура будет содержать локальную переменную K — индекс вводимого элемента:

```
PROCEDURE READM (M, N:INTEGER;  
VAR A:ARMNR);  
VAR K:INTEGER;  
BEGIN  
FOR K:=M TO N DO  
    READ (A [K])  
END;
```

В качестве параметров процедур и процедур-функций можно использовать и массивы. При этом они подчиняются тем же правилам, что и простые переменные: в списке формальных параметров указывается имя массива — формального параметра вместе с именем его типа, который должен быть описан в программе.

При обращении указывается имя массива фактического параметра. Типы фактического и формального параметров должны совпадать.

Но вернемся к нашей задаче о картофеле. Определим в программе тип ARMNR:

```
TYPE ARMNR = ARRAY [M..N] OF REAL.
```

Опишем функцию SUM нахождения суммы элементов массива:

```
FUNCTION SUM (M, N:INTEGER;  
VAR A:ARMNR):REAL;  
VAR K:INTEGER;  
S:REAL;  
BEGIN  
S:=0;  
FOR K:=M TO N DO  
    S:=S+A [K];  
SUM:=S  
END;
```

Опишем процедуру, реализующую план 3, несмотря на то что она будет выполняться только один раз:

```

PROCEDURE PLAN 3 (M, N:INTEGER; S:REAL;
                  VAR Z, ZN:REAL;
                  VAR A, B, C:ARMNR);
VAR I:INTEGER;
FUNCTION NMIN (M, N:INTEGER;
VAR A:ARMNR):INTEGER;
VAR K:INTEGER;
    MIN:REAL;
BEGIN
NMIN := M; MIN := A [M];
FOR K := M + 1 TO N DO
    IF MIN > A [K]
    THEN BEGIN
        MIN := A [K];
        NMIN := K
    END
END;
BEGIN ZN := 0; Z := 0; (*закупленное количество
                        картофеля и затраты по-
                        лагаем равными нулю*)
WHILE ZN < S DO
    BEGIN
        I := NMIN (M, N, C);
        IF ZN + B [I] > S
            THEN A [I] := S - ZN
            ELSE A [I] := B [I];
        Z := Z + A [I] * C [I];
        ZN := ZN + A [I];
        C [I] := MAXINT
    END
END

```

В процедуру PLAN3 включена процедура-функция для вычисления номера минимального элемента. Входной параметр, массив C, описывается как параметр-переменная процедуры PLAN3. Некоторым элементам этого массива присваивается значение константы MAXINT. Это означает, что значения элементов фактического массива C будут изменены в результате выполнения процедуры PLAN3.

Чтобы этого не произошло, параметр C можно описать как параметр-значение, изменив список формальных параметров следующим образом:

```
(M, N:INTEGER; S:REAL; VAR Z, ZN:REAL;
VAR A,B:ARMNR; C:ARMNR).
```

В этом случае при обращении к PLAN3 будет создаваться копия массива C. Если число элементов велико, то копия будет большой. Поэтому нужно решить, какое из явлений менее желательно—изменение элементов массива C или неэкономное расходование памяти.

|| Опишите процедуру PLAN1. ||□

Запишите программу KARTOF с помощью рассмотренных процедур:

```
PROGRAM KARTOF1 (INPUT, OUTPUT);
CONST M=1; N=40;
TYPE ARMNR=ARRAY[M..N] OF REAL;
VAR K:INTEGER;
    S, Z, ZN:REAL;
    A, B, C:ARMNR;
PROCEDURE READM (M, N:INTEGER;
VAR A:ARMNR);
...
END;
FUNCTION SUM (M, N:INTEGER;
VAR A:ARMNR):REAL;
...
END;
PROCEDURE PLAN1 (...);
...
END;
PROCEDURE PLAN3 (...);
...
END;
BEGIN
READ (S);
READM (1, N, B);
READM (1, N, C);
FOR K:=1 TO N DO
    A[K]:=0;
P:=SUM (M, N, B);
IF P<S
    THEN BEGIN
        WRITE ('КАРТОФЕЛЯ НЕ ДОСТА-
        ТОЧНО');
        PLAN1 (M, N, Z, ZN, A, B, C)
        END
    ELSE IF P=S
```

```

THEN PLAN1 (M, N, Z, ZN, A, B, C)
ELSE PLAN3 (M, N, S, Z, ZN, A,
B, C);
FOR K:=1 TO N DO
  WRITELN (K, A [K]);
  WRITELN ('ВСЕГО', ZN);
  WRITELN ('ЗАТРАТЫ', Z)
END.

```

В результате выполнения программы будет выдан план, в соответствии с которым вы сможете действовать, если в городе N... вам случится быть директором овощной базы.

Глава 18 ОТ ЗАДАЧИ К РЕЗУЛЬТАТАМ

Очень часто начинающие программисты переоценивают свои возможности при создании программ. Прочитав текст задачи, многие из них берутся за ручку, чтобы тотчас же написать программу. Затем, после нескольких выходов на машину, когда из программы устранены синтаксические ошибки, они огорчаются, попав в одну из следующих типичных ситуаций.

Выдаются недопустимые результаты. Например, при вычислении площади какой-либо фигуры получается отрицательное число.

Работа завершается аварийно — производится попытка выполнить невыполнимую команду, например деление на ноль.

Не выдавая никакой информации, программа заикливается — несколько ее команд повторяются до тех пор, пока не истечет время, отведенное заданию.

И наконец, программа выдает похожие на истину результаты, и ими начинают пользоваться без всякой проверки, а при дальнейшем использовании программы вновь может возникнуть одна из описанных выше ситуаций. Обычно подобные результаты вызывают недоумение у автора программы. И он вновь и вновь без специального плана пытается найти и исправить ошибку, думая, что она последняя. А сроки работы при этом все затягиваются и затягиваются.

В программистской среде бытует утверждение: «В каждой программе есть по крайней мере одна ошибка». Поэтому имеет смысл стремиться к их минимуму с са-

мого начала решения задачи. Для этого при создании программы следует придерживаться некоторых правил.

Решение задачи проходит в несколько этапов, причем программирование не является первым из них. Рассмотрим последовательность этапов решения простых задач (отметим, что при создании больших программных систем дело обстоит несколько иначе) на следующем примере.

Вычислить площадь треугольника S по заданным сторонам a , b , c .

1. Постановка задачи.

Сущность этого этапа кратко может быть сформулирована так: «Прежде чем пытаться решить задачу, убедитесь, что вы понимаете, в чем она состоит».

Для начала попытайтесь ответить на вопросы.

Что должна делать программа?

Какие у нее исходные данные и какие результаты?

По возможности сформулируйте условия, которым должны удовлетворять исходные данные и результаты работы программы.

И независимо от того, выдано ли задание программисту кем-то или он определил его себе самостоятельно, целесообразно записать его формулировку. Ведь если нет четкой постановки задачи, то и требований к ее решению вроде бы не существует или подчас программист заменяет их другими. При этом нет возможности проверить, насколько решение отвечает требованиям, которые подразумевались при постановке задачи.

Продемонстрируем этот этап на нашем примере.

Итак, разрабатываемая программа должна вычислять площадь треугольника по трем сторонам.

Исходные данные:

переменные a , b , c — стороны треугольника.

Вычисляемый результат:

S — площадь треугольника.

По смыслу задачи a , b , c , S положительны, причем не всякая тройка чисел образует длины сторон треугольника.

Чтобы числа a , b , c могли служить длинами сторон треугольника, необходимо и достаточно, чтобы большее из них было меньше суммы двух других (известное из геометрии неравенство треугольника). Но вот вопрос: следует ли считать, что вводимые значения a , b , c удовлетворяют этому условию или программа должна предусматривать дополнительную проверку? Отвечая на

него, мы обнаруживаем, что постановка задачи не является столь ясной, какой она показалась на первый взгляд. А значит, в нее нужно внести одно из дополнений: « a , b , c заведомо являются сторонами треугольника» или: «Следует вычислить площадь, если a , b , c — стороны треугольника, в противном случае выдавать сообщение: a , b , c не являются сторонами треугольника». Теперь от нашего выбора зависит работа будущей программы.

Обычно программа должна предусматривать защиту от неправильных данных, реагируя на них выдачей соответствующего сообщения. Поэтому выбираем второе дополнение.

2. Выбор метода.

Проанализировав постановку задачи, программист выбирает или разрабатывает метод решения.

В нашем примере в связи с этим нужно рассмотреть два вопроса. Во-первых, каким способом лучше определить, что значения a , b , c могут служить длинами сторон треугольника? По определению, неотрицательные числа a , b , c могут быть длинами сторон треугольника, если максимальное из них меньше суммы двух других.

Пусть $\max(a, b, c) = c$.

Тогда правило может быть записано так: $a + b > c$. Увеличим обе части неравенства на величину c и поделим их на 2:

$$\frac{a+b+c}{2} > c.$$

Переменной P обозначим полупериметр треугольника:

$$P = \frac{a+b+c}{2}.$$

Тогда a , b , c — стороны треугольника, если $P > c$, где c — наибольшая из сторон.

Во-вторых, площадь треугольника также может быть вычислена различными способами. Мы выберем формулу Герона:

$$S = \sqrt{P \cdot (P-a) \cdot (P-b) \cdot (P-c)}.$$

Теперь метод решения поставленной задачи может быть сформулирован так. По заданным сторонам a , b , c вычислить полупериметр $P = \frac{a+b+c}{2}$. Если a , b , c образуют стороны треугольника, то вычислить его площадь $S = \sqrt{P \cdot (P-a) \cdot (P-b) \cdot (P-c)}$ и выдать результаты

расчета, в противном случае выдать сообщение о том, что a , b , c не могут служить сторонами треугольника.

Для сложных задач выбор метода решения, как правило, состоит из нескольких шагов. Сначала обсуждаются наиболее крупные действия, а уж затем последовательно может быть реализовано каждое из них. Для очень простых задач метод решения можно даже не записывать на бумаге.

3. Организация данных.

Прежде чем приступить к разработке алгоритма, следует продумать, какие переменные, массивы или другие виды данных в нем будут использованы. Это во многом определяет будущий алгоритм.

К этому этапу нередко приходится обращаться и во время разработки алгоритма, когда появляется необходимость ввести новые переменные, используемые для получения некоторых промежуточных результатов. Или, наоборот, если некоторые переменные введены неоправданно и их нужно исключить. В ходе организации данных нужно не только привести список используемых переменных, но определить их смысл, тип и условия, которым они должны удовлетворять.

В рассматриваемой задаче мы будем использовать следующие переменные:

- a , b , c — стороны треугольника;
- d — наибольшее из значений a , b , c ;
- P — полупериметр;
- S — площадь.

Для них должны выполняться следующие условия: $P > d$, $s > 0$, $a > 0$, $b > 0$ и $c > 0$.

Все переменные будем считать вещественными.

4. Алгоритмизация.

На этап построения алгоритма иногда смотрят как на некоторое вспомогательное действие, выполняемое непосредственно перед программированием. На самом деле успешная разработка алгоритма позволяет избежать многих ошибок, поскольку именно на этом этапе определяется логика будущей программы. А как известно, труднее всего находить и исправлять логические ошибки.

Вы знаете два средства, с помощью которых можно описывать разрабатываемый алгоритм: язык блок-схем и словесно-формульный способ. Алгоритм достаточно описать одним из них, причем второй способ считается более предпочтительным. Словесное описание легче чи-

тать и понимать. Оно содержит больше информации, нежели блок-схема. Кроме того, такое описание легче переводить на язык программирования.

Конечно, разработать алгоритм можно, не пользуясь никакими особыми приемами. Но вы, безусловно, быстрее и успешнее выполните этот шаг, если воспользуетесь специальными методами построения алгоритмов. Одним из них является метод пошагового уточнения.

Сначала вы пытаетесь взглянуть на задачу в целом и описать алгоритм в структурированной форме, не вдаваясь в мелкие детали. Для нашей задачи, например, это описание будет выглядеть так:

1. Начало;
2. задать (a, b, c) ;
3. вычислить полупериметр;
4. найти наибольшую из сторон a, b, c ;
5. если a, b, c — стороны треугольника
- 5.1 то {вычислить площадь S ;
- 5.2. выдать (S) }
- 5.3. иначе выдать $(a, b, c, \text{'не являются сторонами треугольника'})$;
6. конец.

А теперь переключим свое внимание с общей структуры алгоритма на более мелкие его детали.

3. Вычислить полупериметр P .

Этот шаг может быть сведен к выполнению оператора

$$P := (a + b + c)/2.$$

4. Найти d — наибольшую из сторон a, b, c .

$$d := a;$$

если $d < b$, то $d := b$;

если $d < c$, то $d := c$.

5. Условие: a, b, c — стороны треугольника теперь сформулируем так: $P > d$.

5.1. Вычислить площадь S :

$$S := \text{SQRT}(P * (P - a) * (P - b) * (P - c)).$$

Таким образом, разработка алгоритма состоит из последовательности шагов. Причем каждый из них является шагом в направлении уточнения алгоритма.

Построение алгоритма закончено, если, читая его, каждое действие вы можете заменить операторами языка программирования.

В нашем примере для этого оказалось достаточным сделать два шага. Теперь алгоритм выглядит так:

```

начало;
задать (a, b, c);
(* найти полупериметр*)
 $P := (a + b + c) / 2;$ 
(*найти наибольшую длину*)
 $d := a;$ 
если  $d < b$  то  $d := b;$ 
если  $d < c$ , то  $d := c;$ 
если (* a, b, c—стороны *)  $P > d$ 
    то { (* вычислить площадь *)
         $S := \text{SQRT}(P * (P - A) * (P - B) * (P - C));$ 
        выдать (S)}
    иначе выдать (a, b, c, 'не являются сторонами
        треугольника')
конец.
```

Вы заметили, что названия более крупных шагов внесены в конечный вид алгоритма на уровне комментариев. Таким способом мы делаем его более наглядным и понятным. Имея перед собой подробное описание алгоритма вместе с описанием данных, можно без труда выполнить следующий этап.

5. Программирование.

На предыдущих этапах был детально разработан алгоритм решения задачи, описаны используемые в нем переменные. Теперь написание программы сводится лишь к переводу этого алгоритма на язык программирования. Но если предыдущие этапы были выполнены некачественно, то алгоритм приходится дорабатывать уже на ходу. Это приводит к появлению дополнительных ошибок.

Одним из вопросов, с которым сталкиваются при программировании, является вопрос о выборе языка. На каком языке будет программироваться задача? Это зависит от многих условий. В частности, на какой ЭВМ будет решаться задача? Какие трансляторы там есть? Какой из имеющихся языков предоставляет максимум возможностей для ее решения?

Для написания нашей программы мы выберем, естественно, язык Паскаль.

Теперь основная сложность заключается в том, чтобы учесть все правила и ограничения выбранного языка.

При создании программы, которая могла бы надежно работать, подавляющая часть времени уходит не на ее написание, а на поиск ошибок и внесение исправлений. Поэтому уже при составлении программы

нужно позаботиться о том, чтобы она была наглядной, легко читалась и по выдаваемой ею информации можно было бы без труда обнаружить ошибки.

Вы уже познакомились с некоторыми приемами, используемыми при написании программ. Постарайтесь придерживаться еще и таких правил.

В каждой строчке программы размещайте по одному оператору, за исключением случаев, когда операторы небольшие и по смыслу тесно связаны друг с другом. Каждый следующий составной оператор размещайте со сдвигом на несколько позиций вправо.

А метки лучше всего располагать в самых левых позициях, чтобы они «не загорались» другими операторами. Соответствующие друг другу BEGIN, END располагайте в одних и тех же колонках.

Понять смысл программы вам помогут и комментарии. С их помощью можно указать назначение программы, смысл используемых переменных, пояснить наиболее трудные для понимания участки.

Все вводимые данные сразу распечатывайте. Это самый надежный способ проверить, что фактически поступило на вход программы.

Следуя перечисленным правилам, напомним программу решения рассматриваемой задачи:

```
PROGRAM PLOCHAD (INPUT, OUTPUT);
```

```
(*вычисление площади треугольника по трем сторонам*)
```

```
VAR A,B,C:REAL; (* стороны *)
```

```
    D : REAL ; (* значение большей из сторон *)
```

```
    P : REAL ; (* полупериметр *)
```

```
    S : REAL ; (* площадь *)
```

```
BEGIN
```

```
(* задать A, B, C *)
```

```
READ (A,B,C); WRITE ('ВХОД:' A, B, C);
```

```
(* вычислить полупериметр *)
```

```
P:=(A + B + C)/2;
```

```
(* найти наибольшую длину *)
```

```
D:=A;
```

```
IF D < B THEN D:=B;
```

```
IF D < C THEN D:=C;
```

```
IF(* A, B, C—стороны *) P > D
```

```
    THEN BEGIN (вычислить площадь)
```

```
        S := SQRT (P*(P-A)*(P-B)*(P-C));
```

```
        WRITE ('ПЛОЩАДЬ: ', S)
```

```
        END
```

```
    ELSE WRITE (A,B,C, 'не являются
```

END.

Итак, программа готова. Вы перенесли ее текст на машинный носитель: перфокарты, магнитный диск или ленту. Но еще много усилий придется приложить, прежде чем вы убедитесь в ее правильной работе.

6. Тестирование и отладка.

Для начала попытайтесь проверить свою программу, как говорят, «вручную». Внимательно читая ее текст, испытайте себя в роли исполнителя этого алгоритма на конкретных числовых данных. Возможно, что некоторые ошибки будут обнаружены таким образом.

Далее программу можно отправлять в ЭВМ, но и теперь нет никакой гарантии, что она заработает. Написав текст программы, т. е. создав так называемую исходную программу, программист направляет ее транслятору, который, в свою очередь, переведет ее на язык более низкого уровня—язык ЭВМ. Но транслятор не человек, он понимает лишь то, что написано, а не то, что мы хотели бы написать. Необходимо, чтобы правильными были все запятые, тире, пробелы, их места, а также команды и программа в целом. Первым делом транслятор проверяет синтаксические ошибки: «смотреть», нет ли неправильно использованных предложений языка. Если они есть, то программа отвергается с сообщением о том, что в таком-то месте программы найдены ошибки такого-то типа.

На первом шаге программист стремится добиться безошибочной трансляции, всякий раз изменяя перед ее выполнением несколько операторов. Но вот синтаксические ошибки устранены. Можно ли сказать, что, выполнив программу с заданными исходными данными, мы получим правильные результаты? Конечно, нет. Ведь в ней, помимо синтаксических ошибок, могут быть ошибки логические. Их-то транслятор не замечает. Чтобы выявить такие ошибки, программа проходит этап испытаний, или тестирования.

Тестирование—это процесс исполнения программы с целью обнаружения ошибок. Для проведения этого этапа заранее, обычно еще до написания программы, подготавливается специальная система примеров, просчитанных вручную или каким-либо другим способом, с тем чтобы сравнить их с результатами работы программы. Такие примеры называются тестами.

Желательно, чтобы тесты были простыми и позво-

ляли легко «вручную» вычислить получаемые результаты, а также разнообразными, чтобы достаточно полно проверить программу, все ее логические ветви.

Подбирая тесты, не нужно стремиться показать, что программа работает правильно, ибо с помощью тестирования нельзя доказать отсутствие ошибок, а можно лишь обнаружить их. Для составления как можно более полного набора тестов постараемся так подобрать исходные данные, чтобы программа вынуждена была пройти по всем ветвям алгоритма. В нашей задаче начнем с такого.

1. На вход поступают числа 3, 4, 5, являющиеся, как известно, сторонами треугольника с площадью 6. А что дает счет по программе? То же самое!

При тестировании нужно проверять не только правильные, но неверные и граничные ситуации. При этом следует уделять им внимания не меньше, чем работе с правильными данными. А поэтому рассмотрим и такие тесты.

2. При А, В, С, равных соответственно 1, 1, 2, треугольник вырождается в отрезок.

3. $(A, B, C) = (0, 0, 0)$ — это точка.

4. Зададим величины: 1, 1, 3. Треугольник с такими сторонами построить нельзя.

5. А что если на вход подать отрицательное число: 2, 1, —3?

6. Пусть все числа будут отрицательными: —4, —4, —4.

Для тестов 2—6 в качестве результата должно быть получено сообщение:

«А, В, С не являются сторонами треугольника».

Ну а теперь посмотрим, все ли ветви алгоритма будут пройдены хотя бы по одному разу при выполнении этих тестов. Оказывается, да. Но если бы этого не случилось, пришлось бы добавлять новые примеры.

Для каждого теста нужно выписывать не только входные данные, но и результаты, которые им соответствуют. Иначе, получив правдоподобные, но неверные результаты, вы можете не заметить ошибку. Ошибка обнаружится позже, и тогда для ее исправления потребуется больше усилий. Помните, чем раньше обнаружится ошибка, тем легче ее устранить. А поэтому тщательно изучайте итоги каждого тестового выполнения программы.

Но вот вы получили результаты работы некоторого теста, и они оказались совсем не такими, какие вы

ожидали. Что делать? Теперь начинается работа по выяснению того, какая ошибка или ошибки дали такой эффект, а также устранение этих ошибок. Эта работа называется отладкой. Проведение отладки роднит работу программиста с деятельностью врача, который по некоторым симптомам (выдаваемым сообщениям и результатам) пытается установить болезнь (в данном случае—ошибки в программе). Постановка диагноза, в том числе и в программировании, как известно, требует большого искусства. Однако и здесь можно рекомендовать некоторые приемы для облегчения этой работы.

Во-первых, внимательно просмотрите текст. Возможно, ошибки связаны с искажением программы. Например, если при вычислении полупериметра вместо знака / был поставлен знак *, ошибка такого рода, не замеченная транслятором, будет обнаружена лишь в результате просмотра.

Чтобы выявить логические ошибки, попытайтесь выполнить алгоритм или подозреваемую его часть «вручную». Локализовать ошибку вам поможет и печать промежуточных результатов. Попробуйте объяснить каждый оператор своей программы, просматривая ее с конца. В крайнем случае посоветуйтесь с более опытным программистом.

Итак, вы каким-то образом нашли и исправили ошибку. Теперь все тесты нужно повторить заново, чтобы убедиться, не повлекло ли за собой это исправление других ошибок. Поэтому не спешите выбрасывать тесты, чтобы вам не пришлось изобретать их заново.

В процессе тестирования выявляются ошибки, допущенные на более ранних этапах, например при разработке алгоритма в постановке задачи. А поэтому большой объем работы приходится повторять, снова и снова возвращаясь к более ранним этапам. Таким образом, описываемая последовательность действий при решении задачи не является очень уж строгой. Только простейшие программы проходят все шаги без каких-либо повторений.

Тестирование, или испытание, программы предполагает, что уже есть что испытывать, т. е. какой-то вариант уже создан. Чтобы на ЭВМ проверять программу, мы должны иметь ее уже в готовом виде. Однако надежность программы увеличится, если некоторые ее части проверять отдельно, например, выполняя их «вручную».

9. Документирование.

Если после проверки программы у вас не возникает сомнений по поводу правильности выдаваемых ею результатов, можно использовать ее в дальнейшем для проведения необходимых расчетов. Но, не имея описания программы, будет трудно разобраться в ней уже через некоторое время. Вчитываясь в каждую строку и пытаясь понять, как работает программа, что делает тот или иной оператор, что обозначает та или иная переменная, вы, безусловно, почувствуете себя детективом. Описание в еще большей степени потребуется тому, кто захочет воспользоваться вашей программой или усовершенствовать ее. А потому разработка программы заканчивается ее описанием, или документированием. Что входит в состав описания? Во-первых, инструкция по использованию программы. Если вам приходилось покупать, скажем, утюг или какой-либо другой столь же полезный предмет, то, вскрыв упаковку, вы обнаруживали там не только этот предмет, но и инструкцию по его использованию. Ознакомившись с ней, можно пользоваться утюгом, ничего не зная о его устройстве, о том, как он создавался и проектировался. Такого же принципа следует придерживаться и при написании инструкции по использованию программы. Опишите, что делает программа, каким образом пользователь должен задать исходные данные и какие действия выполнить, чтобы, ничего не зная о ее структуре, он смог бы применить программу для своих расчетов. Помимо инструкции, в документацию следует включить и краткое описание того, что было сделано на каждом этапе, начиная от постановки задачи и заканчивая набором тестов. Это более подробное описание потребуется тому, кто захочет усовершенствовать вашу программу. Подготовка документации не составит большого труда, если о ней позаботиться заранее, еще в период разработки программы. Имея в рабочем состоянии все материалы, вы сможете быстро оформить и эту, заключительную, часть работы. В частности, полезно описание логической структуры программы включать в ее текст в качестве комментариев. Следует также прокомментировать смысл каждой используемой переменной, а иногда и отдельных их значений. В хорошо документированных программах комментарии могут составлять до $\frac{3}{4}$ всего текста.

ОТ ВЫЧИСЛЕНИЙ К СИМВОЛАМ

Все рассмотренные выше задачи являются вычислительными—заданы числовые данные, нужно выполнить над ними некоторые операции для того, чтобы получить результат, тоже числовой. Именно такого класса задачи были первыми, решавшимися с помощью ЭВМ. Однако сегодня не меньшее распространение получили и другие задачи, в которых обрабатываются не числа, а символы. Коротко познакомим читателя с подобного рода задачами.

В словаре русского языка Ожегова насчитывается 57 тысяч слов. Чтобы записать каждое из этих слов, мы используем буквы русского алфавита. Из слов и знаков препинания строятся различные предложения, из предложений—тексты. Чтобы записать предложение: «У лукоморья—дуб зеленый»,—мы использовали восемнадцать различных символов: у, л, к, о, м, р, ь, я, д, б, з, е, н, ы, й, .,—и невидимый символ—пробел.

Запишем некоторый оператор языка Паскаль:

```
FOR I:=1 TO N DO
```

```
  II[I]:=I*I
```

Чтобы записать этот оператор, мы использовали символы F, R, O, I, T, N, D, I, :, =, [,], * и пробел.

Какие же символы допустимы при записи программ на языке Паскаль? Набор символов, используемых при записи программ, зависит от конкретной ЭВМ. Как правило, в этот набор включаются латинские буквы: A, B, C, ..., Z, русские буквы, отличные от латинских, цифры: 0, 1, ..., 9, разделители.

Множество всех символов, допустимых для данной ЭВМ, называется ее литерным набором.

Особенность использования русских букв состоит в том, что они могут входить лишь в состав символьных строк, например:

```
WRITE ('ПЕРЕМЕННАЯ N РАВНА', N)
```

Пример 1. В результате выполнения оператора цикла

```
FOR I:=1 TO 3 DO
```

```
  WRITELN('I', I)
```

будут напечатаны значения:

— символьная константа 'I' и значение переменной I.

Символьная константа — символ, заключенный в апострофы.

Пример 2. Символьные константы: 'A', '0', 'S', '.', '''. Отметим, что при записи символьной константы «апостроф» сам он просто повторяется дважды.

Пример 3. 'AB' не является символьной константой, а представляет собой символьную строку, поскольку это — последовательность символов, заключенная в апострофы.

Используя символьные строки, мы можем выдавать на печать различные предложения, например: WRITELN ('В СИНЕМ НЕБЕ ЗВЕЗДЫ БЛЕЩУТ.').

Предположим, во входном файле записано некоторое предложение, состоящее не более чем из восьмидесяти символов и заканчивающееся одним из символов '.', '?' или '!'. Составим программу, которая прочитает это предложение и выдаст его на печать.

Символ, прочитанный из входного файла, обозначим переменной SYMBOL.

Опишем алгоритм:

```
начало;  
  выдать ('');  
  повторять;  
    читать (SYMBOL);  
    выдать (SYMBOL);  
  до тех пор, пока (SYMBOL = '.') или (SYMBOL = '?')  
или (SYMBOL = '!');  
  выдать ('');  
конец.
```

Переменная, значением которой является символ, называется символьной переменной. В этом случае говорят, что она имеет символьный тип.

Пример 4. A:='A'

Значением символьной переменной A является символ 'A'.

Пример 5. В результате выполнения оператора присваивания $AB := '.'$ значением символьной переменной AB будет точка.

Пример 6. $READ(SYM);$
 $IF\ SYM='D'$ THEN WRITE ('ДА')
ELSE WRITE ('НЕТ')

Вводится символьная переменная SYM . Ее значение сравнивается с константой 'D' и в зависимости от результатов сравнения печатается либо слово 'ДА', либо слово 'НЕТ'.

Итак, символьные переменные могут сравниваться с другими символьными переменными, символьными константами, участвовать в операциях ввода, вывода, присваивания. Сравнивать символьные переменные можно посредством известных уже нам операций отношения: $>$, $>=$, $<$, $<=$, $=$, $<>$. Литерный набор каждой ЭВМ упорядочен. Это означает, что каждому символу литерного набора ставится в соответствие порядковый номер. Из двух символов большим считается тот, чей порядковый номер больше. Узнать порядковый номер символа из литерного набора можно с помощью функции ORD . Значением $ORD(X)$ является порядковый номер значения символьной переменной X .

Пример 7. $I := ORD('A')$.

Переменной I будет присвоен порядковый номер символа 'A'.

Символьные переменные, как и другие переменные Паскаль-программы, подлежат описанию в разделе переменных. Символьный тип указывается с помощью ключевого слова $CHAR$.

$CHAR$ —символ.

Итак, запишем программу для чтения и выдачи на печать предложения, состоящего не более чем из 80 символов.

```
PROGRAM READER(INPUT, OUTPUT);  
VAR SYMBOL:CHAR;  
BEGIN  
  WRITE('');  
  REPEAT  
    READ(SYMBOL); WRITE (SYMBOL)  
  UNTIL (SYMBOL='.') OR (SYMBOL='?') OR  
  (SYMBOL='!');
```

WRITE (""")
END.

В результате выполнения этой программы будет выдано заключенное в апострофах предложение, считанное из входного файла. Чтобы программа могла работать, следует подготовить входной файл: поместить туда предложение, заканчивающееся символом '.', '?' или '!'.

Здесь полезно сказать еще несколько слов об «устройстве» входного файла.

Пример 8. Пусть в некоторой программе описаны переменные M, N, A:

```
VAR M, N:INTEGER;  
      A:REAL
```

и выполняются следующие операторы ввода:

```
READ (M, N);  
READ (A);
```

а переменным нужно задать значение соответственно 1, 2, 3.5.

Тогда во входном файле значения располагаются так:
1 2 3.5

Несимвольные данные во входном файле должны отделяться друг от друга пробелами или начинаться с новой строки. При чтении пробелы перед несимвольными данными игнорируются.

Как во входном файле должны располагаться символы предложения для программы **READER**? Должен ли каждый символ отделяться от другого пробелом или нет?

Если из входного файла читаются символьные данные, то каждый символ представляет сам себя и пробелы являются значимыми, т. е. каждый пробел читается как символ. Таким образом, во входном файле может быть записано такое предложение;

У ЛУКОМОРЬЯ—ДУБ ЗЕЛЕНЬЙ.

Входной файл условно делится на строки, в конце которых помещается символ разделения строк. Может ли программа **READER** прочесть предложение, которое располагается в нескольких строках файла **INPUT**? После того как будет введен последний символ строки,

выполнение приказа READ приведет к чтению пробела. Чтобы перейти к следующей строке, нужно выполнить процедуру READLN.

* * *

Итак, вы познакомились с содержанием предыдущих двадцати глав. Книга прочитана. Значит ли это, что вы уже овладели искусством программирования на языке Паскаль? Конечно же, нет. Вы узнали лишь основы этого языка, простейшие приемы программирования—не зря ведь книга эта называется «Азбука программирования». И сейчас самое подходящее время для дальнейшего совершенствования в области программирования, конечно, если оно вас заинтересовало.

Позвольте дать в заключение несколько полезных советов.

1. Читайте книги по программированию и вычислительной технике. (В качестве первых лучше взять те, что приведены ниже в списке литературы.)

2. Регулярно тренируйтесь в составлении программ.

3. Постарайтесь найти где-нибудь ЭВМ (в своей организации, в соседней школе и т. п.), на которой эти программы можно пропускать.

Помните, что цели достигает тот, кто без усталости идет к ней.

ЛИТЕРАТУРА

1. Йенсен К., Вирт Н. Паскаль.— М.: Финансы и статистика, 1982.
2. Мейер Б., Бодуэн К. Методы программирования. Т. 1—2. М.: Мир, 1982.
3. Ван Тассел Д. Стил, разработка, эффективность, отладка и испытание программ.— М.: Мир, 1981.
4. Рафаэл Б. Думающий компьютер.— М.: Мир, 1979.
5. Каган Б. М. Электронные вычислительные машины и системы.— М.: Энергоатомиздат, 1985.

ОГЛАВЛЕНИЕ

Предисловие	3
От авторов	4
Глава 1. Займемся арифметикой	6
Глава 2. Обратимся по имени	10
Глава 3. Учимся приказывать	12
Глава 4. Доложите обстановку	15
Глава 5. Большую полочку или маленькую?	18
Глава 6. Поиски клада, или что такое алгоритм?	23
Глава 7. ЭВМ печатает таблицы, а читатель впервые встречается с оператором цикла	29
Глава 8. Если..., то...	39
Глава 9. Снова поговорим о циклах	46
Глава 10. Ищем бракованные стержни, а находим... ло- гические переменные и логические операции	56
Глава 11. Самый большой и самый маленький.	62
Глава 12. Алгоритм, который придумал Евклид	73
Глава 13. Если исчезнут стандартные функции	77
Глава 14. Для тех, кто решил разводить кроликов	85
Глава 15. Найдите красный шар	88
Глава 16. Нужно ли изобретать велосипед?	93
Глава 17. Если бы вы были директором	108
Глава 18. От задачи к результатам	126
Приложение. От вычислений к символам	137
Литература	142

Александр Борисович ГОРСТКО
Светлана Викторовна КОЧКОВАЯ
АЗБУКА ПРОГРАММИРОВАНИЯ (Информатика для всех)

Главный отраслевой редактор А. Нелюбов
Редактор Н. Феоктистова
Мл. редактор Н. Карячкина
Художник А. Григорьев
Худ. редактор М. Бабичева
Техн. редактор А. Красавина
Корректор В. Каночкина

ИБ № 9436

Сдано в набор 24.07.87. Подписано к печати 31.12.87. А 13864. Формат бумаги 84×108^{1/8}. Бумага тип. журн. Гарнитура литературная. Печать высокая. Усл. печ. л. 7,56. Усл. кр.-отт. 7,88. Уч.-изд. л. 7,57. Тираж 200 000 экз. Заказ 8-64. Цена 45 коп. Издательство «Знание». 101835, ГСП, Москва, Центр, проезд Серова, д. 4. Индекс заказа 886705.

Отпечатано с матриц МПО 1-ой Образцовой типографии им. А. А. Жданова на Киевской книжной фабрике, 252054, Киев-54, ул. Воровского, 24.

15 к.

А.Б.ГОРСТКО С.В.КОЧКОВАЯ

АЗБУКА ПРОГРАММИРОВАНИЯ



Издательство
Знание