

М.П.ЛАПЧИК

ВЫЧИСЛЕНИЯ.

АЛГОРИТМИЗАЦИЯ.

ПРОГРАММИРОВАНИЕ.



Р е ц е н з е н т ы:

член-корреспондент АН СССР, зав. кафедрой автоматизированных комплексов вычислительных систем факультета вычислительной математики и кибернетики МГУ Л. Н. Королев;
учитель математики школы № 183 Москвы Т. А. Коржикова

Лапчик М. П.

Л24 Вычисления. Алгоритмизация. Программирование: Пособие для учителя.— М.: Просвещение, 1988.—208 с.: ил.
ISBN 5-09-000274-6

В пособии рассматриваются приемы вычислений на микрокалькуляторах различных типов (арифметических, инженерных, программируемых), общие основы алгоритмизации, а также элементы программирования на алгоритмических языках Бейсик, Паскаль и Рапира, т. е. круг наиболее актуальных для школьного учителя тем в связи с введением в учебный процесс школы современных микропроцессорных средств и нового предмета «Основы информатики и вычислительной техники».

Л 4306010000—780
103(03) — 88 127—88

ББК 74.262

ISBN 5-09-000274-6

© Издательство «Просвещение», 1988

Перестройка технологии производства и управления в различных отраслях народного хозяйства, резкое повышение производительности труда в современных условиях немыслимы без широкого внедрения и использования средств микропроцессорной и электронно-вычислительной техники. Решение этих актуальных практических задач предполагает не только техническое перевооружение многих сфер производственной и управленческой деятельности, но и подготовку специалистов различных профилей и уровней к пониманию возможностей современной микропроцессорной техники и овладению методами ее использования в своей практической работе. Общеобразовательные основы такой подготовки должны закладываться в школе. Этой цели служит новый школьный предмет «Основы информатики и вычислительной техники», этой же цели должна служить соответствующая содержательно-методическая перестройка преподавания других школьных дисциплин, прежде всего дисциплин естественно-научного цикла (математики, физики, химии и др.).

Появление микропроцессорной техники (калькуляторы, микро-ЭВМ) на школьном уроке потребовало массовой подготовки учителей в областях, которые совсем еще недавно относились исключительно к «факультативной» сфере деятельности школы и поэтому объединяли лишь энтузиастов. Эта подготовка обеспечивается сегодня двумя путями: с одной стороны, с помощью системы краткосрочных курсов все более повышается уровень и формируется практический опыт работающих учителей, с другой — оперативно налаживается регулярное «компьютерное» образование будущих учителей в педагогических учебных заведениях. И все же пока очень непросто в школе учителю-практику, познания которого в области новой для него науки и методики чаще всего складываются «по ходу дела», в условиях острого недостатка ориентированной на школьную практику специальной и методической литературы. Настоящее пособие, адресуемое преж-

де всего учителям информатики и предметов естественнонаучного цикла, призвано в какой-то мере восполнить этот пробел. В пособии рассматриваются приемы вычислений на микрокалькуляторах, общие основы алгоритмизации, а также элементы программирования на языках Бейсик, Паскаль, Рапира, т. е. круг вопросов и тем, наиболее актуальных для современного этапа внедрения микропроцессорных средств в учебный процесс школы и постановки изучения нового предмета «Основы информатики и вычислительной техники».

В пособии четыре главы. Основное содержание первой главы составляет рассмотрение возможностей калькуляторов различных типов — арифметических, инженерных, программируемых. В условиях постепенного оснащения школ микропроцессорной техникой калькуляторы будут сохранять роль подручного и эффективного вычислительного прибора, а в ряде случаев и единственного средства поддержки школьного курса информатики. Здесь же даются краткие сведения о счетном режиме работы микроЭВМ. К этой главе непосредственно примыкает содержание второй главы, в которой рассмотрены элементарные методы анализа точности вычислений. Этот материал будет полезен учителям всех тех школьных дисциплин, при изучении которых расчеты составляют неотъемлемую часть.

Подробному рассмотрению общих основ и различных учебно-методических средств обучения алгоритмизации посвящена в пособии самая большая, третья глава. Детальное ознакомление с различными способами записи алгоритмов, раскрывающими в том числе генезис идей структуризации, позволяет учителю лучше понять особенности внутренней организации школьного алгоритмического языка и языков программирования. В четвертой главе показаны приемы программирования на алгоритмическом языке Бейсик, а также кратко проиллюстрированы особенности языков Паскаль и Рапира. В пособии рассмотрено большое число примеров, по всем разделам даны контрольные вопросы и упражнения, облегчающие самостоятельную работу с книгой.

Излагаемый в пособии материал имеет содержательно-методическую направленность, поэтому он не только расширит познания учителя в области использования вычислительной техники для решения учебных задач, но и окажет ему методическую помощь при проведении кружковых, факультативных занятий с учащимися.

Вычисления на микрокалькуляторах и микроЭВМ

1.1. Назначение и устройство микрокалькуляторов

Микрокалькулятор (МК) или, коротко, просто калькулятор* — это портативное электронное вычислительное устройство, позволяющее быстро и с большой точностью производить математические расчеты. В настоящее время промышленность выпускает достаточно широкий ассортимент калькуляторов, предназначенных для производственных, учебных и бытовых целей. Среди них различные модели МК, отличающиеся размерами и техническим оформлением: одни размещаются на столе, другие вычислитель в процессе работы держит в руках, третьи могут быть совсем миниатюрными и вмонтированными в бытовые приборы. Сегодня мы уже не мыслим другого подручного способа для выполнения более или менее сложных расчетов, как применение калькулятора (если, правда, не принимать во внимание персональный компьютер), а такие совсем недавно широко распространенные вычислительные средства, как математические таблицы или логарифмическая линейка, становятся анахронизмом и неумолимо уходят в прошлое.

Независимо от таких потребительских свойств, как размеры или способ электропитания, калькуляторы по функциональным возможностям подразделяются на три основных типа: *арифметические, инженерные* (непрограммируемые) и *программируемые*.

Арифметические калькуляторы предназначаются для выполнения обычных арифметических расчетов, они наиболее просты в эксплуатации. Основное назначение этих приборов — выполнение четырех арифметических действий. К этой группе относятся отечественные микрокалькуляторы «Электроника» моделей Б3-14, Б3-23, Б3-26, Б3-30, Б3-39, С3-22, МК-42, МК-44, МК-57, МК-60 и др.

Инженерными МК (или МК для научно-технических расчетов) называют калькуляторы, которые, помимо арифметических операций, имеют встроенные программы, позволяющие автоматически вычислять значения основных элементарных функций. К ним

* Калькулятор, калькуляция — эти связанные с вычислениями термины происходят от латинского *calculus* (галька, галыш), что объясняется весьма просто: один из простейших вариантов древнего вычислительного прибора — абака представлял собой ряд желобков, в которых по позиционному принципу размещались счетные камешки.



Рис. 1

относятся следующие модели «Электроники»: Б3-18, Б3-19, Б3-22, С3-15, Б3-36, Б3-37, Б3-38, МК-41, МК-45, МК-51, МКШ-2 и др.

Программируемые МК обладают всеми основными возможностями калькуляторов указанных выше групп, но, кроме этого, они могут автоматически выполнять целую серию заранее составленных команд (программу), введенную в память МК с помощью клавиатуры. По своему характеру программируемые МК, несмотря на свои миниатюрные размеры, принципиально не отличаются от больших программно-управляемых электронных вычислительных машин (ЭВМ). К программируемым МК относятся отечественные модели «Электроники»: Б3-21, Б3-34, МК-51, МК-46, МК-56, МК-64.

Микрокалькулятор представляет собой портативный вычислительный прибор, имеющий клавиатуру с двумя-тремя десятками клавиш (кнопок) и световой индикатор (дисплей) для чтения результатов (на рисунке 1 показан внешний вид арифметического калькулятора МК-57). В состав МК входит сложное счетно-решающее устройство, содержащее десятки тысяч функциональных элементов и соединителей. Числа в процессе вычислений размещаются в особых устройствах памяти — *registрах*. Схема основных взаимосвязей между клавиатурой, индикатором и дополнительными registros памяти изображена на рисунке 2.

На индикаторе всегда изображено то число, которое в данный момент хранится в регистре индикации (регистре X). Число, набираемое на клавиатуре, попадает в регистр X и тут же высвечивается на индикаторе. Регистр X выполняет роль одного из операционных регистров, роль другого операционного регистра у многих моделей МК выполняет особый рабочий регистр Y. Числа в регистр Y попадают из регистра X (например, после нажатия операционных клавиш сложения, умножения и др.).



Рис. 2

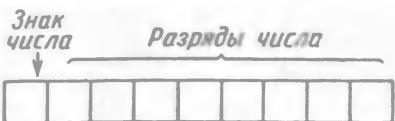


Рис. 3

-				3	2	6	9	7
---	--	--	--	---	---	---	---	---

Рис. 4

Память МК включает также и дополнительные регистры памяти (Π), которых у программируемых МК может быть более десятка, а у простейших и инженерных калькуляторов, как правило, от одного до трех. Одно из назначений дополнительных регистров — хранить промежуточные результаты вычислений, что позволяет производить довольно сложные вычисления без выписывания результатов промежуточных действий на бумагу.

Микрокалькуляторы оперируют числами, вводимыми в память и изображаемыми на индикаторе в десятичной системе счисления. Будем считать для определенности, что МК может работать с 8-разрядными десятичными числами. Если учесть еще один разряд для изображения знака числа, то на бумаге разрядную сетку МК можно изображать так, как показано на рисунке 3. В знаковом разряде индицируется только знак $-$, $+$, при изображении положительных чисел этот разряд остается свободным.

Для отделения целой части числа от дробной используется десятичная запятая (или точка). В большинстве случаев запятая при изображении числа на индикаторе размещается в том же разряде, что и цифра, за которой запятая следует. Так, например, число $-32,697$ на индикаторе МК, разрядная сетка которого изображена на рисунке 3, будет изображаться так, как показано на рисунке 4*.

В микрокалькуляторах используются в основном два способа представления чисел: с естественным размещением запятой и в форме с так называемой плавающей запятой (коротко их называют соответственно естественной и плавающей формой).

При естественной форме представления запятая в изображении числа всегда присутствует явно (как, например, на рисунке 4), может располагаться в любом цифровом разряде сетки, а ее место определяется или при вводе числа с клавиатуры, или в результате выполнения операции. Легко видеть, что в МК с 8-разрядной сеткой в форме с естественным размещением запятой могут быть представлены лишь числа из диапазонов от $-(10^8 - 1)$ до -10^{-7} и от 10^{-7} до $10^8 - 1$. Конечное множество этих чисел покрывает числовую ось двумя участками равномерно расположенных чисел с шагом 10^{-7} (рис. 5). Если в результате выполнения операции получится число, по модулю меньшее чем 10^{-7} , то на индикаторе МК с естественной формой представления чисел высветится нуль, значащие разряды, не вошедшие в разрядную

* Иногда число на индикаторе МК занимает не правые, как изображено на рисунке 4, а левые разряды сетки.



сетку МК, пропадают. Такие числа называют **машинным нулем**. Если в результате получится число, по модулю большее чем $10^8 - 1$, загорится сигнал переполнения — специальный знак в первом слева (знаковом) разряде индикатора. Рабочий диапазон чисел, используемых 8-разрядным калькулятором с естественным размещением запятой, обеспечивает решение многих практических задач. Однако не исключаются случаи, когда полученный результат не может быть зафиксирован на индикаторе.

Наиболее совершенные модели МК используют плавающую форму представления чисел. В этом случае число автоматически представляется в виде $M \cdot 10^p$, где M — мантисса, p — порядок числа, т. е. показатель степени, в которую нужно возвести 10, чтобы, умножив результат на мантиссу, получить данное число. Для изображения знака и величины порядка в разрядной сетке таких МК имеются специальные разряды — один знаковый и два для изображения величины порядка (рис. 6). Значения мантиссы в МК удовлетворяют условию $1 \leq M < 10$. Так, например, число 0,000327 при представлении в МК с плавающей запятой будет преобразовано к виду $3,27 \cdot 10^{-4}$. Максимальная величина порядка выражается двузначным числом 99. Это дает огромные диапазоны представления чисел: от $-9,9999999 \cdot 10^{99}$ до -10^{-99} и от 10^{-99} до $9,9999999 \cdot 10^{99}$. Интересной особенностью этой системы чисел является неравномерность их расположения на числовой оси.

Рис. 6



К числу микрокалькуляторов с плавающей формой представления чисел относятся ориентированный на среднюю школу калькулятор МКШ-2, инженерный калькулятор МК-41 и др.

Контрольные вопросы

- Чем различаются основные типы калькуляторов: арифметические, инженерные и программируемые?
- Где хранятся числа в МК в процессе вычислений?
- Как попадают числа в регистр X? в регистр Y?
- Содержимое какого регистра всегда отражается на индикаторе?
- Какие два способа представления чисел используются в калькуляторах?

6. Каков диапазон представления чисел с естественным размещением запятой в МК с 8-разрядной сеткой? Может ли такой калькулятор хранить величину заряда электрона ($e=1,6022 \times 10^{-19}$)?

7. Какие числа называют машинным нулем?

8. Как размещаются на числовой оси совокупности чисел, изображаемых на индикаторах МК в естественной и плавающей формах?

1.2. Арифметические расчеты на МК

Технические особенности эксплуатации конкретных микрокалькуляторов (обеспечение электропитанием, подготовка к работе и т. п.) несложны и описываются в руководствах по эксплуатации. К тому же общие принципы вычислений с помощью МК одинаковы: дело сводится к последовательному нажиманию клавишей с цифрами и знаками операций и считыванию результатов, которые мгновенно высвечиваются на индикаторе.

Различные микрокалькуляторы простейшего (арифметического) типа обладают множеством схожих функциональных возможностей, круг которых для МК этого типа все-таки ограничен. К МК этого класса принадлежат портативные калькуляторы, допускающие автономное электропитание, а также арифметические МК настольного типа с питанием от сети переменного тока (см. табл. 5). На рисунке 7 изображена схема клавиатуры калькулятора арифметического типа модели «Электроника Б3-26». Она практически не отличается от клавиатуры арифметического калькулятора МК-57 (см. рис. 1). Набор клавиш и функциональные возможности этого МК в известной степени стандартны для арифметических калькуляторов. Вычислительные возможности простейших МК рассмотрим применительно к общему случаю. При необходимости для иллюстрации отдельных операций будут использоваться конкретные модели МК.

Ввод чисел. При вводе чисел используются цифровые клавиши, клавиша с десятичной запятой (точкой), а при вводе отрицательных чисел еще и клавиша $/ - /$. Так, при вводе числа 84,036 последовательно нажимают клавиши:



Последовательность появления знаков на индикаторе при выполнении этого упражнения описана в таблице 1. Для присвоения числу на индикаторе знака $\leftarrow \rightarrow$ или для изменения знака этого числа на противоположный нажимается клавиша $/ - /$. Как уже отмечалось выше, отображаемое на индикаторе МК число

Таблица 1

Ввод	Индикатор (Х)
8	8
4	84
.	84,
0	84,0
3	84,03
6	84,036

Рис. 7

одновременно размещается в операционном регистре Х. Гашение числа на индикаторе (как и в регистре Х) осуществляется однократным нажатием клавиши сброса **C**. Многие МК имеют

клавишу **↔**, нажатие которой производит обмен содержимым между регистрами Х и Y: однократное нажатие этой клавиши меняет содержимое регистров Х и Y местами, при повторном ее нажатии содержимое этих регистров восстанавливается в первоначальном виде. Для гашения содержимого регистра в некоторых МК требуется двукратное нажатие клавиши сброса **C**: первым нажатием гасится содержимое индикатора (т. е. Х), вторым — нуль индикатора записывается в Y. Следует знать, что содержимое всех регистров гасится при отключении МК от сети. Впредь будем предполагать, что перед началом счета операционные регистры X и Y очищены.

Выполнение арифметических действий. Основное назначение арифметического МК — это выполнение операций сложения, вычитания, умножения и деления. Для выполнения каждого из этих арифметических действий над двумя десятичными числами нужно:

- 1) ввести первое число (оно высвечивается на индикаторе);
- 2) нажать одну из клавиш **+**, **-**, **×**, **÷**;
- 3) ввести второе число (при этом первое число с индикатора исчезает, на нем высвечивается второе число);
- 4) нажать клавишу **=** (на индикаторе высвечивается результат операции).

Пример 1.2.1. Для вычисления суммы $48,7 + 6,9$ нужно выполнить следующую последовательность действий:

4 8 . 7 + 6 , 9 =

В будущем при выписывании последовательности клавиш для решения на МК вычислительных задач не будем особо выделять клавиши ввода чисел. Это сделает вычислительные программы для МК более компактными и наглядными. Так, программа выполнения приведенного выше задания будет иметь вид:

48,7 + 6,9 =

После нажатия клавиши $=$ на индикаторе высветится ответ:

55,6. В выполнении двухместных операций, к которым относятся арифметические, участвуют оба операционных регистра МК: регистр X и регистр Y. Для лучшего понимания возможностей микрокалькуляторов полезно знать последовательность пересылок чисел-операндов в ходе выполнения операции. При решении задач на выполнение любой из арифметических операций над двумя числами в простейших МК происходит следующее. После ввода первого числа оно размещается в регистре X (и соответственно высвечивается на индикаторе). При нажатии одной из клавиш $+$, $-$, \times , \div содержимое регистра X пересыпается в регистр Y, причем содержимое X остается без изменения. При введении второго числа оно размещается в X, а его прежнее содержимое стирается. При нажатии на клавишу $=$

микрокалькулятор производит подготовленную операцию и записывает ответ в регистр X (индикатор), а в регистр Y засыпается второе число*. Зная порядок перемещения чисел в памяти МК в ходе выполнения операции, опытный вычислитель может использовать эти числа в последующих вычислениях без дополнительного ввода их с клавиатуры, чем достигается экономия времени и повышается эффективность вычислений. Содержимое операционных регистров X и Y при выполнении программы

48,7 + 6,9 =

на МК БЭ-26 приведено в таблице 2.

Пользуясь клавишей \leftrightarrow , можно наблюдать за содержимым регистров X и Y в процессе счета.

Таблица 2

Ввод	X	Y
48,7	48,7	0
+ \square	48,7	48,7
6,9	6,9	48,7
- \square	55,6	6,9

* У некоторых моделей МК в регистр Y при этом засыпается, как и в X, результат выполненной операции.

Выполнение цепочки операций. Как показывает рассмотренный пример, выполнение установленного действия обеспечивается нажатием клавиши $=$, в этом, собственно, и состоит ее назначение. Однако в МК этим свойством обычно обладают и операционные клавиши $+$, $-$, \times , \div : если вместо клавиши $=$ в рассмотренном примере нажать одну из операционных клавиш, то на индикаторе высветится тот же результат, что и в предыдущем случае, но, кроме этого, МК будет подготовлен к выполнению над полученным результатом арифметической операции, клавиша которой была нажата последней. Это свойство клавишей $+$, $-$, \times , \div (его называют их вторым назначением) позволяет производить цепочку арифметических действий, не выписывая промежуточных результатов и экономя число нажатий клавиш.

Пример 1.2.2. Пусть результат рассмотренного выше действия сложения нужно умножить на 8,31, т. е. выполнить вычисление: $(13,7 + 0,9) \times 8,31$. Задача решается по программе:

$$13,7 \quad + \quad 0,9 \quad \times \quad 8,31 \quad =$$

В этом случае нажатие клавиши \times производит установленную ранее операцию сложения и подготавливает МК к умножению. Это умножение и осуществляется затем нажатием клавиши $=$. Содержимое регистров X и Y в ходе выполнения этого упражнения дано в таблице 3.

Аналогичным образом могут вычисляться значения многих арифметических выражений, в которых порядок выполнения действий соответствует порядку их написания. К таким выражениям относятся, например, выражения типа $(a \pm b)c$, $(a \pm b)/c \pm d$ и др.

А что делать, если нужно вычислить значение выражения, порядок действий в котором не совпадает с порядком их написания, а некоммутативность последней предусмотренной операции не позволяет преобразовать выражение к одному из указанных выше вариантов? К ним относятся, например, следующие три простейших выражения, включающие некоммутативные операции вычитания и деления: $a/(b \pm c)$, $a - b \times c$, $a - b/c$. Что касается двух последних выражений, то они допускают желаемое преобразование: $-(b \times c) + a$, $-(b/c) + a$. Теперь задача может быть решена с использованием клавиши $/-$, нажатие которой изменяет знак числа в регистре X на противоположный. Возьмем для примера выражение $38 - 12 \times 3 = -(12 \times 3) + 38$.

Т а б л и ц а 3

Ввод	X	Y
13,7	13,7	0
<input style="border: 1px solid black; width: 20px; height: 20px;" type="button" value="+"/>	13,7	13,7
0,9	0,9	13,7
<input style="border: 1px solid black; width: 20px; height: 20px;" type="button" value="X"/>	14,6	0,9
8,31	8,31	14,6
<input style="border: 1px solid black; width: 20px; height: 20px;" type="button" value="="/>	121,326	8,31

Т а б л и ц а 4

Ввод	X	Y
8,61	8,61	0
<input style="border: 1px solid black; width: 20px; height: 20px;" type="button" value="-"/>	8,61	8,61
3,25	3,25	8,61
<input style="border: 1px solid black; width: 20px; height: 20px;" type="button" value="÷"/>	5,36	3,25
17,152	17,152	5,36
<input style="border: 1px solid black; width: 20px; height: 20px;" type="button" value="↔"/>	5,36	17,152
<input style="border: 1px solid black; width: 20px; height: 20px;" type="button" value="—"/>	3,2	5,36

Вычисление обеспечивается программой:

$$12 \quad \boxed{\times} \quad 3 \quad \boxed{+} \quad \boxed{/} \quad 38 \quad \boxed{=}$$

Прежде чем выполнить сложение с числом 38, здесь устанавливается знак \leftrightarrow перед числом на индикаторе.

Однако в каждом из трех указанных случаев легко избежать выписывания промежуточных результатов на бумаге, если воспользоваться клавишей $\boxed{\leftrightarrow}$.

П р и м е р 1.2.3. Пусть надо вычислить значение выражения

$$\frac{17,152}{8,61 - 3,25}$$

Сначала нужно выполнить вычитание, а потом деление. Запись промежуточного результата можно избежать, если перед выполнением команды деления поменять содержимое регистров X и Y. Вычисление обеспечивается программой:

$$8,61 \quad \boxed{-} \quad 3,25 \quad \boxed{\div} \quad 17,152 \quad \boxed{\leftrightarrow} \quad \boxed{=}$$

Техника вычислений видна из таблицы 4.

Значительно упрощается выполнение цепочных вычислений в МК, имеющих *дополнительные регистры памяти* (П). При наличии одного дополнительного регистра, как, например, в БЭ-26 или МК-57 (рис. 1 и рис. 7), на клавиатуре имеются клавиши:

СП — гашение содержимого регистра П, **П +** — прибавление числа, хранящегося в регистре X (индикаторе), к содержимому регистра П, **П -** — вычитание из содержимого регистра П числа,

записанного в регистре Х, ИП — вызов числа из регистра П в регистр Х. Этот несложный аппарат позволяет избегать выписывания промежуточных результатов при выполнении сложных цепочных вычислений.

Пример 1.2.4. Вычислить значение выражения $(21,3 + 6,9) \times (8,46 - 5,32)$, используя дополнительный регистр памяти.

Программа:

СП 21,3 + 6,9 = П + 8,46 - 5,32 × ИП =

(Ответ: 88,548.)

Некоторые из моделей МК арифметического типа имеют по несколько дополнительных регистров памяти, использование которых позволяет вычислителю хранить результаты промежуточных действий или даже автоматически накапливать их. Рассмотрим для примера некоторые особенности настольного микрокалькулятора арифметического типа «Электроника МК-44», располагающего тремя дополнительными регистрами памяти. Клавиатура этого МК изображена на рисунке 8.

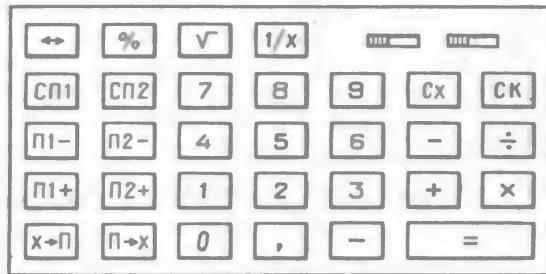


Рис. 8

Работа с дополнительными регистрами памяти П1, П2 и П3 в МК-44 обеспечивается двумя левыми вертикальными рядами клавиш. Содержимое регистра Х записывается в один из трех дополнительных регистров памяти с помощью клавиши $X \rightarrow P$

с последующим нажатием одной из цифровых клавиш 1, 2

или 3, смотря по тому, в какой из регистров — П1, П2 или П3 — надлежит записать содержимое регистра Х. Аналогично при вызове содержимого одного из трех регистров П в регистр Х после нажатия клавиши $P \rightarrow X$ нажимают одну из клавиш 1,

2, 3, определяющую номер регистра П, из которого вызывается информация.

Гашение содержимого любого из трех дополнительных регистров П можно произвести пересылкой нуля из регистра X с помощью клавиш **C**, **x→P** и одной из клавиш **1**, **2**, **3**. Для гашения содержимого регистров П1 и П2 имеются специальные клавиши **СП1** и **СП2**. Нажатием клавиш **P1+** и **P2+** содержимое регистра X добавляется к содержимому регистров П1 и П2, а с помощью клавиш **P1-** и **P2-** содержимое регистра X вычитается соответственно из содержимого дополнительных регистров П1 и П2.

Пример 1.2.5. Вычислить значение выражения

$$24,7 + 3,42 \cdot 0,49 - \frac{103,68}{21,5} .$$

Используем регистр П1. Программа может иметь вид:

24,7 $x \rightarrow \Pi$ 1 3,42 \times 0,49 = $\Pi +$ 103,68 \div
21,6 = $\Pi -$ $\Pi \rightarrow x$ 1

(Ответ: 21,5758.)

В МК-44 предусмотрен также режим автоматического накопления, устанавливаемый специальным переключателем. В этом режиме при каждом нажатии клавиши **=** получаемый результат (содержимое регистра X) автоматически прибавляется к содержимому третьего дополнительного регистра памяти.

Пример 1.2.6. В режиме автоматического накопления значение выражения $Ax + By + Cz$ может быть получено по программе (сначала производится очистка регистра П3):

C x → Π 3 A × x = B × y = C × z =
Π → x 3

Работа с константами. Представление о вычислительных возможностях простейших МК арифметического типа будет неполным, если вычислитель не знаком с особым режимом работы МК, называемым *режимом констант*. В большинстве моделей МК режим констант вводится автоматически. Сущность его состоит в том, что при выполнении двухместной арифметической операции после нажатия клавиши **=** МК запоминает выполненную при этом операцию и число, попавшее в регистр Y, которое становится

Арифметические микрокалькуляторы	С автономным источником				
Функциональные возможности	Б3-14	Б3-23	Б3-24Г	МК-57	Б3-26
4 арифметических действия	+	+	+	+	+
Действия с константой	+	+	+	+	+
Обратная величина числа	+			+	+
Извлечение квадратного корня	+			+	+
Процент от числа	+	+		+	+
Накопление в отдельном регистре памяти			+	+	+
Вычитание из содержимого регистра памяти				+	+
Накопление итоговых сумм процентных результатов					
Обмен содержимым регистров X и Y				+	+
Изменение знака числа				+	+
Округление					
Разрядность чисел	8	8	8	8	8

константой. Если повторно нажать клавишу $=$, то будет выполнена хранящаяся в памяти МК операция над содержимым индикатора и константой. Использование константы существенно экономит время вычисления выражений, в которых повторяется один операнд, так как отпадает необходимость в его повторном вводе.

При использовании режима констант важно знать, какой из двух operandов — первый или второй — в конкретной модели МК остается в памяти после выполнения двухместной операции. В большинстве моделей МК (к ним относятся арифметические МК Б3-23, Б3-24, Б3-26, МК-44 и др.) константой становится второй operand.

Пример 1.2.7. Рассмотрим выполнение программы

$$3 \times 4 = = =$$

на арифметическом калькуляторе Б3-26 или на МК-57. Последовательность действий приведена в таблице 6.

После первого нажатия клавиши $=$ в индикаторе появляется первое произведение (12), а предыдущее содержимое индикатора

Таблица 5

Платин					Настольного типа				
Б3-30	Б3-30	МК-33	МК-10	МК-53	Б3-05М	МК-59	03-22	МК-42	МК-44
+	+	+	+	+	+	+	+	+	+
+	+	+	+	+	+	+	+	+	+
		+			+	+	+	+	+
+	+			+					+
+	+	+	+	+	+	+	+	+	+
		+	+	+	+	+	+	+	+
		+	+	+	+	+	+	+	+
								+	+
			+						+
						+			+
				+					+
8	8	8	10	8	16	16	12	12	12

ра — второй операнд 4 — переходит в регистр Y и становится константой. Последующим нажатием клавиши $=$ умножают содержимое регистра X на константу, что дает последовательно 48 и 192.

Рассмотренный пример показывает, что троекратное нажатие клавиши $=$ в данном случае привело к возведению в третью степень второго множителя, т. е. было вычислено выражение $3 \cdot 4^3 = 192$.

В том, что константой становится в данном случае именно второй operand, легко убедиться, переставив operandы местами:

$$4 \quad \boxed{\times} \quad 3 \quad \boxed{=} \quad \boxed{=} \quad \boxed{=}$$

Выполнение этой программы даст совершенно другой результат:
 $4 \cdot 3^3 = 108$.

Важно заметить, что в некоторых моделях МК арифметического типа константная автоматика срабатывает при нажатии не только клавиши $=$, но и клавиш $+$, $-$, \times , \div .

Т а б л и ц а 6

Ввод	X	Y
3	3	0
<input type="checkbox"/>	3	3
4	4	3
<input type="checkbox"/>	12	4
<input type="checkbox"/>	48	4
<input type="checkbox"/>	192	4

Т а б л и ц а 7

Ввод	X	Y
12	12	0
<input type="checkbox"/>	12	12
<input type="checkbox"/>	144	12
<input type="checkbox"/>	1728	12
<input type="checkbox"/>	20 736	12
<input type="checkbox"/>	248 832	12

В этом смысле выполнение, например, программ

5 2 = = и 5 2

даст одинаковый результат.

Отметим два поучительных примера простейших вычислений, основанных на использовании константной автоматики.

П р и м е р 1.2.8. Пусть требуется вычислить a^n , где n — целое положительное число.

Вычисление обеспечивается программой:

a = = ... =

$\underbrace{\quad\quad}_{n-1 \text{ раз}}$

На МК, у которых режим констант вводится также и клавишами арифметических действий, программа может быть более естественной:

a ...

$\underbrace{\quad\quad}_{n \text{ раз}}$

Рассмотрим для примера вычисление 12^5 . Программа будет иметь вид:

12

Ход вычислений виден из таблицы 7.

П р и м е р 1.2.9. Пусть требуется вычислить обратную величину числа a , т. е. значение $\frac{1}{a}$.

Клавиатура некоторых арифметических МК имеет клавишу $\boxed{1/X}$, с помощью которой обратная величина находится автоматически по программе $a \boxed{1/X}$. Если такой клавиши нет, то выполняют программу:

$$1 \boxed{\div} a \boxed{=}$$

Способ вычисления неудобен из-за необходимости набирать на клавиатуре два числа (1 и a), и особенно если a уже имеется в регистре X (например, как результат предыдущих вычислений). Более рационально задача может быть решена программой:

$$a \boxed{\div} \boxed{\div} \boxed{\div}$$

Неожиданность приема объясняется константной автоматикой. Приводим для примера ход вычислений при нахождении значения $\frac{1}{5}$ (табл. 8).

После второго нажатия клавиши $\boxed{\div}$ на индикаторе появляется 1, что и дает при последующем нажатии этой клавиши число, обратное 5.

Таблица 8

Ввод	X	Y
5	5	0
$\boxed{\div}$	5	5
$\boxed{\div}$	1	5
$\boxed{\div}$	0,2	5

Контрольные вопросы

1. Какие клавиши используются для ввода чисел на индикатор?
2. Каким способом очищаются операционные регистры X и Y?
3. Каково назначение клавиши $\boxed{\leftrightarrow}$?
4. Какова последовательность действий вычислителя при выполнении на МК арифметических операций сложения, вычитания, умножения и деления?
5. Какова последовательность перемещения чисел в операционных регистрах X и Y в процессе выполнения арифметической операции?
6. В чем состоит второе назначение клавишей $\boxed{+}$, $\boxed{-}$, $\boxed{\times}$, $\boxed{\div}$?
7. В чем заключается режим констант и как он может быть использован в вычислениях?

Упражнения

1. Заполнить таблицу 9, проследив с помощью клавиши за содержимым регистра Y.

Пользуясь таблицей, сделать вывод о том, какое из чисел помещает МК в регистр Y при нажатии клавиши .

2. Составить программы вычисления следующих выражений:

- a) $(x \pm y)z$; b) xy/z ;
 б) $(x \pm y)/z$; г) $(x \pm y)z + u$.

3. Составить программы вычисления следующих выражений, используя клавишу :

- a) $x - yz$; б) $x - y/z$.

4. Составить программы вычисления следующих выражений:

1) используя клавишу ; 2) используя дополнительный регистр памяти:

- a) $x(y - z)$; б) $x - yz$; д) $x - (y + z)/u - v$;
 б) $x - y/z$; г) $x - y/(z - u)$; е) $(x/(yz + u) - v)/y - z$.

5. Заполнить таблицу 10, проследив с помощью клавиши за содержимым регистра Y.

Пользуясь таблицей, сделать вывод о том, какой из операндов — первый или второй — становится в данном МК константой.

6. Составить программы вычисления следующих выражений, учитывая константную автоматику:

- а) a^3b ; б) x/y^5 ; в) a^3b^4 .

Таблица 9

Ввод	X	Y
49,8		
18,5		

Таблица 10

Ввод	X	Y
31		
2		

7. По заданной программе определить вид выражений, значения которых вычисляются:

а) x

б) x

в) x

1.3. Калькуляторы для научно-технических вычислений

Рассмотрение вычислительных возможностей калькуляторов инженерного типа (см. табл. 15, с. 31) начнем с модели «Электроника Б3-18». Это наиболее распространенный образец отечественного МК, предназначенный для научно-технических расчетов. На примере этой модели МК пользователи смогли впервые реально ощутить небывалый уровень достижений в области портативной вычислительной техники. Изготовленные массовым тиражом «Электроника Б3-18» и ее модификации Б3-18А и Б3-18М и сейчас широко применяются на практике. Основные свойства этого МК сохраняются неизменными и в последующих моделях калькуляторов этого типа, таких, как Б3-22 и Б3-27.

Клавиатура калькуляторов Б3-18 изображена на рисунке 9. Судя по надписям на клавишиах, калькулятор обладает всеми возможностями рассмотренных в п. 1.2 арифметических МК. Однако возможности инженерного калькулятора значительно богаче: в них предусмотрено непосредственное вычисление натуральных и десятичных логарифмов, антилогарифмов, тригонометрических и обратных тригонометрических функций, корней, степеней и обратных величин (см. надписи над клавишами на рисунке 9). Для многих моделей инженерных МК характерной особенностью является наличие клавиши совмещенной функции

F, позволяющей использовать каждую клавишу для выполнения двух операций — по ее прямому назначению, обозначенному на самой клавише, а также согласно назначению, обозначенному над клавишей. В обычном режиме каждая клавиша используется в соответствии с обозначением, сделанным на самой клавише. При использовании клавиши для выполнения функции, обозначенной над клавишей, сначала нажимается клавиша **F**, устанавливающая режим совмещения. Снятие режима совмещения осуществляется клавишей **CF**.

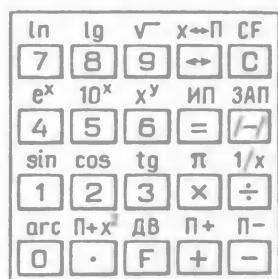


Рис. 9

При изображении вычислительных программ с помощью клавиш мы будем использовать принятые раньше их обозначение — квадрат с вписанной внутри функцией — независимо от того, изображена эта функция на самой клавише или над нею.

Ввод чисел в инженерных МК производится по тем же правилам, что и в арифметических калькуляторах. Клавиатура инженерных МК имеет обычно специальную клавишу для ввода числа π, работающую в прямом режиме или в режиме совмещения. Так, в калькуляторе Б3-18 вызов на индикатор числа π обеспечивается нажатием клавиш **F** **π**.

Выполнение арифметических действий. Наряду с регистрами X и Y инженерные МК, как и калькуляторы арифметического типа, имеют дополнительные регистры памяти (П), что позволяет уменьшить или совсем избежать записей промежуточных результатов в цепочках арифметических действий. В моделях Б3-18, как и у большинства инженерных МК, имеется один дополнительный регистр памяти. Пересылка числа из регистра X (индикатора) в регистр П производится нажатием клавиш **F** **ЗАП**,

извлечение числа из регистра П на индикатор — командой **F** **ИП**,

И в том и в другом случае стирается прежнее содержимое регистра, в который производится запись. По аналогии с клавишей **↔**, осуществляющей обмен содержимым регистров X и Y,

команда **F** **X↔П** производит перестановку содержимого регистров X и П. Гашение содержимого регистра П производится командой **C** **F** **ЗАП**.

Рассмотрим пример арифметического вычисления с использованием дополнительного регистра П.

Пример 1.3.1. Пусть требуется найти значение выражения $15,6 \cdot 2,25 + 0,35 \cdot 24,4$.

Если регистры памяти очищены, вычисление обеспечивается программой:

15,6 **×** **2,25** **=** **F** **ЗАП** **0,35** **×** **24,4** **=** **+** **F** **ИП** **=**

Движение чисел в регистрах в ходе вычислений показано в таблице 11.

Таблица 11

Ввод	X	Y	П
15,6	15,6	0	0
<input type="button" value="X"/>	15,6	15,6	0
2,25	2,25	15,6	0
<input type="button" value="="/>	35,1	2,25	0
<input type="button" value="F"/> <input type="button" value="ЗАП"/>	35,1	2,25	35,1
0,35	0,35	2,25	35,1
<input type="button" value="X"/>	0,35	0,35	35,1
24,4	24,4	0,35	35,1
<input type="button" value="="/>	8,54	24,4	35,1
<input type="button" value="+"/>	8,54	8,54	35,1
<input type="button" value="F"/> <input type="button" value="ИП"/>	35,1	8,54	35,1
<input type="button" value="="/>	43,64	35,1	35,1

Микрокалькулятор Б3-18 имеет клавиши , ,

, с помощью которых регистр П можно использовать в качестве сумматора для накопления результатов. Команды

, ,

осуществляют соответственно прибавление к содержимому регистра П числа на индикаторе, вычитание из содержимого П числа на индикаторе, прибавление к содержимому П квадрата числа на индикаторе. Использование этих клавиш позволяет вести вычисления с одновременным накоплением результатов.

Пример 1.3.2. Значение выражения вида

$$x_1 x_2 + y_1 y_2 - z_1 z_2$$

может быть вычислено по программе:

x_1 x_2 y_1 y_2 z_1 z_2

Пример 1.3.3. Пусть надо вычислить значение выражения $2,1^2 + 3,2^2 + 1,6^2$.

Задача решается программой:

2,1 [F] [П+х²] 3,2 [F] [П+х²] 1,6 [F] [П+х²] [F] ИП

Рассмотрим теперь порядок вычисления значений выражений, содержащих элементарные функции.

Для получения значений функций $\ln x$, $\lg x$, \sqrt{x} , e^x , 10^x , $\sin x$, $\cos x$, $\tg x$, $1/x$ на МК с совмещённой функцией надо ввести x на индикатор, затем последовательно нажать клавишу [F] и клавишу, над которой написано название нужной функции. Например, на калькуляторе Б3-18 значение функции $\lg x$ вычисляется следующим образом:

x [F] lg

При вычислении значения функции x^y в калькуляторе используется представление $x^y = e^{y \ln x}$, поэтому y в данном случае может быть произвольным, но x обязательно должен быть положительным. Для получения x^y на МК Б3-18 выполняется программа:

x [F] [x^y] y [=]

Для вычисления значений обратных тригонометрических функций ($\arcsin x$, $\arccos x$, $\text{arc}\tg x$) пользуются клавишей [агс]. Сначала вводят на индикатор значение аргумента x , затем нажимают клавиши [F] [агс] и одну из клавиш [sin], [cos], [tg]. Вычисления с тригонометрическими функциями обычно могут вестись в двух режимах — для значений аргумента, выраженного в градусной или радианной мере. Для установки того или иного режима на передней панели МК имеется специальный переключатель «град/рад».

Примеры простейших вычислений с использованием функций приведены в таблице 12. В тех случаях, когда положение переключателя «град/рад» безразлично, в соответствующей графе таблицы проставлен прочерк.

При вычислении значений сложных выражений, содержащих элементарные функции, для запоминания промежуточных результатов можно использовать дополнительный регистр П. Дело, однако, осложняется тем, что при вычислении на МК значений элементарных функций (кроме \sqrt{x} и $1/x$) используется регистр X. Это обстоятельство заставляет перед каждым вычислением значения элементарной функции предыдущий промежуточ-

Таблица 12

Выражение	«град/рад»	Программа	Ответ
$\cos 64^\circ$	«град»	64 [F] [cos]	0,438371
$\sin 0,8$	«рад»	0,8 [F] [sin]	0,717356
$\sqrt{469,9}$	—	469,9 [F] [$\sqrt{}$]	21,68581
$3,4^{1,2}$	—	3,4 [F] [x^y] 3,2 [=]	50,203311
$\arcsin 0,83$	«град»	0,83 [F] [arc] [sin]	56,09873°
$\arcsin 0,83$	«рад»	0,83 [F] [arc] [sin]	0,97907
$\cos 79^\circ 18'$	«град»	18 [\div] 60 [$+$] 79 [F] [cos]	0,190809

ный результат переслать в регистр П, что удлиняет процесс счета (как было отмечено выше, это не относится к вычислениям значений функций \sqrt{x} и $1/x$).

Пример 1.3.4. Пусть надо вычислить значение $\frac{28,35}{\sqrt{84,92}}$.

Задача решается программой:

$$28,35 \quad \div \quad 84,92 \quad [F] \quad [\sqrt{]} \quad [=]$$

(Ответ: 3,0761373.)

Как видно, в этом случае удалось избежать использования дополнительного регистра, так как при вычислении значения квадратного корня регистр X не используется.

Пример 1.3.5. Вычислить значение выражения $\frac{8,62}{\lg 9,44}$.

С виду это выражение аналогично предыдущему, однако для его вычисления потребуется уже иная по характеру программа:

$$9,44 \quad [F] \quad [\lg] \quad [F] \quad [ЗАП] \quad 8,62 \quad \div \quad [F] \quad [ИП] \quad [=]$$

(Ответ: 8,8412796.)

Заметим, что для выполнения этого задания можно составить более короткую программу, если воспользоваться клавишей \leftrightarrow :

$$9,44 \quad [F] \quad [\lg] \quad \div \quad 8,62 \quad \leftrightarrow \quad [=]$$

Следует учитывать, что при вычислении сложных выражений с функциями на калькуляторах с ограниченными возможностями запоминания во многих случаях целесообразнее выписать промежуточный результат на бумагу, вместо того чтобы терять время на обдумывание и реализацию хитроумной программы. Последнее оправдывается, пожалуй, лишь при неоднократном использовании таких программ для различных исходных данных.

Дальнейшее совершенствование функциональных свойств инженерных калькуляторов привело к появлению новых дополнительных возможностей: расширению числового диапазона путем удлинения разрядной сетки и введения плавающей формы представления чисел; увеличению числа дополнительных регистров памяти; исключению режима совмещения клавиш, что экономит время вычислителя, хотя и приводит к расширению клавиатуры; введению приоритета операций, а также скобок, регулирующих порядок выполнения действий. Последнее существенно облегчает производство цепочных, или, как говорят иногда, смешанных, вычислений. В той или иной мере перечисленные возможности реализованы на разных моделях современных калькуляторов инженерного типа, таких, как МКШ-2, МК-41, МК-45 и др. Рассмотрим более подробно характерные особенности одного из них — настольного калькулятора «Электроника МК-41».

Микрокалькулятор МК-41 имеет 14-разрядный индикатор, предусматривающий два режима представления чисел — с фиксированной запятой и с плавающей запятой (см. п. 1.1). Левый крайний разряд, как обычно, предназначен для изображения знака «—» отрицательного числа (знак «+» не высвечивается). Для изображения значащей части числа используются следующие десять разрядов индикатора, последние три разряда в режиме с естественным размещением запятой не задействуются. Отсюда следует, что этот режим допускает работу с числами из довольно-таки внушительного диапазона: от $-(10^{10}-1)$ до -10^{-9} и от 10^{-9} до $10^{10}-1$.

Для работы с числами, выходящими из указанных границ, используется представление чисел в форме с плавающей запятой. В этом случае десять разрядов, вмещающие значащую часть числа, используются для изображения мантиссы, а следующие три — для изображения знака и величины порядка, которая должна быть не более чем двузначной (на рисунке 10 показан пример изображения на индикаторе МК-41 числа $-2,314501952 \times 10^{13}$, знаковый разряд порядка не занят, потому что, как и при изображении знака самого числа, знак «+» не фиксируется). Очевидно, что использование плавающей формы представления чисел сильно расширяет формальный диапазон их представления.



Рис. 10

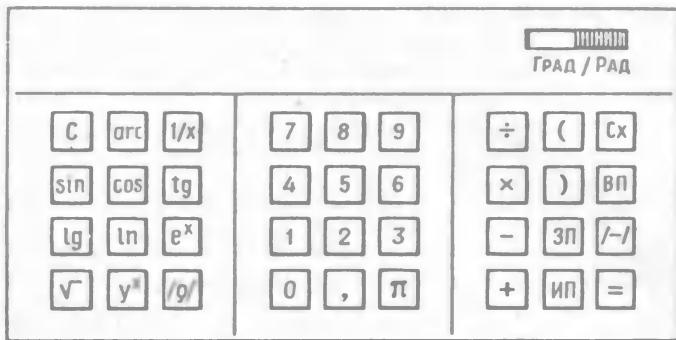


Рис. 11

Рассмотрим сейчас некоторые особенности вычислений на калькуляторе МК-41. Клавиатура МК-41 работает без режима совмещения и имеет 36 клавиш (рис. 11), на передней панели размещен также переключатель режимов «град/рад».

Для ввода чисел используются цифровые клавиши от 0 до 9, клавиша десятичной запятой , клавиша изменения знака /-/, и клавиша ввода порядка ВП. При вводе числа в форме с плавающей запятой после ввода мантиссы нажимается клавиша ВП и цифры порядка. Если порядок отрицателен, нажимается клавиша /-/ (или сразу после нажатия клавиши ВП, или после набора первой или второй цифры порядка).

Пример 1.3.6. Для ввода на индикатор МК-41 числа, изображенного на рисунке 10, клавиши можно нажимать в следующем порядке:

2 , 3 1 4 5 0 1 9 2 /-/ ВП 1 3

Следует заметить, что в МК-41 при вводе отрицательной мантиссы знак «-» не может вводиться первым: клавишу /-/ следует нажимать после любой по порядку цифры мантиссы. Для гашения содержимого индикатора в МК-41 имеется специальная клавиша Сх, ею можно пользоваться, в частности, при исправлении ошибок ввода путем повторного набора числа.

Как было отмечено выше, разрядная сетка МК-41 вмещает 10-разрядные числа. Не всегда, однако, в вычислениях требуется

сохранять столь большое количество цифр после запятой. Для регулирования количества десятичных знаков после запятой в МК-41 предусмотрена возможность установки любого желаемого режима отображения чисел на индикаторе. При использовании представления чисел с естественным размещением запятой это достигается путем последовательного нажатия клавиш **агс**, **,** **N**, где **N** — одна из цифровых клавиш, определяющая в данном случае устанавливаемое количество знаков после запятой. Заметим, что при включении калькулятора индикатор автоматически устанавливается в режим **агс**, **,** **2**, т. е. после запятой будут высвечиваться два десятичных знака.

Перевод в режим чисел с плавающей запятой производится командой **агс**, **N**, где **N** — одна из цифровых клавиш, определяющая количество десятичных знаков после запятой в мантиссе **M** при условии, что $1 \leq M < 10$.

Замечательной особенностью рассмотренных режимов индикации чисел в МК-41 является то, что изображение результатов вычислений с установленным количеством разрядов после запятой сопровождается округлением младшего сохраняемого разряда по правилу: если первая из отбрасываемых цифр больше или равна 5, последняя сохраняемая цифра увеличивается на единицу.

Исключением является лишь режим **агс**, **9**, при котором округление не производится.

Таблица 13

Режим индикации	Представление числа	
агс , , 2	46,81	
агс , , 3	46,810	
агс , , 4	46,8097	
агс , 2	4,68	01
агс , 6	4,680975	01

Пример 1.3.7. Введем на индикатор МК-41 какое-либо число, например 46,80974821. Задавая теперь разные режимы индикации, можно пронаблюдать на индикаторе соответствующее им представление заданного числа (см. табл. 13), в том числе и процедуру автоматического округления.

Примеры простейших вычислений на калькуляторе МК-41 приведены в таблице 14 (все вычисления даны в режиме **агс**, **,** **2**).

Основные двухместные арифметические операции **+**, **-**, **×**, **÷** выполняются на МК-41 обычным образом: сначала вводится первое число, затем нажимается клавиша соответ-

ствующей операции и вводится второе число. Результат операции получается после нажатия клавиши **=**. В МК-41 имеются еще две двухместные операции: возведение числа в степень (клавиша **y^x**) и извлечение квадратного корня из суммы квадратов (клавиша **$/p/$**); порядок выполнения этих операций понятен из третьей и четвертой строк таблицы 14.

Таблица 14

Выражение	«град/рад»	Программа	Ответ
$42,68 + 6,887$	—	42,68 + 6,887 =	49,57
$84,7 : 18,49$	—	84,7 ÷ 18,49 =	4,58
$8,9^{0,7}$	—	8,9 y^x 0,7 =	4,62
$\sqrt{6,81^2 + 15,9^2}$	—	6,81 $/p/$ 15,9 =	17,30
$\cos 56^\circ$	«град»	56 cos	0,56
$\operatorname{tg} 2,46$	«рад»	2,46 tg	-0,81
$\arccos 0,83$	«рад»	0,83 arc cos	0,59
$\lg 58,2$	—	58,2 lg	1,76

Что же касается вычисления значений, имеющихся на клавиатуре элементарных функций, то порядок этих действий также обычен, а отсутствие режима индикации даже упрощает действия вычислителя.

Микрокалькулятор МК-41 имеет три дополнительных (адресуемых) регистра памяти для хранения исходных данных, промежуточных результатов или констант, неоднократно используемых в ходе вычислений. Дополнительные регистры памяти имеют номера 1, 2 и 3. Для обращения к этим регистрам используются клавиши **ЗП** (запись числа из регистра X в дополнительный регистр памяти) и **ИП** (вызов числа из дополнительного регистра памяти в регистр X).

Для записи в дополнительный регистр памяти числа, хранящегося на индикаторе, необходимо нажать клавишу **ЗП** и одну из цифровых клавиш **1**, **2** или **3**, определяющих номер адресуемого дополнительного регистра. Для вызова числа из

дополнительного регистра памяти следует нажать клавишу **ИП** и одну из клавиш **1**, **2** или **3**. При этом следует помнить, что при вызове числа из адресуемого регистра памяти содержимое этого регистра сохраняется, а при записи нового числа в адресуемый регистр его прежнее содержимое пропадает.

Особенностью 1-го дополнительного регистра памяти в МК-41 является то, что его содержимое может использоваться в качестве второго операнда любой двухместной операции как константа.

Пример 1.3.8. Пусть надо выполнить ряд двухместных операций, в которых вторым операндом является одно и то же число 8:

$$25 + 8, \quad 14 \cdot 8, \quad 56 : 8, \quad 2^8.$$

Введем вначале число 8 в регистр X и перепишем это значение в 1-й дополнительный регистр памяти:

$$8 \quad \boxed{ЗП} \quad 1$$

Теперь сразу вслед за этим можно произвести все требуемые действия, не вводя уже на индикатор значение второго операнда:

$$25 \quad \boxed{+} \quad \boxed{=} \quad (\text{Ответ: } 33.) \quad 14 \quad \boxed{\times} \quad \boxed{=} \quad (\text{Ответ: } 112.)$$

$$56 \quad \boxed{\div} \quad \boxed{=} \quad (\text{Ответ: } 7.) \quad 2 \quad \boxed{y^x} \quad \boxed{=} \quad (\text{Ответ: } 512.)$$

Инженерный калькулятор МК-41 обладает рядом новых характерных особенностей, которые мы рассмотрим подробнее.

Иерархия операций. Для выполнения промежуточной двухместной операции в МК-41, как и в других калькуляторах, не обязательно нажимать клавишу **=**. При выполнении смешанных вычислений клавишу **=** нажимают, как правило, в конце задачи.

В промежуточных действиях вместо **=** можно нажимать клавиши следующих по формуле двухместных операций, однако исполнение операций производится с учетом установленного логикой МК старшинства операций. Самыми старшими из двухместных операций в МК-41 являются операции возведения в степень и извлечения квадратного корня из суммы квадратов двух чисел. Затем следуют операции умножения и деления, а за ними — сложения и вычитания. Если вслед за двухместной операцией нажимают клавишу операции того же или меньшего старшинства, то предыдущая операция исполняется. В противном случае МК только запоминает введенные числа и операции, до тех пор пока не будет нажата клавиша с меньшим или равным приоритетом или клавиша **=**.

Таблица 15

Инженерные калькуляторы		С автономным источником питания						Настольные					
Функциональные возможности		B3-16A	B3-16M	B3-37	B3-19M	B3-32	B3-35	C3-36	B3-15	B3-14	МШК-2	МК 41	МК-45
4 арифметических действия, изменения знака числа, действия с константой		+	+	+	+	+	+	+	+	+	+	+	+
Вычисления с применением скобок						+	+	+	+	+	+	+	+
Обмен содержимых операционных регистров		+	+	+	+	+	+	+					
Математические функции	x^2				+				+				
	$x^y (Y)$, $1/x$, \sqrt{x}	+	+	+	+	+	+	+	+	+	+	+	+
	$x^{1/y}$								+				
	$\sqrt{x^2 + y^2}$							+					
	$\ln x$, $\lg x$, e^x	+	+	+	+	+	+	+	+	+	+	+	+
	10^x	+	+	+	+	+	+	+	+	+	+	+	+
	тригонометрические (прямые и обратные)	+	+	+	+	+	+	+	+	+	+	+	+
Представление аргумента тригонометрических функций	в радианах	+	+	+	+	+	+	+	+	+	+	+	+
	в градусах	+	+	+		+	+	+	+	+	+	+	+
	в градах									+			
Решение системы линейных уравнений с двумя неизвестными						+					+		
Решение квадратного уравнения					+						+		
Число π		+	+	+	+	+	+	+	+	+	+	+	+
Факториал n!						+	+	+	+	+			
Работа с отдельными регистрами памяти	запись	+	+	+	+	+	+	+	+	+	+	+	+
	сложение	+	+	+		+	+	+	+	+	+	+	+
	вычитание	+	+	+		+	+	+					
	умножение							+	+				
	деление							+	+				

Введение иерархии операций существенно облегчает действия вычислителя. При выполнении цепочных вычислений на таких МК уже нет надобности обращаться к осмысленному запоминанию результатов промежуточных действий, а достаточно нажимать клавиши в той последовательности, которая определена самой формулой.

Пример 1.3.9. Рассмотрим порядок вычислений по двум простым и похожим по виду формулам: $12 \cdot 23 + 15$ и $12 + 23 \cdot 15$. И в том и в другом случае программа вычислений на МК-41 будет состоять из последовательности действий, указанной в самих формулах:

$$12 \boxed{\times} 23 \boxed{+} 15 \boxed{=} \quad (\text{Ответ: 291.})$$

$$12 \boxed{+} 23 \boxed{\times} 15 \boxed{=} \quad (\text{Ответ: 357.})$$

Однако с учетом старшинства операций нажатие клавиши $\boxed{+}$ в первой программе приведет к исполнению предыдущей операции умножения, в то время как нажатие клавиши $\boxed{\times}$ во второй программе приводит лишь к запоминанию компонентов предыдущей операции сложения. Особенности вычислений по этим двум формулам хорошо просматриваются при сравнении показаний индикатора (см. табл. 16).

Таблица 16

$12 \cdot 23 + 15 = 291$		$12 + 23 \cdot 15 = 357$	
Ввод	X	Ввод	X
12	12	12	12
$\boxed{\times}$	12	$\boxed{+}$	12
23	23	23	23
$\boxed{+}$	276	$\boxed{\times}$	23
15	15	15	15
$\boxed{=}$	291	$\boxed{=}$	357

Новые возможности МК-41 позволяют избегать использования дополнительных регистров памяти в тех случаях, когда их использование, например, в вычислениях на инженерных калькуляторах типа Б3-18 было необходимо (см. пример 1.3.5 на с. 25).

Пример 1.3.10. Вычислить на МК-41 в режиме $\boxed{\text{агс}}$ $\boxed{,}$

4 значение выражения $\frac{8,62}{\lg 9,44}$.

Программа: 8,62 \div 9,44 \lg $=$ (Ответ: 8,8414.)

П р и м е р 1.3.11. Вычислить на МК-41 с двумя знаками после запятой:

$$19 + 6 \cdot 2,1^3 \cdot \sqrt{0,6^2 + 4,7^2}.$$

Программа:

$$19 + 6 \times 2,1 y^3 3 \times 0,6 / \rho / 4,7 =$$

(Ответ: 282,28.)

Использование скобок. Наличие на клавиатуре специальных клавиш $($ и $)$ позволяет при необходимости изменять естественный порядок выполнения операций, определяемый их старшинством.

Например, для вычисления значения выражения $(21 + 13) \cdot 5$ на МК со скобками последовательность нажатия клавиш будет в точности совпадать с порядком следования соответствующих действий в самой формуле:

$$(21 + 13) \times 5 =$$

Следуя логике вычислительных формул, клавиши открывающих и закрывающих скобок можно нажимать по несколько раз подряд. Рассмотрим пример.

П р и м е р 1.3.12. Вычислить значение выражения

$$((21 + 35) / 4 + 75) \cdot 16 + 144) / 28.$$

Программа:

$$(((21 + 35) \div 4 + 75) \times 16 + 144) \div 28 =$$

(Ответ: 56.)

Существующие при этом правила позволяют в данном случае не нажимать первые три клавиши открывающих скобок.

Для запоминания операций в вычислениях со скобками в МК используется особым образом организованная *стековая память**. Стековые регистры служат также для запоминания операций и чисел в том случае, когда в цепочных вычислениях старшинство последующей операции превосходит старшинство предыдущей операции. Емкость стековой памяти МК-41 позволяет запоминать

* Подробнее об организации стековой памяти см. в следующем пункте.

до 8 чисел и до 7 операций. При переполнении стековой памяти на индикаторе появляется обычный сигнал переполнения.

В заключение приведем несколько примеров программ смешанных вычислений на МК-41 (в режиме **агс**, **,** **2**).

Пример 1.3.13. Вычислить

$$\ln 9,3 \cdot \sin 3,79 + \frac{16,48}{\cos 0,092}.$$

Программа:

9,3 **ln** **×** 2,79 **sin** **+** 16,48 **÷** 0,092 **cos** **=**

(Ответ: 17,32.)

Пример 1.3.14. Вычислить

$$32,4 \cdot (8,24 + 0,72)^{4,1}.$$

Программа:

32,4 **×** **(** 8,24 **+** 0,72 **)** **y^x** 4,1 **=**

(Ответ: 260 020,66.)

Пример 1.3.15. Вычислить

$$\sqrt[5]{(7,91 + \ln 6,94) \cdot 3,85}.$$

Программа:

(7,91 **+** 6,94 **ln** **)** **×** 3,85 **=** **y^x** 5 **1/x** **=**

(Ответ: 2,07.)

Контрольные вопросы

1. Какие дополнительные функциональные возможности имеют инженерные микрокалькуляторы по сравнению с простейшими?

2. В чем состоит назначение клавиши совмещенной функции

F ?

3. Чем различаются действия, осуществляемые нажатием клавиш **П+** и **ЗАП**?

4. Каков порядок нажатия клавиш при вычислении значений одноместных элементарных функций?

5. Каков порядок ввода чисел в форме с плавающей запятой?

6. Как производятся цепочные действия на МК: а) с установленной иерархией арифметических операций; б) с использованием клавиш **(** и **)**?

Упражнения

1. Составить программы вычислений следующих выражений, используя дополнительные регистры памяти:

$$a) \frac{a}{x} - by; \quad b) \frac{a-x}{b+y}; \quad в) (a+x)^2 - (b+y)^2; \quad г) \frac{(a-x)^3}{b^2 + y^2}.$$

Составить аналогичные программы для МК с установленной иерархией операций и скобками.

2. Как установить с помощью МК, что больше: e^x или π^x ?

3. Составить программы вычисления значений выражений, используя клавишу \leftrightarrow и дополнительный регистр памяти:

$$a) \frac{2x}{\ln y}; \quad б) \frac{\sqrt{x}}{\ln y}; \quad в) (x+y)^{\frac{1}{2}}; \quad г) \frac{a \cdot \sin x - b \cdot e^x}{\arccos(x+y^2)}.$$

Составить аналогичные программы для МК с иерархией памяти и скобками.

1.4. Особенности школьного калькулятора МКШ-2

Предназначенный для средней школы калькулятор «Электроника МКШ-2» обладает основными свойствами микрокалькулятора инженерного типа, приспособленного для выполнения научно-технических расчетов в достаточно широком числовом диапазоне. Калькулятор МКШ-2 использует естественную и плавающую формы представления чисел, но способ представления чисел в разрядной сетке индикатора имеет свои особенности.

Индикатор МКШ-2 содержит 9 разрядов. При изображении числа в естественной форме (рис. 12) крайний левый разряд используется для изображения знака «+» (знак «-» не высвечивается), а в остальных 8 разрядах размещаются цифры числа, точнее, его модуля. Плавающая форма представления чисел в МКШ-2 использует 9 разрядов индикатора, но для записи мантиссы при этом отводится лишь 5 разрядов (рис. 13). Это означает, что у числа в плавающей форме на индикаторе МКШ-2 мантисса не может содержать более 5 цифр. Пусть, к примеру, имеется число $15,93478 \cdot 10^{-5}$ и мы собираемся ввести его на индикатор. Введя последовательно знаки мантиссы 25,93478 и нажав вслед за этим клавишу **ВП** (ввод порядка), мы увидим, что последние две цифры 7 и 8 исчезли с индикатора.



Рис. 12



Рис. 13

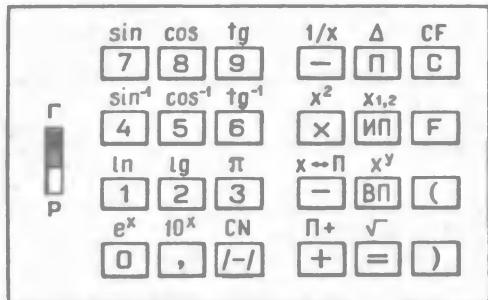


Рис. 14

Расположение клавиш на клавиатуре МКШ-2 показано на рисунке 14. Большинство клавиш используется в режиме совмещения, устанавливаемом клавишей **F**. Назначение многих клавиш понятно из соответствующих надписей. У некоторых модификаций МКШ-2 обозначения клавиш **Π** и **ИП** заменены соответствующими обозначениями **x→Π** и **Π→x**, имеющими тот же смысл. Есть и новые клавиши. Среди них клавиша **CN**, которая используется для изменения формы представления чисел. Переход от одной формы записи числа на индикаторе к другой производится последовательным нажатием клавиш **F** и **CN**.

Наряду с традиционными для инженерных калькуляторов клавишами вычисления значений элементарных функций (обозначения \sin^{-1} , \cos^{-1} , \tg^{-1} использованы для обратных тригонометрических функций) МКШ-2 имеет встроенные программы для вычисления действительных корней квадратного уравнения (клавиша **X_{1,2}**) и вычисления определителя системы двух линейных уравнений с двумя переменными (клавиша **Δ**). Рассмотрим порядок использования этих клавиш.

При вычислении действительных корней квадратного уравнения $ax^2+bx+c=0$ ($a \neq 0$) на МКШ-2 исполняется программа:

a (**b** (**c** **F** **X_{1,2}** ↓ **F** **X_{1,2}** ↓

В этой программе клавиша открывающей скобки **(** служит как бы условным обозначением переменной x . Знак **↓** использован для обозначения момента записи результата на бумаге. Если уравнение $ax^2+bx+c=0$ не имеет действительных корней, то

после первого нажатия клавиш **F** **X_{1,2}** во всех разрядах индикатора высветятся минусы. То же самое произойдет, если будет введено значение $a=0$.

Пример 1.4.1. Решить на МКШ-2 уравнение

$$3x^2 + 5x - 22 = 0.$$

Выполним программу:

3 **(** 5 **(** 22 **/-** **F** **X_{1,2}** **↓** **F** **X_{1,2}** **↓**

Получим ответ: $x_1 = 3,6666666$, $x_2 = 2$.

Для вычисления определителя второго порядка

$$\Delta = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

в МКШ-2 используется клавиша **Δ**. Вычисление производится по программе:

a **(** b **(** c **(** d **F** **Δ**

Пример 1.4.2. Вычислить определитель

$$\Delta = \begin{vmatrix} 3 & 2 \\ 1 & 4 \end{vmatrix}.$$

Программа:

3 **(** 2 **(** 1 **(** 4 **F** **Δ**

(Ответ: 10.)

Используя программу вычисления определителя, на МКШ-2 можно достаточно быстро решать системы двух линейных уравнений с двумя переменными по формулам Крамера.

Пример 1.4.3. Решить систему

$$\begin{cases} 12x - 3y = 15, \\ x + 8y = 26. \end{cases}$$

Выполним программы:

12 **(** 3 **/-** **(** 1 **(** 8 **F** **Δ** **↓** ($\Delta = 99$)

15 **(** 3 **/-** **(** 26 **(** 8 **F** **Δ** **↓** ($\Delta_x = 198$)

12 **(** 15 **(** 1 **(** 26 **F** **Δ** **↓** ($\Delta_y = 297$)

198 **÷** 99 **=** **↓** ($x = 2$)

297 **÷** 99 **=** **+** ($y = 3$)

(Ответ: $x = 2$, $y = 3$.)

Контрольные вопросы

1. Каковы особенности представления чисел в естественной и плавающей формах в разрядной сетке калькулятора МКШ-2?
2. Каким образом осуществляется переход от одной формы представления числа на индикаторе к другой?
3. Как с помощью клавиши $X_{1,2}$ на МКШ-2 вычисляются действительные корни квадратного уравнения?
4. Как с помощью клавиши Δ вычисляется значение определителя второго порядка?

Упражнения

1. Ввести на индикатор МКШ-2 числа: а) 25; б) -493; в) 25,698; г) -324,01; д) $52,341 \cdot 10^{-12}$; е) $-45,34869 \cdot 10^5$; ж) 0,0000032491.
2. Решить квадратные уравнения (в области действительных чисел), используя МКШ-2:
 - а) $x^2 + 2x - 3 = 0$;
 - б) $4x^2 - 3x + 15 = 0$;
 - в) $1,5x^2 - 13x - 14,5 = 0$.
3. Пользуясь МКШ-2, решить системы двух линейных уравнений с двумя переменными:
 - а) $\begin{cases} x - 4y = -7, \\ -2x + 5y = 8, \end{cases}$
 - б) $\begin{cases} 4,3x + 8,6y = 3,7, \\ -6,2x + 4,3y = -13,7. \end{cases}$

Произвести проверку решения, подставив найденные значения x и y в исходные системы уравнений.

1.5. Программируемый микрокалькулятор

Основное отличие программируемых МК (ПМК) от арифметических и инженерных калькуляторов состоит в том, что они допускают ввод и автоматическое исполнение заранее составленной последовательности команд — программы, включающей как арифметические (вычислительные) действия, так и проверку условий, обеспечивающую выбор направления дальнейших вычислений. Умелое составление программы для МК этого класса позволяет во многих случаях с помощью весьма короткой последовательности команд заставить ПМК автоматически выполнить большое количество вычислительных операций, что значительно повышает эффективность расчетов.

Первой отечественной моделью программируемых калькуляторов является «Электроника Б3-21». В последующих моделях ПМК этого типа (Б3-34, МК-54 — с автономным источником

питания, МК-46, МК-56 — с питанием от сети переменного тока) функциональные возможности значительно расширены, но вместе с тем логика и входной язык у всех этих моделей имеют много общего.

На программируемых калькуляторах можно вести и обычные расчеты без ввода программы в память. При этом особенностью всех ПМК является использование нестандартной вычислительной логики, основанной на так называемой *обратной польской записи* вычисляемых выражений. В 1921 г. польский математик и логик Ян Лукасевич, разрабатывая особый язык формализации логических и математических выражений, предложил размещать знаки арифметических операций или перед, или после операндов. Первый из этих способов стали называть прямой, а второй — обратной польской записью. В обратной польской записи, например, выражение $a + b$ принимает вид $ab+$. Применение такого необычного представления выражений в МК обусловлено более простой технической реализацией вычислений (среди инженерных МК по такому принципу работает «Электроника Б3-19»).

Клавиатура одной из современных моделей ПМК «Электроника МК-56» изображена на рисунке 15. Содержимое регистра X, как обычно, отображается на индикаторе. Для пересылки содержимого индикатора в рабочий регистр Y используют клавишу

B↑. При выполнении арифметической операции на ПМК необходимо:

- 1) ввести в X первое число;
- 2) нажать клавиши **B↑** переслать первое число в регистр Y;
- 3) ввести в регистр X второе число;
- 4) нажать одну из клавиш **+**, **-**, **×**, **÷**.

Как видно, клавиша **=** в данном случае оказывается не нужной (ее и нет на клавиатуре). Вычисление значений одноместных элементарных функций производится обычным обра-

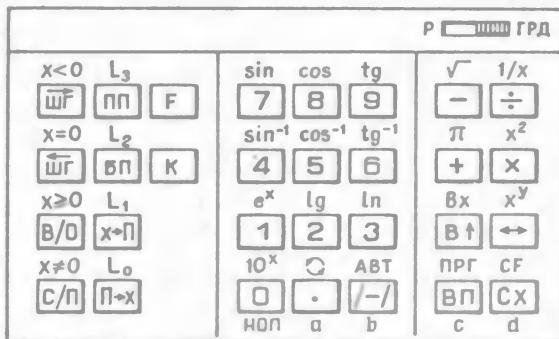


Рис. 15

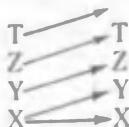
Таблица 17

Задача	Программа	Индикатор (Х)
$0,34 + 9,72$	0,34 [B↑] 9,72 [+]	10,06
$12,4 \cdot 8,5$	12,4 [B↑] 8,5 [X]	105,4
$2,46^2$	2,46 [F] [x^2]	6,0516
$10^{-0,41}$	0,41 [/–/] [F] [10 ⁰]	3,8904512 –01
$24,1^{13,5}$	13,5 [B↑] 24,1 [F] [x^y]	4,541822 –18
$\sin 0,9$	0,9 [F] [sin]	7,83322691 –01
$\arccos 0,2$	0,2 [F] [cos ⁻¹]	1,3694383

зом: в регистр X вводится аргумент, нажимается клавиша [F] и нужная клавиша с обозначением функции (\sin^{-1} , \cos^{-1} , \tg^{-1} обозначают соответствующие обратные тригонометрические функции). Примеры простейших вычислений на МК-56 приведены в таблице 17 (аргумент в радианах).

Несмотря на некоторую необычность порядка ввода данных при выполнении двухместных операций, обратная польская запись в ряде случаев оказывается даже более удобной для вычислений. Однако, чтобы понять эти особенности работы ПМК, необходимо познакомиться подробнее с организацией стековой памяти.

Стековую память МК-56 образуют четыре регистра: известные уже нам операционные регистры X и Y, а также еще два регистра Z и T. Ввод чисел всегда производится в регистр X. При нажатии клавиши [B↑] копия числа из регистра X передается в регистр Y, то, что было в Y, пересыпается в Z, а прежнее содержимое Z — в регистр T. При этом содержимое регистра X сохраняется, а содержимое регистра T исчезает. Это передвижение можно изобразить так:



Указанное передвижение происходит при каждом нажатии клавиши [B↑]. Пользуясь этим, в регистры стека можно записать любые числа.

Так, в результате нажатия клавиш

4 $\boxed{B \uparrow}$ 3 $\boxed{B \uparrow}$ 2 $\boxed{B \uparrow}$ 1

произойдет запись чисел 4, 3, 2, 1 соответственно в регистры T, Z, Y, X. Очистка всех регистров стека производится так:

\boxed{Cx} $\boxed{B \uparrow}$ $\boxed{B \uparrow}$ $\boxed{B \uparrow}$

В дополнение к четырем стековым регистрам в МК-56 имеется еще один регистр, называемый *регистром предыдущего результата* (X1). Этот регистр всегда сохраняет значение числа, которое находилось в регистре X до выполнения операции.

При выполнении одноместных операций ПМК оперирует с числом, находящимся в регистре X. При этом содержимое регистров Y, Z, T сохраняется, а число, находившееся до выполнения операции в регистре X, передается в регистр X1. При выполнении двухместных операций ПМК оперирует числами, находящимися в регистрах X и Y. При этом после нажатия операционной клавиши происходит обратное перемещение информации: содержимое регистра T пересыпается в Z, то, что было в Z, переходит в регистр Y, в регистр X заносится результат выполненной операции, а прежнее содержимое регистра X передается в X1. К сказанному нужно добавить, что если число на индикаторе является результатом предыдущих вычислений, то набор на клавиатуре нового числа автоматически передвигает информацию из регистра X в регистр Y. Таким образом, результат выполнения предыдущей операции может участвовать в качестве второго числа при выполнении последующей операции.

Пример 1.5.1. Рассмотрим с учетом указанных перемещений порядок вычислений по формуле

$$A \cdot B + C \cdot D.$$

Эта формула, несмотря на кажущуюся простоту, является, как известно, непростым «орешком» для МК, не имеющих иерархии операций. На ПМК вычисления по этой формуле выполняются весьма просто:

A $\boxed{B \uparrow}$ B $\boxed{\times}$ C $\boxed{B \uparrow}$ D $\boxed{\times}$ $\boxed{+}$

В таблице 18 приведено подробное описание всех перемещений при выполнении этой программы (для удобства принято, что перед вычислениями во всех регистрах стека лежат нули). Аналогичным образом обеспечивается и вычисление по формуле $(A+B)(C+D)$, которое может быть выполнено по программе:

A $\boxed{B \uparrow}$ B $\boxed{+}$ C $\boxed{B \uparrow}$ D $\boxed{+}$ $\boxed{\times}$

Таблица 18

Ввод	X	Y	Z	T
A	A	0	0	0
	A	A	0	0
B	B	A	0	0
	AB	0	0	0
C	C	AB	0	0
	C	C	AB	0
D	D	C	AB	0
	CD	AB	0	0
	AB+CD	0	0	0

Для хранения исходных данных и промежуточных результатов ПМК, как и калькуляторы других типов, имеют дополнительные *адресуемые регистры памяти*. В МК-56 таких регистров 14, они имеют номера (адреса) 0, 1, 2, ..., 9, a , b , c , d^* . Запись числа из регистра X в адресуемые регистры осуществляется нажатием клавиши и одной из клавиш, обозначающих номер адресуемого регистра. При этом число, переданное в адресуемый регистр, сохраняется в регистре X. Так, например, запись числа из регистра X в регистры R4 и Ra на МК-56 осуществляется командами:

и

Для вызова числа, хранящегося в адресуемом регистре, в регистр X необходимо нажать клавишу и клавишу, обозначающую номер адресуемого регистра.

Однако главное назначение ПМК не в выполнении разовых вычислений по формуле (для этих целей можно успешно использовать инженерные и арифметические калькуляторы), а в том, чтобы запоминать целые программы и автоматически выполнять их. Для запоминания команд программы у ПМК имеется специальная программная память. Например, программная память

* В дальнейшем адресуемые регистры будем обозначать буквой с расположенным за ней номером: R0, R1, R2 и т. д., а содержимое регистров — с помощью его номера, заключенного в круглые скобки. Например, $(R9)$ обозначает содержимое регистра с номером R9.

МК-56 состоит из 98 ячеек. Первая ячейка имеет номер 00, последняя — 97. Чтобы получить общее представление об использовании ПМК в режиме программирования, рассмотрим примеры.

Пример 1.5.2. Составить программу для автоматического вычисления на МК-56 длии окружностей по заданному радиусу r .

Как известно, длина окружности вычисляется по формуле $L = 2\pi r$. Программа ручных вычислений по формуле на МК-56 весьма проста:

2

Предположим теперь, что значение радиуса r заранее записано в регистр X (т. е. находится на индикаторе). При этом условии действия по вычислению L сведутся к следующему:

2

Программа начинается с того, что хранящееся (по предположению) в регистре X значение r отправляется в регистр Y. Запишем теперь все команды этой программы шаг за шагом в программную память ПМК. Для этого калькулятор нажатием клавиш

переводится в режим «Программирование». В процессе пошагового ввода программы на индикаторе будет наблюдаться движение двузначных кодов команд (слева направо). При этом крайнее правое двузначное число регистрирует значение счетчика вводимых команд программы и принимает последовательные значения: 00, 01, 02 и т. д.*. Текущее состояние счетчика команд выражает адрес ячейки программной памяти, по которому будет записана следующая команда программы. Покомандное изложение приведенной выше программы вычисления длины окружности $L = 2\pi r$ в предположении, что значение радиуса r находится в регистре X, дано в таблице 19. Здесь же указаны адреса ячеек, хранящих команды, а также дано описание действий, которые производит каждая команда. Таким образом, вся программа состоит из шести команд и занимает шесть ячеек памяти с адресами 00, 01, 02, 03, 04, 05.

Когда ввод всей программы закончен, микрокалькулятор нужно вернуть в счетный режим работы, для чего нажимают клавиши (этот режим носит название «Автоматическая работа»).

* Программу можно записать в память с произвольного адреса. Для этого, прежде чем перейти в режим «Программирование», нужно нажать клавишу , а затем значение адреса первой команды.

Адрес команды	Команда	Выполняемое действие
00		Пересылка значения r из регистра X в регистр Y
01		Ввод числа 2 в регистр X
02		Вычисление $2 \cdot r$
03	 	Ввод числа π в регистр X
04		Вычисление $2 \cdot \pi \cdot r$
05		Остановка и индикация ответа в регистре X

Программа пускается нажатием клавиш (клавиша устанавливает нулевое значение счетчика команд, клавишей производится пуск). Однако, прежде чем сделать это, как следует из предположения, принятого перед составлением программы, необходимо в регистр индикации X записать значение радиуса. Пусть, к примеру, нужно вычислить длину окружности радиуса $r = 5$. Тогда после выполнения всех описанных выше действий по вводу программы нужно нажать клавиши:

5

На индикаторе высветится ответ: 31,415926. Если нужно получить длины окружностей для значений радиуса r_1, r_2, \dots, r_n , то необходимо n раз обратиться к введенной в программную память ПМК программе с помощью команд:

r_i ($i = 1, 2, \dots, n$).

Записанная в память ПМК программа будет храниться там до тех пор, пока на ее место не будет записана другая программа или произойдет отключение ПМК.

На программируемых микрокалькуляторах весьма эффективно решается задача вычисления таблицы значений функции (эту часто встречающуюся практическую задачу называют еще *табулированием функции*). Задача табулирования ставится обычно следующим образом. Дается функция $f(x)$, отрезок $[a; b]$, на котором функция определена, и шаг h . Требуется вычислить значения функции в точках $a, a+h, a+2h, \dots$ до тех пор, пока очередное значение аргумента не выйдет за правую границу отрезка. При этом оформляется таблица 20.

Таблица 20

Поскольку в вычислениях на микрокалькуляторе вычислитель сам управляет ходом вычислительного процесса, можно считать, что для постановки задачи табулирования функции на ПМК, помимо задания аналитического выражения функции, достаточно указать лишь левую границу отрезка $x=a$ и шаг h . Фактически в этом случае задача сводится к задаче табулирования функции на интервале $[a; \infty)$.

Пример 1.5.3. Составить программу табулирования функции $f(x) = \frac{3.1 \ln x}{x^2 - 0.2}$ на МК-56 с начальным значением $x=1$ и шагом $h=0.1$.

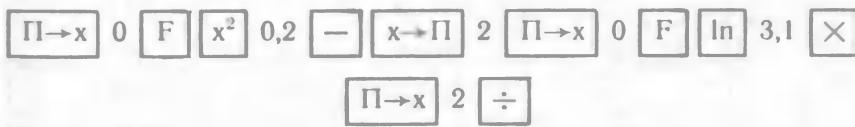
При составлении программы будем предполагать, что значения x и h записаны соответственно в адресуемые регистры R0 и RI:

x	R0
h	RI

Принимая во внимание характер заданной формулы и вычислительные возможности МК-56, будем придерживаться следующей последовательности вычисления значения $f(x)$:

- 1) вычислить значение знаменателя $x^2 - 0.2$ и запомнить его в одном из адресуемых регистров памяти (R2);
- 2) вычислить значение числителя $3.1 \ln x$;
- 3) вызвать в регистр X значение знаменателя из регистра R2 (значение числителя при этом перейдет в регистр Y);
- 4) выполнить операцию деления.

Программа вычисления одного значения $f(x)$ будет иметь вид:



(после вычисления знаменателя его пришлось временно поместить в регистр R2).

Итак, программа, по существу, готова. Достаточно завершить ее командой **C/П** (остановка и индикация значения $f(x)$)

и можно записывать в память. Остается очевидным лишь одно неудобство: после однократного прохода этой программы (и снятия очередного значения функции с индикатора) перед новым пуском придется вручную вводить в регистр R0 новое значение аргумента x . Этого неудобства, впрочем, можно избежать, если вслед за остановкой и индикацией значения функции в программе предусмотреть автоматическое вычисление нового значения аргумента и пересылку его в регистр. Теперь остановка по команде

x	$f(x)$
a	$f(a)$
$a+h$	$f(a+h)$
$a+2h$	$f(a+2h)$
$a+3h$	$f(a+3h)$
...	...

С/П будет сопровождаться индикацией нового значения аргумента. Вслед за этим остается осуществить автоматический переход к началу программы. Для этой цели в языке МК-56 есть команда безусловного перехода **БП**, вслед за которой нужно указать адрес ячейки программной памяти — 00, в которой находится первая команда программы. Полностью искомая программа представлена в таблице 21.

Таблица 21

Адрес	Команда	Выполняемое действие
00	П→x 0	Вызов x в регистр X из R0
01	F x^2	Вычисление x^2
02	0.2	Запись числа 0.2 в регистр X
03	—	Вычисление $x^2 - 0.2$
04	x→П 2	Запись $x^2 - 0.2$ в регистр R2
05	П→x 0	Вызов x в регистр X из R0
06	F ln	Вычисление $\ln x$
07	3.1	Запись числа 3.1 в регистр X
08	×	Вычисление $3.1 \ln x$
09	П→x 2	Вызов $x^2 - 0.2$ в регистр X из R2
10	÷	Вычисление $f(x) = 3.1 \ln x / (x^2 - 0.2)$
11	С/П	Остановка и индикация $f(x)$
12	П→x 0	Вызов x в регистр X из R0
13	П→x 1	Вызов h в регистр X из R1
14	+	Вычисление $x = x + h$
15	x→П 0	Запись x в регистр R0
16	С/П	Остановка и индикация x
17	БП	Безусловный переход
18	00	Адрес безусловного перехода

Таким образом, для табулирования функции $f(x) = \frac{3,1 \ln x}{x^2 - 0,2}$ с начальным значением аргумента $x = 1$ и шагом $h = 0,1$ с помощью программы, изложенной в таблице 21, необходимо:

- 1) ввести программу в память ПМК;
- 2) записать значения $x = 1$ и $k = 0,1$ в регистры R0 и R1 соответственно (командами 1 $\boxed{x \rightarrow P}$ 0 и 0,1 $\boxed{x \rightarrow P}$ 1);
- 3) пустить программу (команда $\boxed{B/O}$ $\boxed{C/P}$).

Пункт 1 выполняется в режиме $\boxed{\text{ПРГ}}$, пункты 2 и 3 — в режиме $\boxed{\text{АВТ}}$. Программа составлена так, что выходные данные индицируются на световом табло в последовательности

$$x, f(x), x+h, f(x+h), \dots .$$

При этом индикация каждого члена последовательности, начиная со второго, осуществляется в результате выполнения команды $\boxed{C/P}$. После каждой остановки и снятия показания индикатора работы программы возобновляется нажатием клавиши $\boxed{C/P}$.

Результаты работы программы табулирования для заданных в примере 1.5.3 условий частично представлены в таблице 22. Процесс табулирования прекращается по усмотрению самого вычислителя.

Для задания нового участка или шага табулирования достаточно перед пуском программы ввести новые значения в регистры R0 и R1. Однако на этом, к сожалению, и заканчивается вариативность составленной программы. Для того чтобы перейти к табулированию другой функции, всю программу придется составлять заново. Этот недостаток программы можно исправить, если при ее организации использовать следующий подход: вычисление значения функции $f(x)$ оформить в виде самостоятельного блока — подпрограммы, а в основной программе обращаться к этой подпрограмме. Тогда для перехода к табулированию другой функции достаточно будет заменять только блок вычисления $f(x)$.

Для перехода на подпрограмму в языке МК-56 имеется специальная команда, реализуемая клавишей $\boxed{\text{ПП}}$. С помощью этой команды происходит переход на подпрограмму по адресу, ука-

Таблица 22

x	$f(x) = \frac{3,1 \ln x}{x^2 - 0,2}$
1	0
1,1	2,9253618 —01
1,2	4,5580387 —01
1,3	5,4585852 —01
1,4	5,9264991 —01
1,5	6,1314234 —01
1,6	6,1737762 —01
1,7	6,1150164 —01
...	...

занному непосредственно после команды перехода, и, кроме того запоминается адрес следующей команды основной программы. По этому адресу и происходит возврат в основную программу после исполнения подпрограммы. Подпрограмма всегда завершается командой возврата **B/O**. Подпрограмму вычисления

$f(x)$ в общей программе табулирования естественно располагать в конце: это облегчит затем замену подпрограммы при переходе к табулированию новой функции.

В таблице 23 приведена программа табулирования функции с подпрограммой, реализующей вычисление значений функции $f(x)$ из примера 1.5.3. В ячейках 00—10 размещена основная программа табулирования. Подпрограмма вычисления $f(x)$ начинается с ячейки 11 — с этого адреса должна располагаться новая подпрограмма вычисления значения функции при переходе к другой задаче табулирования.

При решении некоторых задач требуется автоматическое неоднократное выполнение (или, как говорят, зацикливание) отдельных участков одной и той же программы. Язык программирования ПМК предусматривает такую возможность.

Пример 1.5.4. Требуется найти сумму всех тех членов ряда

$$S = \sum_{n=1}^{\infty} \frac{1}{\sqrt{n}}, \text{ величина которых не меньше заданного числа } \varepsilon.$$

Выпишем несколько членов заданного ряда:

$$1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} + \frac{1}{2} + \frac{1}{\sqrt{5}} + \dots$$

Если, например, $\varepsilon = 0,5$, то согласно условию задачи искомую сумму составят четыре первых члена ряда: $1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} + \frac{1}{2} = 2,7844571$.

Предположим, что значение ε находится на индикаторе (в регистре X). Сумма членов ряда будет накапливаться в регистре R1, который предварительно нужно обнулить, текущее значение номера члена n — в регистре R2, а значение очередного члена ряда в виде $1/\sqrt{n}$ будет находиться в регистре R3. Исходя из этих предположений, в общем-то, несложно составить программу, которая вычисляла бы последовательные члены заданного ряда, помещая их в регистр R3, проверяла условие $(R3) < \varepsilon$ * и, если оно не выполняется, добавляла значение этого члена к содержимому регистра R1. Если же окажется, что $(R3) < \varepsilon$, должна происходить остановка с индикацией содержимого регистра R1.

Как видно, для реализации этой идеи надо уметь определять истинность условия $(R3) < \varepsilon$ и в зависимости от исхода проверки либо прекращать счет, либо продолжать его. На языке

* $(R3)$ обозначает содержимое (число) из регистра R3.

команд это означает, что нужно проверить заданные условия и в зависимости от результата перейти к указанному для каждого из этих двух случаев адресу очередной команды.

Таблица 23

Адрес	Команда	Выполняемое действие
00	0	Вызов x в регистр X из R0
01		Переход на подпрограмму
02	11	Адрес подпрограммы
03		Остановка и индикация $f(x)$
04	0	Вызов x в регистр X из R0
05	1	Вызов h в регистр X из R1
06		Вычисление $x = x + h$
07	0	Запись x в регистр R0
08		Остановка и индикация x
09		Безусловный переход к началу
10	00	Адрес безусловного перехода
11		Вычисление x^2
12	0,2	Запись числа 0,2 в регистр X
13		Вычисление $x^2 - 0,2$
14	2	Запись $x^2 - 0,2$ в регистр R2
15	0	Вызов числа x в регистр X из R0
16		Вычисление $\ln x$
17	3,1	Запись числа 3,1 в регистр X
18		Вычисление $3,1 \ln x$
19	2	Вызов $x^2 - 0,2$ в регистр X из R2
20		Вычисление $f(x) = 3,1 \ln x / (x^2 - 0,2)$
21		Возврат к команде с адресом 03

Язык программируемых калькуляторов имеет соответствующие средства. Таким средством являются, во-первых, команды перехода по условию, которые реализуются клавишей **F** и одной из клавиш с обозначением условий: **$x \geq 0$** , **$x < 0$** , **$x = 0$** , **$x \neq 0$** . Каждая из этих команд проверяет содержимое регистра X на выполнение заданного условия. Если условие не выполняется, то следующей по программе будет исполнена команда, адрес которой указан непосредственно за командой условного перехода. Если условие выполняется, то следующей будет исполняться команда, записанная после адреса перехода. Другой командой, необходимой для организации циклических программ, является команда безусловного перехода, реализуемая клавишей **БП**.

В таблице 24 приведена программа вычисления суммы ряда по заданному числу ε , предварительно введенному в регистр X . Ход вычислений и порядок использования команд условного и безусловного перехода можно уяснить из подробных пояснений действия команд, приведенных в правом столбце.

Для пуска этой программы нужно ввести значение ε в регистр X и нажать клавиши **B/O** **C/P**. Так, при $\varepsilon = 0,1$ результатом работы программы будет 18,589606 (время счета на МК-56 более семи с половиной минут).

Мы рассмотрели далеко не все свойства программируемых МК. Эти калькуляторы обладают возможностями, умелое использование которых предоставляет вычислителю мощное средство решения задач, многие из которых до недавнего времени были под силу лишь «большим» ЭВМ. Впрочем, для полного использования возможностей ПМК, кроме усвоения их логики и языка, необходимо еще овладение специальными навыками и культурой программирования, для формирования которых требуется знакомство с общими вопросами конструирования алгоритмов и программистская практика.

Контрольные вопросы

1. В чем основное отличие программируемых микрокалькуляторов (ПМК) от МК арифметического и инженерного типа?
2. Каков порядок выполнения на ПМК двухместной арифметической операции?
3. Какие регистры образуют стековую память МК-56?
4. Каков порядок перемещения информации в стековых регистрах при нажатии клавиши **B↑?**

Таблица 24

Адрес	Команда	Выполняемое действие
00	$x \rightarrow \Pi$ 0	Запись числа π из регистра X в регистр R0
01	0	Ввод числа 0 в регистр X
02	$x \rightarrow \Pi$ 1	Запись числа 0 в регистр R1 (обнуление суммы S)
03	$x \rightarrow \Pi$ 2	Запись числа 0 в регистр R2 (в будущем в R2 будет замещаться значение n)
04	1	Ввод числа 1 в регистр X
05	$\Pi \rightarrow x$ 2	Пересылка числа 1 в регистр Y и вызов числа из регистра R2
06	+	Вычисление текущего значения n
07	$x \rightarrow \Pi$ 2	Запись n в регистр R2
08	F $\sqrt{}$	Вычисление \sqrt{n}
09	F $1/\sqrt{}$	Вычисление $1/\sqrt{n}$
10	$x \rightarrow \Pi$ 3	Запись $1/\sqrt{n}$ в регистр R3
11	$\Pi \rightarrow x$ 0	Пересылка $1/\sqrt{n}$ в регистр Y и вызов числа в регистр X
12	-	Вычисление $1/\sqrt{n} - \epsilon$
13	F $x > 0$	Проверка $1/\sqrt{n} - \epsilon > 0$ (надо ли продолжать?)
14	21	Адрес перехода при $1/\sqrt{n} - \epsilon > 0$ (к окончанию цикла)
15	$\Pi \rightarrow x$ 1	Вызов числа S из регистра R1
16	$\Pi \rightarrow x$ 3	Вызов числа $1/\sqrt{n}$ из регистра R3
17	+	Вычисление $S + 1/\sqrt{n}$
18	$x \rightarrow \Pi$ 1	Запись $S + 1/\sqrt{n}$ в регистр R1
19	БП	Безусловный переход
20	04	Адрес безусловного перехода
21	$\Pi \rightarrow x$ 1	Вызов найденной суммы из регистра R1
22	C/P	Остановка и индикация результата

5. Какие перемещения происходят в стековых регистрах ПМК при выполнении одноместной операции?

6. Что происходит с результатом предыдущих вычислений, хранящимся в регистре X , при вводе в X нового числа?

7. Как производятся запись и вызов чисел из адресуемых регистров памяти?

8. В каком режиме производится ввод программы в программную память ПМК? В каком режиме производится пуск программы?

9. Как записываются и работают команды безусловного и условного переходов?

Упражнения

1. Составить программы для однократных вычислений на МК-56 значений следующих выражений:

a) $\frac{43,26}{15,73}$;

в) $7,62^3 \cdot \ln 49,47$;

б) $\sqrt{12,5^2 - 14,8}$;

г) $13,86^{1,3} + \frac{6,84}{3,27}$.

2. Составить программу для ввода чисел a , b , c , d в регистры стека X , Y , Z , T соответственно.

3. Написать программу для ручных вычислений на ПМК общего сопротивления цепи R при параллельном включении сопротивлений R_1 , R_2 , R_3 и R_4 по формуле

$$R = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4}}.$$

4. Составить программу для автоматического вычисления значений площади круга $S = \pi R^2$ по заданным значениям радиуса R , вводимым в регистр индикации X .

5. Дано уравнение $0,5 \ln(x+1) - \frac{1}{x} = 0$. Методом табулирования функции $f(x) = 0,5 \ln(x+1) - \frac{1}{x}$ на МК-56 локализовать отрезок, содержащий корень уравнения.

Указание. Составить программу табулирования функции $f(x)$ на отрезке $[1; 2]$ с шагом 0,1. Непрерывная функция имеет на отрезке корень, если она меняет знак на его концах. Если функция к тому же монотонна на этом отрезке, корень единственный.

6. Составить циклическую программу вычисления суммы всех членов ряда $\sum_{n=1}^{\infty} \frac{n}{n+1}$, величина которых не меньше заданного числа E .

7. Составить циклическую программу вычисления наибольшего общего делителя (НОД) двух целых положительных чисел a и b .

Указание. Если натуральные числа m и n таковы, что $m > n$, то НОД чисел m , n такой же, как и чисел $m - n$, n .

1.6. Вычисления на персональных микроЭВМ

Если в распоряжении вычислителя имеется персональная электронно-вычислительная машина (ПЭВМ), то достаточно освоить очень простые правила, чтобы научиться использовать ее для выполнения вычислений по формулам. Составление сложных вычислительных программ, содержащих ветвления и циклы, основывается на специальных языках программирования. С ними мы познакомимся позже (см. главу 4), а сейчас рассмотрим лишь самые простейшие приемы использования персональной ЭВМ для вычислений (или, как иногда говорят, приемы использования ПЭВМ в режиме *непосредственного счета*).

Персональная микроЭВМ размещается на столе, а минимальная ее конфигурация, достаточная для выполнения расчетов, содержит клавиатуру (1), устройство для выполнения операций — процессор (2), а также устройство визуального отображения информации — дисплей (3) (рис. 16). На рисунке 16 изображен минимальный комплект отечественного диалогового вычислительного комплекса (ДВК), именуемый ДВК-1. В ДВК-1 процессор выполнен в виде отдельного устройства. Во многих моделях ПЭВМ процессор конструктивно соединен с клавиатурой, что делает весь комплекс более компактным.



Рис. 16

Команды процессору микроЭВМ передаются с клавиатуры и поэтому могут состоять лишь из тех символов, которые имеются на клавиатуре. Клавиатура у разных микроЭВМ может иметь свои особенности, но общий принцип их организации примерно одинаков.

На рисунке 17 изображена достаточно стандартная клавиатура персональной микроЭВМ учебного назначения «Корвет».

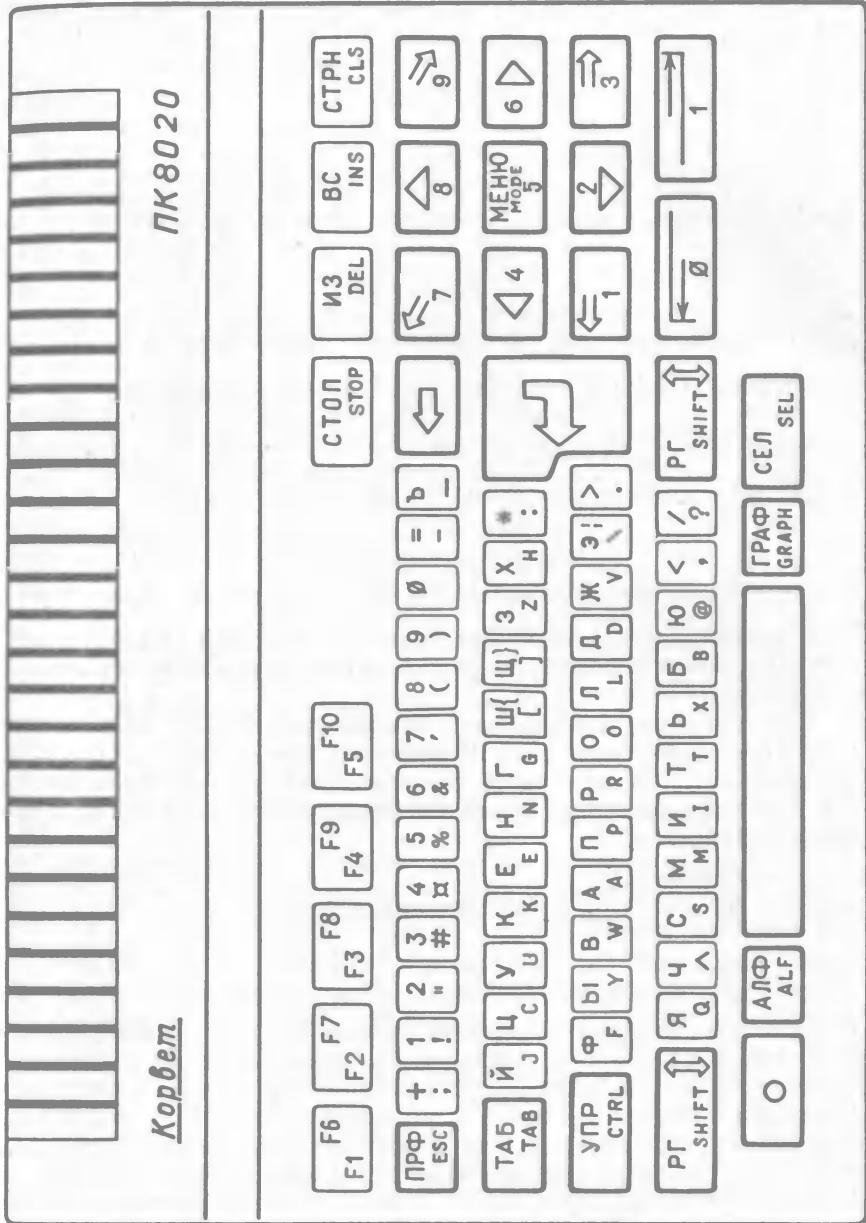
На клавиатуре много специальных клавиш, назначение которых нужно знать при основательном знакомстве с различными возможностями микроЭВМ. Многое, однако, понятно уже потому, что клавиатура микроЭВМ в значительной своей части совпадает с клавиатурой обычной пишущей машинки. Здесь есть клавиши для набора цифр от 0 до 9, а также клавиши для набора букв — и русских, и латинских. Как и в пишущей машинке, клавиши могут работать в режиме совмещения, регулируемом регистром

(на рисунке 17 клавиша **РГ/SHIFT**) : на нижнем регистре будут вводиться символы, изображенные в нижней части клавиши, на верхнем — в верхней части. При вводе русских или латинских букв, кроме того, используется переключатель алфавитов **АЛФ/ALF** ; регистр **РГ/SHIFT** в данном случае переводит или на строчные или на прописные буквы алфавита. Как и на клавиатуре пишущей машинки, имеются клавиши «пробел» (длинная клавиша в нижней части клавиатуры) и клавиша перехода на новую строку **↓**.

Полезно обратить внимание, что расположение букв русского алфавита на клавиатуре микроЭВМ стандартно и совпадает с их расположением на клавиатуре пишущих машинок. Это означает, что навык работы с русскими текстами, сформированный за клавиатурой пишущей машинки, автоматически переносится на микроЭВМ.

После включения микроЭВМ на экране дисплея появляется специальный символ — курсор, который показывает, что ЭВМ готова к работе и ждет указаний. Курсор всегда сопровождает протокол общения человека и ЭВМ, изображаемый на экране. Курсор у разных моделей микроЭВМ может выглядеть по-разному; распространена форма курсора в виде светящегося квадрата (рис. 18, а). Если теперь по очереди нажимать цифровые и буквенные клавиши, то на экране будут появляться соответствующие символы. При этом курсор будет перемещаться по строке вправо и всегда указывать место, на котором появится очередной вводимый символ.

Для выполнения вычислений на экране дисплея формируется, а затем исполняется специальная команда. Эта команда начинается обычно с ключевого слова **PRINT** (печать, вывод), которое может быть образовано путем последовательного нажатия клавиш с соответствующими латинскими буквами. Поскольку это ключе-



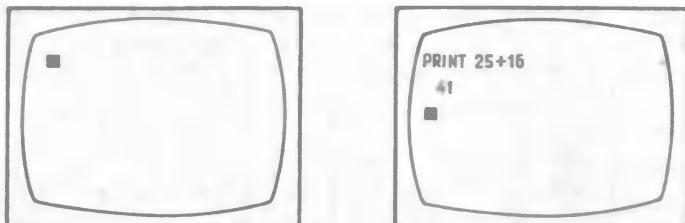


Рис. 18

a)

б)

вое слово используется в общении с ЭВМ часто, в некоторых случаях предусмотрена возможность автоматического вызова его на экран с помощью только одной клавиши (например, **F1**) из числа специальных клавиш **F1** — **F10**. Вслед за словом PRINT набирается вычисляемое выражение, например:

PRINT 25 + 16

В конце нажимается клавиша перехода на новую строку **↓**. ЭВМ высвечивает на экране результат вычислений (в следующей строке) и снова переходит в режим готовности для продолжения работы (рис. 18, б).

При составлении выражений для команды PRINT необходимо строго придерживаться установленных правил их записи. Правила эти просты. При записи чисел для отделения целой части от дробной используется точка. Знаки арифметических операций изображаются символами:

- +** — сложение;
- — вычитание и изменение знака;
- *** — умножение;
- /** — деление;
- ^** — возведение в степень.

В бескобочных записях порядок действий определяется следующим образом: сначала выполняются все возведения в степень, потом умножение и деление, последними — сложение и вычитание. Операции одного приоритета выполняются слева направо. В любом случае порядок действий может регулироваться круглыми скобками: (и). Для того чтобы не путать букву *O* с нулем, нуль в алфавите микроЭВМ обычно перечеркнут: \emptyset . Нулевую целую часть при записи чисел, как правило, разрешено не писать. Если вычисляемое выражение не вместилось на одной строке экрана, нужно, не смущаясь, продолжать набор символов дальше: микроЭВМ сама в нужный момент перейдет на новую строку. Однако повторять при переносе набираемые символы не следует.

П р и м е р 1.6.1. Составить команду для вычисления на микроЭВМ значения выражения

$$\frac{8.42 \cdot 39.1}{196.5 - 47.7} + 4.9^2.$$

Решение:

PRINT $8.42 * 39.1 / (196.5 - 47.7) + 4.9^2$.

После нажатия на клавишу перехода на новую строку на экране высвечивается ответ:

107.30271379812

Точность, с которой выдается результат, зависит от типа микро-ЭВМ.

В состав вычисляемых выражений могут включаться элементарные функции. Перечень функций определяется для каждого типа ЭВМ. Аргумент функции обычно требуется заключать в круглые скобки. Ниже приведен некоторый (неполный) перечень стандартных функций с обозначениями, принятymi для ряда микроЭВМ:

ABS (X)	Абсолютная величина x
SIN (X)	Синус x
COS (X)	Косинус x
TAN (X)	Тангенс x
ATN (X)	Арктангенс x
SQR (X)	Корень квадратный из x
LOG (X)	Логарифм натуральный x
EXP (X)	Экспонента (e^x)
INT (X)	Целая часть x
RND (X)	Случайное число из интервала (0; 1)

Если вычисляемое выражение содержит функции, отсутствующие в перечне стандартных функций микроЭВМ, его нужно преобразовать к виду, содержащему только разрешенные функции.

П р и м е р 1.6.2. Составить команду для вычисления на микро-ЭВМ значения выражения, используя приведенный выше перечень стандартных функций:

$$\sqrt{8.6 \ln 14.3} + \frac{\operatorname{arctg} 8.94}{\sin 0.8}.$$

Решение:

PRINT $\operatorname{SQR}(8.6 * \operatorname{LOG}(14.3)) + \operatorname{ATN}(8.94) / \operatorname{SIN}(.8)$

После нажатия клавиши J будет получен ответ: 6.8175384497837

Перед нажатием клавиши перехода на новую строку набранный текст вычисляемого выражения можно редактировать. Для этого нужно усвоить назначение еще нескольких клавиш.

В нижнем правом углу клавиатуры (рис. 17) находится группа из четырех клавиш, вызывающих направленное перемещение курсора в пределах рабочего поля экрана в направлениях, указанных стрелками на самих клавишиах: влево, вверх, вниз, вправо. С помощью этих клавиш курсор может быть установлен на любой уже набранный символ.

Уничтожение символа, на котором находится курсор, осуществляется нажатием клавиши **DEL** (*delete* — вычеркивать).

При этом вся правая часть строки сдвигается на один символ влево и заполняет образовавшийся пробел. Если требуется сделать вставку между символом, расположенным слева, и символом, на котором находится курсор, нажимается клавиша **BC/INS** (*inset* — вставлять). Все набираемые (при нажатой клавише **BC/INS**) символы будут помещаться между указанными выше, при этом правая часть набранной строки будет автоматически сдвигаться вправо. Отредактированное выражение может быть вычислено обычным способом — нажатием клавиши перехода на новую строку. Для гашения всей информации на экране используется клавиша **СТРН/CLS**. Правила редактирования текста на экране несложны, но с ними нужно знакомиться по описанию конкретной микроЭВМ.

Контрольные вопросы

1. Какие основные устройства входят в состав персональной микроЭВМ?
2. Как составляется команда для вычислений на микроЭВМ, какие основные клавиши при этом используются?
3. Каковы правила составления выражений, содержащих арифметические операции и функции?
4. Как с помощью клавиатуры микроЭВМ можно редактировать текст, содержащийся на экране?

Упражнения

Записать приведенные формулы на языке микроЭВМ (при необходимости преобразовать формулы к виду, содержащему только функции из списка стандартных функций микроЭВМ, приведенного в тексте):

a) $\frac{31,6 \cdot 20,5 + 8,1^{2,3}}{79,8 - 42,7};$

b) $\frac{8,2 + \cos(x^2 - 1)}{\sqrt{x^2 - 2x - 1}};$

6) $\frac{\lg x - \operatorname{ctg} x (x - \ln y)}{|x^2 + ax + b|};$

г) $e^{y^2 + \operatorname{arccos}(|ax^2 - b| - 3,6)}.$

Анализ точности вычислений

Рассматривая в предыдущей главе технику вычислений, мы не затрагивали вопросов точности. Понятно, однако, что большое количество цифр, высвечиваемых при выполнении расчетов на индикаторе МК или дисплее микроЭВМ, не делает менее актуальной задачу оценки точности результатов. Это относится и к учету точности исходных данных, и к анализу накопления вычислительных ошибок, т. е. затрагивает весь круг вопросов, традиционно относимых к методам приближенных вычислений.

2.1. Откуда берутся ошибки в вычислениях по формуле

Наибольшее влияние на точность расчетов по формуле оказывает точность исходных данных. Дело в том, что используемые в вычислениях числовые значения носят в большинстве случаев приближенный характер, так как возникают на практике в результате измерений, взвешиваний или других способов инструментального получения числовых значений каких-либо реальных величин. В тех исключительных случаях, когда требуется особо высокая точность исходных значений, ее достижение связано обычно с немалыми затратами. В обычных случаях точность исходных значений не очень высока. Часто она существенно ниже тех потенциальных возможностей точности, которыми обладают при обработке числовых значений калькуляторы или микроЭВМ. Именно здесь и кроется опасность возникновения дилетантских заблуждений о «могуществе» современной счетной техники. Нас всегда поражает, например, с какой легкостью и к тому же практически мгновенно калькуляторы выдают числовые результаты, выражаемые внушительными количествами цифр. Например, используя простейший калькулятор МК-57, мы можем легко установить, что $5,2 : 1,7 = 3,0588235$. Можно ли доверять каждой цифре этого результата? Грамотный вычислитель не сделает здесь поспешного вывода, а подойдет к оценке точности полученного значения с учетом точности исходных данных.

Другие причины, влияющие на точность вычислений,— это технические возможности вычислительного прибора. Здесь прежде всего приходится упомянуть одно очевидное препятствие точности, носящее объективный характер — конечно́сть разрядной сетки,

используемой для представления чисел. Это обстоятельство приводит к тому, что выходящие за пределы сетки разряды результата остаются в ходе вычислений неведомыми для вычислителя. В большинстве случаев эти разряды попросту отбрасываются. Допустим, что использованные в приведенном выше примере исходные значения 5,2 и 1,7 — точные числа. Тогда их частное 3,0588235 содержит ошибку, единственной причиной которой служит сам калькулятор, поскольку его разрядная сетка не вместила всех цифр результата. В том, что ошибка допущена, легко убедиться, проверив деление умножением:

$$3,0588235 \times 1,7 = 5,199999.$$

Положение усугубляется еще и тем, что вычислитель в данном случае никак не может судить об истинной величине допущенной погрешности. Из сложившейся ситуации можно вывести только то, что ошибка не превышает единицы самого младшего из изображенных на индикаторе разрядов результата.

В отдельных случаях, как следует из инструкций по эксплуатации, МК выдают результаты с точностью, не использующей даже всех разрядов индикатора. Так, например, инженерный калькулятор Б3-18 при 8-разрядном индикаторе вычисляет значения элементарных функций лишь с шестью знаками после запятой, т. е. с точностью до единицы в шестом младшем разряде.

К числу причин, искажающих результат вычислений, следует отнести также всевозможные промахи, допускаемые иногда в процессе счета: ошибочный ввод данных, смещение клавиш операций, использование неверной программы вычислений и т. п. Сюда относятся также ошибки, возникающие из-за сбоев в самом вычислительном приборе. Средством борьбы против промахов разного рода служит или предварительная грубая прикидка ожидаемого результата, или способ двойных вычислений, а также специально организуемые системы текущего контроля, связанные с существом решаемой задачи.

Для исчерпывающего представления о точности результата вычислений следует учитывать влияние всех возможных ошибок. Подобный анализ нельзя провести без использования начальных понятий приближенных вычислений.

2.2. Первоначальные понятия приближенных вычислений

Если X — истинное значение некоторой величины, а x — ее известное приближение, то абсолютная величина ошибки приближения x определяется так:

$$e_x = |X - x|. \quad (2.1)$$

Величина e_x , называемая *абсолютной погрешностью* приближенного значения x , в большинстве случаев остается для вычислителя неизвестной, так как для ее вычисления нужно точное

значение X . Вместе с тем на практике обычно удается установить верхнюю границу абсолютной погрешности, т. е. такое (по возможности наименьшее) число Δx , для которого справедливо

$$|X - x| \leq \Delta x. \quad (2.2)$$

Число Δx в этом случае называют *границей абсолютной погрешности* (или *пределной абсолютной погрешностью*) приближения x .

Пример 2.2.1. Возьмем число $\pi = 3,14159265358\dots$. Если же вызвать π на индикатор 8-разрядного микрокалькулятора, получим приближение этого числа:

$$\pi' = 3,1415926.$$

Попытаемся выразить абсолютную погрешность значения π' :

$$e_{\pi'} = |\pi - \pi'| = 0,00000005358\dots.$$

Получили бесконечную дробь, непригодную для практических расчетов. Очевидно, что

$$e_{\pi'} < 0,00000006 = \Delta \pi',$$

следовательно, число $\Delta \pi' = 0,6 \cdot 10^{-7}$ можно считать границей абсолютной погрешности приближения, используемого микрокалькулятором вместо числа π .

Неравенство (2.2) позволяет установить приближения к точному значению X по недостатку и по избытку:

$$x - \Delta x \leq X \leq x + \Delta x, \quad (2.3)$$

которые называют также соответственно нижней границей (НГ) и верхней границей (ВГ) приближения x :

$$NG_x = x - \Delta x, \quad VG_x = x + \Delta x. \quad (2.4)$$

Качество приближенных значений измеряется с помощью *относительной погрешности*, которая определяется как отношение ошибки e_x к модулю значения X (когда оно неизвестно — к модулю приближенного числа называется отношение *пределной абсолютной погрешности* к *абсолютному значению приближения x*):

$$\delta x = \frac{\Delta x}{|x|}. \quad (2.5)$$

Относительную погрешность выражают обычно в процентах.

Пример 2.2.2. Вычислим границу относительной погрешности приближения к числу e , используемого калькулятором Б3-37.

Значение e , записанное 8 значащими цифрами, равно 2,7182818... . На Б3-37, вычислив e^x при $x = 1$, получим приближение, выраженное 7 значащими цифрами: $e' = 2,718281$. Это означает, что граница абсолютной погрешности $\Delta e' = 0,9 \cdot 10^{-6}$.

Тогда

$$\delta e' = \frac{0,9 \cdot 10^{-6}}{2,718281} < 0,4 \cdot 10^{-6},$$

т. е. можно принять $\delta_e = 0,00004\%$. Это чрезвычайно высокая точность, если учесть, что для ординарных технических расчетов считается приемлемым уровень точности от 0,1 до 5%.

Цифра числа называется *верной*, если абсолютная погрешность числа не превосходит единицы разряда, в котором стоит эта цифра. *Значащими* цифрами в записи числа называются все цифры в его десятичном изображении, отличные от нуля, и нули, если они расположены между значащими цифрами или стоят в конце выражения верных знаков. Запись приближенного числа только верными знаками называют его *правильной* записью.

Пример 2.2.3. Приближенное число $x = 308,7060$ дано в правильной записи. Отсюда следует, что в числе 7 значащих цифр, а его абсолютная погрешность не превышает единицы 4-го разряда после запятой, т. е. за границу абсолютной погрешности можно принять $\Delta_x = 0,0001$.

Исключение из последующих вычислений неверных цифр производится путем *округления* приближенных чисел. Округление — это замена числа его значением с меньшим количеством значащих цифр. При округлении возникает погрешность, называемая *погрешностью округления*. Пусть x — данное число, а x_1 — результат его округления. Погрешность округления определяется как модуль разности прежнего и нового значений числа:

$$\Delta_{\text{окр}} = |x - x_1|. \quad (2.6)$$

В отдельных случаях вместо $\Delta_{\text{окр}}$ приходится использовать ее верхнюю оценку.

Пример 2.2.4. Выполним на МК действие $5 \div 6$. На индикаторе высветится число 0,8333333. Произошло автоматическое округление бесконечной десятичной дроби 0,8(3) до количества разрядов, вмещающихся в регистре МК. При этом можно принять $\Delta_{\text{окр}} = 0,4 \cdot 10^{-7}$.

Рассмотренный пример «принудительного» округления называется *округлением методом отбрасывания*. Очевидно, что сам по себе метод отбрасывания оставляет все сохраняемые цифры округленного числа верными, так как то, что отбрасывается, не может быть больше единицы последнего сохраняемого разряда.

Если вычисления ведутся в пределах разрядной сетки МК, т. е. с точностью, меньшей чем машинная точность микрокалькулятора, целесообразнее пользоваться способом *симметричного округления*, который приводит к меньшей величине ошибки округления, чем способ отбрасывания. Симметричное округление выполняется по правилам:

1. Если первая слева из отбрасываемых цифр меньше 5, то сохраняемые десятичные знаки остаются без изменения.
2. Если первая слева из отбрасываемых цифр больше или равна 5, то последняя сохраняемая цифра увеличивается на единицу.

Из правил симметричного округления следует, что его погрешность не превышает половины единицы последнего сохраняемого разряда. Это обстоятельство позволяет вести счет с точностью большей, чем единица последнего сохраняемого разряда. По этой причине наряду с понятием «верная цифра», соответствующим методике округления путем отбрасывания, используется понятие «цифра, верная в строгом смысле», применяемое в вычислениях с симметричным округлением.

Цифра числа называется *верной в строгом смысле*, если абсолютная погрешность этого числа не превосходит половины единицы разряда, в котором стоит эта цифра.

Пример 2.2.5. Вычислим на МК $x = \sqrt{561}$. Получим $x = 23,685438$. Округлим результат до десятых методом симметричного округления: $x_1 = 23,7$. Очевидно, что можно принять $\Delta x_1 = -0,02$. Оказалось, таким образом, что все цифры округленного числа x_1 верны в строгом смысле.

Абсолютная погрешность числа x_1 , получаемого в результате округления приближенного значения x , складывается из абсолютной погрешности первоначального числа x и погрешности округления. Действительно, из неравенства

$$|x - x_1| \leq |x - x| + |x - x_1| \leq \Delta x + \Delta_{окр}$$

следует, что если в результате округления приближенного числа x получено значение x_1 , то границей абсолютной погрешности числа x_1 можно считать сумму границы абсолютной погрешности числа x и погрешности округления.

Пример 2.2.6. Пусть в приближенном значении $a = 16,395$ все цифры верны в широком смысле. Округлим a до сотых: $a_1 = 16,40$. Погрешность округления $\Delta_{окр} = 0,005$. Для нахождения полной погрешности исходного значения a , которая находится из условия, что все цифры в записи a верны, $\Delta a = 0,001$. Таким образом, $\Delta a_1 = \Delta a + \Delta_{окр} = 0,001 + 0,005 = 0,006$. Отсюда следует, между прочим, что в значении $a_1 = 16,40$ цифра 0 не верна в строгом смысле.

Упражнения

1. В результате измерения длины бруска метром с сантиметровым делением установлено, что значение длины находится между делениями 96 и 97 см. Указать границы абсолютной и относительной погрешности полученного значения.

2. У приближенного числа 24,010 все цифры верны в строгом смысле. Указать границы его абсолютной и относительной погрешности.

3. Округлить приближенное число 8,495, у которого все цифры верны в строгом смысле, до сотых и определить в округленном значении количество цифр, верных в строгом смысле.

2.3. Определение количества верных цифр по относительной погрешности приближенного числа

Количество верных значащих цифр в приближенном числе и величина относительной погрешности этого числа взаимосвязаны. Эта связь со всей очевидностью вытекает уже из того, что по величине δx , учитывая формулу (2.5), можно вычислять абсолютную погрешность:

$$\Delta x = |x| \cdot \delta x, \quad (2.7)$$

величина которой, как следует из определения верных значащих цифр, явно влияет на их количество в приближенном числе. Впрочем, иногда на практике удобнее пользоваться правилом, устанавливающим взаимосвязь количества верных цифр непосредственно с величиной относительной погрешности. Рассмотрим этот вопрос применительно к приближенному числу x , записанному в «плавающем» виде (см. п. 1.2):

$$x = M \cdot 10^p, \quad (2.8)$$

в предположении, что мантисса M удовлетворяет условию $0,1 \leq M < 1$ (число x в этом случае называют нормализованным). У нормализованного числа первый разряд мантиссы после запятой всегда отличен от нуля. Очевидно, что хранение в машине чисел именно в нормализованном виде позволяет сохранить больше значащих цифр мантиссы. Заметим также, что для нормализованного числа x вида (2.8) справедливо

$$|x| < 10^p. \quad (2.9)$$

Итак, имеется приближенное число x и его относительная погрешность δx . Нужно установить количество верных в строгом смысле значащих цифр в числе x .

Для каждого известного значения x можно подобрать такое наибольшее натуральное n , чтобы имело место

$$\delta x \leq 10^{-n}. \quad (2.10)$$

Тогда

$$\Delta x \leq |x| \cdot 10^{-n} < 10^p \cdot 10^{-n} = 10^{p-n} < \frac{1}{2} 10^{p-(n-1)},$$

т. е.

$$\Delta x < \frac{1}{2} 10^{p-(n-1)}. \quad (2.11)$$

Сопоставляя теперь (2.7) и (2.11) и используя определение цифры, верной в строгом смысле, можно сделать вывод, что в мантиссе приближенного числа x верны по крайней мере $n-1$ цифр после запятой (а поскольку x нормализовано, то все эти цифры значащие). Таким образом, для того чтобы по заданной величине относительной погрешности δx найти количество верных значащих цифр в числе x , достаточно подобрать наибольшее натуральное n так, чтобы имело место неравенство $\delta x \leq 10^{-n}$, а потом полученное значение уменьшить на единицу.

При мер 2.2.7. Пусть $x = 897$, $\delta x = 0,009$. Очевидно, что $0,009 \leq 10^{-2}$. Это означает, что число x имеет по крайней мере одну верную в строгом смысле цифру (это первая слева цифра 8). Полученный результат легко подтвердить, используя определение цифры, верной в строгом смысле. На МК вычислим $\Delta x = 897 \cdot 0,009 = 8,073$, откуда следует, что в числе 897 цифра 8 действительно верна в строгом смысле.

Полученное правило в отдельных случаях проявляет завышенную «осторожность» — при выполнении условия (2.10) в числе x могут оказаться верными все n цифр. Зависит это от величины первых значащих цифр числа x .

При мер 2.2.8. Пусть $x = 286$, $\delta x = 0,007 \leq 10^{-2}$ (т. е. $n = 2$). Согласно правилу в числе 286 лишь одна верная в строгом смысле цифра ($n - 1 = 1$). Вычислим $\Delta x = 286 \times 0,007 < 2,1$. Как показывает величина предельной абсолютной погрешности, в числе верны в строгом смысле две цифры: 2 и 8.

В отдельных случаях, когда первая значащая цифра в относительной погрешности δx меньше 5, вместо условия (2.10) удается установить более сильное условие:

$$\delta x \leq \frac{1}{2} \cdot 10^{-n}, \quad (2.12)$$

Легко показать, что в этом случае число x имеет по крайней мере n верных в строгом смысле цифр. Действительно, с учетом (2.7), (2.9) и (2.12) имеем:

$$\Delta x \leq \frac{1}{2} |x| \cdot 10^{-n} < \frac{1}{2} 10^n 10^{-n} = \frac{1}{2} 10^{n-n}, \quad \text{т. е.}$$

$$\Delta x < \frac{1}{2} 10^{n-n},$$

а это означает, чтоmantисса M нормализованного числа x (см. 2.8) имеет по меньшей мере n верных в строгом смысле цифр.

При мер 2.2.9. Пусть $x = 94,27$, $\delta x = 0,0002$. Имеем $0,0002 < 0,0005 = \frac{1}{2} \cdot 10^{-3}$, т. е. в числе x верны в строгом смысле 3 цифры. Действительно, в данном случае $\Delta x = 94,27 \cdot 0,0002 < 0,019$, что подтверждает полученный результат.

Упражнения

1. По заданным значениям приближенных чисел и их относительных погрешностей установить количество цифр, верных в строгом смысле:

- а) $x = 57,39$, $\delta x = 0,08\%$;
- б) $y = 2,978$, $\delta y = 0,03\%$;
- в) $z = 408,32$, $\delta z = 0,009\%$.

Округлить значения x , y и z до верных цифр с учетом одной запасной.

2. Со сколькими верными в строгом смысле десятичными знаками после запятой нужно взять указанные значения, чтобы относительная погрешность не превышала 0,1%:

а) $\frac{1}{17,2}$; б) $\sqrt{0,039}$; в) $\sin 1,34$?

2.4. Подсчет погрешностей арифметических действий с приближенными данными

Основная задача, возникающая в ходе приближенных вычислений и имеющая большое практическое значение, состоит в следующем: как влияют на точность конечных результатов вычислений по формулам погрешности исходных данных? Рассмотрим этот вопрос последовательно для четырех основных арифметических действий и элементарных функций.

Сложение и вычитание

Пусть $S = X + Y$ — сумма точных чисел, среди которых могут быть как положительные, так и отрицательные, а $s = x + y$ — сумма их приближений. Составим разность:

$$S - s = (X - x) + (Y - y),$$

или, переходя к модулям:

$$|S - s| \leq |X - x| + |Y - y|, \quad \text{т. е.} \quad e_s \leq e_x + e_y,$$

или тем более $e_s \leq \Delta x + \Delta y$. Отсюда следует, что можно принять

$$\Delta s = \Delta x + \Delta y, \quad (2.13)$$

т. е. границей абсолютной погрешности алгебраической суммы можно считать сумму границ абсолютных погрешностей слагаемых.

Пример 2.4.1. Даны приближенные значения $x = 854,2$ и $y = 17,2138$, у которых все цифры являются верными в широком смысле. Найдем на МК их сумму:

$$s = 854,2 + 17,2138 = 871,4138.$$

Для оценки точности результата вычислим сумму погрешностей слагаемых: $10^{-1} + 10^{-4} = 0,1001 < 0,11 = \Delta s$. Величина ошибки показывает, что в результате уже первый знак после запятой является сомнительным. Стоило ли терять время на учет в вычислениях всех знаков после запятой у второго слагаемого?

Из рассмотренного примера следует поучительный вывод: при вычислении сумм и разностей чисел с сильно различающимися абсолютными ошибками с целью экономии времени целесообразно «уравнивать» точность исходных данных путем округления более точных до точности менее точных (с одной-двумя запасными цифрами). Так, в рассмотренном выше примере имело смысл перед выполнением действия сложения округлить y до сотых: 17,21.

Руководствуясь только что указанным правилом, следует иметь в виду, что при последовательном вычитании и сложении нескольких чисел выгоднее производить действия над числами в порядке возрастания их абсолютных величин.

Относительные погрешности суммы и разности можно вычислять через абсолютные, пользуясь формулой (2.5), но можно использовать и специальные формулы. Получим их:

$$\begin{aligned}\delta(x+y) &= \frac{\Delta x + \Delta y}{|x+y|} = \frac{|x|}{|x+y|} \cdot \frac{\Delta x}{|x|} + \frac{|y|}{|x+y|} \cdot \frac{\Delta y}{|y|} = \\ &= \frac{|x|}{|x+y|} \cdot \delta x + \frac{|y|}{|x+y|} \cdot \delta y;\end{aligned}\quad (2.14)$$

$$\begin{aligned}\delta(x-y) &= \frac{\Delta x + \Delta y}{|x-y|} = \frac{|x|}{|x-y|} \cdot \frac{\Delta x}{|x|} + \frac{|y|}{|x-y|} \cdot \frac{\Delta y}{|y|} = \\ &= \frac{|x|}{|x-y|} \cdot \delta x + \frac{|y|}{|x-y|} \cdot \delta y.\end{aligned}\quad (2.15)$$

Формулы (2.14) и (2.15) позволяют сделать полезные для практического использования выводы.

Пусть слагаемые x и y одного знака, а $\delta = \max(\delta x, \delta y)$. Тогда из (2.14) следует:

$$\delta(x+y) \leq \frac{\delta(|x|+|y|)}{|x+y|} = \delta, \quad \text{т. е.} \quad \delta(x+y) \leq \delta.$$

Это означает, что если приближенные слагаемые имеют одинаковый знак, то граница относительной погрешности их суммы не превышает наибольшей из границ относительных погрешностей слагаемых.

Как видно из формулы (2.15), при вычитании близких чисел может произойти большая потеря точности. Действительно, когда вычитаемые числа почти одинаковы, то даже при условии, что их собственные ошибки малы, относительная ошибка может оказаться большой.

Пример 2.4.2. Найдем разность чисел $x=274,598$ и $y=274,581$, у которых все цифры верны в строгом смысле. Имеем:

$$x - y = 274,598 - 274,581 = 0,017.$$

Граница абсолютной погрешности разности:

$$\Delta(x-y) = 0,0005 + 0,0005 = 0,001,$$

поэтому в числе 0,017 из двух значащих цифр в строгом смысле верна лишь одна. Сравним границы относительных погрешностей результата и исходных данных:

$$\delta x = \frac{0,0005}{274,598} \approx 0,000002; \quad \delta y = \frac{0,0005}{274,581} \approx 0,000002;$$

$$\delta(x-y) = \frac{0,001}{0,017} \approx 0,0588.$$

Таким образом, в данном случае граница относительной погрешности разности оказалась почти в 30 тысяч раз больше границы относительной погрешности исходных данных. Это означает, что в приближенных вычислениях нужно исключать вычитание близких по величине значений (например, путем преобразования вычисляемых выражений).

Умножение и деление

Пусть $p = xy$ — произведение двух приближенных чисел, а $q = x/y$ — их частное. Знаки чисел x и y не влияют на величину ошибки, поэтому для простоты примем $x, y > 0$. Имеем:

$$\ln p = \ln x + \ln y, \quad \ln q = \ln x - \ln y.$$

Принимая во внимание (2.13), а также используя приближенную формулу $\Delta \ln z \approx d \ln z = \frac{\Delta z}{z}$, получим:

$$\Delta \ln p = \Delta \ln q = \Delta \ln x + \Delta \ln y, \quad \text{т. е.}$$

$$\frac{\Delta p}{p} = \frac{\Delta q}{q} = \frac{\Delta x}{x} + \frac{\Delta y}{y}, \quad (2.16)$$

откуда следует:

$$\delta(xy) = \delta(x/y) = \delta x + \delta y, \quad (2.17)$$

т. е. границей относительной погрешности произведения (частного) можно считать сумму границ относительных погрешностей сомножителей (делимого и делителя).

Из формулы (2.16) легко получаются формулы для вычисления границ абсолютных погрешностей произведения и частного:

$$\Delta(xy) = x \cdot \Delta y + y \cdot \Delta x; \quad (2.18)$$

$$\Delta(x/y) = \frac{x \cdot \Delta y + y \cdot \Delta x}{y^2}. \quad (2.19)$$

Пример 2.4.3. Числа 74,9 и 128 заданы верными цифрами. Найдем на МК их частное: $q = 0,5851562$. Для определения числа верных знаков результата вычислим:

$$\Delta q = \frac{74,9 \cdot 1 + 128 \cdot 0,1}{128} = \frac{87,7}{16384} = 0,0053527.$$

Частное q имеет два верных знака. Округляя с одной запасной, получим $q = 0,535$.

Как следует из формулы (2.17), относительная погрешность произведения и частного не может быть меньше, чем относительная погрешность наименее точного из компонентов действий. По аналогии с тем, как при нахождении алгебраической суммы не имеет смысла сохранять в более точных слагаемых излишнее количество десятичных знаков, при умножении и делении приближенных чисел нет смысла сохранять в менее точных данных излишнее количество значащих цифр.

Для удобства все формулы для вычисления погрешностей арифметических действий сведены в общую таблицу (см. табл. 25). Знак \odot в таблице обозначает одну из операций $+$, $-$, \times , $/$.

Таблица 25

$x \odot y$	$\Delta(x \odot y)$	$\delta(x \odot y)$
$x+y$	$\Delta x + \Delta y$	$\frac{ x }{ x+y } \delta x + \frac{ y }{ x+y } \delta y$
$x-y$	$\Delta x - \Delta y$	$\frac{ x }{ x-y } \delta x + \frac{ y }{ x-y } \delta y$
$x \cdot y$	$x \cdot \Delta y + y \cdot \Delta x$	$\delta x + \delta y$
x/y	$\frac{x \cdot \Delta y + y \cdot \Delta x}{y^2}$	$\delta x + \delta y$

Упражнения

Произвести указанные действия и определить абсолютные и относительные погрешности результатов (исходные числа заданы верными в строгом смысле цифрами):

- а) $87,24 - 19,23$; в) $4,694 - 4,692$; д) $42,84 \cdot 0,8$;
 б) $43,586 + 47,8$; г) $846,5 \cdot 121,8$; е) $48,3 : 0,82$.

2.5. Оценка погрешностей значений функций

Вычисления по формулам нередко предполагают нахождение значений элементарных функций. Используя для расчетов калькулятор или персональную ЭВМ, вычислитель в этом случае преобразует вычисляемое выражение так, чтобы оно содержало только те функции, которые имеются на клавиатуре калькулятора или в списке стандартных функций ЭВМ. При этом становится актуальным вопрос о методах подсчета погрешностей значений элементарных функций.

Пусть функция $f(x)$ дифференцируема в некоторой окрестности приближенного значения аргумента x , а e_x — абсолютная ошибка значения аргумента. Тогда абсолютная ошибка значения функции $e_f = |f(x+e_x) - f(x)|$. Поскольку на практике ошибка e_x обычно существенно мала по сравнению со значением x , воспользуемся приближенным равенством $e_f \approx |df| = |f'(x)| \cdot e_x$. Заменим e_x на Δx :

$$e_f \leq |f'(x)| \cdot \Delta x.$$

Это означает, что можно принять

$$\Delta f = |f'(x)| \cdot \Delta x. \quad (2.20)$$

Равенство (2.20) позволяет получить целую серию формул для оценки границ абсолютных погрешностей значений элементарных функций. Пусть, например, $f(x) = \sqrt{x}$. Тогда $\Delta(\sqrt{x}) = |(\sqrt{x})'| \Delta x = \frac{\Delta x}{2\sqrt{x}}$. Или аналогично:

$$\Delta(\sin x) = |(\sin x)'| \cdot \Delta x = |\cos x| \cdot \Delta x;$$

$$\Delta(\cos x) = |(\cos x)'| \cdot \Delta x = |- \sin x| \cdot \Delta x = |\sin x| \cdot \Delta x;$$

$$\Delta(\operatorname{tg} x) = |(\operatorname{tg} x)'| \cdot \Delta x = \left| \frac{1}{\cos^2 x} \right| \cdot \Delta x = \frac{\Delta x}{\cos^2 x};$$

$$\Delta(\operatorname{ctg} x) = |(\operatorname{ctg} x)'| \cdot \Delta x = \left| -\frac{1}{\sin^2 x} \right| \cdot \Delta x = \frac{\Delta x}{\sin^2 x};$$

$$\Delta(\ln x) = |(\ln x)'| \cdot \Delta x = \frac{\Delta x}{x} \quad (x > 0);$$

$$\Delta(e^x) = |(e^x)'| \cdot \Delta x = e^x \cdot \Delta x$$

и т. д. Для вывода формулы погрешности функции x^y воспользуемся представлением $x^y = e^{y \ln x}$, а затем формулами погрешностей экспоненты и произведения (предполагается, что $x > 0$):

$$\begin{aligned}\Delta(x^y) &= \Delta(e^{y \ln x}) = e^{y \ln x} \cdot \Delta(y \ln x) = \\ &= x^y \left(\frac{y \cdot \Delta x}{x} + \ln x \cdot \Delta y \right).\end{aligned}$$

Формула (2.20) позволяет сделать важное для практики вычислений наблюдение. Если значение модуля производной функции $f'(x)$ в точке x меньше единицы, то $\Delta f < \Delta x$, т. е. абсолютная ошибка значения функции оказывается меньше абсолютной ошибки значения аргумента. Если же $|f'(x)| > 1$, то значение функции будет иметь ошибку, большую ошибки аргумента. Этот факт имеет простое геометрическое толкование. При $|f'(x)| < 1$ функция изменяется медленно, т. е. «коридор» колебаний значений функции меньше соответствующего «коридора» изменений аргумента (рис. 19, а). При $|f'(x)| > 1$ малым отклонениям аргумента будут соответствовать большие колебания значения функции (рис. 19, б).

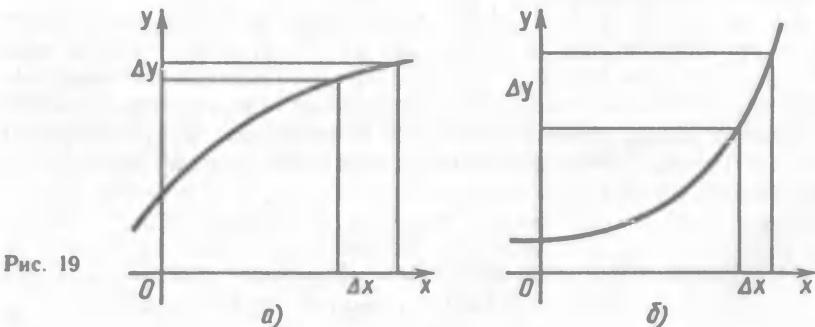


Рис. 19

значениях $|f'(x)|$ приводит к резкой потере точности и потому должен особо учитываться в практике вычислений (см. пример 2.5.2 на с. 72).

Формулы для вычисления границ абсолютных погрешностей значений некоторых функций одной переменной приведены в таблице 26.

Таблица 26

$f(x)$	$\Delta f(x)$	$\delta f(x)$
\sqrt{x}	$\frac{\Delta x}{2\sqrt{x}}$	$\frac{1}{2} \cdot \delta x$
$\frac{1}{x}$	$\frac{\Delta x}{x^2}$	δx
$\sin x$	$ \cos x \cdot \Delta x$	$ x \cdot \operatorname{ctg} x \cdot \delta x$
$\cos x$	$ \sin x \cdot \Delta x$	$ x \cdot \lg x \cdot \delta x$
$\operatorname{tg} x$	$\frac{\Delta x}{\cos^2 x}$	$\frac{2 x }{ \sin 2x } \cdot \delta x$
$\operatorname{ctg} x$	$\frac{\Delta x}{\sin^2 x}$	$\frac{2 x }{ \sin 2x } \cdot \delta x$
$\ln x$	$\frac{\Delta x}{x}$	$\frac{\delta x}{ \ln x }$
$\lg x$	$\frac{\Delta x}{x \cdot \ln 10}$	$\frac{\delta x}{ \lg x \cdot \ln 10}$
e^x	$e^x \cdot \Delta x$	$ x \cdot \delta x$
10^x	$10^x \cdot \ln 10 \cdot \Delta x$	$ x \ln \delta x$
$\arcsin x$	$\frac{\Delta x}{\sqrt{1-x^2}}$	$\frac{ x \delta x}{ \arcsin x \sqrt{1-x^2}}$
$\arccos x$	$\frac{\Delta x}{\sqrt{1-x^2}}$	$\frac{ x \delta x}{ \arccos x \sqrt{1-x^2}}$
$\operatorname{arctg} x$	$\frac{\Delta x}{1+x^2}$	$\frac{ x \delta x}{ \operatorname{arctg} x \cdot (1+x^2)}$
x^y	$x^y (y \frac{\Delta x}{x} + \ln x \Delta y)$	$y \ln x \cdot \delta y + y \cdot \delta x$

Во всех случаях предполагается, что значения x принадлежат области допустимых значений аргумента. Получаемые в результате вычислений значения границ погрешностей округляются для последующего использования до одной-двух значащих цифр (по избытку!).

Пример 2.5.1. С помощью МК получаем $\ln 1,9 = 0,641854$. Если 1,9 — точное значение, то в соответствии с точностью вычислительного прибора полученный результат имеет точность $\pm 10^{-6}$. Если же 1,9 — приближенное значение, у которого цифра 9 верна, например, в строгом смысле, то граница абсолютной

погрешности значения аргумента $x = 0,05$, а погрешность полученного значения логарифма в соответствии с формулой оценки границы абсолютной погрешности будет:

$$\Delta(\ln x) = \frac{\Delta x}{x} = \frac{0,05}{1,9} = 0,0263157 < 0,03.$$

Отсюда следует, что во втором случае полученное на МК значение $\ln 1,9 = 0,641854$ имеет лишь одну верную значащую цифру. Округляя результат с одной запасной цифрой, получим 0,64.

В тех случаях, когда производная функции вблизи приближенного значения аргумента имеет большие по модулю значения, произойдет большая потеря точности (так называемый случай катастрофической потери точности).

Пример 2.5.2. Пусть $x = 0,03$, причем $\Delta x = 0,005$, т. е. цифра 3 в числе x верна в строгом смысле. Нужно вычислить $\operatorname{ctg} x$. С помощью МК получаем $\operatorname{ctg} 0,03 = 33,323336$. Для определения верных цифр в результате оценим его абсолютную погрешность:

$$\Delta(\operatorname{ctg} x) = \frac{\Delta x}{\sin^2 x} \approx \frac{0,005}{0,0009} \approx 5,6,$$

откуда следует, что в полученном значении 33,323336 ни одну цифру нельзя считать верной (в строгом смысле).

Прокомментируем полученный результат. Заданная точность исходного значения аргумента определяет «коридор» его возможных значений: $0,025 \leq x \leq 0,035$. Найдем на МК значение $\operatorname{ctg} x$ для граничных значений x из этого «коридора»:

$$\operatorname{ctg} 0,025 = 39,992001;$$

$$\operatorname{ctg} 0,035 = 28,560004.$$

Как видно, диапазон изменения $\operatorname{ctg} x$ составляет более 10 единиц, что подтверждает отсутствие смысла в полученном выше результате вычислений.

В подобных случаях надо использовать все имеющиеся для этого возможности и увеличить число верных знаков в исходном данном (например, использовать более точный измерительный прибор, если исходное данное получается в результате измерений).

Пусть в условиях рассмотренного выше примера 2.5.2 $x = 0,0295$, $\Delta x = 0,00005$. Тогда

$$\operatorname{ctg} x = \operatorname{ctg} 0,0295 = 33,889114,$$

$$\Delta(\operatorname{ctg} x) < \frac{0,00005}{0,0008} < 0,063,$$

т. е. в полученном результате в строгом смысле верны 2 цифры. Округляя его с одной запасной цифрой, получим 33,9.

Из формулы (2.20) легко получается оценка границы относительной погрешности функции через границу относительной погрешности значения аргумента:

$$\delta f = \frac{\Delta f}{|f(x)|} = \frac{|f''(x)| \cdot \Delta x}{|f(x)|} = \frac{|f''(x)|}{|f(x)|} \cdot |x| \cdot \frac{\Delta x}{|x|} =$$

$$= \frac{|f''(x)|}{|f(x)|} \cdot |x| \cdot \delta x, \quad \text{т. е.}$$

$$\delta f = \frac{|f''(x)|}{|f(x)|} \cdot |x| \cdot \delta x, \quad \text{или} \quad (2.21)$$

$$\delta f = \frac{|f''(x)|}{|f(x)|} \cdot \Delta x. \quad (2.22)$$

Так, пользуясь формулами (2.21) и (2.22), можно, например, получить:

$$\delta(\sqrt{x}) = \frac{|x|}{2\sqrt{x} \cdot \sqrt{x}} \cdot \delta x = \frac{1}{2} \delta x = \frac{\Delta x}{2|x|};$$

$$\delta\left(\frac{1}{x}\right) = \left|\frac{x^2}{x^2}\right| \cdot |x| \cdot \delta x = \delta x = \frac{\Delta x}{|x|};$$

$$\delta(\sin x) = \frac{|\cos x|}{|\sin x|} \cdot |x| \cdot \delta x = |x \cdot \operatorname{ctg} x| \delta x = |\operatorname{ctg} x| \cdot \Delta x;$$

$$\delta(e^x) = \frac{e^x}{e^x} |x| \delta x = |x| \delta x = \Delta x;$$

$$\delta(\ln x) = \frac{x \cdot \delta x}{x |\ln x|} = \frac{1}{|\ln x|} \cdot \delta x;$$

$$\delta(x^y) = \delta(e^{y \ln x}) = |y \cdot \ln x| \cdot \delta(y \ln x) = |y \ln x| [\delta y + \delta(\ln x)] =$$

$$= |\ln x| \Delta y + y \cdot \Delta(\ln x) = |y \ln x| \delta y + |y| \delta x \text{ и т. д.}$$

В зависимости от способа задания исходных данных (с абсолютными или относительными погрешностями) вычислитель может использовать для подсчета границ относительных погрешностей тот или иной вариант формулы. Некоторые из этих формул приведены в таблице 26.

Пример 2.5.3. Значение аргумента $x = 0,63$ имеет относительную ошибку около 0,1%. Оценить величину относительной ошибки $\sin 0,63$.

Используя соответствующую формулу из таблицы 26, с помощью МК получим:

$$\delta(\sin 0,63) = |0,63 \cdot \operatorname{ctg} 0,63| \cdot 0,001 \approx 0,000864 \approx 0,08\%.$$

Используя величину найденной относительной погрешности, можно оценить количество верных в строгом смысле значащих цифр в искомом значении $\sin 0,63$. Поскольку имеет место $0,000864 < 10^{-3}$, то (см. формулу 2.10, с. 64) можно сделать вывод, что в числе $\sin 0,63 = 0,589145$ по крайней мере две цифры после запятой верны в строгом смысле.

Формулу оценки погрешности значения функции можно получить для общего случая. Пусть f — дифференцируемая функция n переменных; X_1, X_2, \dots, X_n — точные значения; x_1, x_2, \dots, x_n — приближенные значения переменных, а e_{x_i} ($i = 1, 2, \dots, n$) — их абсолютные погрешности. Тогда абсолютная погрешность значения функции f в точке (x_1, x_2, \dots, x_n) — это ее приращение $e_f = |f(X_1,$

$X_1, \dots, X_n) - f(x_1, x_2, \dots, x_n)|$. Считая, что погрешности e_{x_i} значительно меньше абсолютных величин значений x_1, x_2, \dots, x_n , заменим приращение функции e_f ее дифференциалом:

$$e_f \approx |df(x_1, x_2, \dots, x_n)| =$$

$$= \left| \sum_{i=1}^n \frac{\partial f}{\partial x_i} e_{x_i} \right| \leq \sum_{i=1}^n \left| \frac{\partial f}{\partial x_i} \right| \cdot e_{x_i}.$$

Это неравенство усиливается после замены абсолютных погрешностей e_{x_i} границами абсолютных погрешностей Δx_i :

$$e_f \leq \sum_{i=1}^n \left| \frac{\partial f}{\partial x_i} \right| \Delta x_i.$$

Отсюда следует, что можно принять

$$\Delta f = \sum_{i=1}^n \left| \frac{\partial f}{\partial x_i} \right| \Delta x_i. \quad (2.23)$$

Формула (2.23) имеет общий характер. Из нее можно получить не только все формулы для вычисления границ абсолютных погрешностей элементарных функций (см. табл. 26), но и формулы для границ абсолютных погрешностей арифметических действий. Пусть для примера $f(x_1, x_2) = x_1 \cdot x_2$. Тогда

$$\begin{aligned} \Delta f &= \left| \frac{\partial f}{\partial x_1} \right| \Delta x_1 + \left| \frac{\partial f}{\partial x_2} \right| \Delta x_2 = \\ &= x_2 \cdot \Delta x_1 + x_1 \cdot \Delta x_2, \end{aligned}$$

что в точности совпадает с формулой (2.18).

Получим общую формулу для относительной погрешности функции:

$$\delta f = \frac{\Delta f}{|f(x_1, x_2, \dots, x_n)|} = \sum_{i=1}^n \left| \frac{\partial f}{\partial x_i} \cdot \frac{1}{f} \right| \cdot \Delta x_i = \sum_{i=1}^n \left| \frac{\partial \ln f}{\partial x_i} \right| \cdot \Delta x_i. \quad (2.24)$$

Для выражения δf через относительные погрешности аргументов формулу (2.24) можно переписать в виде

$$\delta f = \sum_{i=1}^n |x_i| \cdot \left| \frac{\partial \ln f}{\partial x_i} \right| \cdot \delta x_i. \quad (2.25)$$

Предположив, что в формуле (2.23) все частные дифференциалы одинаково влияют на образование погрешности Δf , получим:

$$\Delta f = n \cdot \left| \frac{\partial f}{\partial x_i} \right| \Delta x_i,$$

откуда

$$\Delta x_i = \frac{\Delta f}{n \left| \frac{\partial f}{\partial x_i} \right|} \quad (i = 1, 2, \dots, n). \quad (2.26)$$

Формула (2.26) позволяет решать так называемую *обратную задачу теории погрешностей*: по заданной точности значения функции находить необходимые для достижения этой точности допустимые величины погрешностей значений аргументов.

Пример 2.5.4. Пусть x близко к значению $\pi/4 = 0,7853981\dots$. Установить, какова должна быть точность x , чтобы $\sin x$ мог быть получен на МК с максимальной (машинной) точностью $\pm 10^{-6}$.

Для $f(x) = \sin x$ по формуле (2.26) имеем ($n=1$):

$$\Delta x = \frac{\Delta f}{|\cos x|},$$

т. е.

$$\Delta x = \frac{10^{-6}}{\cos \frac{\pi}{4}} \approx 0,14 \cdot 10^{-5}.$$

Отсюда следует, что для получения максимальной точности значений синуса при x , близких к $\frac{\pi}{4}$, значения аргумента должны иметь не менее пяти верных (в строгом смысле) знаков после запятой.

Контрольные вопросы

1. В какой зависимости находится абсолютная погрешность значения функции одной переменной от абсолютной погрешности значения аргумента?
2. Как объясняется резкая потеря точности значения $f(x)$ для приближенных значений аргумента x на участках с большими по модулю значениями $f'(x)$?
3. Как формулируется обратная задача теории погрешностей?

Упражнения

1. Исходные числовые значения аргумента заданы цифрами, верными в строгом смысле. Определить, пользуясь МК, количество верных в строгом смысле цифр в следующих значениях элементарных функций:

- a) $\log 23,6$; в) $\frac{1}{4,09}$; д) $\operatorname{arctg} 8,45$;
б) $e^{2,01}$; г) $\arccos 0,79$; е) $3,4^{2,5}$.

2. Значение $x = 4,53$ имеет относительную ошибку 0,02%. Оценить количество верных в строгом смысле значащих цифр в значениях:

- а) $\ln x$; б) e^x ; в) x^x .

2.6. Способы приближенных вычислений

Наиболее распространенный вид вычислений — это вычисления по готовой формуле. В ЭВМ вычисление при любой громоздкости формулы обеспечивается, как правило, одной командой (оператором). Если при этом программно не предусматривается контроль за вычислительными погрешностями, вычислитель анализирует результат в конце счета.

Иногда условия вычислительной задачи заставляют вести пооперационный учет движения вычислительной погрешности. До появления калькуляторов и персональных ЭВМ пооперационный подход к оценке точности вычислений по необходимости рассматривался еще и как способ рационализации вычислений, ибо позволял на каждом шагу избавляться от сомнительных цифр, способных лишь загромождать и, что было особенно не приятно, удлинять процесс ручных вычислений. Однако то, ради чего преимущественно использовался пооперационный подход — исключение сомнительных цифр из последующих вычислений, при широком распространении быстродействующих вычислительных устройств с памятью утратило свой изначальный смысл, так как при умелом использовании дополнительных регистров памяти и других возможностей, представляемых современными калькуляторами, нет необходимости выписывать и заново вводить промежуточные результаты даже в случаях вычислений по достаточно громоздким формулам. Не говоря уже о том, что для микро-ЭВМ этих проблем вовсе не существует. И все же возможны ситуации (или потому, что этого требуют особенности вычислительной задачи, или это связано с желанием самого вычислителя), когда пооперационный учет движения ошибок вычислений необходим. Рассматривая в дальнейшем приемы вычислений, мы будем учитывать как пооперационную, так и итоговую методику оценки точности.

Вычисления по правилам подсчета цифр

При вычислении этим методом явного учета погрешностей не ведется, правила подсчета цифр показывают лишь, какое количество значащих цифр или десятичных знаков в результате можно считать надежным. Сами эти правила основываются на выводах, вытекающих из формул для оценки погрешностей арифметических действий и функций (см. п. 2.4 и 2.5). Приведем эти правила здесь в систематизированном виде.

1. При сложении и вычитании приближенных чисел младший из сохраняемых десятичных разрядов результата должен являться наибольшим среди десятичных разрядов, выражаемых последними верными значащими цифрами исходных данных*.

* Когда точность исходных данных такова, что все они имеют десятичные знаки после запятой, т. е. являются десятичными дробями, правило формулирует-

При этом следует избегать вычитания близких по величине чисел, а также при пооперационном применении правила для сложения и вычитания нескольких чисел подряд стараться производить действия над числами в порядке возрастания их абсолютных величин.

2. При умножении и делении приближенных чисел нужно выбрать число с наименьшим количеством значащих цифр и округлить остальные числа так, чтобы в них было лишь на одну значащую цифру больше, чем в наименее точном числе.

В результате следует считать верными столько значащих цифр, сколько их в числе с наименьшим количеством значащих цифр.

3. При определении количества верных цифр в значениях элементарных функций от приближенных значений аргумента следует прикинуть значение модуля производной функции. Если это значение не превосходит единицы или близко к ней, то в значении функции можно считать верными столько знаков после запятой, сколько их имеет значение аргумента. Если же модуль производной функции в окрестности приближенного значения аргумента превосходит единицу, то количество верных десятичных знаков в значении функции меньше, чем в значении аргумента, на величину k , где k — наименьший показатель степени, при котором имеет место

$$|f'(x)| < 10^{-k}.$$

4. При записи промежуточных результатов следует сохранять на одну цифру больше, чем рекомендуют правила 1—3. В окончательном результате эта запасная цифра округляется.

Правила подсчета цифр носят оценочный характер и не являются методом строгого учета точности вычислений. Обычно их применяют тогда, когда быстро и без особых затрат нужно получить результат, не особенно беспокоясь о его достоверности. Между тем практическая надежность этих правил достаточно высока в результате значительной вероятности взаимопогашения ошибок, не учитываемой при строгом подсчете границ погрешностей.

При пооперационном учете ошибок вычислений используется обычная расчетная таблица, так называемая *расписка формулы*.

ся более доступно: при сложении и вычитании приближенных чисел в результате следует считать верными столько десятичных знаков после запятой, сколько их в приближенном данном с наименьшим числом знаков после запятой. Количество десятичных знаков после запятой перед выполнением действия целесообразно уравнивать, округляя до одной запасной исходные данные с большим количеством десятичных знаков.

* Действительно, при этом условии с учетом формулы $\Delta f = |f'(x)| \Delta x$ можно вывести, что увеличение k на единицу допускает увеличение $|f'(x)|$, а значит, и Δf примерно в 10 раз, что в данном случае сокращает в значении $f(x)$ количество верных десятичных знаков по сравнению со значением x на единицу.

Пример 2.6.1. Вычислить значение величины

$$A = \frac{x + \sin y}{\sqrt{x^2 - y^2}} \quad (2.27)$$

по правилам подсчета цифр для приближенных значений $x = 2,53$ и $y = 0,764$, у которых все цифры верные.

Вычисления приведены в таблице 27. Прокомментируем ход вычислений. На МК получаем $\sin 0,764 = 0,691815$. Сразу же прикинем величину производной в этой же точке: $\cos 0,764 = -0,722075 < 1$, т. е. в полученном значении следует сохранить столько же десятичных знаков, сколько их в значении аргумента. Округляя с одной запасной, получаем 0,6918 (запасная цифра выделена) и заносим результат в таблицу. При вычислении суммы в числителе находим $2,53 + 0,6918 = 3,2218$ и согласно правилам 1 и 4 округляем результат до тысячных: 3,222. При вычислении x^2 пользуемся правилом 2, при нахождении разности $x^2 - y^2$ — правилом 1. При определении количества верных цифр в значении $\sqrt{5,637}$ снова применяем правило 3 (учитываем, что производная \sqrt{x} при $x > 1$ имеет значение, меньшее единицы). Округляя окончательный результат без запасной цифры, получим $A = 1,36$ (три верные цифры).

Таблица 27

x	y	$\sin y$	$x + \sin y$	x^2	$x^2 - y^2$	$\sqrt{x^2 - y^2}$	A
2,53	0,764	0,6918	3,222	6,401	5,637	2,374	1,36

Легко видеть, что с помощью, например, МК Б3-18 вычисления по формуле (2.27) можно произвести и без выписывания промежуточных результатов:

y [F] [sin] [+ x =] [F] [ЗАП] x [X] [—]
 y [=] [F] [$\sqrt{}$] [\div] [F] [ИП] [\leftrightarrow] [=]

Выполнив эту программу при заданных выше значениях, получим на индикаторе МК число 1,3570018. Как выделить в полученном числе верные цифры? Сделать это можно и без подробного поэтапного анализа, который приведен в примере.

Действительно, так как выражение A представляет собой дробь, то последнее действие при его вычислении — деление, а следовательно, результат будет содержать верных значащих цифр не более, чем в наименее точном из операндов — числитеle или знаменателе. Замечаем, что синус в данном случае дает верных цифр столько же, сколько и его аргумент (три), причем все они расположены после запятой. Второе слагаемое числителя также имеет три значение цифры, из которых после запятой размещены

две. Отсюда следует, что сумма в числителе сохранит три верные значащие цифры. Нетрудно видеть, что в знаменателе число верных цифр благодаря свойствам производной квадратного корня также наверняка не менее трех. Следовательно, значение A должно быть округлено до трех верных знаков: $A = 1,36$.

Там, где возможен подобный анализ, при использовании МК в вычислениях по правилам подсчета цифр можно избежать по-операционного учета верных знаков.

Контрольные вопросы

1. Как формулируются правила подсчета цифр?
2. В каких случаях рекомендуется применять правила подсчета цифр?
3. Какие два способа применения правил подсчета цифр возможны в вычислениях на калькуляторах и ЭВМ?
4. Какова последовательность действий на каждом промежуточном этапе расчетной таблицы в вычислениях по правилам подсчета цифр с пооперационным учетом ошибок? на заключительном этапе?

Упражнения

Вычислить на МК значения заданных выражений по правилам подсчета цифр двумя способами: 1) с пооперационным анализом результатов; 2) с итоговой оценкой окончательного результата (у числовых данных все цифры верные):

$$a) \frac{0,62 + \sqrt{16,9}}{\lg 41,3};$$

$$b) \frac{12,47 + \sqrt{12,5^2 + 14,8^2}}{\sin 0,97 + \cos 2,63};$$

$$6) \frac{\ln(6,91 + 3,35)}{\sqrt{62,3}};$$

$$g) \frac{\sqrt[3]{26,88}}{\frac{3,94}{8,04} - 6,19^{1,34}}.$$

Вычисления со строгим учетом границ погрешностей

Этот метод предусматривает строгий подсчет границ погрешностей по правилам вычисления погрешностей, рассмотренных в п. 2.4 и 2.5.

При пооперационном учете ошибок промежуточные результаты, так же как и их погрешности, заносятся в специальную таблицу, состоящую из двух параллельно заполняемых частей — для результатов и их погрешностей. В таблице 28 приведены вычисления со строгим учетом границ абсолютных погрешностей по той же формуле, что и в примере 2.16 (с. 68), и в предположении, что исходные данные x и y имеют границы абсолютных погрешностей $\Delta x = 0,005$, $\Delta y = 0,0005$ (т. е. у x и y все цифры верны в строгом смысле). Промежуточные результаты вносят в таблицу после округления до одной запасной (с учетом вычисленной параллельно величины погрешности), значения погрешностей

для удобства округляются (с возрастанием) до двух значащих цифр. Проследим ход вычислений на одном этапе.

С помощью МК имеем $\sin 0,764 = 0,691815$. Подсчитаем границы абсолютной погрешности (см. табл. 28):

$$\Delta(\sin 0,764) = \cos 0,764 \cdot 0,0005 = 0,000361 \approx 0,0004.$$

Судя по величине погрешности, в полученном значении синуса в строгом смысле верны три знака после запятой. Округляем это значение с одной запасной цифрой: $\sin 0,764 \approx 0,691\bar{8}$ (запасная цифра выделена) — и вносим его в таблицу. Вслед за этим вычисляется полная погрешность полученного результата (погрешность действия плюс погрешность округления: $0,0004 + 0,000015 < 0,00042$), которая также вносится в таблицу. Все последующие действия выполняются аналогично с применением соответствующих формул для границ абсолютных погрешностей.

Таблица 28

x	y	$\sin y$	$x + \sin y$	x^2	$x^2 - y$	$\sqrt{x^2 - y}$	A
2,53	0,764	0,6918	3,22	6,41	5,65	2,38	1,35
Δx	Δy	$\Delta(\sin y)$	$\Delta(x + \sin y)$	$\Delta(x^2)$	$\Delta(x^2 - y)$	$\Delta(\sqrt{x^2 - y})$	ΔA
0,005	0,0005	0,00042	0,0056	0,026	0,028	0,019	0,04

Округляя окончательный результат до последней верной цифры, а также округляя погрешность до соответствующих разрядов результата, окончательно получаем:

$$A = 1,4 \pm 0,1.$$

Рассмотрим теперь, как можно получить итоговую оценку границы погрешности результата вычислений по формуле без операционного учета движения ошибок.

Пример 2.6.2. Значения $a = 23,1$ и $b = 5,24$ даны цифрами, верными в строгом смысле. Вычислить значение выражения

$$B = \frac{\sqrt{a}}{b \cdot \ln a}.$$

С помощью МК по непрерывной программе

23,1 [F] [ln] [X] 5,24 [=] [F] [ЗАП]

23,1 [F] [$\sqrt{}$] [\div] [F] [ИП] [=]

получаем $B = 0,2921247$. Используя формулы относительных погрешностей частного и произведения, запишем:

$$\delta B = \delta(\sqrt{a}) + \delta b + \delta(\ln a), \quad \text{т. е.}$$

$$\delta B = \frac{1}{2} \delta a + \delta b + \frac{\delta a}{|\ln a|}.$$

Пользуясь МК, получим $\delta B = 0,003$, что дает

$$\Delta B = B \cdot \delta B = 0,0008.$$

Это означает, что в результате две цифры после запятой верны в строгом смысле:

$$B = 0,29 \pm 0,001.$$

Контрольные вопросы

1. Как оформляются вычисления со строгим учетом границ погрешностей при пооперационном учете ошибок?
2. Какова последовательность действий на каждом промежуточном этапе расчетной таблицы в вычислениях по методу строгого учета границ погрешностей с пооперационным учетом ошибок? на заключительном этапе?
3. Как вычисляются границы погрешностей результата при использовании методики итоговой оценки ошибки вычислений?

Упражнения

У значений $a = 2,674$ и $b = 31,48$ все цифры верны в строгом смысле. Вычислить значения заданных выражений со строгим учетом границ погрешностей двумя способами: 1) с операционным учетом погрешностей; 2) с итоговой оценкой точности результата:

$$a) \frac{ab}{\sqrt{a+b^2}}; \quad b) \frac{a+\sqrt{b}}{\lg(a^2+b^2)}; \quad c) \frac{a-\sqrt{b}}{\ln(1+a^2)}; \quad d) \lg \frac{\cos a + b}{a^2 + b^2}.$$

Вычисления по методу границ

Если нужно иметь абсолютно гарантированные границы возможных значений вычисляемой величины, используют специальный метод вычислений — метод границ.

Пусть $f(x, y)$ — функция, непрерывная и монотонная в некоторой области допустимых значений аргументов x и y . Нужно получить ее значение $f(a, b)$, где a и b — приближенные значения аргументов, причем совершенно достоверно известно, что

$$НГ_a < a < ВГ_a, \quad НГ_b < b < ВГ_b.$$

Здесь $НГ$, $ВГ$ — обозначения соответственно нижней и верхней границ значений параметров. Итак, вопрос состоит в том, чтобы найти строгие границы значения $f(a, b)$ при известных границах значений a и b .

Допустим, что функция $f(x, y)$ возрастает по каждому из аргументов x и y . Тогда

$$f(НГ_a, НГ_b) < f(a, b) < f(ВГ_a, ВГ_b).$$

Пусть теперь $f(x, y)$ возрастает по аргументу x и убывает по аргументу y . Тогда будет строго гарантировано неравенство

$$f(\text{НГ}_a, \text{ВГ}_b) < f(a, b) < f(\text{ВГ}_a, \text{НГ}_b).$$

Указанный принцип особенно очевиден для основных арифметических действий. Пусть, например, $f(x, y) = x + y$. Тогда очевидно, что

$$\text{НГ}_a + \text{НГ}_b < a + b < \text{ВГ}_a + \text{ВГ}_b.$$

Точно так же для функции $f(x, y) = x - y$ (она по x возрастает, а по y убывает) имеем:

$$\text{НГ}_a - \text{ВГ}_b < a - b < \text{ВГ}_a - \text{НГ}_b.$$

Аналогично для умножения и деления:

$$\text{НГ}_a \cdot \text{НГ}_b < ab < \text{ВГ}_a \cdot \text{ВГ}_b,$$

$$\text{НГ}_a / \text{ВГ}_b < a/b < \text{ВГ}_a / \text{НГ}_b.$$

Рассмотрим функцию $\frac{1}{\ln(x-y)}$. Замечаем, что при увеличении x она убывает, а с увеличением y возрастает (разумеется, при соблюдении условий существования). Следовательно, имеет место:

$$\frac{1}{\ln(\text{ВГ}_a - \text{НГ}_b)} < \frac{1}{\ln(a-b)} < \frac{1}{\ln(\text{НГ}_a - \text{ВГ}_b)}.$$

Вычисляя по методу границ с пооперационной регистрацией промежуточных результатов, удобно использовать обычную вычислительную таблицу, состоящую из двух строк — для вычисления НГ и ВГ результата (по этой причине метод границ называют еще и *методом двойных вычислений*). При выполнении промежуточных вычислений и округления результатов используются все рекомендации правил подсчета цифр с одним важным дополнением: округление нижних границ ведется по недостатку, а верхних — по избытку. Окончательные результаты округляются по этому же правилу до последней верной цифры.

Таблица 29

	x	y	$\sin y$	$x + \sin y$	x^2	$x^2 - y$	$\sqrt{x^2 - y}$	A
НГ	2,525	0,7635	0,69145	3,2164	6,3756	5,6111	2,3687	1,1351
ВГ	2,535	0,7645	0,69218	3,2272	6,4263	5,6628	2,3797	1,363

В таблице 29 приведены вычисления по формуле

$$A = \frac{x + \sin y}{\sqrt{x^2 - y}}$$

методом границ. Нижняя и верхняя границы значений x и y определены из условия, что в исходных данных $x = 2,53$ и $y = 0,764$ все

Рис. 20



цифры верны в строгом смысле ($\Delta x = 0,005$, $\Delta y = 0,0005$), т. е. $2,525 < x < 2,535$, $0,7635 < y < 0,7645$.

Способ границ связан со способом строгого учета границ погрешностей следующим образом. Пусть X — точное значение некоторой величины, x — его приближение с известными границами $B\Gamma_x$ и $H\Gamma_x$. Примем x равным значению $\frac{H\Gamma_x + B\Gamma_x}{2}$; тогда абсолютная погрешность ϵ_x этого приближения (рис. 20) будет заведомо не больше полуразности $\Delta x = \frac{B\Gamma_x - H\Gamma_x}{2}$. Так, по результатам вычислений в таблице 29 получаем:

$$A = \frac{1,351 + 1,363}{2} = 1,357, \quad \Delta A = \frac{1,363 - 1,351}{2} = 0,006,$$

что дает

$$A = 1,357 \pm 0,006.$$

Вычисления по методу границ можно вести и без пооперационного фиксирования промежуточных результатов. Пусть, например, нужно найти границы значения выражения

$$Z = \frac{\sqrt{x}}{\ln(x-y^2)},$$

если $8,643 < x < 8,645$; $2,109 < y < 2,112$. Имеем:

$$\frac{\sqrt{H\Gamma_x}}{\ln(B\Gamma_x - (H\Gamma_y)^2)} < Z < \frac{\sqrt{B\Gamma_x}}{\ln(H\Gamma_x - (B\Gamma_y)^2)}.$$

С помощью МК, используя непрерывную программу вычисления

$$\begin{array}{ccccccccc} x & \boxed{F} & \boxed{\sqrt{}} & \boxed{F} & \boxed{\text{ЗАП}} & y & \boxed{\times} & \boxed{-} \\ x = & \boxed{/ - /} & \boxed{F} & \boxed{\ln} & \boxed{\div} & \boxed{F} & \boxed{\text{ИП}} & \boxed{\leftrightarrow} & \boxed{=} \end{array}$$

получим:

$$2,0495677 < Z < 2,0548186.$$

Если нет нужды держать в результате слишком большое количество значащих цифр, его можно округлить (нижнюю границу по убыванию, верхнюю — по возрастанию). Так, округляя границы Z до сотых, будем иметь:

$$2,04 < Z < 2,06, \quad \text{т. е.}$$

$$Z = 2,05 \pm 0,01.$$

Контрольные вопросы

1. В каких случаях в приближенных вычислениях применяется метод границ?
2. Как оформляются расчеты по методу границ при пошаговом учете результатов вычислений?
3. Какова последовательность действий на каждом промежуточном этапе расчетной таблицы в вычислениях по методу границ с пошаговым учетом результатов вычислений? на заключительном этапе?
4. Как обнаруживается связь метода границ с методом строгого учета границ погрешностей?

Упражнения

Значения a и b заключены в границах: $4,92 < a < 4,96$, $1,83 < b < 1,85$. Вычислить значения заданных выражений по методу границ двумя способами: 1) с пошаговой регистрацией промежуточных результатов; 2) без выписывания промежуточных результатов:

$$a) \frac{\ln(a^2 - b)}{\frac{a}{\epsilon^2}};$$

$$b) \frac{\sqrt{a} + \sqrt{b}}{\ln(a^2 + b^2)};$$

$$6) \frac{\sqrt{a+b}}{\lg(a^2 + b^2)};$$

$$г) \sqrt{\frac{1+b^2}{1+a^2}} + \sqrt{\frac{a}{b}}.$$

Основы алгоритмизации

3.1. Метод алгоритмизации

Электронные вычислительные машины — это высокопроизводительные средства обработки информации, предназначенные для решения большого круга самых разнообразных задач. Однако ЭВМ — это всего лишь автомат (хотя и достаточно «интеллектуальный»), который быстро и точно выполняет предписания, составленные человеком. Разработка таких предписаний, т. е. заведомое проектирование всего хода решения задач,— неотъемлемая часть деятельности, связанной с использованием вычислительных машин.

В самом деле, даже при использовании такого вычислительного прибора, как простейший микрокалькулятор, иногда полезно заранее составить и записать на бумаге программу, т. е. перечень действий, которые должны быть проделаны вычислителем в процессе счета. Такую программу особенно целесообразно иметь тогда, когда предстоит произвести несколько вычислений по одной и той же формуле. В этом случае сначала составляется и тщательно выверяется программа вычислений, затем вычислитель может использовать ее механически, не вникая в последовательность производимых действий, что облегчает и ускоряет процесс счета.

Более совершенные вычислительные машины запоминают программы вычислений, и тогда человеку остается только составить программу и поместить ее в память машины, а все остальное — собственно решение задачи — машина делает автоматически, т. е. без участия человека. Такие вычислительные машины называют программно-управляемыми.

Таким образом, общим для программно-управляемой вычислительной техники является то, что решение задачи на ней осуществляется посредством составления программы. В основу программы для вычислительной машины кладется алгоритм решения данной задачи, т. е. точное предписание о последовательности действий, которые должны быть произведены для получения результата. Алгоритм является более общим понятием, чем программа, в этом смысле программа для вычислительной машины — это запись алгоритма решения некоторой задачи в виде, пригодном для данной вычислительной машины. Отсюда следует, что основная часть процесса решения задач с помощью программно-управляемой техники — это разработка алгоритмов решения этих

задач. Когда алгоритм решения задачи ясен, он без особого труда может быть представлен на языке программирования. Говоря иными словами, главное в решении задач на программно-управляемых вычислительных машинах — это **алгоритмизация**, т. е. составление алгоритмических предписаний.

Если рассматривать процесс решения задачи в целом, то разработка алгоритма для вычислительной машины является на самом деле лишь составной частью того сложного процесса, который обеспечивает полное решение задачи на ЭВМ. Этап алгоритмизации в общем случае наступает лишь тогда, когда ясна постановка задачи, когда имеется четкая математическая модель, в рамках которой будет проходить собственно решение задачи. С этой общей точки зрения процесс решения задачи на ЭВМ включает в себя следующие основные этапы:

1. Постановка задачи.
2. Построение математической модели.
3. Разработка алгоритма (алгоритмизация).
4. Составление программы.
5. Реализация программы на ЭВМ.
6. Анализ (интерпретация) результатов.

Полный процесс решения всегда сложен уже потому, что он включает извечную проблему: как решать задачу? С этой точки зрения первые два пункта из указанной выше последовательности этапов в определенной ситуации могут оказаться решающими. В самом деле, во многих случаях точная формулировка задачи, построение адекватной математической модели составляют основную трудность в поиске метода решения, который затем и приводит к разработке алгоритма. В то же время хоть сколько-нибудь однозначных рекомендаций для выполнения этих первых шагов не существует. При обучении методам разработки алгоритмов и программирования по этой причине часто, к сожалению, используются задачи, формулировка которых заведомо освобождает от прохождения этих **«неприятных»** этапов. В решении таких (учебных) задач основная роль действительно принадлежит алгоритмизации.

Следует заметить, что всевозрастающий **«интеллект»** ЭВМ, а также учет разработчиками новых алгоритмических языков всего предшествующего опыта практической алгоритмизации приводят к тому, что языки программирования приобретают свойства, позволяющие использовать эти языки для записи алгоритмов уже на стадии их разработки и поиска. Если это удается, то алгоритм сразу приобретает вид программы, и, следовательно, разделение пунктов 3 и 4 в приведенном выше перечне этапов решения задачи на ЭВМ становится условным. И все же алгоритмизация как метод, на который опирается взаимодействие человека с программно-управляемой техникой, имеет более широкие сферы применения в современном мире, выводящие его за пределы программирования для вычислительных машин. Так же как и моде-

лирование, алгоритмизация — это общий метод кибернетики. К реализации определенных алгоритмов сводятся процессы управления в различных системах. С построением алгоритмов связано и создание самых простейших автоматических устройств, и разработка автоматизированных систем управления сложнейшими производственными процессами, и построение автоматизированных систем обучения (АСО). Истоки алгоритмизации лежат в сугубо теоретической области современной математики — теории алгоритмов, однако сама она выступает скорее всего как набор определенных практических приемов, особых специфических навыков рационального мышления об алгоритмах в рамках заданных языковых средств.

Роль алгоритмизации в жизни человека определяется не только рассмотренными выше научно-техническими аспектами ее использования. Алгоритмический подход, обращение к «бытовым» алгоритмам неотделимы от повседневной жизни людей, от их обычной работы. В подавляющем большинстве случаев результат деятельности человека зависит от того, насколько четко он чувствует алгоритмическую сущность своих действий: что делать в каждый момент, в какой последовательности, каким должен быть итог действий и т. п. Все это определяет особый аспект культуры мышления и поведения, характеризующийся умением составлять и использовать различные алгоритмы.

Не могут быть оставлены без внимания принципы алгоритмизации и в процессе обучения. И речь идет не только об автоматизации алгоритмов самого обучения (построение и внедрение АСО), но и о выделении линии алгоритмов в содержании обучения. Это относится и к преподаванию школьных дисциплин. Хорошо известно, что алгоритмы буквально пронизывают содержание всех школьных предметов. А это означает, что формулирование, изучение и применение (выполнение) алгоритмов составляют существенный компонент содержания школьного обучения. Отсюда следует, что само содержание школьных наук при соответствующей его подаче предоставляет большие возможности для систематического формирования алгоритмических навыков. При этом умелое введение ученика в алгоритмическую природу понятий не только помогает более четко уяснить их смысл, но и подготовливает к последующему восприятию основных идей использования программно-управляемой вычислительной техники.

Контрольные вопросы

1. Каковы основные этапы решения задачи с помощью программно-управляемой вычислительной техники?
2. В чем состоит значение этапа алгоритмизации в этом процессе?
3. Какова роль алгоритмизации в современном мире?

3.2. Алгоритмы и их свойства

Понятие алгоритма относится к числу фундаментальных математических понятий и является объектом исследования специального раздела математики — теории алгоритмов. Вместе с тем для ознакомления с методами алгоритмизации в связи с составлением программ для вычислительных машин нет необходимости обращаться к строгому определению этого понятия.

Что же такое алгоритм? С уверенностью можно сказать, что, не употребляя самого слова «алгоритм», многие интуитивно пользуются этим понятием и правильно представляют его смысл.

В зависимости от характера занятий людям в своей повседневной жизни встречаются различные практические задачи. Примеры таких задач могут быть самыми разнообразными: приготовление кофе, проезд от дома до школы, изготовление детали на токарном станке, решение квадратного уравнения, пеленание ребенка, оплата проезда в городском транспорте, исполнение танца, стрельба из пневматического ружья в тире, поиск слова в словаре и т. д. Примеры таких задач можно успешно продолжить дальше. Здесь важно отметить, что при решении любой подобной задачи человек обращается к тщательно продуманным заранее со всеми возможными вариантами предписаниям о том, какие действия и в какой последовательности должны быть выполнены для получения решения задачи. Эти системы предписаний, пригодные для неоднократного использования разными людьми, являются примерами алгоритмов.

Например, при домашнем крашении изделий из хлопчатобумажных и льняных тканей с помощью анилинового красителя (в таблетках) можно руководствоваться следующей последовательностью указаний:

1. Таблетки размельчить и растворить с одной столовой ложкой соли в 0,5 л горячей воды.

2. Раствор процедить в просторную посуду и добавить воды, чтобы раствором был полностью покрыт выстиранный и отжатый материал.

3. Поставить емкость с материалом на огонь и при слабом кипении красить 1 ч и еще 20 мин без нагревания, все время переворачивая материал.

4. После окрашивания ткань прополоскать в холодной воде.

Этот алгоритм дословно переписан с инструкции, которая сопровождает упаковку таблеток красителя; разница лишь в том, что здесь для наглядности отдельные описания алгоритма пронумерованы. Аналогичным способом могут быть описаны алгоритмы решения и других приведенных выше задач.

Итак, с точки зрения неформальных представлений алгоритм — это система точных и понятных предписаний о содержании и последовательности выполнения конечного числа действий, необходимых для решения любой задачи данного типа.

Много примеров алгоритмов имеется в школьных предметах. Но более всего таких примеров в школьном курсе математики. Это и понятно: ведь математика и занимается, по существу, изучением различных алгоритмов и созданием новых. К числу алгоритмов школьного курса математики относятся, например, правила выполнения арифметических действий, правила решения определенных видов уравнений или неравенств, правила построения определенных геометрических фигур и т. п. Происхождение самого термина «алгоритм» тоже связано с математикой. Слово «алгоритм» появилось в результате искажения (после перевода на европейские языки) имени арабского математика IX в. аль-Хорезми, которым были описаны правила (или, как мы теперь говорим, алгоритмы) выполнения основных арифметических действий в десятичной системе счисления. Эти алгоритмы изучаются сегодня в начальных разделах школьной математики.

Таким образом, понятие алгоритма возникло и используется давно, значительно раньше появления ЭВМ*. Тем не менее широким распространением это понятие обязано основополагающей идеи — идею автоматизации поведения исполнителя-автомата, реализуемой на основе алгоритма. В ряду всевозможных автоматов ЭВМ является лишь частным (хотя и наиболее впечатляющим) примером такого исполнителя. Задача обучения алгоритмизации заключается в том, чтобы научить составлять записи алгоритмов, причем делать это так, чтобы воображаемый при этом исполнитель (человек, робот, ЭВМ) мог однозначно и точно следовать предписаниям алгоритма и эффективно получать определенный результат. Это накладывает на записи алгоритмов целый ряд обязательных требований, суть которых вытекает, вообще говоря, уже из приведенного выше неформального толкования понятия алгоритма. Сформулируем эти требования в виде перечня свойств, которым должны удовлетворять алгоритмы, адресуемые заданному исполнителю.

1. Одно из первоначальных требований, которое предъявляется к алгоритмической записи, состоит в том, что описываемый процесс должен быть разбит на последовательность отдельных шагов. Возникающая в результате такого разбиения запись представляет собой упорядоченную совокупность четко разделенных друг от друга предписаний (директив, команд), образующих прерывную (или, как говорят, дискретную) структуру алгоритма: только выполнив требования одного предписания, можно приступить к выполнению следующего**.

* Появление первых моделей ЭВМ в нашей стране относится к началу 50-х годов этого столетия.

** Следует иметь в виду, что это естественное на первый взгляд свойство характерно лишь для частного (хотя и весьма широкого) класса так называемых последовательных алгоритмов и не является обязательным для алгоритмов с параллельным исполнением. Принцип параллелизма здесь не рассматривается.

Эта особенность алгоритмической записи наглядно видна из приведенного выше примера алгоритма домашнего крашения. Дискретная структура алгоритмической записи в данном случае подчеркивается сквозной нумерацией отдельных предписаний алгоритма, хотя это требование может быть необязательным.

Рассмотренное свойство алгоритмов называют свойством *дискретности*.

2. Используемые на практике записи алгоритмов составляются с ориентацией на определенного исполнителя. Чтобы составить для него алгоритм, нужно знать, какие предписания этот исполнитель может понять и выполнить, а какие не может. Дело в том, что у каждого исполнителя имеется свой перечень предписаний, которые для него понятны и которые он может выполнить. Такой перечень называют *системой предписаний* (или *системой команд*) исполнителя алгоритмов. Понятно, что, составляя запись алгоритма для определенного исполнителя, можно использовать лишь те предписания, которые имеются в его системе предписаний.

Так, обычные адресуемые человеку «бытовые» алгоритмы (на подобие приведенного выше алгоритма домашнего крашения) по меньшей мере предполагают, чтобы их исполнитель умел читать на использованном для записи алгоритма языке, т. е. обладал элементарной грамотой. Кроме того, подразумевается, что он в состоянии понять и выполнить все те действия, которые предусмотрены предписаниями алгоритма.

Это свойство алгоритмов будем называть свойством *понятности*.

3. Будучи понятным, алгоритм не должен все же содержать предписаний, смысл которых может восприниматься неоднозначно. Это означает, что одно и то же предписание, будучи понятным разным исполнителям, после выполнения каждым из них должно давать одинаковый результат.

В данном случае речь фактически идет о том, что запись алгоритма должна быть настолько четкой, настолько полной и продуманной в деталях, чтобы у исполнителя никогда не могло возникнуть потребности в принятии каких-либо самостоятельных решений, не предусмотренных составителем алгоритма. Говоря иначе, алгоритм не должен оставлять места для произвола исполнителя. Поэтому такие предписания, как, например, «Взять две-три ложки сахарного песку», «Через несколько минут снять трубку с рычага», «Умножить x на одно из двух данных чисел a или b », не могут встречаться в алгоритмах. Очевидно, что понятные в определенных ситуациях для человека предписания такого типа могут поставить в тупик робота. Кроме того, в алгоритмах недопустимы также ситуации, когда после выполнения очередного предписания алгоритма исполнителю неясно, какое из предписаний алгоритма должно выполняться на следующем шаге.

Отмеченное свойство алгоритмов называют свойством определенности или детерминированности.

4. Наиболее предпочтительными являются алгоритмы, обеспечивающие решение широкого класса задач данного типа. В простейшем случае эта вариативность алгоритма обеспечивается возможностью использовать различные допустимые значения исходных данных. Так, например, алгоритм (или программа) вычисления по формуле $y = (\ln \sqrt{x})^2$ на микрокалькуляторе «Электроника МК-41»



может применяться не для одного значения x , а для всех $x > 0$. Другой пример: решение квадратного уравнения $ax^2 + bx + c = 0$ в области действительных чисел может быть найдено по формулам

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a},$$

которые применимы не для одного, а для многих квадратных уравнений с коэффициентами a , b и c , удовлетворяющих условию $D = b^2 - 4ac \geq 0$.

Это свойство алгоритмов называется свойством массовости.

5. Обязательное требование к алгоритмам — результативность. Смысл этого требования состоит в том, что при точном выполнении всех предписаний алгоритма процесс должен прекратиться за конечное число шагов и при этом должен быть получен какой-либо определенный ответ на вопрос задачи.

Итак, приступая к разработке алгоритмов, составитель должен придерживаться перечисленных выше требований. Учитывая, что сама задача составления алгоритма не имеет смысла до тех пор, пока четко не определены возможности исполнителя, следует начинать с уточнения системы его предписаний. Система предписаний исполнителя определяет характер языка, на котором должна выполняться запись алгоритма. По этой причине понятия «алгоритм» и «язык» являются понятиями взаимосвязанными. В этом смысле можно отметить, что сформулированные выше свойства алгоритмов имеют значение не столько сами по себе, сколько в приложении к конкретным формам представления алгоритмов.

Ознакомление с простейшими типами алгоритмических предписаний мы будем вести с ориентацией на исполнителя, возможности которого (там, где это не имеет принципиального значения) определены не слишком строго. В дальнейшем при переходе к средствам представления алгоритмов, приближающихся к «машинному» стандарту требований, особенностям языка будет уделено больше внимания.

Упражнения

1. Перечислить основные требования, предъявляемые к алгоритмам.
2. Привести примеры предписаний, в которых нарушено свойство определенности.
3. Привести примеры немассовых алгоритмов.

3.3. Алгоритмы для вычислений

Простейший класс алгоритмов составляют алгоритмы вычислений по формулам. Задача вычисления по формуле ставится так: задается формула и значения входящих в нее величин, требуется произвести предусмотренные формулой действия и получить окончательный числовoy ответ. Рассмотрим формы представления алгоритмов вычислений по формулам, традиционно используемых в условиях школьного обучения и рассчитанных на «ручное» исполнение (т. е. с участием человека). Обсуждение уже этих простых (и не очень жестко регламентированных) способов задания алгоритмов затрагивает некоторые важные общие вопросы алгоритмизации, имеющие продолжение и при составлении программ для электронных вычислительных машин. К тому же такой обзор позволяет подчеркнуть связь между алгоритмизацией и содержанием школьного обучения.

Формулы. Математические формулы вместе с правилами их написания представляют собой своеобразный алгоритмический язык, с успехом используемый для задания вычислительных алгоритмов некоторого специального вида. Так, например, формула

$$S = \frac{\pi D}{2} + \pi DH \quad (3.1)$$

задает алгоритм вычисления площади поверхности цилиндрического тела с диаметром D и высотой H .

Строго говоря, формула определяет последовательность действий не столь однозначно, как того требует понятие алгоритма (в частности, свойство определенности). Вместе с тем известно, что выбор любого порядка действий при соблюдении установленных в математике правил не отразится на результате. В обычном случае порядок действий по формуле регулируется исполнителем-человеком. В алгоритмических языках, используемых в программировании для ЭВМ, обычно предусматривается возможность задания вычислений непосредственно формулами в обычном математическом смысле; это означает, что ЭВМ в данном случае «обучена» определять правильный порядок действий по виду формулы.

Алгоритм, изображаемый обычной формулой, относится к некоторому, весьма узкому классу алгоритмов, называемых линейными. Так называются алгоритмы, у которых последовательность

операций при исполнении совпадает с порядком их следования в записи алгоритма и не зависит от конкретных значений входных данных. Таким является, например, приведенный в п. 3.1 «бытовой» алгоритм домашнего крашения. К линейному сводится также приведенный алгоритм вычисления по формуле площади поверхности цилиндра.

Одних лишь линейных алгоритмов оказывается недостаточно для описания уже самых простейших вычислительных процессов. Так, например, алгоритм вычисления функции, заданный формулами

$$y = \begin{cases} \ln x, & \text{если } x \geq 1, \\ 1-x, & \text{если } x < 1 \end{cases} \quad (3.2)$$

(рис. 21) уже не является линейным, так как в нем заложена операция выбора одной из формул в зависимости от заданного значения x . Такого рода алгоритмы называют разветвляющимися или ветвящимися.

Табличный способ. Запись вычислительного алгоритма в форме таблицы широко используется при организации вычислений по формуле с пооперационной регистрацией промежуточных результатов. В этом случае расписка формулы на последовательность элементарных действий, обеспечиваемых имеющимися в наличии вычислительными средствами, есть не что иное, как определение последовательности шагов (указаний) вычислительного алгоритма. Так, табличный алгоритм вычисления по формуле (3.1) может иметь вид таблицы 30. В этой таблице приведены все этапы вычисления площади поверхности цилиндра по правилам подсчета цифр для трех пар исходных значений D и H . (Если они выражены, например, в сантиметрах, то результат S имеет наименование см².)

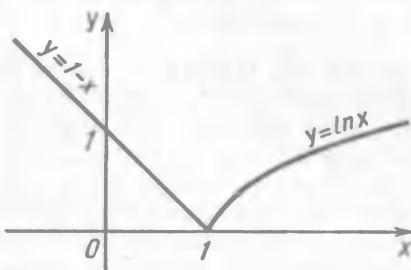


Рис. 21

Таблица 30

D	H	πD	πD^2	$\pi D^2/2$	πDH	S
12,6	8,9	39,58	498,7	249,4	352	600
19,3	14,2	60,63	1170	585,0	860,9	1446
6,8	5,4	21,4	146	73,0	116	190

Табличная форма записи алгоритма особенно удобна тогда, когда требуется вычислять не одно, а несколько значений одного и того же выражения для различных значений входных величин.

Программы для МК. В связи с привлечением для вычислений микрокалькуляторов возник язык для записи программ вычислений. Программа для МК — это, по сути дела, зафиксированный на бумаге перечень клавиш, нажатие которых в заданной последовательности приводит к решению данной вычислительной задачи. Так, программа вычислений площади поверхности цилиндра на МК-41 по формуле (3.1) для различных значений D и H может быть записана следующим образом:

$$\boxed{\text{л}} \times \boxed{D} \times \boxed{(} \boxed{D} + \boxed{2} \times \boxed{H} \boxed{)} \div \boxed{2} =$$

Итоговая таблица в данном случае, помимо столбцов для исходных данных, содержит лишь столбец для записи окончательного результата (табл. 31). Запись алгоритмов для программируемых МК также выполняется на основе некоторых условных обозначений.

Таблица 31

D	H	S
12,6	8,9	600
19,3	14,2	1466
6,8	5,4	190

Упражнения

По заданным формулам составить вычислительные алгоритмы в виде таблиц и программ для МК:

a) $S = \frac{\pi D^2}{4} + \frac{\pi D l}{2}$

(S — площадь боковой поверхности конуса, D — диаметр основания, l — образующая);

b) $f = \frac{1}{2\pi \sqrt{LC}}$

(f — частота собственных колебаний в контуре, L — индуктивность катушки, C — емкость конденсатора).

3.4. Схемы алгоритмов

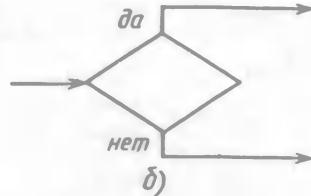
Схема алгоритма — это наглядный графический способ его представления. Отдельные предписания алгоритма изображаются в данном случае в форме определенных плоских геометрических фигур. Переходы от предписания к предписанию изображаются с помощью линий связи, а направления переходов — стрелками. Особое свойство схем заключается в том, что этот способ представления алгоритмов предусматривает явное выделение двух основных типов предписаний — арифметических и логических (рис. 22).

Арифметическое предписание (рис. 22, a) предусматривает выполнение некоторой последовательной работы, завершающееся естественным переходом в одном направлении, без ветвлений. Применительно к различным (а не только вычислительным) за-



Рис. 22

a)



да

нет

б)

дачам соответствующий элемент схемы называют также **элементом общей обработки**. Изображается он в виде прямоугольника с одним входом и одним выходом. Содержание действий, предусматриваемых арифметическим элементом, записывается обычно внутри прямоугольника.

Логические предписания (рис. 22, б) специально используются для организации ветвлений. Соответствующий элемент схемы называют **элементом принятия решения**. Изображать его принято в форме ромба с одним входом и двумя выходами. Назначением логического элемента является проверка заданного условия, которое записывается обычно внутри ромба. Если проверяемое условие выполняется (истинно), то происходит переход по стрелке *да*, если не выполняется (ложно) — по стрелке *нет*.

С помощью описанного аппарата схем можно наглядно изображать самые различные алгоритмы.

Пример 3.4.1. Перед выходным днем пapa сказал своему сыну: «Давай спланируем свой завтрашний день. Если будет хорошая погода, то проведем день в лесу. Если же погода будет плохая, то сначала займемся уборкой квартиры, а во второй половине дня посмотрим фильм по телевизору». Эта простая альтернатива может быть изображена так, как показано на рисунке 23.

Схема помогает наглядно показать важное свойство разветвляющихся алгоритмов: исполнение разветвляющегося алгоритма всегда проходит только по одному из возможных путей, который определяется конкретными текущими условиями. Применительно к алгоритму, изображеному на рисунке 23, это означает, что если погода действительно будет хорошая, то вылазка в лес займет весь день и тогда уж уборку квартиры придется



Рис. 23



Рис. 24

отложить, а также пожертвовать телевизионным фильмом. И наоборот, если придется из-за плохой погоды весь день просидеть дома, то уборка квартиры и просмотр телевизионного фильма состоятся, но тогда уже не будет прогулки в лес.

Перестановка элементов схемы алгоритма может привести к изменению его сути. Допустим, что, хорошо сознавая отмеченное выше свойство разветвляющегося алгоритма и проявляя беспокойство о судьбе интересного фильма, сын вносит предложение: «Папа, если мы отправимся на прогулку, то хорошо бы постараться вернуться домой к началу телефильма». Предлагаемый при этом план выходного дня изображен на рисунке 24. Ситуация становится иной: что бы ни предшествовало этому — прогулка в лес или уборка квартиры, просмотр телевизионного фильма обязательно состоится. Допустим теперь, что, беспокоясь все же прежде всего о прогулке на свежем воздухе, пapa подводит итог составлению плана выходного дня следующим образом: «Если будет хорошая погода, то мы весь день проведем в лесу. Если же погода с утра будет неблагоприятной, то займемся сначала уборкой квартиры. И если при этом во второй половине дня погода улучшится, то совершим прогулку в лес. Если же погода весь день будет плохой, то после обеда посмотрим фильм по телевизору». В форме схемы этот план изображен на рисунке 25.

Наглядность, понятность и общедоступность, какими обладают схемы, позволяют знакомить с этим способом представления алгоритмов детей младшего возраста.



Рис. 25

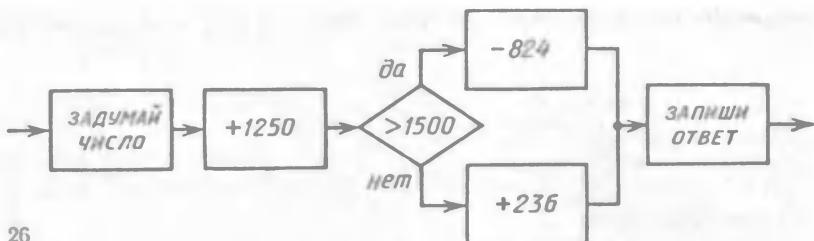


Рис. 26

Пример 3.4.2. На рисунке 26 изображена схема, пользуясь которой младший школьник наряду с усвоением техники простейших вычислений на микрокалькуляторе может успешно уяснить суть разветвленной алгоритмической записи и научиться понимать (и даже строить) схемы подобных алгоритмов.

При записи вычислительных алгоритмов удобно использовать специальный знак присваивания $:=$. Этот знак используется для изображения особой операции — операции присваивания, смысл которой состоит в следующем. Пусть имеется предписание вида

$$y := A$$

(читается: « y присвоить A »), где y — переменная, A — некоторое выражение*. Предписание означает следующее: выполнить все действия, предусмотренные формулой A , и полученный результат (число) считать значением (т. е. присвоить) переменной y . Пример записи команды присваивания:

$$\begin{aligned} x &:= \ln \sqrt{a^2 + 1}; \\ y &:= 2x^2 + 7. \end{aligned}$$

В левой части команды присваивания всегда должна стоять переменная. Выражение в правой части в частном случае может быть переменной или числом, например: $y := a$; $x := 12$.

Пример 3.4.3. Схема разветвляющегося алгоритма вычислений по формулам

$$y = \begin{cases} \ln x, & \text{если } x > 1, \\ 1 - x, & \text{если } x \leq 1 \end{cases}$$

имеет вид простой альтернативы (рис. 27). Вычислительные указания сформулированы с помощью знака присваивания.

В форме схем могут быть наглядно изображены многие алгоритмы, изучаемые в школе.

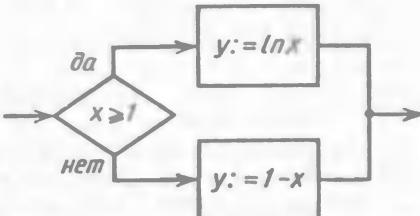


Рис. 27

* Операцию присваивания иногда обозначают обычным знаком равенства, а в некоторых случаях для этой цели используют стрелки. Например, указание $x := 5$ может иметь вид $x \leftarrow 5$ или $5 \Rightarrow x$.

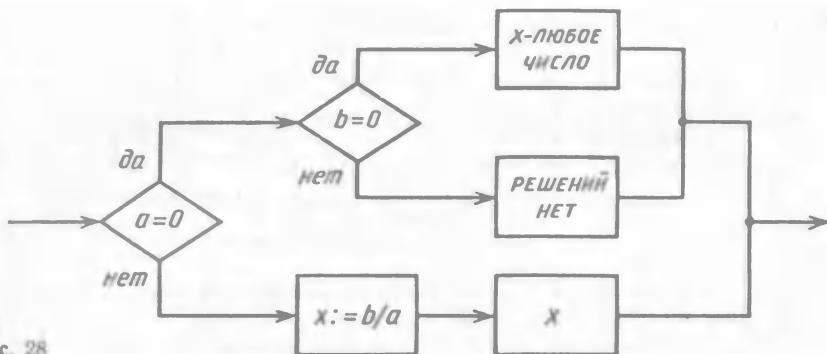


Рис. 28

При этом использование представления алгоритмов в форме схем часто позволяет серьезно облегчить задачу не только ученика (понять суть алгоритма), но и учителя (наглядно показать алгоритм во всех его проявлениях).

Пример 3.4.4. Рассмотрим процесс решения уравнения $ax=b$, где a и b — произвольные числа.

Если $a \neq 0$, то решение находится просто: $x = b/a$. Если же $a = 0$, то уравнение приобретает вид $0 \cdot x = b$, и теперь уже надо посмотреть, равно ли нулю b . При $b = 0$ решением будет любое число, а если $b \neq 0$, то решений нет. Схема алгоритма полного исследования и решения уравнения $ax=b$ приведена на рисунке 28. Из схемы хорошо видно, что при решении этого уравнения существуют три возможных исхода.

Контрольные вопросы

1. Из каких двух основных типов предписаний составляются алгоритмы? Как изображаются эти предписания в схемах алгоритмов?
2. В чем заключается основное свойство разветвляющихся алгоритмов?
3. Что обозначает и как используется в записи алгоритмов знак присваивания?

Упражнения

1. Составить схему алгоритма, описанного следующим образом: «Если Павлик дома, будем решать задачи по математике. В противном случае следует позвонить Марине и взяться за выпуск стенгазеты. Если же Марина нет дома, то надо сесть за сочинение».

2. Составить схемы алгоритмов: а) решения уравнения $ax = 7$; б) решения неравенства $ax > 5$.

3. Составить схему алгоритма решения квадратного уравнения $ax^2 + bx + c = 0$ ($a \neq 0$) в области действительных чисел.

3.5. Словесная запись алгоритмов

Самой распространенной формой представления алгоритмов, адресуемых человеку, является обычная словесная запись. Форму словесной записи имеют многие так называемые «бытовые» алгоритмы, часто используемые в повседневной практике: как выкрасить изделие (см. с. 88), как позвонить по междугородному телефону-автомату, как пользоваться стиральной машиной и т. п. Особенность словесных представлений алгоритмов в том, что таким путем при желании могут быть описаны любые алгоритмы, в том числе и вычислительные.

Пример 3.5.1. Пусть требуется записать последовательность элементарных действий для вычислений по формуле

$$y = \frac{3x}{\sqrt{8x+1}}.$$

При составлении требуемой алгоритмической записи предположим, что правила выполнения арифметических операций сложения, умножения и деления, а также действие извлечения квадратного корня понятны вычислителю, т. е. являются для него элементарными. При этом предположении искомый алгоритм в обычной словесной записи может иметь вид:

1. Прочитать заданное значение x .
2. Умножить x на 8.
3. Из результата второго действия извлечь квадратный корень.
4. К результату третьего действия прибавить 1.
5. Умножить x на 3.
6. Результат пятого действия разделить на результат четвертого действия.
7. Записать значение результата y .

Получили словесную запись линейного вычислительного алгоритма. Следует особое внимание обратить на первое и последнее предписания в этой записи, обеспечивающие задание значения исходного данного и выдачу полученного ответа. При всей кажущейся излишней педантичности этих предписаний они имеют первостепенное значение в алгоритмах, адресуемых исполнителям-автоматам. Формулируя эти предписания, автор алгоритма должен четко определить для себя, что в этой задаче *дано*, а что *требуется получить*. Или, говоря иными словами, какие величины являются носителями исходных значений (входные величины, или *аргументы*), а какие — носителями значений результатов (выходные величины, или *результаты*). В приведенной выше записи вычислительного алгоритма входной величиной является x , а выходной — y .

Отметим теперь другое важное обстоятельство. Несмотря на то что в приведенной записи алгоритма ни в одном из предписаний ничего не сообщается о том, к какому предписанию

следует переходить в следующий момент, представляется вполне естественным, что всюду предполагается переход к предписанию, номер которого является следующим в порядке возрастания номеров предписаний. Именно этого правила мы и будем придерживаться в дальнейшем: если при исполнении текущего предписания алгоритма никак не определяется номер очередного предписания, то это будет означать автоматический переход к предписанию, следующему за данным в порядке возрастания номеров. Если же было исполнено предписание с наибольшим номером, то процесс выполнения алгоритма заканчивается. Для прекращения исполнения алгоритма используется также специальное предписание «конец».

Приведенную выше словесную запись вычислительного алгоритма можно сделать более компактной, если воспользоваться операцией присваивания $:=$, которая была введена в предыдущем пункте. Однако при этом мы приходим к весьма важному приему, широко используемому в практике алгоритмизации,— использованию *вспомогательных переменных* (букв) для запоминания промежуточных числовых значений. Действительно, возьмем, к примеру, предписание с номером 2: «Умножить x на 8». Выберем переменную a для обозначения результата этого действия, тогда само действие с использованием знака присваивания может быть записано значительно короче:

$$a := 8x,$$

что означает: умножить текущее значение x на 8 и полученный результат считать впредь значением переменной a . Используя всюду, где можно, знак присваивания и вводя новые вспомогательные переменные, получим следующее представление приведенного выше алгоритма:

1. чтение x
2. $a := 8x$
3. $b := \sqrt{a}$
4. $c := b + 1$
5. $d := 3x$
6. $y := d/c$
7. запись y
8. конец

В дальнейшем всюду, где в словесной записи алгоритмов будет использоваться знак присваивания, будет предполагаться, что для исполнителя понятна эта операция.

Составляя записи алгоритмов, нужно постоянно хорошо понимать, какие действия исполнителя на самом деле будут вызваны предписаниями алгоритма и к каким изменениям значений используемых переменных эти действия приведут. Умение вести такой анализ в течение всей разработки алгоритма является исключительно важным требованием к составителю алгоритмов. Это

требование принуждает автора постоянно перевоплощаться в хладнокровного и педантичного исполнителя и является, по сути дела, единственным работающим в процессе составления записи алгоритма средством контроля и обоснования.

На первых порах для тренировки необходимого навыка понимания процесса исполнения алгоритма полезно обращаться к специальной процедуре проигрывания алгоритмической записи. Как мы увидим дальше, использование этой процедуры до приобретения опыта алгоритмизации может существенно облегчить контроль правильности составления алгоритмов. Процесс проигрывания алгоритмической записи рассмотрим на примере только что описанного алгоритма вычисления значения y .

Суть этого процесса состоит в том, что выбираются конкретные исходные данные и описанный алгоритм педантично исполняется со строгим соблюдением всех содержащихся в его записи предписаний. При этом каждое выполняемое элементарное действие фиксируется не где-нибудь в уме, а в специальном бланке, на котором регистрируются не только номера выполняемых предписаний алгоритма, но и все получаемые промежуточные результаты. Форма бланка для проигрывания алгоритма вычисления значения y показана в таблице 32. В таблице отведены столбцы для номеров предписаний (шаги алгоритма), для всех фигурирующих в описании величин — и входных, и промежуточных, и выходных (в данном случае это переменные x, a, b, c, d, y). Отдельная графа отведена для пояснений. В таблице 32 показано исполнение алгоритма вычисления значения y при $x=2$.

Таблица 32

Шаги алгоритма	Аргумент	Промежуточные величины					Результат	Пояснения
		x	a	b	c	d		
1	2							чтение x
2		16						
3				4				
4					5			
5						6		
6							1,2	
7								
8								запись 1,2 остановка

Отметим сейчас важное свойство операции присваивания, которое вытекает из правил ее выполнения, описанных в п. 3. Дело в том, что эта операция вполне допускает случаи, когда одна и та же переменная находится и слева и справа от знака присваивания. Так, например, можно написать

$$n := n + 1,$$

и это означает, что требуется к значению переменной a , которое она имела к началу выполнения операции присваивания, прибавить число 1 и считать полученное значение новым значением переменной a . Прежнее значение a при этом пропадает.

Легко заметить, что, используя указанное свойство операции присваивания, можно избежать в записи алгоритмов введения избыточных вспомогательных переменных. Так, вместо того чтобы в предписании 3 описанного выше алгоритма вводить новую переменную b , можно воспользоваться той же переменной a , тем более что ее прежнее значение $8x$ в этом алгоритме уже нигде не потребуется. Очевидно, что эту же переменную a можно использовать и в предписании 4. Ясно также, что при использовании знака присваивания новую переменную следует вводить лишь в случае, если полученный результат требуется «запомнить», т. е. сохранить ее для последующих вычислений.

С учетом сделанных соглашений запись алгоритма вычисления y будет иметь вид:

1. чтение x
2. $a := 8x$
3. $a := \sqrt{a}$
4. $a := a + 1$
5. $y := 3x$
6. $y := y / a$
7. запись y
8. конец

В данном случае удалось обойтись лишь одной вспомогательной переменной a , которая используется в процессе счета для запоминания промежуточного результата, последовательно вычисляемого в знаменателе. Для обозначения промежуточного результата, полученного в числителе, с успехом используется «ответная» переменная y .

Весь механизм происходящих при этом присваиваний наглядно вскрывается в процессе проигрывания алгоритма (см. табл. 33).

Таблица 33

Шаги алгоритма	Аргумент	Промежуточная величина		Результат	Пояснение
		x	a		
1	2				чтение x
2			16		
3			4		
4			5		
5				3	
6				1,2	
7					запись 1,2
8					остановка

Теряющиеся при повторном присваивании прежние значения переменных *a* и *у* для наглядности перечеркнуты. При проигрывании вычислительных действий целесообразно использовать микрокалькулятор.

Контрольные вопросы

1. Каково назначение предписаний «чтение» и «запись» в описании алгоритмов? Какие величины в записи алгоритма называют входными (аргументами) и выходными (результатами)?
2. Как при исполнении словесной записи алгоритма производится выбор очередного предписания?
3. Когда заканчивается процесс исполнения алгоритма в словесной записи?
4. Какую роль играют в записи алгоритмов вспомогательные переменные?
5. Каким путем в записи алгоритмов достигается экономия вспомогательных переменных?

Упражнения

1. Привести примеры словесной записи «бытовых» алгоритмов с четким выделением отдельных предписаний (например, пользование автоматом «газвода», пользование междугородным телефоном-автоматом, запуск двигателя автомобиля и т. п.).
2. Составить словесные записи (с выделением всех элементарных арифметических операций и функций и с использованием знака присваивания) алгоритмов вычислений по формулам:

$$a) \ y = \frac{2x - 3}{\sqrt{x} + 1}; \quad b) \ y = \frac{\ln(x^2 + 1)}{2x^3 - x + 5}.$$

Проиграть полученные записи алгоритмов для конкретных значений исходных данных, используя для вычислений микрокалькулятор.

3.6. Запись ветвлений

В предыдущем пункте были рассмотрены правила организации словесной записи линейных алгоритмов. Большой интерес с точки зрения алгоритмизации представляют разветвляющиеся алгоритмические процессы, причем в данном случае вызывает особую сложность организация переходов от предписания к предписанию.

Дело в том, что в разветвляющихся алгоритмах принцип линейного автоматического перехода от предписания к предписанию в порядке естественного возрастания их номеров уже не является всеобщим, так как время от времени возникает потребность перехода к предписанию с произвольными, а необяза-

тельно следующими по порядку номерами. В схемах алгоритмов все переходы организуются одним способом — с помощью стрелок. В словесной записи алгоритмов наряду с механизмом линейной передачи управления используются также и предписания специального вида — для организации так называемых безусловных и условных переходов.

Предписание *безусловного перехода* имеет вид:

идти к N,

где N — номер одного из предписаний в записи алгоритма. Само по себе предписание *безусловного перехода* никаких действий не вызывает, кроме передачи управления предписанию, номер которого явно указан в предписании *безусловного перехода*. Например, вслед за предписанием «идти к 5» будет выполняться предписание с номером 5. Чтобы не занимать отдельных строк, предписание *безусловных переходов* будем размещать в строках вслед за другими предписаниями, отделяя их точкой с запятой.

Предписание *условного перехода* соответствует логическому элементу схем алгоритмов (один вход, два выхода) и имеет вид:

если P идти к N*.

Здесь P — проверяемое условие, N — номер предписания, к которому происходит переход, когда условие P соблюдается (истинно). Условием P может быть предложение с одной или несколькими переменными или какое-либо другое содержательное логическое условие (в случае записи нематематических алгоритмов). Если условие P ложно, происходит переход к предписанию, расположенному вслед за предписанием *условного перехода* в порядке возрастания номеров. Пусть, к примеру, текущее значение переменной *x* равно 2 и нужно выполнить предписание.

4. если $x < 0$ идти к 7

Условием P в данном случае является неравенство $x < 0$, которое при $x = 2$ будет ложным. Следовательно, данное предписание осуществит переход (или, как говорят еще, передаст управление) не на предписание 7, а на предписание, следующее за предписанием с номером 4, т. е. на предписание с номером 5. Полезно обратить внимание, что здесь фактически срабатывает тот же принцип, который был молчаливо принят нами в предыдущем параграфе (см. с. 99) для линейных алгоритмов: если при исполнении текущего предписания алгоритма номер очередного предписания не определяется, то происходит автоматический переход к предписанию, следующему за данным в порядке возрастания номеров.

* Согласно правилам пунктуации русского языка в этой фразе после P должна стоять запятая. В записи алгоритмов отсутствие этого знака препинания никаких неприятностей не сулит, поэтому мы будем его опускать. Так поступают и в официальных алгоритмических языках, где экономия используемых символов является предметом особой заботы.



Рис. 29

Пример 3.6.1. Составить словесную запись алгоритма решения уравнения $ax=5$ (a — произвольное действительное число).

На первых порах перед составлением словесной записи алгоритма удобно предварительно вычерчивать его схему. В данном случае вся суть алгоритма сводится к проверке условия $a=0$ (если оно не соблюдается, решение находится trivialно: $x=5/a$; если же $a=0$, ни одно x не удовлетворяет решению). Схема искомого алгоритма изображена на рисунке 29.

Итак, вслед за чтением значения исходного данного a (в схеме алгоритма нет такого блока) нужно сразу приступить к проверке условия $a=0$. Появляется следующее начало будущего алгоритма:

1. чтение a
2. если $a=0$ идти к ...

Многоточие здесь означает, что нам пока неизвестен номер предписания, к которому нужно будет переходить при выполнении условия $a=0$. Однако на основании правил выполнения предписания условного перехода нам известно, что в случае несоблюдения условия $a=0$ (т. е. при соблюдении условия $a \neq 0$) исполнитель перейдет к предписанию с номером 3,— вот эту логическую ветвь мы и станем пока продолжать. Учитывая, что при $a \neq 0$ решением уравнения будет $x=5/a$, напишем продолжение алгоритма так:

1. чтение a
2. если $a=0$ идти к ...
3. $x = 5 / a$
4. запись x ; идти к ...

После записи решения x будет произведен переход к окончанию алгоритма (соответствующее предписание мы разместим в конце, так что его номер тоже пока неизвестен).

Теперь предписание, следующее за предписанием 4, можно использовать для выполнения действий алгоритма при $a=0$. Это позволяет исключить многоточие в предписании 2. Окончание записи алгоритма:

1. чтение a
2. если $a=0$ идти к 5
3. $x := 5 / a$

4. запись x ; иди к 6
5. запись "решений нет"
6. конец

Предписание «конец» получило номер 6, поэтому строка 4 заканчивается предписанием «идти к 6». А в строке 5 такого предписания не потребовалось, так как переход к окончанию алгоритма в данном случае произойдет автоматически.

Сделаем замечание относительно предписания 5, имеющего вид:

5. запись "решений нет"

Это предписание выводит результат не в виде числового ответа, а в форме текстового сообщения. Во всех таких случаях в записи предписания выводимый текст берется в кавычки. В этом смысле, например, предписания

запись x и
запись " x "

приведут к совершенно различным действиям исполнителя. В первом случае будет выведено значение переменной x (число), а во втором — сам символ x . Будем впредь считать, что исполнителю алгоритмов понятны предписания такого типа, он умеет их различать и правильно исполнять. Учитывая сделанное соглашение, можно добиваться выдачи комбинированной информации — и текстов, и чисел. Так, например, если текущее значение x равно 12, то результатом выполнения предписания

запись " $x =$ ", x

будет текст:

$$x = 12.$$

Элементы в строке «запись» (" $x =$ " и x) разделены запятыми.

Таблица 34

Шаги алгоритма	Аргумент	Результат	Пояснение
1	20		чтение a
2			$20 = 0$ (ложно)
3			запись 0,25
4			остановка
6			
1	0		чтение a
2			$0 = 0$ (истинно)
5			запись: решений нет
6			остановка

Для понимания особенностей исполнения разветвляющихся алгоритмов особенно полезно обратиться к процедуре исполнения. В таблице 34 показаны примеры исполнения алгоритма решения уравнения $ax=5$ для двух случаев: $a=25$ и $a=0$. Из первого столбца таблицы хорошо видно, что получение результата решения задачи в каждом из этих двух случаев алгоритм обеспечивает разными путями: в первом случае «работают» предписания 1, 2, 3, 4, 6, во втором — 1, 2, 5, 6.

При мер 3.6.2. Составим словесную запись алгоритма решения уравнения $ax=b$ (см. рис. 28, с. 98):

1. чтение a, b
2. если $a=0$ идти к 5
3. $x:=b/a$
4. запись x ; идти к 8
5. если $b=0$ идти к 7
6. запись "решений нет"; идти к 8
7. запись " x — любое"
8. конец

При составлении словесных записей алгоритмов с несколькими разветвлениями требуется еще больше внимания и навыка организации таких записей.

Приведенную выше запись алгоритма полезно проиграть для различных сочетаний исходных данных a и b , приводящих к каждому из трех возможных случаев решения.

Контрольные вопросы

1. Как записываются и как исполняются предписания безусловного и условного перехода?
2. Что является общим в правилах перехода от предписания к предписанию в линейных алгоритмах и в правилах выполнения предписания условного перехода?
3. Каким образом с помощью предписания «запись» может быть выдано текстовое сообщение?

Упражнения

Составить схемы и словесные записи алгоритмов решения следующих задач:

1. Вычисление абсолютной величины числа.
- Указание. Воспользоваться определением абсолютной величины:

$$|x| = \begin{cases} x, & \text{если } x \geq 0, \\ -x, & \text{если } x < 0. \end{cases}$$

2. Решение неравенства $ax > 7$.
3. Решение уравнения $ax^2 + bx + c = 0$ ($a \neq 0$).
4. Решение неравенства $ax > b$.

3.7. Циклические алгоритмы

При составлении алгоритмов решения практических задач не редко возникают случаи, когда приходится неоднократно повторять одни и те же предписания.

Пример 3.7.1. Пусть требуется вычислить несколько значений функции $y = 2x^3 - 5$ для значений x , начиная с $x = 1$ и с шагом 0,5. Последовательность необходимых для этого действий может быть записана так:

1. $x := 1$
2. $y := 2x^3 - 5$
3. запись x, y
4. $x := x + 0,5$
5. $y := 2x^3 - 5$
6. запись x, y
7. $x := x + 0,5$
8. $y := 2x^3 - 5$

и т. д.

Желая получить значения функции для последующих значений x , мы должны были бы и дальше записывать абсолютно одинаковые тройки предписаний, таких, как тройки с номерами 2, 3, 4 или с номерами 5, 6, 7 и т. д. Этого можно, однако, и не делать, если попытаться предъявить исполнителю одну и ту же тройку предписаний несколько раз. Используем для этой цели предписание безусловного перехода:

1. $x := 1$
2. $y := 2x^3 - 5$
3. запись x, y
4. $x := x + 0,5$
5. идти к 2

Действительно, в результате выполнения написанной группы предписанного получается тот же, как если бы исполнитель имел перед собой выписанную подряд последовательность троек предписаний, таких, как предписания с номерами 2, 3 и 4. То, что при этом происходит, наглядно видно из схемы, изображенной на рисунке 30.

Мы получили пример так называемого циклического алгоритма. В процессе исполнения этого алгоритма (например, методом обычного проигрывания с микрокалькулятором в руках) быстро устанавливается, что он, по существу, правильно решает поставленную задачу, но обладает существенным недостатком.

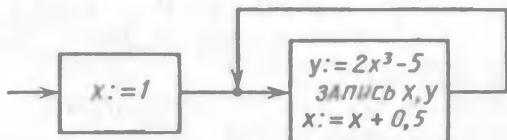


Рис. 30

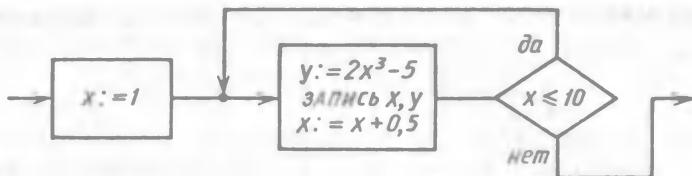


Рис. 31

статком — никогда не приводит к остановке! Такого рода циклы обычно принято называть бесконечными, они не удовлетворяют требованиям, предъявляемым к алгоритмическим предписаниям. Недостаток этот, однако, легко устраним. Пусть требуется вычислить значения функции с заданным шагом изменения аргумента на ограниченном справа отрезке, например на отрезке $[1; 10]$. Дополним схему алгоритма логическим блоком так, как показано на рисунке 31, т. е. в каждом «обороте» цикла будем устраивать проверку условия $x \leq 10$. Как только оно перестанет соблюдаться, процесс прекращается. Словесная запись этого алгоритма приведена ниже.

1. $x := 1$
2. $y := 2x^3 - 5$
3. запись x, y
4. $x := x + 0,5$
5. если $x \leq 10$ идти к 2
6. конец

Полученный алгоритм является типичным примером простого циклического алгоритма с управляемым окончанием. Путем незначительных добавлений его можно преобразовать так, чтобы он был применим на любом отрезке $[a; b]$ из области определения функции и с любым шагом h . Для этого достаточно ввести предписание «чтение», с помощью которого значения параметров a, b и h будут устанавливаться по желанию пользователя:

1. чтение a, b, h
2. $x := a$
3. $y := 2x^3 - 5$
4. запись x, y
5. $x := x + h$
6. если $x \leq b$ идти к 3
7. конец

Как и при первом ознакомлении с разветвляющимися алгоритмами, лучше понять характер циклических алгоритмов помогает использование процедуры проигрывания. В таблице 35 показан процесс исполнения алгоритма вычислений значений функции $y = 2x^3 - 5$ на отрезке $[2; 3]$ с шагом 0,5 (т. е. $a = 2$, $b = 3$, $h = 0,5$). Содержимое первого столбца таблицы 35 наглядно демонстрирует основное отличительное свойство циклических алгоритмов: количество исполняемых в процессе выпол-

Таблица 35

Шаги алгоритма	Аргументы			Результаты		Пояснения
	<i>a</i>	<i>b</i>	<i>h</i>	<i>x</i>	<i>y</i>	
1	2	3	0,5			чтение <i>a</i> , <i>b</i> , <i>h</i>
2				2		
3					и	запись 2; 11
4						$2 \leq 3$ (истинно)
5						запись 2,5; 26; 25
6						$3 \leq 3$ (истинно)
7						запись 3; 49
3						$3,5 \leq 3$ (ложно)
4						остановка
5						
6						
7						

нения циклического алгоритма предписаний может существенно перекрывать количество предписаний, из которых составлена запись такого алгоритма.

К циклическим алгоритмам сводится решение многих задач, и не только вычислительных.

Пример 3.7.2. В урне хранится некоторое количество черных и белых шаров. Требуется сделать запись алгоритма рассортировки этих шаров по двум корзинам (черного и белого цвета) так, чтобы в результате выполнения алгоритма белые шары оказались в белой корзине, черные — в черной.

Составим сначала схему алгоритма выполнения этой работы.

Очевидно, что для рассортировки шаров по корзинам с каждым шаром должны быть проделаны следующие действия: вынуть шар из урны (в соответствии с условием задачи предполагаем, что урна с самого начала не пуста), определить цвет вынутого шара и опустить его в соответствующую корзину. Если после этого в урне еще остаются шары, описанные выше действия повторить еще раз и т. д. Схема искомого алгоритма изображена на рисунке 32. Можно заметить, что группа повторяющихся в цикле блоков образует довольно-таки сложную конструкцию, так как включает разветвление.

Пользуясь схемой, легко составить словесную запись этого алгоритма:

1. вынуть из урны один шар
2. если шар белый, идти к 4

3. опустить шар в черную корзину; идти к 5
4. опустить шар в белую корзину
5. если урна не пуста, идти к 1
6. конец

Циклический характер имеют многие алгоритмы школьного курса математики.

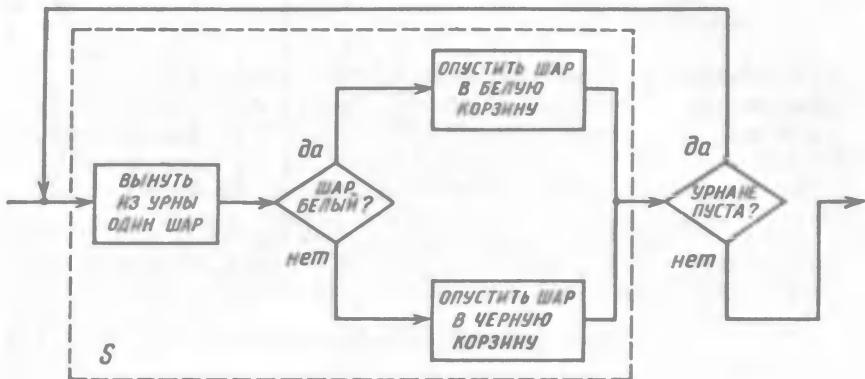


Рис. 32

При мер 8.7.3. Составить запись алгоритма Евклида нахождения наибольшего общего делителя (НОД) двух целых положительных чисел a и b .

Воспользуемся известным алгоритмом вычитания, суть которого состоит в следующем: заданные числа сравниваются между собой и в случае равенства одно из них объявляется результатом. Если же числа не равны, то из большего числа вычитается меньшее, после чего большее число заменяется полученной разностью. Вслед за этим процесс повторяется: снова числа сравниваются между собой и т. д.

Циклический характер этого алгоритма хорошо виден из его схемы, изображенной на рисунке 33.

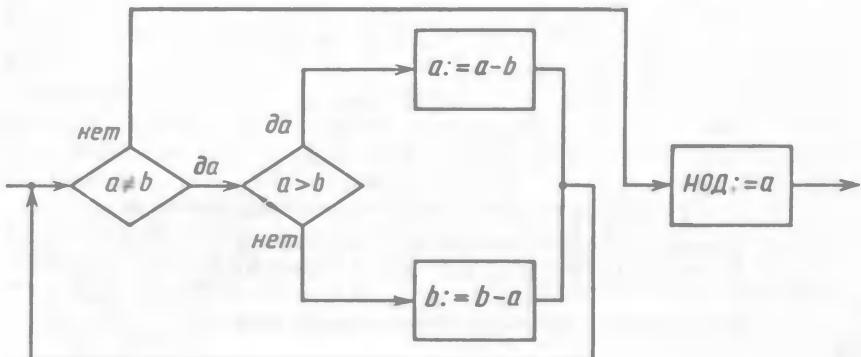


Рис. 33

Словесная запись:

1. чтение a, b
2. если $a=b$ идти к 6
3. если $a>b$ идти к 5
4. $b:=b-a$; идти к 2
5. $a:=a-b$; идти к 2
6. запись "НОД =", a
7. конец

В таблице 36 приведено выполнение этого алгоритма для исходных значений $a=12$, $b=18$.

Таблица 36

Шаги алгоритма	a	b	Письменное
1	12	18	чтение a, b
2		12	$12=18$ (ложно)
3		18	$12>18$ (ложно)
4		6	
2		6	$12=6$ (ложно)
3		6	$12>6$ (истинно)
5	6		
2	6		$6=6$ (истинно)
6			запись: НОД=6
7			остановка

Массу интересных и полезных с точки зрения не только алгоритмизации, но и углубления понятий курса школьной математики дает тема курса алгебры «Последовательности».

Пример 3.7.4. Составить алгоритм вычисления суммы первых 20 членов последовательности с общим членом $a_k = \frac{k+1}{(2k+1)^2}$.

Для циклического накапливания сумм при составлении соответствующих алгоритмов используется предписание стандартного вида:

сумма := *сумма* + *слагаемое*

Если повторять такое предписание требуемое количество раз, изменяя соответствующим образом *слагаемое*, то и будет получена искомая *сумма*. Понятно, что *сумма* перед началом работы цикла должна иметь нулевое значение. В схеме, изображенной на рисунке 34, роль *суммы* выполняет переменная S , а роль *слагаемого* — формула общего члена последовательности $(k+1)/(2k+1)^2$. Изменение *слагаемого* достигается увеличением в каждом обороте цикла номера члена k на единицу. Словесная запись этого алгоритма:

1. $k:=1; S:=0$
2. $S:=S+(k+1)/(2k+1)^2$
3. $k:=k+1$
4. если $k \leq 20$ идти к 2
5. запись " $S=$ ", S
6. конец

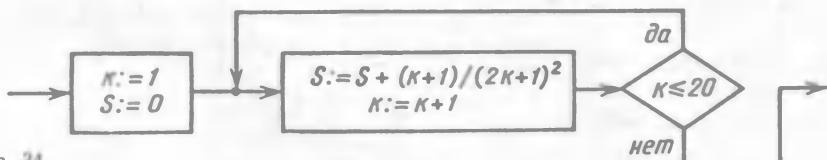


Рис. 34

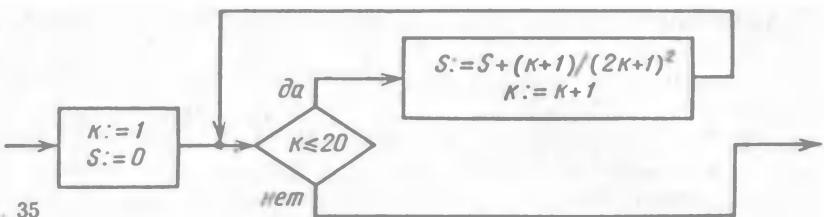


Рис. 35

В этом циклическом алгоритме управление окончанием цикла осуществляется проверкой условия $k \leq 20$, и производится эта проверка всегда после выполнения рабочей части цикла (см. рис. 34). Такие циклы называют циклами с *пост-условием*. Этот же алгоритм можно организовать иначе: поставить проверку окончания перед рабочей частью цикла (рис. 35). Такие циклы называют циклами с *пред-условием*. Ниже приведена словесная запись алгоритма, соответствующего схеме, изображенной на рисунке 35.

1. $k := 1; S := 0$
2. если $k > 20$ идти к 5
3. $S := S + (k+1)/(2k+1)^2$
4. $k := k + 1$; идти к 2
5. запись " $S =$ ", S
6. конец

При составлении предписания условного перехода оказалось удобнее взять не условие $k \leq 20$, а его отрицание $k > 20$.

В рамках рассмотренной задачи нахождения суммы 20 членов последовательности оба приведенных алгоритма равносильны в том смысле, что они дадут одинаковый результат. Впрочем, нужно отдавать себе отчет в том, что одинаковые по смыслу алгоритмы, составленные с пред-условием и пост-условием, все-таки различны. Очевидно, что рабочая часть (или, как еще говорят, *тело*) циклов, построенных по схеме (см. рис. 34), всегда выполнится по меньшей мере один раз независимо от результатов проверки условия. В то же время тело цикла с пред-условием в определенной ситуации может не выполниться ни разу. Если допустить, что в задачах, подобно рассмотренной выше, верхняя граница параметра k заранее не известна и определяется, скажем, в результате предыдущего счета, то может оказаться, что алгоритм с пост-условием в этой ситуации даст нелепый результат. Действительно, если окажется, например, что заданная граница значения параметра k неположительна, то последовательность становится неопределенной. А алгоритм с пост-условием в этом случае выдаст в качестве значения суммы последовательности ее первый член. Подобные ситуации требуют особого внимания при составлении программ, суммирующих последовательность.

Контрольные вопросы

1. В чем заключается основное отличительное свойство циклических алгоритмов?
2. Каково принципиальное различие между циклами с предусловием и пост-условием?

Упражнения

1. Составить алгоритм решения следующих задач, используя циклы с пред-условием и пост-условием:

- а) вычисление значений функции $f(x)=3\sin 2x$ на отрезке $[a; b]$ с шагом h ;
- б) вычисление 30 первых членов последовательности с общим членом

$$a_k = \frac{2k+1}{3k^2-1};$$

- в) вычисление суммы 50 членов последовательности с общим членом

$$a_k = \frac{2k}{k^2+1}.$$

Проиграть полученные алгоритмы для двух-трех первых оборотов цикла.

2. Составить схему и словесную запись алгоритма вычисления произведения 20 членов последовательности с общим членом $a_k = \frac{2k+1}{3k-2}$, используя циклы с пред-условием и пост-условием.

Указание. Для циклического получения значений произведений меняющихся множителей необходимо начальным значением произведения сделать 1, а затем повторять требуемое количество раз предписание вида:

произведение := произведение × множитель

3.8. Построчная алгоритмическая нотация

Как следует из всего предыдущего изложения, задача составления алгоритма всегда связана с использованием некоторого языка, на котором алгоритм записывается. В достаточно общем случае язык — это средство выражения и передачи некоторых сведений. В этом смысле можно говорить о языке общения между людьми, о языке математики, о языке общения между людьми и автоматами. Можно говорить даже о существовании особых языков, на которых автоматы общаются между собой.

Выше уже отмечалось, что характер языка, используемого для записи алгоритмов, так или иначе связан с возможностями исполнителя, на которого ориентируются эти описания. Можно

сказать, что возможности исполнителей выделяют различные уровни языковых средств; обсудим сейчас некоторые основные характеристики языков для записи алгоритмов.

Говоря об уровне языка, мы имеем здесь в виду, в частности, степень детализации предписаний в алгоритмической записи (т. е. фактический уровень «элементарности» действий, считающихся для данного исполнителя элементарными). Если, например, исполнитель, ориентированный на выполнение вычислительных алгоритмов, способен «за один шаг» производить вычисление корня квадратного, то для него будет понятно указание: $y := \sqrt{x}$. Если же вычислительные возможности исполнителя ограничены рамками четырех основных арифметических действий, то для нахождения корня квадратного такому исполнителю придется предъявить более подробную систему указаний, использующую только те операции, которые исполнителю под силу.

Говоря об уровне языка, имеют в виду также и такое их свойство, особенно важное с точки зрения машинного исполнения алгоритмов, как степень *формализации*. В тех случаях, когда исполнителем алгоритмов является человек, средства языка, используемого для представления этих алгоритмов, обычно строго не ограничены, так как учитывают «здравый смысл». По усмотрению составителя алгоритмов в процессе описания могут привлекаться любые общепонятные выражения и словесные обороты, принятые в естественном человеческом языке. Если же алгоритмическое описание ориентируется, например, на ЭВМ, приходится выполнять ряд обязательных требований:

- при формулировании предписаний можно требовать от машины выполнения лишь тех операций, которые для данной машины жестко определены;
- можно использовать лишь принятые для языка данной машины правила построения предписаний;
- ничего не предусмотренного правилами использовать нельзя, ибо машина все равно такие предписания не выполнит.

Эти требования, которые обычно могут быть сведены в строгую систему формальных правил, образуют синтаксис языка, сам язык в подобных случаях становится, как говорят, формализованным.

Как известно, естественный человеческий язык, не являясь языком формализованным, иногда может содержать выражения, смысл которых может быть истолкован неоднозначно. Подобные явления должны быть полностью исключены в языках, используемых для общения с автоматами. Машину трудно вооружить таким опытом, каким обладает человек, поэтому для общения с машиной приходится придумывать специальный язык, сообщения на котором могли бы истолковываться лишь строго однозначно и были бы доступны ее восприятию.

Само собой разумеется, что формализация языка обедняет его выразительные возможности по сравнению с естествен-

ным языком человека. Таким языком становится неудобно пользоваться, ибо каждый раз нужно помнить о строгих правилах образования предложений. Тем не менее такие языки создаются, и человеку приходится пользоваться ими, если он намерен управлять действиями используемых автоматов.

Таким образом, поскольку одним из пользователей языка общения с машиной так или иначе становится человек, говоря об уровне языка, имеют в виду также и уровень *доступности* языка для человека. Общение с первыми электронными вычислительными машинами проходило на уровне языка цифр. В дальнейшем массовое распространение ЭВМ потребовало новых форм общения между лицом, сформулировавшим задачу, и машиной. Однако требование доступности языка находится в противоречии с требованием его формализации, поэтому проблема «очеловечивания» языка, который в то же время оставался бы понятным и для машины,— сложная проблема. Проблема эта решается параллельно с совершенствованием самих машин, и к настоящему времени на этом пути имеются ощутимые результаты. К числу языков, построенных на сочетании некоторых элементов обычного человеческого языка и языка математики, относятся, например, широко известные алгоритмические языки Алгол, Фортран, Бейсик, Паскаль и др., ориентированные на описание алгоритмов решения широкого класса научных и инженерно-технических задач.

В п. 5—7 мы познакомились с основными правилами организации построчных словесных записей алгоритмов всех основных типов: линейных, разветвляющихся, циклических. Эти правила вместе с теми изобразительными средствами, которые используются для записи алгоритмов, образуют своеобразный алгоритмический язык (алгоритмическую нотацию). Этот язык не формализован, пока еще свободен от целого ряда «машинных» стандартов, а привлекаемые для записи алгоритмов изобразительные средства предоставляют большую свободу составителю алгоритмов. В то же время такой язык может включать в себя все основные общеобразовательные свойства «настоящих» алгоритмических языков, что делает его удобным инструментом для обучения основам алгоритмизации. Проведем сейчас систематизацию и уточнение основных понятий и изобразительных средств построчной алгоритмической нотации, рассмотренной в предыдущих параграфах.

При определении алгоритмического языка должны быть описаны, как минимум, следующие основные понятия: из каких символов состоит алфавит, как из чисел, переменных конструируются выражения, каков перечень, назначение и характер исполнения различного вида предписаний, каковы правила общей организации алгоритмической записи и т. п.

При описании алгоритмического языка прежде всего определяется его *алфавит*, т. е. перечень символов (знаков), разре-

шенных к использованию в данном языке. Алфавит учебной алгоритмической нотации специально не ограничен, в него в зависимости от характера описываемых алгоритмов и по желанию составителя по необходимости могут быть введены любые понятные школьнику символы: строчные и прописные буквы русского, латинского, греческого алфавита, математические и прочие специальные знаки, которые могут понадобиться при записи алгоритмов. Из алфавита выделяются несколько особых символов, которые называются служебными (или ключевыми) словами языка. Это следующие символы, введенные в рассмотрение в предыдущих пунктах: чтение, запись, если, идти к, конец. Хотя служебные слова и имеют вид обычных слов, в алгоритмическом языке они относятся к неделимым символам, т. е. рассматриваются в целом, без связи с буквами, которыми они написаны. Для того чтобы выделять служебные слова среди других слов языка, что делает запись алгоритмов более наглядной и понятной, их при письме подчеркивают (а в книгах обычно используют особый шрифт). Среди знаков алфавита особо популярны знаки арифметических операций $+$, $-$, \times , $/$, а также знаки операций отношений $>$, \geq , $=$, \neq , \leq , $<$.

Переменные величины и *числа* обозначаются и используются в учебном алгоритмическом языке в соответствии с обычным порядком их употребления в тех предметных областях, к которым относится запись алгоритма.

Никаких ограничений также не накладывается и на форму записи выражений — здесь разрешено использовать любые математические и логические знаки, а также подстрочные и надстрочные символы. Среди выражений выделяются *арифметические выражения*, значениями которых являются числа, например: $2x\sqrt{\sin^2 x+1}$. К *логическим выражениям* (или *условиям*) относятся прежде всего отношения, составляемые из двух арифметических выражений, соединенных знаком операции отношения, например: $2x < 3$, $a \neq b$, $x^2 + y^2 \geq r$ и т. п. Условиями могут быть не только математические отношения, но и отношения, описывающие самые разнообразные реальные ситуации, например: в трубке короткие гудки, сигнал светофора зеленый и т. п. При необходимости могут быть привлечены знаки алгебры логики, с помощью которых из отношений могут конструироваться более сложные составные логические выражения, например: $(2x > 3) \wedge (x \leq 8)$. Здесь \wedge — знак логической операции конъюнкции (И). Значениями логических выражений являются два значения — «истинно» и «ложно». В алгоритмической нотации используется и третий тип выражений — *текстовой* (или *литерный*), который использовался в предписаниях *запись*, например:

запись " $S =$ ", S

То, что заключено в кавычки, исполнитель алгоритма должен понимать как текст, который нужно записать.

Основными объектами в алгоритмической нотации являются операторы, которые в записи алгоритмов играют роль предложений. Выделяются два основных вида операторов: а) операторы общей обработки; б) операторы управления.

Оператор общей обработки представляет собой в общем случае конечное описание некоторого реально выполнимого элементарного действия. Столь широкое толкование этого вида операторов позволяет записывать средствами учебной алгоритмической нотации многие наглядные и поучительные алгоритмы из числа трудно формализуемых «бытовых» задач. Частным случаем оператора общей обработки является оператор присваивания, имеющий вид:

$$A := B,$$

где A — обозначение переменной, B — обозначение некоторого выражения (того же типа). Примеры операторов общей обработки:

снять трубку с рычага,
вынуть из урны один шар,

$$y := x^2 + 3x - 1.$$

Особую группу операторов общей обработки в алгоритмической нотации составляют операторы чтения исходных данных и записи результатов:

чтение a, b, \dots, c и
запись x, y, \dots, z .

Здесь чтение и запись — служебные слова языка, а a, b, \dots, z — обозначения переменных. Предполагается, что в результате выполнения оператора чтение всем величинам, перечисленным в нем через запятую, присваиваются числовые значения, заранее заготовленные для этого конкретного исполнения алгоритма. Точно так же каждое исполнение оператора запись означает фиксирование каким-либо способом (например, посредством записи на бумаге) тех конкретных числовых значений перечисленных в операторе величин, которые эти величины имеют в момент выполнения оператора. Как видно, использование операторов чтения данных и записи результатов делает завершенным весь процесс выполнения алгоритмов — от задания исходным данным конкретных числовых значений до выдачи полученных результатов.

К операторам управления относятся: оператор безусловного перехода, оператор условного перехода и оператор остановки.

Оператор безусловного перехода записывается по форме

идти к N,

где идти к — ключевые слова языка (слово идти для сокращения записи может опускаться), а N — обозначение целочисленной метки, которой снабжаются операторы (или строки) в записи

алгоритма. После выполнения оператора безусловного перехода следующим всегда выполняется оператор, помеченный меткой N.

Оператор условного перехода имеет вид:

если Р идти к N,

где если, идти к — ключевые слова, Р — условие, N — номер оператора (строки), к которому происходит переход, когда условие Р истинно. Оператор условного перехода также допускает сокращенную форму записи: если Р к N.

Прекращение процесса исполнения алгоритма происходит после перехода к оператору остановки, записываемому с помощью служебного слова конец.

Сделаем теперь уточнения в связи с оформлением записи алгоритмов.

Операторы в записи алгоритма располагаются по строкам, которые нумеруются последовательными номерами, начиная с единицы. Если один оператор не вмещается на одной строке, он может быть перенесен на новую строку, однако эта новая строка уже не получает номера. Операторы общей обработки можно размещать в одной строке более чем по одному (если они записываются коротко), в этом случае операторы отделяются друг от друга точкой с запятой. Среди операторов строки может быть и оператор безусловного перехода, если только он стоит последним. Например, одна строка в записи алгоритма может иметь вид:

S := S + a; n := n + 1; к 2

В этой строке последовательно размещены три оператора, которые выполняются в порядке их следования слева направо. Номер строки отделяется точкой, в конце строки никакого знака не ставится. Оператор условного перехода обычно занимает отдельную строку, но может (если это позволяет логика алгоритма) размещаться на месте последнего оператора в строке.

Для облегчения ссылок на ранее составленные алгоритмы будем, кроме того, снабжать каждый алгоритм заголовком, который располагается всегда перед первой строкой в записи алгоритма и имеет вид:

Алгоритм ИМЯ,

где Алгоритм — это служебное слово языка, а ИМЯ — краткое название алгоритма, отражающее его содержание (часто назначением может служить какое-нибудь символическое или мнемоническое обозначение). Пример заголовка:

Алгоритм НОД

Рассмотрим теперь примеры записи алгоритмов с учетом принятых выше уточнений и соглашений относительно правил использования построчной алгоритмической нотации.

Пример 3.8.1. Алгоритм БОЛЬШЕЕ ИЗ ТРЕХ (коротко БИТ) поиска большего из трех чисел:

Алгоритм БИТ

1. чтение a, b, c
2. если $a < b$ к 4
3. $y := b$; к 5
4. $y := a$
5. если $y > c$ к 7
6. $y := c$
7. запись y
8. конец

Пример 3.8.2. Алгоритм решения квадратного уравнения (КВУР) $ax^2 + bx + c = 0$ ($a \neq 0$) в области действительных чисел:

Алгоритм КВУР

1. чтение a, b, c
2. $D := b^2 - 4ac$
3. если $D \geq 0$ к 5
4. запись "решений нет"; к 9
5. $x_1 := (-b - \sqrt{D}) / (2a)$
6. $x_2 := (-b + \sqrt{D}) / (2a)$
7. запись " $x_1 =$ ", x_1
8. запись " $x_2 =$ ", x_2
9. конец

Пример 3.8.3. Приведем один из простейших вариантов записи алгоритма перехода через улицу, который дает пример организации цикла:

Алгоритм ПЕРЕХОД УЛИЦЫ

1. подойти к перекрестку и определить значение (цвет) сигнала светофора
2. если сигнал зеленый к 4
3. подождать смены сигнала, к 2
4. перейти улицу (на первой половине пути смотреть налево, на второй — направо)
5. конец

Пример 3.8.4. Символ $n!$ (читается « n факториал») используется для обозначения произведения последовательных натуральных чисел от 1 до фиксированного числа n , т. е.

$$n! = 1 \cdot 2 \cdot 3 \cdots \cdot n.$$

Составим алгоритм вычисления факториала по заданному значению n .

Искомый алгоритм является циклическим. Действительно, если, положив вначале $N = 1$ и изменяя последовательно параметр

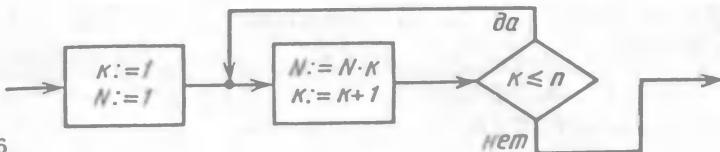


Рис. 36

k от 1 до n , выполнить n раз оператор $N := N \cdot k$, то значением переменной N как раз и будет $n!$. Схема искомого алгоритма, организованного как цикл с пост-условием, приведена на рисунке 36. Этот же алгоритм в построчной алгоритмической нотации:

Алгоритм ФАКТ

1. чтение n
2. $k := 1; N := 1$
3. $N := N \cdot k; k := k + 1$
4. если $k \leq n$ к 3
5. запись " $n! =$ ", N
6. конец

В таблице 37 описан процесс исполнения алгоритма ФАКТ для $n=3$.

Таблица 37

Шаги алгоритма	Аргумент		Промежуточные величины	Результат	Пояснения
	n	k			
1	3				чтение n
2		1		1	
3		2		2	$2 \leq 3$ (истинно)
4		3		6	$3 \leq 3$ (истинно)
3		4			$4 \leq 3$ (ложно)
4					запись $n! = 6$
5					

Контрольные вопросы

1. Какими характеристиками определяются различные уровни языковых средств, используемых для записи алгоритмов?
2. Какими основными понятиями определяется алгоритмический язык?
3. Как составляются выражения, каковы их типы?
4. Как определяется алфавит учебного алгоритмического языка?
5. Какие виды операторов используются в построчной алгоритмической нотации, как они записываются и действуют?

Упражнения

Составить записи алгоритмов решения следующих задач, используя построчную алгоритмическую нотацию:

1. Вычисление одного значения функции:

$$f(x) = \begin{cases} 3 \ln 4x \sqrt{x-1}, & \text{если } x \geq 2, \\ 2x^3 - \sin x, & \text{если } x < 2. \end{cases}$$

2. Решение уравнения $ax = b$.
3. Решение неравенства $ax > b$.
4. Вычисление значений функции $y = 2 \ln x - 3x^3 + 1$ на отрезке $[a, b]$ с шагом h ($a, h > 0$).
5. Вычисление НОД (a, b), где a, b — положительные целые числа.
6. Вычисление суммы и произведения n первых членов последовательности с общим членом $a_k = \frac{k+1}{2k^3-1}$.

3.9. Массивы

В подавляющем большинстве практических задач, связанных с обработкой числовой информации, приходится иметь дело с конечными числовыми последовательностями, образование членов в которых не всегда подчиняется какой-либо зависимости и не может быть связано с заранее заданной формулой, как это было в примерах предыдущего пункта. В общем случае элементы каждой такой совокупности чисел могут быть обозначены одной и той же буквой с различными индексами, например:

$$x_1, x_2, x_3, \dots, x_n.$$

Такие конечные совокупности чисел будем называть *массивами*. Число n в данном случае называют *длиной массива*. С обработкой массивов связаны многие задачи алгоритмизации.

Поскольку каждый элемент массива определяется через свой индекс, то в описании алгоритмов, связанных с массивами, индексы играют существенную роль. При описании алгоритмов индексы могут оказываться в роли переменных, значения которых определяют содержание описываемых действий. Например, оператор

$$y := x_i + 5$$

будет побуждать выполнение различных действий при различных значениях индекса i . Если, скажем, $i=2$, то этот оператор будет означать действие $y := x_2 + 5$, а при $i=10$ будет $y := x_{10} + 5$ и т. д. Прежде чем перейти к рассмотрению примеров, нам необходимо *установиться* о правилах оформления операторов чтения и записи, когда их выполнение связано с массивами.

Если исходными данными задачи является массив определенной длины, то чтение его элементов записывается так:

чтение $x_{1:50}$,

что означает в данном случае чтение последовательности 50 чисел и присвоение их значений в порядке поступления переменным x_1, x_2, \dots, x_{50} . Если же длина массива заранее не известна и определяется в процессе чтения, то чтение должно обязательно начинаться с чтения значения длины массива, например:

чтение $n, x_{1:n}$.

В данном случае сначала прочитывается число n , выражающее длину массива, потом в соответствии с этим прочитываются n очередных чисел, и их значения фигурируют затем в алгоритме под обозначениями x_1, x_2, \dots, x_n .

Аналогично оформляется и оператор запись, если среди результатов оказываются массивы. Так, например, оператор

запись $y_{1:m}$

означает запись последовательных чисел массива длины m (разумеется, это можно сделать лишь тогда, когда параметр m , так же как и все y_i при $i=1, 2, \dots, m$, получил перед этим определенные числовые значения). В заключение необходимо отметить, что вместе с массивами в состав операторов чтения и записи могут входить и обычные переменные. Например, оператором

чтение $a, b, x_{1:20}$

прочитываются вначале значения a и b , а затем 20 элементов массива: x_1, x_2, \dots, x_{20} .

Пример 3.9.1. Пусть имеется массив x_1, x_2, \dots, x_n . Требуется составить алгоритм вычисления среднего арифметического этих чисел, т. е.

$$S = \frac{x_1 + x_2 + \dots + x_n}{n}.$$

Для вычисления суммы элементов массива может быть применена обычная схема: сумма := сумма + слагаемое. Действительно, если вначале выполнить $S := 0$, а затем выполнять в цикле оператор $S := S + x_i$, изменяя при этом индекс i от 1 до n , то и будет получена сумма элементов массива. После этого остается разделить ее на n , и задача решена. Схема алгоритма (цикл с пост-условием) изображена на рисунке 37.

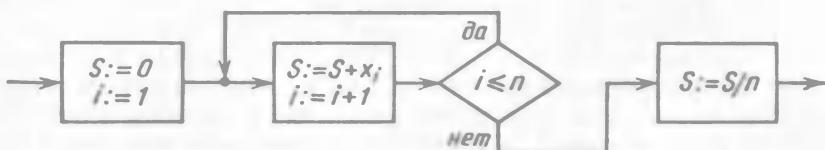


Рис. 37

Запись алгоритма в построчной нотации:

Алгоритм СРЕДАР

1. чтение n , $x_1:n$
2. $S := 0$; $i := 1$
3. $S := S + x_i$; $i := i + 1$
4. если $i \leq n$ к 3
5. $S := S/n$
6. запись " $S =$ ", S
7. конец

В таблице 38 содержится описание процесса выполнения алгоритма СРЕДАР для массива, состоящего из четырех элементов: 2,9; 3,6; 6,2; 4,3. Как видно из таблицы, в процессе исполнения алгоритма изменяют свои значения только переменные S и i .

Таблица 38

Шаги алгоритма	Аргументы					i	S	Пояснения
	n	x_1	x_2	x_3	x_4			
1	4	2,9	4,6	6,2	4,3			чтение данных
2						1	0	
3						2	2,9	
4								$2 \leq 4$ (истинно)
3						3	7,5	
4								$3 \leq 4$ (истинно)
3						4	12,7	
4								$4 \leq 4$ (истинно)
3							18,0	
4						5	4,5	$5 \leq 4$ (ложно)
5								запись
6								$S = 4,5$
7								остановка

Пример 3.9.2. Рассмотрим задачу вычисления значения многочлена степени n :

$$y = a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n.$$

Исходными данными для этой задачи являются массив коэффициентов a_0, a_1, \dots, a_n (его длина равна $n+1$) и значение аргумента x . Для составления алгоритма используем известную схему Горнера, суть которой рассмотрим на простых примерах.

Если при вычислении квадратного трехчлена $a_0x^2 + a_1x + a_2$ пользоваться непосредственно его формулой, то придется выполнить 5 арифметических действий (3 умножения и 2 сложения). Однако если учесть, что $a_0x^2 + a_1x + a_2 = (a_0x + a_1)x + a_2$, то потребуются уже только 4 действия (2 умножения и 2 сложения). Аналогично по формуле $y = a_0x^3 + a_1x^2 + a_2x + a_3$ нужно выполнить 9 действий, а по формуле $y = ((a_0x + a_1)x + a_2)x + a_3$ только 6. Выгода от применения рассмотренного подхода очевидна. В общем случае многочлен, представленный по схеме Горнера, имеет вид:

$$y = (\dots((a_0x + a_1)x + a_2)x + \dots + a_{n-1})x + a_n.$$

Однако особое удобство схемы Горнера состоит в том, что она хорошо укладывается в циклический процесс. Действительно, последовательное выполнение вычислений по этой схеме состоит в повторении двух основных операций: 1) умножения на x ; 2) прибавления очередного коэффициента. Схема этого процесса изображена на рисунке 38. Приводим также запись алгоритма ГОРНЕР в построчной нотации:

Алгоритм ГОРНЕР

1. чтение $x, n, a_{0:n}$
2. $i := 1; y := a_0$
3. $y := y \cdot x + a_i; i := i + 1$
4. если $i \leq n$ к 3
5. запись " $y =$ ", y
6. конец

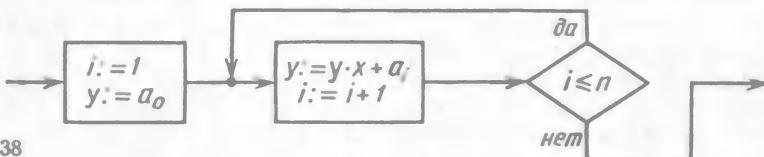


Рис. 38

Широкий класс практически важных задач алгоритмизации, связанных с массивами,— это задачи не столько вычислительного, сколько логического характера. К таким задачам относятся самые разные задачи обработки числовых массивов: сортировка, поиск, упорядочивание и др. Рассмотрим простейшие из них.

Пример 3.9.3. Пусть имеется массив длины n : x_1, x_2, \dots, x_n . Требуется составить алгоритм, выполнение которого обеспечивало бы запись большего элемента этого массива.

Воспользуемся следующей идеей. Выбирается переменная M и выполняется оператор $M := x_1$. После этого проверяется неравенство $M \geq x_i$ для всех $i = 2, 3, \dots, n$. В результате каждой проверки M или остается без изменения (если неравенство соблюдается), или заменяется на x_i (если неравенство не соблюдается). Легко понять, что, после того как таким способом

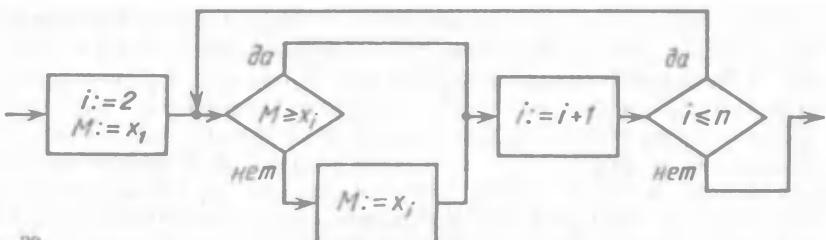


Рис. 39

будет просмотрен весь массив, переменная M будет иметь своим значением значение большего элемента. Схема алгоритма изображена на рисунке 39. Приводим запись алгоритма БЭМ (больший элемент массива) в построчной алгоритмической нотации:

Алгоритм БЭМ

1. чтение $n, x_{1:n}$
2. $i := 2; M := x_1$
3. если $M \geq x_i$ к 5
4. $M := x_i$
5. $i := i + 1$
6. если $i \leq n$ к 3
7. запись " $M =$ ", M
8. конец

Пример 3.9.4. Имеется массив a_1, a_2, \dots, a_n . Требуется переставить значения его элементов так, чтобы массив стал упорядоченным, например, по возрастанию, т. е. чтобы имело место $a_1 \leq a_2 \leq \dots \leq a_n$.

Существуют различные методы организации алгоритма упорядочивания, мы воспользуемся одним из них. Сравниваются два соседних элемента a_i и a_{i+1} , $i = 1, 2, \dots, n - 1$. Если $a_i > a_{i+1}$, то значения элементов меняются местами, в противном случае (когда $a_i \leq a_{i+1}$) значения элементов остаются прежними. Если в результате прохода слева направо происходит хотя бы одна перестановка, процесс повторяется.

Здесь показано применение метода упорядочивания для четырех чисел: 8, 3, 2 и 5. После сравнения первой же пары чисел происходит перестановка: 8 и 3 меняются местами, после чего просмотр начинается сначала. Теперь перестановка происходит после сравнения второй пары: 8 и 2. И снова просмотр начинается сначала и т. д. Процесс упорядочивания оказывается законченным, когда в результате просмотра массива слева направо не возникает потребности ни в одной перестановке. Схема алгоритма упорядочивания изобра-

8	\leftrightarrow	3	2	5
3		\leftrightarrow	2	5
3	\leftrightarrow	2	8	5
2		3	\leftrightarrow	5
2		3	5	8

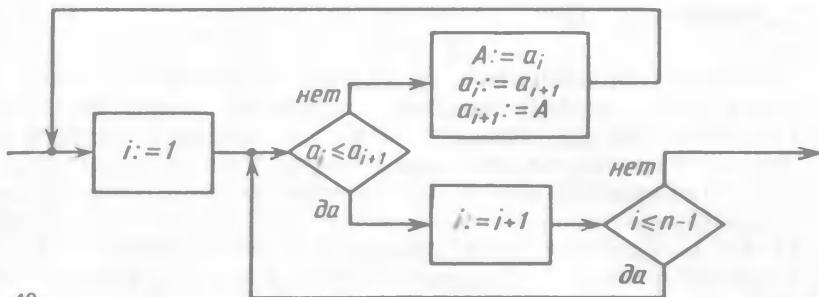


Рис. 40

жена на рисунке 40. Ниже приведена запись алгоритма упорядочивания массива (УМ) в построчной алгоритмической нотации:

Алгоритм УМ

1. чтение n , $a_1:n$
2. $i := 1$
3. если $a_i \leq a_{i+1}$ к 5
4. $A := a_i$; $a_i := a_{i+1}$; $a_{i+1} := A$; к 2
5. $i := i + 1$
6. если $i \leq n - 1$ к 3
7. запись $a_1:n$
8. конец

Результатом выполнения алгоритма является запись массива, упорядоченного по возрастанию.

Контрольные вопросы

1. Что такое массив? длина массива?
2. Как записываются и действуют операторы чтение и запись, работающие с массивами?

Упражнения

Составить схемы, записать в построчной алгоритмической нотации и выполнить для ограниченного набора исходных данных алгоритмы решения следующих задач:

1. Вычисление произведения всех элементов массива.
 2. Вычисление сумм положительных и отрицательных элементов массива.
 3. Проверка упорядоченности массива a_1, a_2, \dots, a_n по возрастанию.
 4. Задан массив a_1, a_2, \dots, a_n . Сформировать массив b_1, b_2, \dots, b_n , у которого $b_1 = a_1$, $b_2 = a_1 + a_2$, $b_3 = a_1 + a_2 + a_3$ и т. д.
- Указание. Воспользоваться рекуррентной формулой $b_i = b_{i-1} + a_i$, $i = 2, 3, \dots, n$.
5. Даны два упорядоченных по возрастанию массива a_1, a_2, \dots, a_m и b_1, b_2, \dots, b_n . Требуется слить эти массивы в один массив c_1, c_2, \dots, c_p (его длина будет $p = m + n$) так, чтобы он тоже был упорядочен по возрастанию.

3.10. Подчиненные алгоритмы

При записи алгоритмов могут использоваться алгоритмы, составленные раньше. Алгоритмы, целиком используемые в составе других алгоритмов, будем называть *подчиненными алгоритмами* или, коротко, *подалгоритмами*. В принципе не исключено, что алгоритм, содержащий в своем описании подчиненные алгоритмы, сам в определенной ситуации может выступать в роли подалгоритма. Составление и использование подалгоритмов находят широкое применение в практике алгоритмизации и являются одним из наиболее значительных и интересных в идейном отношении ее приемов. Действительно, такой подход позволяет исключить при записи алгоритмов повторение тех частей, описания которых уже имеются. Однако при этом возникают вопросы оформления подчиненных алгоритмов и техники включения их в основные алгоритмы в процессе их исполнения. Рассмотрим, как могут быть решены эти вопросы в рамках учебной алгоритмической нотации.

Оформление подчиненных алгоритмов несколько отличается от оформления обычных алгоритмов. Прежде всего в подалгоритмах отпадает потребность в использовании операторов чтения данных и записи результатов, так как вопросы ввода в действие исходных данных и фиксирование результатов должны будут решаться в том основном алгоритме, где подалгоритм будет использоваться.

Кроме этого, у подалгоритмов по-особому записывается заголовок, который в данном случае имеет вид:

Подалгоритм ИМЯ,

где Подалгоритм — служебное слово, а ИМЯ — краткое символическое обозначение подалгоритма. Вместо обычного оператора конец, прекращающего выполнение алгоритма, последним оператором подалгоритма является специальный управляющий оператор возврат, назначение которого состоит в осуществлении перехода к дальнейшему выполнению основного алгоритма. Для уточнения всего механизма обращения к подалгоритму рассмотрим сначала простой пример.

Пример 3.10.1. Алгоритм поиска большего из трех чисел (БИТ, см. пример 3.8.1 на с. 120) состоит фактически из двухкратного применения алгоритма поиска большего из двух чисел — сначала для данных a и b , а затем для y и c . Составим алгоритм БИТ посредством обращения к подалгоритму поиска большего из двух чисел (БИД).

Подалгоритм БИД может быть записан следующим образом:

Подалгоритм БИД

1. если $\alpha \geqslant \beta$ к 3
2. $\gamma := \beta$; к 4
3. $\gamma := \alpha$
4. возврат

При записи подалгоритма мы использовали параметры: α и β — для обозначения входных величин, γ — для обозначения результата. Понятно, что, обращаясь теперь к этому подалгоритму, мы каждый раз должны будем сначала соответствующим образом задавать значения переменным α и β , носителем результата после выхода из подалгоритма БИД всегда будет переменная γ . Для обращения к подалгоритму достаточно в соответствующей строке основного алгоритма поставить имя подалгоритма. С учетом сказанного запись алгоритма БИТ с использованием приведенного выше подалгоритма БИД может иметь вид:

Алгоритм БИТ — БИД

1. чтение a, b, c
2. $\alpha := a; \beta := b$
3. БИД
4. $\alpha := \gamma; \beta := c$
5. БИД
6. запись γ
7. конец

Механизм обращения к подалгоритму хорошо виден из таблицы проигрывания алгоритма БИТ для конкретных исходных данных ($a = 3, b = 9, c = 5$). В графе «Шаги» выделены отдельные места для записи номеров строк основного алгоритма и подалгоритма. Как видно, основной алгоритм (БИТ) в данном случае оказался совсем простым — он является обычным линейным алгоритмом.

Составляя подалгоритм БИД, мы умышленно ввели обозначения для переменных α , β и γ , не совпадающие с обозначе-

Таблица 30

Шаги алгоритма		Аргументы основного алгоритма			Аргументы подалгоритма		Результат	Пояснения
исх. алг.	подалг.	a	b	c	α	β	γ	
1		3	9	5				чтение данных
2								переход к БИД
3								$3 > 9$ (ложно)
	1							возврат к БИТ
	2							
	4							
4								переход к БИД
5								$9 > 5$ (истинно)
	1							возврат к БИТ
	3							
	4							
6								запись 9
7								остановка

ниями a , b , c , используемыми в основном алгоритме. Однако руководствовались мы при этом только методическими соображениями. На самом деле обозначения переменных в подалгоритме и основном алгоритме могут выбираться независимо одни от других. В этом можно убедиться, если заменить в подалгоритме БИД буквы α , β и γ , например, на a , b , c , а потом снова проиграть алгоритм БИТ по принципу, использованному в таблице 39.

Пример 3.10.2. Составить алгоритм поиска большего элемента в массиве x_1, x_2, \dots, x_n ,

используя подалгоритм БИД.

Действительно, большее в массиве из n чисел можно отыскать путем циклического применения подалгоритма нахождения большего из двух чисел — очередного элемента массива и результата предыдущего применения подалгоритма БИД (в схеме этого алгоритма, изображенной на рисунке 41, обращение к подалгоритму обозначено особым блоком — «флажком»). Приведем также запись алгоритма в построчной нотации:

Алгоритм БЭМ — БИД

1. чтение $n, x_1..n$
2. $i := 2; \alpha := x_i$
3. $\beta := x_i$
4. БИД
5. $\alpha := \gamma; i := i + 1$
6. если $i \leq n$ к 3
7. запись γ
8. конец

В определенных ситуациях оказывается удобнее использовать не рассмотренный выше тип подалгоритма, а так называемый подалгоритм с параметрами. Заголовок этого подалгоритма имеет вид:

Подалгоритм ИМЯ ($\alpha, \beta, \dots, \gamma$)

где $\alpha, \beta, \dots, \gamma$ — перечень формальных параметров. В этот перечень включаются переменные величины, используемые в алгоритме, но не все, а только те, которые являются в нем носи-

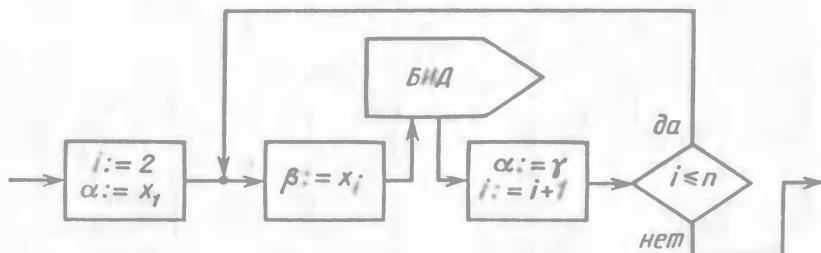


Рис. 41

телями значений исходных данных и результатов. Говоря иными словами, в список параметров включаются только аргументы и результаты (см. п. 5) подалгоритма, причем в этом списке перечисляются вначале имена аргументов, а потом имена результатов. Так, запись подалгоритма БИД (α , β , γ) будет отличаться от записи подалгоритма БИД только заголовком:

Подалгоритм БИД (α , β , γ)

1. если $\alpha > \beta$ к 3
2. $\gamma := \beta$; к 4
3. $\gamma := \alpha$
4. возврат

Как следует из смысла этой задачи, параметры α и β здесь являются именами аргументов, а γ — именем результата.

Для вызова подчиненного алгоритма с параметрами в нужном месте основного алгоритма указывается уже не только имя подалгоритма, но и следуемый за ним перечень фактических параметров, порядок размещения которых должен соответствовать их смыслу и назначению. Так, например, если бы потребовалось с помощью подалгоритма БИД (α , β , γ) найти большее из чисел c и d , а результат присвоить переменной z , то следовало бы написать следующий оператор вызова: БИД (c , d , z).

Оператор вызова подалгоритма с параметрами действует следующим образом. Происходит переход к первой строке подалгоритма, и одновременно с этим происходит замена формальных параметров подалгоритма фактическими (согласно порядку их размещения в списке). После этого обычным образом выполняется подалгоритм (но уже с новыми обозначениями аргументов и результатов!). С помощью оператора возврат происходит переход к строке основного алгоритма, следующей за обращением.

При мер 3.10.3. Составить алгоритм поиска большего из четырех чисел, используя подалгоритм БИД (α , β , γ).

Задача решается троекратным обращением к подалгоритму:

Алгоритм БИЧ

1. чтение a , b , c , d
2. БИД (a , b , y)
3. БИД (y , c , y)
4. БИД (y , d , y)
5. запись y
6. конец

При использовании подчиненных алгоритмов нужно отдавать себе отчет в том, что перед исполнением алгоритмов, включающих обращение к подалгоритмам, исполнителю алгоритма должны быть предъявлены записи всех необходимых алгоритмов: основного алгоритма, всех подчиненных алгоритмов, а также полный перечень конкретных значений всех исходных данных.

Контрольные вопросы

1. Что такое подчиненный алгоритм?
2. Как записываются и используются подалгоритмы без параметров?
3. Как записываются и используются подалгоритмы с параметрами?

Упражнения

1. Оформить в виде подалгоритма (без параметров и с параметрами) алгоритм нахождения абсолютной величины числа (обозначить АБС (a, m)). Используя этот подалгоритм, записать алгоритм вычисления значения y по формуле

$$y = \frac{|x^3 - 3x|}{|x^3 + x + 1| + 4x^2}.$$

2. Четырехугольник задан сторонами и длинами отрезков, соединяющих вершины с внутренней точкой. Составить алгоритм вычисления площади четырехугольника.

Указание. Искомая площадь может быть найдена как сумма площадей треугольников, вычисляемых по формуле Герона (оформить подалгоритм с параметрами).

3. Составить алгоритм вычисления числа сочетаний $C_m^n = \frac{m!}{n!(m-n)!}$, используя подалгоритм вычисления факториала.

3.11. Базовые алгоритмические структуры

Практика разработки и составления алгоритмов (и в форме схем, и в форме последовательной алгоритмической нотации) показывает, что в отдельных случаях одна и та же задача может быть успешно решена по алгоритмам, имеющим различную организационную структуру. Это уже отмечалось и в рассмотренных выше примерах (см., например, циклические алгоритмы, управляемые пост-условием и пред-условием). То, что подобные ситуации могут возникать часто, показывает и пример, изображенный на рисунке 42. На рисунке изображены фрагменты двух схем, имеющие различную структуру, однако как в первом, так и во втором случае последним элементом, добавленным к сумме S , будет элемент a_n . Это потребовало, правда, изменить и содержимое логического блока (вместо $i < n$ там стало $i \leq n$). Но как всё-таки лучше организовывать такой цикл — по методу а) или методу б)?

До сих пор мы не затрагивали вопросов структурной организации алгоритмов, а между тем эти вопросы имеют весьма существенное значение, причем значимость их особенно возрастает по мере усложнения разрабатываемых алгоритмов. Главное

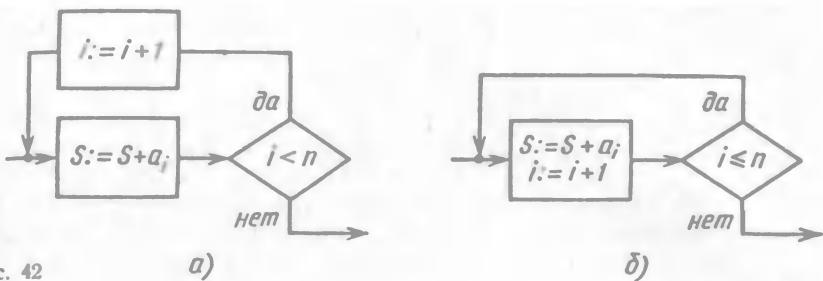


Рис. 42

требование к алгоритму — чтобы он правильно решал поставленную задачу. Но дело как раз в том и состоит, что для удовлетворения этого естественного требования алгоритм должен быть легким для понимания, прост для доказательства правильности и удобен для модификаций, если такая необходимость по каким-либо причинам возникает.

Опыт практической алгоритмизации, накопленный в связи с составлением программ для ЭВМ, привел к формированию особой методики структурной организации алгоритмов, использование которой уменьшает вероятность ошибок в процессе разработки и записи алгоритмов, упрощает их понимание и модификацию. Эту методику (или, как говорят, дисциплину) алгоритмизации называют *структурным подходом*.

При структурном подходе к конструированию алгоритмов они как бы «собираются» из трех основных (базовых) структур: РАЗВИЛКА, ЦИКЛ, СЛЕДОВАНИЕ, каждая из которых имеет один вход и один выход.

РАЗВИЛКА (рис. 43) состоит из логического элемента с проверкой некоторого условия P и функциональных блоков S_1 , S_2 , которые в простейшем случае являются арифметическими элементами. Развилка может быть двух видов: полная условная конструкция (рис. 43, а) и неполная условная конструкция (рис. 43, б). Расстановка значений истинности логического выражения P «да» и «нет» на приведенном рисунке условна, на практике она вытекает

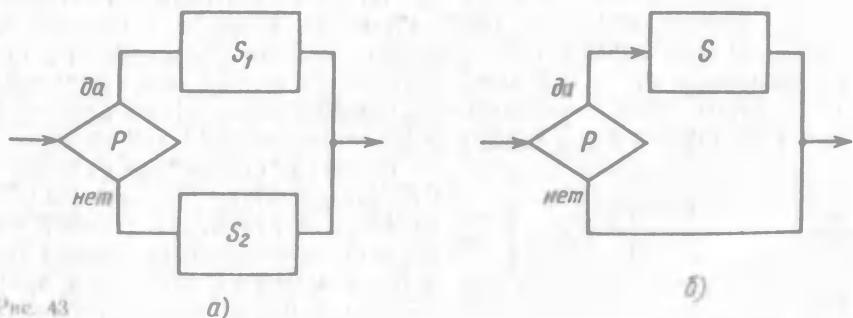


Рис. 43

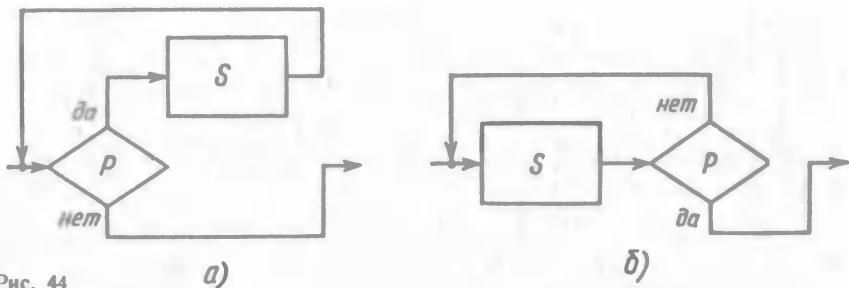


Рис. 44

из смысла конкретной программируемой задачи. В отдельных случаях для удобства последующей записи средствами алгоритмической нотации условие Р заменяется его отрицанием \bar{P} , и тогда значения «да» и «нет» меняются местами.

Базовая структура ЦИКЛ также может быть двух видов (рис. 44). В состав цикла входит логический элемент с проверкой условия Р и функциональный блок S, называемый *телом цикла*. В простейшем случае S является обычным арифметическим элементом. В первом случае (рис. 44, а) блок S размещены после проверки условия Р (цикл с пред-условием) так, что может оказаться, что тело S при определенных условиях не выполняется ни разу. Этот вариант базовой структуры ЦИКЛ, управляемый пред-условием, называют ЦИКЛ-ПОКА. Во втором случае (рис. 44, б) блок S расположен до проверки условия Р (цикл с пост-условием) так, что в этом варианте цикла тело S в любом случае будет выполнено по крайней мере один раз. Этот вариант базовой структуры ЦИКЛ называют ЦИКЛ-ДО. Тот или иной вариант структуры ЦИКЛ используется при составлении алгоритмов в зависимости от особенностей конкретной задачи. Расстановка значений истинности «да» и «нет» в изображении структур циклов, вообще говоря, может быть произвольной. Однако способ их расстановки, принятый на рисунке 44, не случаен, а является выражением определенного стандарта, объясняющего в том числе и сами названия структур (ЦИКЛ-ПОКА и ЦИКЛ-ДО). В первом случае (рис. 44, а) тело цикла исполняется, ПОКА — условие Р истинно; условие Р в этой базовой структуре называют *условием продолжения цикла*. Во втором случае (рис. 44, б) тело цикла исполняется ДО истинности Р, условие Р в этой структуре называют *условием окончания цикла*.

Базовая структура СЛЕДОВАНИЕ изображена на рисунке 45. Эта структура состоит из двух функциональных блоков S_1 и S_2 , каждый из которых в простейшем случае может быть



Рис. 45

арифметическим элементом. Структура СЛЕДОВАНИЕ означает, что два функциональных блока могут быть размещены друг за другом.

Основой методики структурного подхода являются следующие соглашения. Для построения более сложных алгоритмов из указанных простейших базовых структур разрешается пользоваться двумя способами: 1) подсоединить одну структуру к другой, образуя последовательность структур (реализуемость такой методики очевидна, поскольку каждая из структур имеет один вход и один выход); 2) заменить функциональные блоки S_1 и S_2 любой из базовых структур (понятно, что это не приведет к разрушению общей организации внешней структуры по той же причине, что и в первом случае).

Сформулированные выше правила позволяют строить как угодно сложные по структуре алгоритмы, развивая их не только «вширь», но и «вглубь». При этом, разумеется, нужно вести дело таким образом, чтобы получаемый в итоге алгоритм правильно решал поставленную задачу,— сам по себе структурный подход не дает автоматического разрешения этой проблемы.

Конструируемые по методике структурного подхода алгоритмы имеют четкую и ясную структуру, легко поддаются проверке, так как состоят из ограниченного числа различных блоков, устроенных одинаково. Алгоритм, организованный по этой методике, всегда легко подвергнуть структурному анализу и выделить составные элементы. Так, например, на рисунке 46 изображен структурный алгоритм, который представляет собой следование функционального блока S_1 и структуры ЦИКЛ-ПОКА. При этом тело цикла является полной условной конструкцией (обведено пунктиром).

Структурный подход к составлению алгоритмов основывается на молчаливо принятом (но верном по существу) предположении, что каждый алгоритм может быть представлен в структурном виде. Однако в процессе поиска алгоритма это может удаваться

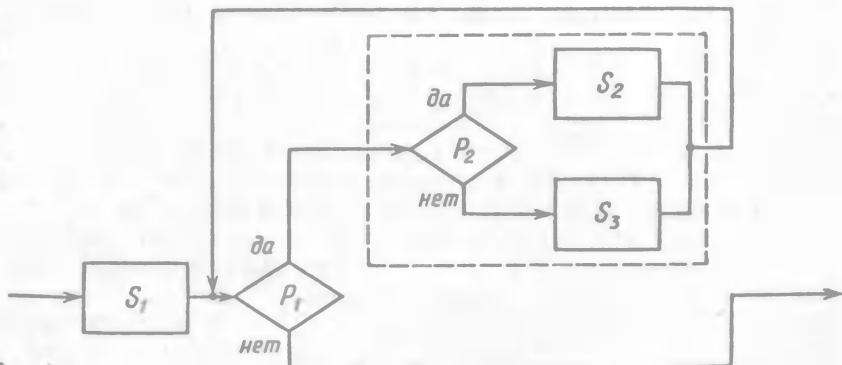


Рис. 46

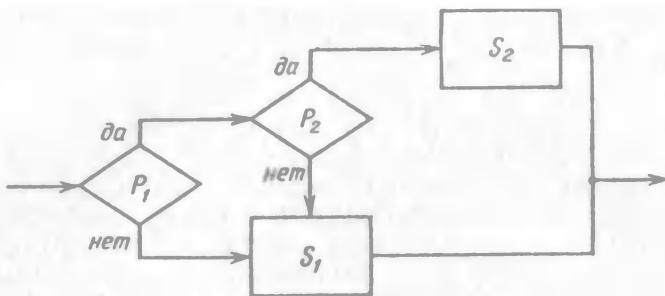


Рис. 47

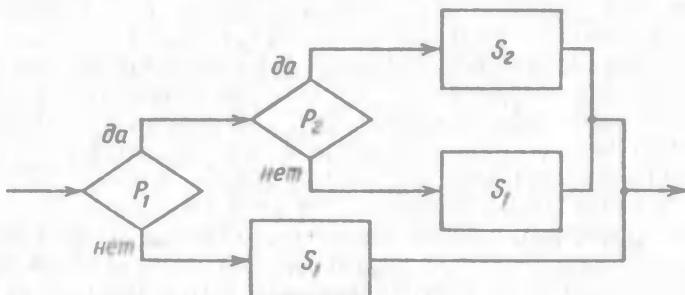


Рис. 48

не сразу. Иногда приходится предпринимать специальные приемы, помогающие преобразовать неструктурные алгоритмы в структурные. Простейший прием — размножение блоков.

На рисунке 47 изображена схема неструктурного алгоритма. Его легко преобразовать к структурному виду, если продублировать блок S_1 . Алгоритм, изображенный на рисунке 48, эквивалентен данному в том смысле, что во всех возможных случаях он предписывает выполнять те же действия.

Иногда для получения структурных алгоритмов приходится применять более изысканные методы. Рассмотрим примеры.

Пример 3.11.1. Даны три числа a , b и c . Определить, имеется ли среди них хотя бы одна пара взаимно обратных чисел.

Как известно, числа являются взаимно обратными, если их произведение равно 1. Для решения поставленной задачи надо производить все попарные проверки и, как только искомая пара найдется, прекратить процесс и выдать ответ «да». Если же ни одна проверка не выявит пары взаимно обратных чисел, выдать ответ «нет». Используя приведенные выше естественные рассуждения, получаем алгоритм, изображенный на рисунке 49. Этот алгоритм верно решает поставленную задачу, но он не является структурным.

Полученный алгоритм легко преобразовать к структурному виду, если продублировать блок выдачи ответа «да». Можно поступить и иначе: ввести дополнительную переменную — признак

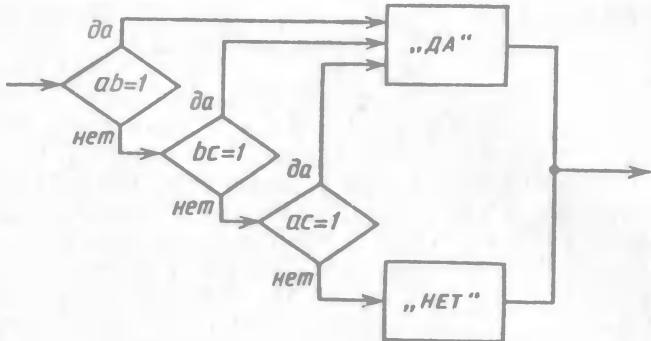


Рис. 49

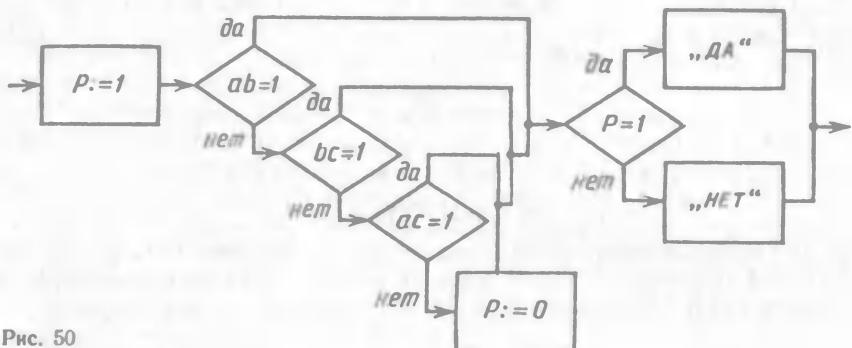


Рис. 50

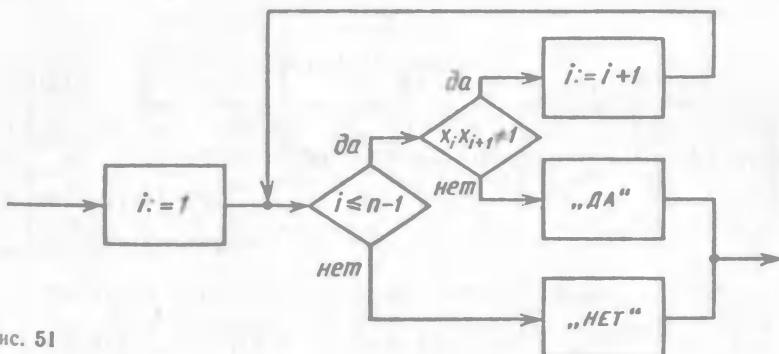


Рис. 51

Р (см. рис. 50). С самого начала принимается $P = 1$, но, если окажется, что ни одной пары взаимно обратных среди чисел a, b и c нет, значение P становится равным 0. Вслед за этим уже с помощью признака P выявляется окончательный ответ на поставленную задачу («да» или «нет»). Заметим, что структурный алгоритм, изображенный на рисунке 50, представляет собой следование простого арифметического блока и двух развилок, первая из которых — структура с глубиной вложения три.

Пример 3.11.2. Дан массив x_1, x_2, \dots, x_n . Требуется определить, имеется ли в этом массиве хотя бы одна пара взаимно обратных соседних чисел.

Судя по постановке задачи, надо циклически проверять выполнение равенства $x_i x_{i+1} = 1$ при $i=1, 2, \dots, n-1$. Как только равенство окажется истинным, процесс прекращается и выдается сигнал о том, что искомая пара элементов имеется («да»). Если же процесс доходит до конца, но равенство не выполняется, должен быть выдан сигнал «нет». Подходя к организации этого циклического алгоритма обычным образом (т. е. начиная цикл с проверки условия окончания $i \leq n-1$), получим схему, изображенную на рисунке 51. Этот алгоритм действует верно, но структурным не является.

Причиной неструктурности в данном случае является то, что по смыслу задачи имеется не одно, а два условия окончания цикла — исчерпание всех элементов и выполнение равенства $x_i x_{i+1} = 1$, причем в зависимости от того, каким условием закончился цикл, необходимо предпринимать различные действия.

Одни из способов преодоления подобного затруднения — объединение обоих условий продолжения цикла в одно:

$$(i \leq n-1) \wedge (x_i x_{i+1} \neq 1),$$

а для принятия окончательного решения по окончании цикла необходимо проверить, какое именно условие привело к выходу из цикла (рис. 52).

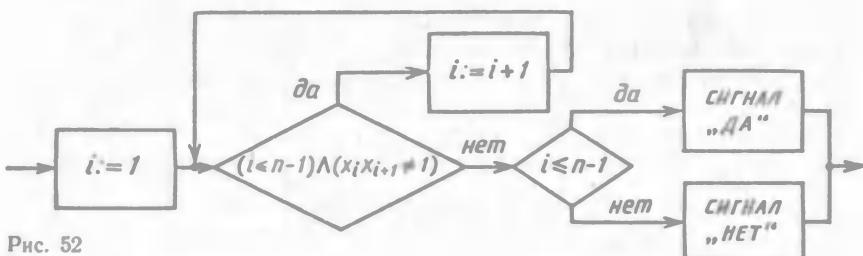
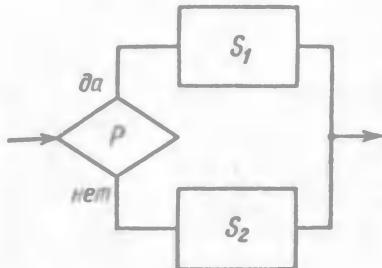


Рис. 52

Для того чтобы следовать методике структурного подхода применительно к алгоритмической нотации, лучше всего, если используемая алгоритмическая нотация будет располагать специальными средствами, просто и адекватно реализующими базовые структуры алгоритмов. Если таких специальных средств нет, то для облегчения последующей работы проще всего следует хорошо отработать технику организации языковых шаблонов, эквивалентно представляющих базовые алгоритмические структуры. Рассмотрим способы реализации базовых структур алгоритмов средствами построчной алгоритмической нотации, введенной в п. 3.8.

РАЗВИЛКА. Полной условной конструкции соответствует следующий шаблон, построенный средствами алгоритмической нотации (c — текущий номер строки):

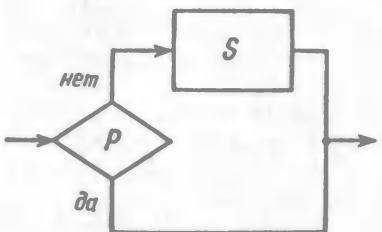


с. если Р к $c+2$
с+1. S_2 ; к $c+3$
с+2. S_1
с+3. выход

Разумеется, в конкретных случаях может оказаться, что для записи содержимого каждого из блоков S_1 и S_2 понадобится не по одной, а по несколько строк программы. Это приведет к увеличению количества строк в полной записи структуры РАЗВИЛКА, но общая логика построения шаблона останется неизменной.

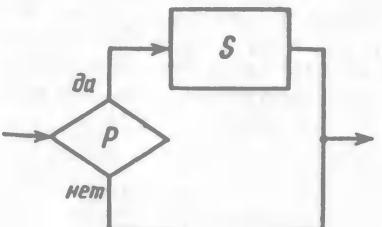
В неполной условной конструкции блок может быть расположен на линии «да» (см. рис. 43, б), но может быть расположен на линии «нет». В построчной реализации этих двух случаев есть свои особенности.

Наиболее просто описывается случай, когда блок расположен на линии «нет»:



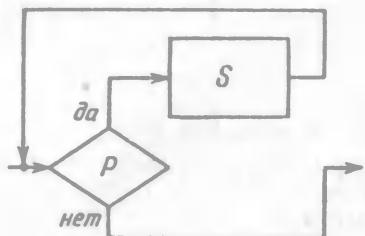
с. если Р к $c+2$
с+1. S
с+2. выход

Если же блок расположен на линии «да», целесообразно в качестве условия в операторе условного перехода взять не Р, а его отрицание \bar{P} :



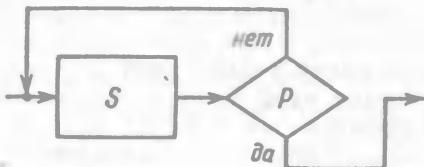
с. если \bar{P} к $c+2$
с+1. S
с+2. выход

Базовая структура ЦИКЛ-ПОКА:



c. если \bar{P} к $c+2$
 $c+1.$ $S;$ $\underline{k} c$
 $c+2.$ выход

Базовая структура ЦИКЛ-ДО:



c. S
 $c+1.$ если \bar{P} к c
 $c+2.$ выход

Замена записи оператора условного перехода условия P его отрицанием, как и при построчной записи структуры РАЗВИЛКА, обусловлена выбором наиболее целесообразного способа организации шаблона.

Базовая структура СЛЕДОВАНИЕ в рамках построчной алгоритмической нотации реализуется либо последовательными строками в алгоритмической записи, либо (если это позволяет объем содержания блоков S_1 и S_2) в одной строке, в которой S_1 и S_2 разделены точкой с запятой:



c. S_1
 $c+1.$ S_2 или c. $S_1; S_2$

Пример 3.11.3. Составим в построчной алгоритмической нотации структурный алгоритм решения задачи (см. рис. 52 на с. 138): определить, имеется ли среди элементов массива x_1, x_2, \dots, x_n хотя бы пара взаимно обратных соседних чисел:

Алгоритм ВЗОБР

1. чтение $n, x_{1:n}$
2. $i := 1$

ЦИКЛ-ПОКА { 3. если $(i > n - 1) \vee (x_i x_{i+1} = 1)$ к 5
 4. $i := i + 1; \underline{k} 3$

РАЗВИЛКА { 5. если $i \leq n - 1$ к 7
 6. запись "нет"; $\underline{k} 8$
 7. запись "да"
 8. конец

При записи структуры ЦИКЛ-ПОКА в соответствующем операторе условного перехода (см. строку 3 алгоритма ВЗОБР) используется отрицание условия ($i \leq n - 1$) \wedge ($x_i, x_{i+1} \neq 1$) (отрицание конъюнкции равно дизъюнкции отрицаний).

Контрольные вопросы

1. Каким требованиям должна удовлетворять разработка алгоритмов на ЭВМ?
2. В чем суть основных структур алгоритмов: РАЗВИЛКА, ЦИКЛ, СЛЕДОВАНИЕ?
3. В чем заключается структурный подход к разработке алгоритмов? Как обеспечивается соблюдение основных требований к разработке алгоритмов с использованием этого подхода?
4. Как реализуются базовые структуры алгоритмов средствами построчной алгоритмической нотации?

Упражнения

Составить структурные схемы и записи в построчной алгоритмической нотации алгоритмов решения задач:

1. Определить, имеется ли в массиве x_1, x_2, \dots, x_n хотя бы одна пара противоположных соседних чисел.
2. Упорядочить заданный массив a_1, a_2, \dots, a_n по возрастанию (см. пример 3.9.4 на с. 126).

3.12. Структурная алгоритмическая нотация

Построчная алгоритмическая нотация вместе со всеми ее изобразительными возможностями, рассмотренными выше, бесспорно обладает целым рядом методических достоинств с точки зрения обучения начальным представлениям и навыкам алгоритмизации, поскольку в явной форме раскрывает анатомию описываемых алгоритмов и показывает динамизм алгоритмических структур во всей их полноте. Тем не менее построчная алгоритмическая нотация, наиболее ярким выражителем которой является широко распространенный алгоритмический язык Бейсик, не отражает магистральных тенденций развития современных производственных алгоритмических языков.

Как было показано выше (п. 3.11), структурные алгоритмы можно составлять и на «неструктурированных» языках. Главное неудобство в использовании неструктурных средств записи алгоритмов состоит в том, что в алгоритмических записях так или иначе возникает скопление операторов безусловного перехода, которые сильно затрудняют чтение и понимание алгоритмов. Это хорошо видно даже из описания приведенных в п. 3.11 структурированных шаблонов. Структурные средства записи алгоритмов могут иметь в своем составе оператор безусловного перехода, однако позволяют вести конструирование алгоритмов, полностью

исключая этот оператор из записей. На этом принципе основано построение современных языков программирования Паскаль, Рапира (см. гл. 4), а также учебного алгоритмического языка, используемого в школьном курсе информатики [1], [2], [14].

Рассмотрим инструментарий структурной алгоритмической нотации, отражающей черты новых алгоритмических языков.

Заведомо исключая частое употребление оператора идти к, структурная нотация не следует принципу построчной записи алгоритмов со сквозной нумерацией строк, как это делалось раньше. В тех редких случаях, когда может возникнуть потребность в переходе идти к адрес перехода может быть помечен целочисленной (или буквенной) меткой, отделяемой от оператора-адресата двоеточием, например:

$$5:x:=2a$$

Если в одной строке расположено несколько операторов, то между ними ставится разделительный знак — точка с запятой:

$$x:=2a; y:=4x; z:=-\sqrt{y^2+1}$$

Последовательное расположение операторов фактически реализует базовую структуру СЛЕДОВАНИЕ. Группа последовательных операторов называется серией.

И все же главная особенность структурной нотации — это наличие специальных средств для записи базовых структур алгоритмов. Ниже рассматриваются новые изобразительные средства (операторы) алгоритмической нотации, непосредственно реализующие базовые структуры РАЗВИЛКА, ЦИКЛ. Новые операторы показываются в сравнении с прежними способами их реализации (см. рис. 43, 44, 45), которые при этом выполняют роль определений новых понятий.

РАЗВИЛКА. Полная условная конструкция:

c. если P идти к c+2
c+1. S₂; идти к c+3
c+2. S₁
c+3. в выход

если P
то S₁
иначе S₂
все

Расположение ключевых слов если, то, иначе, все, принятое в приведенной справа структурной записи, не случайно. Хотя эта (чисто оформительская) сторона дела и не влияет на суть операторов, тем не менее показанное размещение ключевых слов — это наиболее наглядная форма записи, облегчающая чтение и понимание алгоритмов. Ключевые слова если и то ограничивают условие структурного оператора, слова то и иначе — серию S₁, а слова иначе и все — серию S₂. Это, разумеется, не исключает того, что в простых случаях может допускаться и обычное расположение ключевых слов:

если P то S₁ иначе S₂ все

Неполная условная конструкция:

<i>c. если P идти к c+2</i>	<u>если</u> P
<i>c+1. S</i>	<u>то</u> S
<i>c+2. выход</i>	<u>все</u>

ЦИКЛ. Базовая структура ЦИКЛ-ПОКА:

<i>c. если P идти к c+2</i>	<u>пока</u> P
<i>c+1. S; идти к c</i>	<u>цикл</u> S
<i>c+2. выход</i>	<u>все</u>

Базовая структура ЦИКЛ-ДО:

<i>c. S</i>	<u>цикл</u> S
<i>c+1. если P идти к c</i>	<u>до</u> P
<i>c+2. выход</i>	<u>все</u>

Каждый из операторов S_1 , S_2 , S может быть серией. Операторы серии, стоящие в одной строке, разделяются точкой с запятой; точка с запятой перед ключевыми словами все и до не ставится.

В структурной алгоритмической нотации используется еще один вид цикла — цикл с параметром, определяемый следующим образом:

<i>c. x := a</i>	<u>для</u> x <u>от</u> a <u>до</u> b <u>шаг</u> h
<i>c+1. если x > b идти к c+4</i>	<u>цикл</u> S
<i>c+2. S; x := x + h</i>	<u>все</u>
<i>c+3. идти к c+1</i>	
<i>c+4. выход</i>	

Описание всех рассмотренных алгоритмических структур вместе с их схемами и построчными представлениями дано в таблице 40. Перейдем теперь к примерам составления алгоритмов с использованием структурной нотации.

Пример 3.12.1. Алгоритм поиска большего из трех чисел:

Алгоритм БИТ
чтение a, b, c
если a \geqslant b
 то y := a
 иначе y := b
 все
 если y < c
 то y := c
 все
 запись y
конец

В структурной нотации также может быть использован описанный в п. 3.10 аппарат подчиненных алгоритмов. Приводим запись подалгоритма БИД (α , β , γ) и алгоритма БИТ с его использованием:

Подалгоритм БИД (a , b , y)
если $a \geq b$
 то $y := a$
 иначе $y := b$
все
возврат

Алгоритм БИТ
чтение a , b , c
БИД (a , b , y)
БИД (y , c , y)
запись y
конец

П р и м е р 3.12.2. Алгоритм решения неравенства $ax > b$:

Алгоритм НЕРАВЕНСТВО
чтение a , b
если $a \neq 0$
 то $c := b/a$
 если $a > 0$
 то запись " $x >$ ", c
 иначе запись " $x <$ ", c
 все
 иначе если $b < 0$
 то запись "решений беск. мн."
 иначе запись "решений нет"
 все
 все
конец

Как видно, довольно-таки разветвленный алгоритм записан с точки зрения структурного анализа всего лишь тремя операторами: чтение, если — то — иначе — все, конец, и при этом отпала потребность в операторе идти к. С учетом тенденций современной алгоритмизации подобные записи алгоритмов выглядят предпочтительнее по сравнению с аналогичными записями из предыдущих разделов. Тем не менее вряд ли можно отрицать, что составление и понимание таких алгоритмов требуют достаточно высокого уровня алгоритмической культуры.

Столь же эффектно выглядят записи циклических алгоритмов.

П р и м е р 3.12.3. Алгоритм нахождения наибольшего общего делителя двух целых положительных чисел:

Алгоритм НОД
чтение a , b
пока $a \neq b$
 цикл если $a > b$
 то $a := a - b$
 иначе $b := b - a$
 все
 все
 запись "НОД =", a
конец

П р и м е р 3.12.4. Алгоритм сложения всех элементов числового массива x_1 , x_2 , ..., x_n :

Алгоритм СМ-1
чтение n , $x_{1:n}$
 $i := 1$; $S := 0$
пока $i \leq n$
цикл
 $S := S + a_i$
 $i := i + 1$
все
запись " $S =$ ", S
конец

Более естественным для подобных алгоритмов является использование оператора цикла с параметром:

Алгоритм СМ-2
чтение n , $x_{1:n}$
для i от 1 до n шаг 1
цикл $S := S + a_i$
все
запись " $S =$ ", S
конец

В тех случаях, когда шагом цикла является число +1, указание «шаг 1» обычно опускают.

С помощью структурных операторов цикла легко описываются вложенные циклы — так называемые циклические алгоритмы, в которых телом цикла является цикл. Рассмотрим пример.

Пример 3.12.5. До сих пор мы имели дело с одномерными массивами, т. е. такими, каждый элемент которых имеет один индекс. Существуют массивы большей размерности. Например, матрица

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

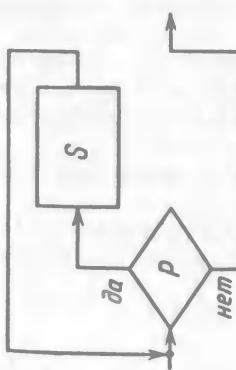
является примером двухмерного массива. Составим алгоритм вычисления суммы всех элементов матрицы.

Алгоритм СУМАТР
чтение m , n , $x_{1:m, 1:n}$
 $S := 0$
для i от 1 до m
цикл
для j от 1 до n
цикл
 $S := S + a_{ij}$
все
все
запись " $S =$ ", S
конец

Таблица 40

Базовая структура	Построение нотации	Структурная нотация
СЛЕДОВАНИЕ	$c. S_1 \text{ или } c. S_1; S_2$ $c+1. S_1$	$S_1; S_2$
РАЗВИЛКА (полная)	$c. \text{если } P \text{ идти к } c+2$ $c+1. S_2 \text{ идти к } c+3$ $c+2. S_1$ $c+3. \text{Выход}$	<u>если</u> <u>P</u> <u>то</u> <u>S₁</u> <u>иначе</u> <u>S₂</u> <u>все</u>
РАЗВИЛКА (неполная)	$c. \text{если } P \text{ идти к } c+2$ $c+1. S$ $c+2. \text{Выход}$	<u>если</u> <u>P</u> <u>то</u> <u>S</u> <u>все</u>

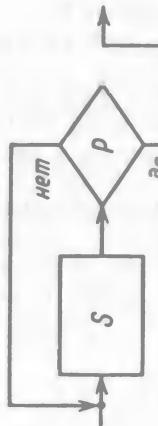
ЦИКЛ-ПОКА



$c.$ если P идти к $c+2$
 $c+1.$ S идти к c
 $c+2.$ выход

пока P
 цикл S
 все

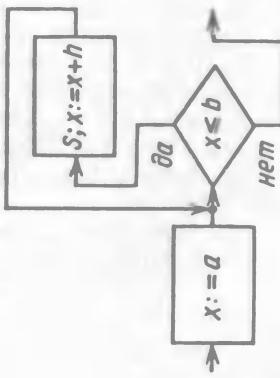
ЦИКЛ-ДО



$c.$ S
 $c+1.$ если P идти к c
 $c+2.$ выход

цикл S
 до P
 все

ЦИКЛ С ПАРАМЕТРОМ



$c.$ $x:=a$
 $c+1.$ если $x > b$ идти к $c+4$
 $c+2.$ $S; x:=x+h$
 $c+3.$ идти к $c+1$
 $c+4.$ выход

для x от a до b шаг h
 цикл S
 все

Контрольные вопросы

1. Каковы положительные и отрицательные стороны (с учебной и производственной точек зрения) построчной алгоритмической нотации?

2. Как записываются в структурной алгоритмической нотации операторы, реализующие базовые структуры РАЗВИЛКА (полная и неполная конструкции), ЦИКЛ-ПОКА, ЦИКЛ-ДО, цикл с параметром?

Упражнения

Записать в структурной алгоритмической нотации алгоритмы решения следующих задач:

1. Решить уравнение $ax = b$.

2. Решить систему уравнений

$$\begin{cases} ax + by = c, \\ dx + ey = f. \end{cases}$$

3. Определить, имеется ли среди чисел a , b и c хотя бы одна пара равных чисел.

4. Определить количество целых чисел среди чисел a , b и c .

Указание. Использовать функцию ЦЕЛ(x) (целая часть x).

5. Вычислить сумму 20 первых членов последовательности, заданной формулой общего члена

$$a_k = \frac{2k+1}{4k^3 - 3}.$$

6. Вычислить сумму 30 членов последовательности, заданной рекуррентно: $a_1 = 2$, $a_{n+1} = 2a_n - 1$.

7. Вычислить сумму всех членов ряда $a_k = k/(k+1)^2$ ($k = 1, 2, \dots$), не меньших заданного числа e .

8. Найти произведение ненулевых элементов одномерного числового массива.

9. Многочлен степени n задан массивом своих коэффициентов. Найти значение производной многочлена в заданной точке.

Указание. Для вычисления значения многочлена использовать схему Горнера.

10. Определить сумму положительных элементов прямоугольной матрицы.

11. Проверить, имеется ли в данном одномерном числовом массиве хотя бы одна пара чисел, совпадающих по величине.

Программирование

В этой главе рассматриваются элементы программирования на алгоритмическом языке Бейсик, который построен по принципу построчной алгоритмической нотации, а также проводится первоначальное знакомство с языками структурного программирования Паскаль и Рапида. Язык Бейсик, так же как языки Паскаль и Рапида, реализуется на отечественных микроЭВМ, которые получают в настоящее время распространение в средней школе.

4.1. Начальные понятия языка Бейсик

Язык программирования Бейсик^{*} существует уже более 20 лет и является в настоящее время наиболее распространенным языком микроЭВМ. Отличительные его черты — простота и доступность. В то же время это — мощное средство для решения большого круга задач в режиме диалога человек — ЭВМ. Следует также принять во внимание, что развитие Бейсика привело к появлению таких его версий, которые захватили многие идеи и средства программирования из современных языков.

Программы на языке Бейсик могут быть реализованы на отечественных микроЭВМ типа «Электроника-60», «Электроника Д3-28», «Агат», «Искра-226», ДВК и др. Язык Бейсик является одним из рабочих языков для школьных комплектов вычислительной техники КУБТ-86, «Корвет», получающих в настоящее время распространение в средних учебных заведениях. Ниже рассматриваются первоначальные сведения о языке Бейсик, необходимые для составления простейших программ.

Алфавит языка Бейсик включает символы, из которых могут быть условно выделены следующие основные группы:

- а) латинские буквы А, В, С, ... Z;
- б) русские буквы А, Б, В, ..., Я (кроме Ё, а иногда и Ъ);
- в) цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;

* Аббревиатура BASIC объясняется английским словосочетанием „Beginner's All-Purpose Symbolic Instruction Code“ (многоцелевой язык символьических инструкций для начинающих). Кроме того, слово basic означает «основной», «базовый».

г) знаки арифметических операций:

- ^ (возвведение в степень)*;
* (умножение);
/ (деление);
+ (сложение);
- (вычитание);

д) символы, из которых составляются знаки отношений (отсутствующие в алфавите знаки \geq , \leq , \neq формируются путем комбинаций из указанных):

- = (равно);
< (меньше);
> (больше);

е) специальные символы . , : " () ! ? % # & @ \$

Алфавит Бейсика включает также внушительный перечень зарезервированных служебных слов, включающих, в частности, ключевые слова, из которых составляются операторы языка. Некоторые версии Бейсика разрешают использовать как прописные, так и строчные буквы русского и латинского алфавита, мы для простоты будем пользоваться только прописными (заглавными) буквами. В группу специальных символов входит символ «пробел», не имеющий начертания (когда этот символ хотят все же выделить специально, используют обычно перевернутую скобку «_»). Группа специальных символов (как, впрочем, и состав других групп) всегда связана с конкретной реализацией языка. В любом случае, собираясь работать на Бейсике, знакомство с его алфавитом полезно начать с изучения клавиатуры микроЭВМ.

Из символов алфавита Бейсика конструируются все его объекты: *константы, переменные, выражения, функции и массивы*. Каждый из этих объектов имеет в языке *имя, значение и тип*.

Константы

Язык Бейсик работает с константами двух типов: *числовыми и литерными* (или *символьными*).

Числа. В языке Бейсик используются целые и действительные числа. Знак «+» перед числом не ставится, а для отделения целой части при записи десятичных дробей используется точка. Нулевую целую часть дробного числа можно опускать.

Примеры записи чисел:

Целые

0
143
-362

Действительные

.01
12.0
-24.356

* В некоторых версиях Бейсика операция возведения в степень обозначается символом \uparrow или \rightarrow .

Действительные числа могут представляться в экспоненциальной форме. Так, например, одно и то же число 0,005342 может быть представлено несколькими различными способами:

.5342E—2
.0005342E2
534.2E—5

Здесь латинская буква Е имеет смысл «возвести 10 в степень». Показатели степени (порядок) может не иметь знака, если он положителен, отрицательный порядок обязательно должен иметь знак «—».

Разрядность чисел зависит от конкретной модели микроЭВМ. Так, например, «Искра-226» имеет 13 десятичных разрядов. Диапазон целых чисел — от 0 до 7999, дробных — от 10^{-12} до $(10 - 10^{-12}) \cdot 10^{10}$. Версия Бейсика стандарта MSX, реализуемая на КУВТ «Ямаха», использует целые числа из диапазона от — 32 768 до 32 767. Кроме того, в этой версии предусмотрена возможность использовать наряду с целыми десятичными числами также и целые двоичные, восьмеричные, шестнадцатеричные константы, а для дробных десятичных чисел (фиксированных и плавающих) устанавливать два режима точности — одинарную и двойную.

Литерные константы. Литерной константой в Бейсике может быть цепочка символов (литер), заключенная в кавычки, например:

"ABC"
"ИНФОРМАТИКА"
"18 ДЕКАБРЯ 1986 Г., 2.45"

В цепочке символов могут быть любые символы языка (пробелы, запятые, точки и т. п.), кроме кавычек. То, что находится между кавычками, называется значением литературной константы, а количество символов, из которых составлено значение, — ее длиной.

Допускается случай, когда литературная константа не содержит ни одного символа; соответствующий текст (пусто) обозначается двумя кавычками, следующими друг за другом"". Длина пустого текста равна нулю. Очевидно, что текст, состоящий, например, из пробела " ", отличается от пустого текста; длина литературной величины " " равна единице.

Переменные

Как и константы, переменные в Бейсике бывают числовые и литературные. Переменная имеет имя (идентификатор, обозначение).

Имя числовой переменной — это любая буква латинского алфавита или любая латинская буква, за которой следует одна цифра. Например:

A1 X5 B2

В MSX-Бейсике имя может состоять из произвольной совокупности латинских букв и цифр, начинающихся с буквы, но значащими являются лишь первые два символа. Числовые переменные, как и сами числа, могут быть двух типов: целые и действительные. Значениями целых переменных всегда являются целые числа, значениями действительных — действительные. Признаком целой переменной в ее обозначении является присутствие символа %, располагаемого вслед за именем переменной. Примеры простых переменных целого типа:

X% I% A5%

Носителями значений литерных констант в Бейсике являются литерные переменные. Идентификатор литерной переменной состоит из буквы латинского алфавита или буквы и цифры, за которыми следует символ \$ (в MSX-Бейсике эту роль выполняет символ \$). Примеры:

A\$ B2\$ X7\$ или соответственно
A\$ B2\$ X7\$

Наряду с рассмотренными выше простыми переменными в Бейсике могут использоваться переменные с индексами, служащие для обозначения элементов массивов. Переменная с индексом обозначается именем массива, за которым в круглых скобках указываются индексы — числовые и буквенные. Именем массива может быть любая простая переменная — целая, действительная или символьная. Примеры:

X (2) A% (1,3) B (2*I, J) C\$ (M, N)

В Бейсике допускается использование только одномерных и двухмерных массивов. В общем случае индексом в переменной с индексами может быть произвольное арифметическое выражение.

Выражения

В языке Бейсик различают арифметические, литерные и логические выражения. Каждое выражение имеет одно значение.

Арифметические выражения составляются из числовых констант и переменных с помощью знаков арифметических действий и круглых скобок. Например:

Обычная запись

$$\frac{by}{ax^2 + bx + c}$$

$$x/y$$

$$x + 5.3y$$

Запись на Бейсике

$$6*Y$$

$$A*X^2 + B*X + C$$

$$X*Y/(X + 5.3*Y)$$

Внутри скобок или в бесскобочных записях порядок действий совпадает с общепринятым: сначала выполняются все возведения в степень, потом умножение и деление и, наконец, сложение и вычитание. Операции одного приоритета выполняются слева направо.

Наряду с числами и переменными в состав арифметических выражений могут входить *стандартные функции* Бейсика. Каждая из функций имеет имя, за которым следует аргумент в круглых скобках. Ниже приводится перечень из десяти стандартных числовых функций Бейсика ЭВМ серии ДВК; эти же функции имеются и в Бейсике стандарта MSX.

Название функции	Обозначение
Синус X	SIN (X)
Косинус X	COS (X)
Арктангенс X	ATN (X)
Случайное число между 0 и 1	RND (X)
Абсолютное значение X	ABS (X)
Целая часть X	INT (X)
Сингум (знак) X	SGN (X)
Натуральный логарифм X	LOG (X)
Экспонента X (e^x)	EXP (X)
Квадратный корень из X	SQR (X)

Аргументом может быть число, переменная, арифметическое выражение (в том числе содержащее функции). Тригонометрические функции в обычных условиях вычисляются для аргументов, выраженных в радианах*. Аргумент функции RND (X) не используется и может быть любым числом. Функция SGN (X) вычисляется по правилу:

$$SGN(X) = \begin{cases} 1, & \text{если } X > 0 \\ 0, & \text{если } X = 0 \\ -1, & \text{если } X < 0 \end{cases}$$

Примеры записи арифметических выражений с функциями:

Обычная запись

$$4 \cos \frac{y}{8}$$

$$\frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Запись на Бейсике

$$4 * COS (Y/8)$$

$$(-B - SQR(B^2 - 4 * A * C)) / (2 * A)$$

Литерные выражения образуются из литерных констант и переменных с помощью литерных операций и функций.

Язык Бейсик имеет одну литерную операцию — операцию соединения литерных величин. Эта операция называется *конкатенацией* и обозначается, как и арифметическое сложение, знаком «+».

* В некоторых версиях Бейсика при желании может быть установлен любой из трех режимов вычислений тригонометрических и обратных тригонометрических функций: градусы, радианы или грады (1 град = $\pi/200$). Для этой цели используется специальный оператор-описатель SELECT (SELECT D — градусы, SELECT R — радианы, SELECT G — грады). При отсутствии указаний в форме оператора SELECT автоматически устанавливается режим R (радианы).

При конкатенации один текст без пробела подсоединяется к другому. Так, например, значением литерного выражения

"АЛГО" + "РИТМ"

будет текст АЛГОРИТМ. В некоторых версиях Бейсика имеется целый ряд специальных функций для обработки литерных величин. В их числе функции:

LEN (α) — вычисляет длину значения литерного выражения α ;

VAL (α) — преобразует литерное значение α в соответствующий ему десятичный код;

ASC (α) — вычисляет десятичный код первого символа литерного значения α ;

CHR\$ (X) — имеет значением символ (литеру), десятичный код которого равен X;

MID (α , M, N) — вырезает часть литерной величины α длиной в N символов, начиная с M-го символа.

Логические выражения в Бейсике могут составляться из арифметических и литерных величин и имеют своими значениями лишь два значения — истинно (TRUE) или ложно (FALSE).

Простейший случай логических выражений — это *отношения*. В зависимости от состава входящих величин могут быть два вида отношений — арифметические и литерные. И в том и в другом случае отношение конструируется одинаково — оно состоит из левой и правой частей, соединенных между собой одной из операций отношения: < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), = (равно), <> (не равно).

Левой и правой частью арифметического отношения может быть произвольное арифметическое выражение. Примеры арифметических отношений:

Обычная запись

$$ax^2 = bx + c$$

$$\ln x < 2 \sin x$$

Запись на Бейсике

$$x >= 0$$

$$A * X^2 = B * X + C$$

$$\text{LOG}(X) < 2 * \text{SIN}(X)$$

Литерные отношения составляются из литерных констант и переменных, например:

$$X\$ = Y\$$$

$$A\$ < "B"$$

Сравнение литерных (символьных) величин ведется слева направо в порядке следования символов, их составляющих (если в сравнении участвует литерная переменная, то имеется в виду последовательность литер, которая является ее текущим значением). Сравнение ведется до первой пары символов, не удовлетворяющей проверяемому условию. Так, например, процесс сравнения двух констант из условия

"ABBA" = "ALFA"

остановится на второй паре символов, причем будет получено значение «ложно», так как символы В и L не совпадают. Если сравнение доходит до конца, то значением условия будет значение «истинно».

При проверке истинности отношений с операциями $<$, $<=$, $>$, $>=$ учитывается порядок старшинства символов алфавита в соответствии с их шестнадцатеричными кодами. Самую низкую степень старшинства имеет символ «пробел». Ниже перечислены некоторые символы языка MSX-Бейсик в порядке возрастания степени их старшинства (слева направо и сверху вниз):

«	!	”	#	\$	%	&	'	0	*	+	.	-	.	/		
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	@
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
Q	R	S	T	U	V	W	X	Y	Z	Ю	А	Б	Ц	Д	Е	Ф
Г	Х	И	Й	К	Л	М	Н	П	Я	Р	С	Т	У	Ж	В	Ь
Ы	З	Ш	Э	Щ	Ч	О										

Пользуясь приведенным порядком старшинства символов, можно, например, заключить, что значением истинности условия

“КВАДРАТ” $<$ “КУБ”

будет значение FALSE, так как В < У (оба символа — русские буквы) неверно. Это же значение будет иметь и условие

“КУБИК” $<$ “КУБ”,

так как, дойдя до четвертого символа, процесс сравнения приведет к ложному неравенству И < «пробел».

Из отношений в MSX-Бейсике могут составляться более сложные логические выражения с помощью логических операций NOT (не), AND (и), OR (или) и др. Примеры:

($X^2 < 0$) AND ($\text{SIN}(X) > = 1$)
(A\$ = "ДА") OR (B\$ = "НЕТ")

Контрольные вопросы

1. Какие основные группы символов составляют алфавит языка Бейсик?
2. Какие типы величин используются в языке Бейсик, как они записываются?
3. Как обозначаются простые (целые, действительные и символьные) переменные, переменные с индексами?
4. Как определяется порядок действий в арифметических выражениях?
5. Как составляются отношения в языке Бейсик?
6. Каков порядок сравнения символьных величин?

Упражнения

1. Записать на Бейсике следующие числа:

2. Выделить из заданных чисел целые и действительные:

а) $-\emptyset$; в) 425; д) -265 ;
б) \emptyset ; г) .425; е) $-26.3E5$.

3. Записать на языке Бейсик следующие арифметические выражения:

a) $x^{1.37}$; b) $(0.1275 + 2 \sin \sqrt{\alpha})^3 + 2^{0.3}$;
 6) $a_0 + a_1 x + a_2 x^2$; r) $\frac{\sqrt{\sin(x-y)-1}}{1+|x^2-y^2|}$.

4. Записать на языке обычных математических обозначений:

a) $(Y - A \cdot X)/B$; b) $ABS(X - Y)/(1 + X)^A (A^2)$;
 6) $A/B \cdot C/D \cdot E$; f) $(4 \cdot A (I - J)^{-2}) / LOG(ABS(X))$.

5. Даны текущие значения переменных: $X=2$, $Y=3$, $A=5$.
Определить истинность отношений:

a) $X \leq 2$; b) $A + X^2 = Y^2$;
 6) $X^2Y > X \cdot Y$; f) $\text{INT}(Y/X) = Y - X$.

6. Определить значения истинности отношений:

6) ("ДА" = "НЕТ") OR ("ДА" > "НЕТ").

4.2. Организация программы

Программа на Бейсике представляет собой последовательность строк. Каждая строка программы снабжается десятичным номером. Выполнение программы осуществляется в порядке возрастания номеров строк^{*}, причем не требуется, чтобы строки программы нумеровались непременно последовательными натуральными номерами. Чтобы облегчить в процессе составления программы вставление новых строк между уже имеющимися, строки программы обычно нумеруют с каким-либо шагом, например через 10 номеров: 10, 20, 30 и т. д.

Основными компонентами программы, побуждающими ЭВМ к действию, являются *операторы*, из которых и состоят строки программы. Одна строка программы может содержать один или несколько операторов, операторы в строке программы отделяются друг от друга двоеточием.

- Если не предусмотрено программное изменение последовательности выполнения строк.

Наиболее употребительный оператор, обеспечивающий в программах производство вычислений,— это *оператор присваивания*, который применяется в Бейсике как для числовых, так и для символьных переменных. Оператор имеет вид:

[LET] <переменная> = <выражение>

Квадратные скобки используются в определении конструкций алгоритмического языка для указания тех объектов, которые в записи данной конструкции могут опускаться. Угловыми скобками выделены объекты языка, участвующие в определении.

Действие оператора LET (пусть) заключается в присваивании арифметической (или литерной) переменной, стоящей слева, значения арифметического (или литерного) выражения, стоящего справа.

Примеры:

- LET I% = 1 — целой переменной I% присваивается значение 1;
LET A = 23.7 — действительной переменной А присваивается значение 23.7;
LET X = A * SIN(T) — действительной переменной X присваивается значение выражения A sin T при текущих значениях переменных А и Т;
LET A\$ = "КОНЕЦ" — символьной переменной A\$ присваивается символьное значение КОНЕЦ;
LET C\$ = A\$ + B\$ — символьной переменной C\$ присваивается результат конкатенации текущих значений символьных переменных A\$ и B\$.

В некоторых версиях Бейсика с помощью оператора LET можно присвоить одно и то же значение не только одной переменной, но и нескольким переменным сразу; в этом случае переменные перечисляются в левой части оператора LET через запятую.

LET X, Y = A * LOG(B) — значение A * ln B присваивается каждой из переменных X и Y.

Как следует из определения, служебное слово LET в записи оператора присваивания может опускаться. При присваивании числовых величин тип величины, стоящей справа, преобразуется к типу левой величины (целому или действительному).

В программах для ЭВМ обычно предусматривается *ввод* начальных значений и *вывод* результатов. Для ввода значений в ходе исполнения программы используется оператор INPUT, имеющий вид:

INPUT ["γ"] <список>

Здесь INPUT (ввод) — служебное слово, γ — произвольный текст, <список> — список переменных, разделенных запятыми. В этом списке могут быть как числовые, так и сим-

вольные переменные. Встретив в программе оператор INPUT, ЭВМ делает паузу, выводит на экран сообщение γ (если оно есть), знак вопроса и ждет, когда с клавиатуры вводного устройства будет введено столько значений, сколько переменных в списке оператора INPUT. Вводимые значения в процессе ввода разделяются запятыми. При вводе смешанных (числовых и символьных) значений порядок их чередования должен в точности соответствовать порядку следования соответствующих переменных в списке переменных оператора INPUT. После ввода последнего по списку значения нажимается клавиша перехода на новую строку, и ЭВМ продолжит выполнение программы.

Так, например, в ответ на запрос оператора ввода

INPUT X, Y, D\$

на клавиатуре может быть набрана следующая последовательность трех значений:

6.5, 12.8, ВАРИАНТ

В результате ввода эти значения будут присвоены переменным из списка оператора INPUT в порядке их появления в нем:

X=6.5, Y=12.8, D\$=ВАРИАНТ

Использование текстового сообщения γ не только позволяет оживить диалог человека и ЭВМ, но и дает возможность своевременно напоминать пользователю о смысле запрашиваемых значений. Например, оператор

INPUT "СИЛА ТОКА"; I

потребует значение переменной I, обратившись к пользователю с вопросом

СИЛА ТОКА?

Вывод (печать) результатов осуществляется в Бейсике оператором PRINT, имеющим вид:

PRINT <список>,

где <список> может содержать переменные (числовые или символьные), арифметические выражения (в частности, числовые константы), строки текста (т. е. символьные константы) или то и другое.

Если в список оператора PRINT входит числовая переменная, то при выполнении оператора произойдет вывод значения этой переменной (число). Так, например, в результате выполнения оператора

PRINT A%, B, C

может быть выдана строка числовых значений вида

23 8.324109283572 -124.0803251723

Список оператора PRINT может содержать и выражение, например:

PRINT X + ABS(ATN(T))

В этом случае оператор PRINT произведет все предусмотренные выражением операции и выдаст полученный числовой результат.

Наличие запятой между элементами списка оператора PRINT показывает, что информация выводится в зонном формате (длина зоны в символах зависит от особенностей конкретного выводного устройства и определяется по-разному для разных ЭВМ, длина зоны может быть, например, от 14 до 16 позиций). Элемент PRINT, перед которым имеется запятая, выводится в начале следующей зоны. Если между элементами строки оператора PRINT стоит точка с запятой, то информация выводится в уплотненном формате.

Весьма удобной особенностью оператора PRINT в Бейсике является то, что этот оператор может использоваться для вывода сообщения, комментария или любой строки символов. Это делается путем включения в список оператора PRINT символьных констант или переменных. Пусть, например, текущее значение числовой переменной U в программе равно 46. Тогда выполнение оператора

PRINT "НАПРЯЖЕНИЕ"; U; "ВОЛЬТ"

повлечет выдачу текста:

НАПРЯЖЕНИЕ 46 ВОЛЬТ

В программы на Бейсике можно включать строки с необходимыми пояснениями, комментариями, что облегчает чтение текста программы (листинга). Для этой цели используется оператор REM, записываемый по форме

REM <цепочка символов>,

где <цепочка символов> может содержать текст (без кавычек), составленный из любых символов, кроме двоеточия. Оператор REM не исполняется машиной и может размещаться в любом месте программы.

Пример:

140 REM БЛОК СОРТИРОВКИ

Ключевое слово REM для простоты разрешается заменять апострофом:

140 'БЛОК СОРТИРОВКИ

Составим теперь несложную вычислительную программу.

Пример 4.2.1. Пусть d — диаметр основания цилиндра, а h — его высота. Площадь поверхности цилиндра и объем вычисляются по формулам

$$S = \frac{\pi d^2}{2} + \pi dh, \quad V = \frac{\pi d^2}{4} h.$$

Программа вычисления значений S и V по задаваемым путем ввода значениям D и H может иметь вид:

```

10 REM ЦИЛИНДР
20 INPUT D, H
30 S=3.1416*(D/2+H)
40 V=3.1416*D^2*H/4
50 PRINT "S=", S, "V=", V
60 END

```

Программа состоит из 6 строк. Первая строка содержит неисполняемый оператор **REM** с комментарием, адресуемым человеку и выполняющим в данном случае роль заголовка программы. Во второй строке расположен оператор ввода **INPUT**, с помощью которого определяются начальные значения переменных D и H . В строках с номерами 30 и 40 расположены операторы присваивания, вычисляющие искомые значения переменных S и V . И наконец, в строке с номером 50 расположен оператор вывода **PRINT**, который составлен таким образом, что машина выведет не просто сами значения S и V , а снабдит эти значения принятыми для подобных случаев обозначениями: $S=$, $V=$. В последней строке программы поставлен оператор **END**, который обозначает конец программы. Использование оператора **END** в конце текста программы необязательно, так как выполнение программы автоматически заканчивается после выполнения всех строк. Однако при наличии оператора **END** по окончании счета по программе в некоторых версиях Бейсика ЭВМ дает сообщение об объеме свободной памяти.

Для того чтобы заставить ЭВМ выполнить программу, ее нужно сначала тщательно набрать с клавиатуры, нажимая после каждой строки клавишу перехода на новую строку. При этом текст программы можно редактировать — исправлять или вносить изменения (см. п. 1.6, с. 53). Для пуска программы в конце (без номера) набирается команда **RUN** (пуск) и нажимается клавиша перехода на новую строку.

После пуска программы **ЦИЛИНДР** машина выведет на экран знак вопроса и остановится. Следует набрать на клавиатуре значение первой переменной из списка оператора **INPUT**, т. е. диаметра D . Затем нужно набрать запятую и ввести значение второй переменной H . В конце нажимается клавиша перехода на новую строку. При вводе значений $D=0.6$, $H=1$ машина выведет следующие результаты:

$$S = 2.4504422698 \quad V = .28274338823$$

Приведенная выше программа **ЦИЛИНДР** составлена так, что в результате ее выполнения вычисляется лишь одна пара зна-

чений S и V. Если бы понадобилось вычислить значения S и V для другой пары значений переменных D и H, пришлось бы вводить команду RUN заново. Впрочем, программу нетрудно видоизменить, чтобы без повторного ввода команды RUN ее можно было бы использовать для повторных вычислений. Достаточно сделать так, чтобы после выполнения оператора вывода (строка 50) снова выполнялся оператор ввода (строка 20). Это достигается заменой оператора END (строка 60) оператором *безусловного перехода*, который в Бейсике имеет вид:

GOTO <номер строки>

Действие этого оператора заключается в безусловном переходе к строке с указанным номером. Преобразованная программа будет иметь вид:

```
10 ЦИЛИНДРЫ
20 INPUT D, H
30 S = 3.1426 * D * (D/2 + H)
40 V = 3.1416 * D^2 * H/4
50 PRINT "S="; S, "V="; V
60 GOTO 20
```

Программу ЦИЛИНДРЫ можно использовать для неоднократных вычислений, так как каждый раз после вывода очередной пары значений S и V машина будет переходить к строке 20 и запрашивать новые значения исходных данных D и H. Ниже приведен протокол выполнения программы для двух пар значений D и H:

```
? 0.6, 1
S = 2.4504422698      V = .282743338823
? 1.4, 2.3
S = 13.19469814508    V = 3.540574920595
```

Вычисления в данном случае можно организовать так, чтобы все исходные данные (если их не слишком много) были сразу размещены в тексте программы и автоматически выбирались машиной по мере потребности в них. Для этой цели в Бейсике имеется пара специальных операторов DATA и READ, которые всегда используются совместно и имеют вид:

```
DATA <список констант>
READ <список переменных>
```

Строка с оператором DATA вводит в программу список констант*, которые перечисляются после слова DATA через запятую, например:

```
DATA 2, 4, 3.2, 10.3, 2.71 E-3
```

* Как и оператор INPUT, операторы DATA и READ могут работать с числовыми и символьными величинами.

Сами по себе операторы DATA никакого действия не вызывают и могут располагаться в любом месте программы. Чтение размещенных в операторе DATA констант осуществляется оператором READ, содержащим перечень переменных, расположенных вслед за словом READ через запятую, например:

READ A, B, C, D

При выполнении программы операторы DATA игнорируются до тех пор, пока встретится оператор READ. Затем отыскивается первый по порядку оператор DATA и перечисленные в нем константы в порядке их следования последовательно присваиваются переменным в операторе READ. Так, если в программе будут использованы указанные выше примеры операторов DATA и READ, то в результате выполнения оператора READ произойдут следующие присваивания: $A = 2$, $B = 4$, $C = 3.2$, $D = 10.3$. Операторов DATA и READ в программе может быть не по одному, и число их может не совпадать, однако количество констант, вводимых в программу операторами DATA, должно быть по крайней мере не меньше, чем количество переменных, содержащихся в операторах READ. Если оператор DATA содержит больше значений, чем имеется переменных в списке оператора READ, то следующий оператор READ начинает присвоение с первого неиспользованного при предыдущем чтении значения в операторе DATA*. Если же окажется, что оператор READ содержит больше переменных, чем имеющихся в операторе DATA значений, то разыскивается следующий оператор DATA, а в случае его отсутствия выдается сообщение об ошибке.

Ниже приводится программа ЦИЛИНДРЫ, в которой используются операторы DATA и READ:

```
10 ЦИЛИНДРЫ
20 READ D, H,
30 S=3. 1416 * D * (D/2+H)
40 V=3. 1416 * D^2 * H/4
50 PRINT "S="; S, "V="; V
60 GOTO 20
70 DATA 0.6, 1, 1.4, 2.3, 0.9, 11.2
```

В ходе выполнения программы переменные D и H сначала получают значения соответственно 0.6 и 1, затем 1.4 и 2.3 и т. д.

Каждый раз при повторном прохождении программы оператор READ будет обращаться к первой неиспользованной паре данных в операторе DATA. Печать результатов для заданных оператором DATA исходных значений будет иметь вид:

* В Бейсике имеется оператор RESTORE, с помощью которого взаимосвязь между операторами READ и DATA в программах может устанавливаться более изощренным способом.

$S = 2.454422698$

$V = .2827433338823$

$S = 13.19468914508$

$V = 3.540574920595$

$S = 32.93959897289$

$V = 7.125132138343$

После исчерпания всех чисел в операторе DATA машина выдаст сообщение о том, что данные в списке DATA использованы и остановится.

Контрольные вопросы

1. Какова общая структура программы на Бейсике?
2. Каким оператором Бейсика обеспечивается ввод исходных данных с клавиатуры в ходе исполнения программы?
3. Как работает оператор вывода PRINT?
4. Каким образом в программу на Бейсике включаются пояснения, комментарии?
5. Как с помощью оператора GOTO организуется неоднократная работа одной и той же программы?
6. Каким образом с помощью операторов DATA и READ организуется запись и считывание исходных данных в тексте программы?

Упражнения

1. Составить программу вычисления площади поверхности S и объема V конуса, заданного диаметром основания D и длиной образующей L :
 - а) с использованием оператора INPUT;
 - б) с использованием операторов DATA и READ (для трех пар произвольно выбранных значений D и L).
2. Радиус окружности, вписанной в равносторонний треугольник, равен R . Составить программу вычисления стороны, высоты и площади треугольника.
3. Сосуд имеет форму опрокинутого конуса, осевое сечение которого — равносторонний треугольник. В сосуд брошен железный шар радиуса R и налита вода так, что поверхность воды касается погруженного в сосуд шара. Составить программу вычисления уровня воды после того, как шар будет вынут.

4.3. Программирование ветвлений

Для программирования разветвляющихся алгоритмов в Бейсике имеется *оператор условного перехода*, или, коротко, оператор IF (если), имеющий вид:

IF <условие> THEN <номер строки>

Действует этот оператор так: если условие при текущих значениях входящих в него переменных истинно, то происходит пе-

переход к строке, номер которой указан после THEN (то), в противном случае происходит переход к оператору программы, расположенному вслед за оператором IF (таким оператором может оказаться либо оператор из этой же строки, что и оператор IF, но расположенный вслед за ним через двоеточие, либо, если оператор IF в своей строке является последним, первый оператор из следующей по порядку строки программы). Условием в операторе IF является логическое выражение.

Образец записи оператора условного перехода:

20 IF X<0 THEN 50

Пусть, к примеру, в момент выполнения оператора IF текущее значение переменной $X = -3$. Тогда условие $X < 0$ будет истинно и произойдет переход к строке с номером 50. Если же, например, $X = 5$, то условие $X < 0$ будет ложно и произойдет переход к строке, следующей в программе за строкой с номером 20.

В приведенном выше примере условие оператора IF составлено из арифметических выражений, т. е. является условием арифметического типа. Рассмотрим фрагмент:

40 IF A\$= "ДА" THEN 70
50 PRINT "ВЕРНИСЬ К РАЗДЕЛУ 8"

Условием оператора IF в данном случае является условие символьного типа $A\$ = \text{"ДА"}$. Если к моменту выполнения оператора IF значением символьной переменной $A\$$ будет «ДА», то условие $A\$ = \text{"ДА"}$ будет истинно и оператор IF осуществит переход к строке 70. Если же значением переменной $A\$$ будет не константа «ДА», то IF передаст управление строке 50.

Приведенная выше форма оператора IF — лишь один из возможных способов его задания. В развитых версиях Бейсика оператор IF обладает расширенными возможностями и может, в частности, записываться в одной из форм:

IF <условие> THEN {
 <номер строки>
 <серия>
|ELSE {
 <номер строки>
 <серия> } |

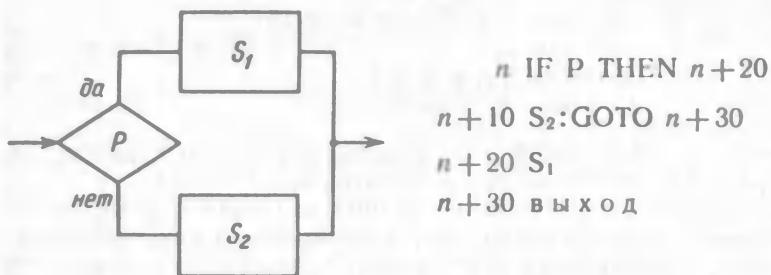
Здесь ELSE (иначе) — ключевое слово языка, <серия> — последовательность операторов, разделенных двоеточием. При составлении конкретного вида оператора IF может использовать любой из вариантов, указанных в фигурных скобках, например:

IF <условие> THEN <серия> ELSE <номер строки>

Прежде чем перейти к составлению программ, содержащих ветвления, покажем, что приемы программирования базовой

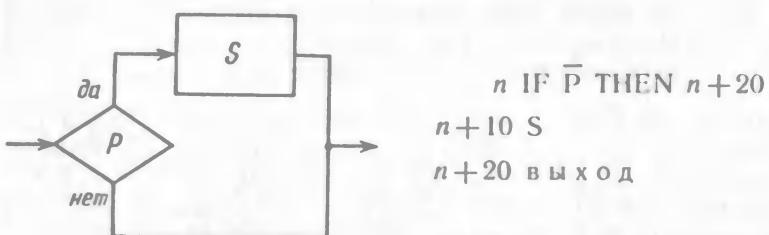
структуры РАЗВИЛКА средствами Бейсика в точности совпадают с соответствующими приемами конструирования алгоритма средствами построчной алгоритмической нотации (гл. 3, п. 3.8). Действительно, так как Бейсик наследует все основные принципы построчной алгоритмической нотации, то программные шаблоны структуры РАЗВИЛКА получаются механически.

Полная условная конструкция



В конкретных случаях может оказаться, что для программного представления каждого из блоков S_1 и S_2 понадобится не по одной, а по несколько строк программы. Это приведет к увеличению количества строк в программной реализации структуры РАЗВИЛКА. Общая же логика построения фрагмента программы останется неизменной.

Неполная условная конструкция



(Читателю предоставляется возможность самостоятельно убедиться в том, что использование в операторе IF самого условия P привело бы к более громоздкой и неестественной программной реализации структуры).

Пример 4.3.1. Составить программу решения уравнения $ax = b$ для произвольных значений числовых параметров a и b .

Схема этого алгоритма изображена на рисунке 28 (с. 98). Составим программу с использованием операторов DATA и READ для трех пар значений a и b , иллюстрирующих работу алгоритма в каждом из трех логически возможных случаев:

```

10 'УРАВНЕНИЕ AX=B
20 READ A, B
30 PRINT "УРАВНЕНИЕ"; A; "* X="; B
40 IF A=0 THEN 70
50 PRINT "РЕШЕНИЯ НЕТ"
60 GOTO 110
70 IF B=0 THEN 100
80 PRINT "РЕШЕНИЯ НЕТ"
90 GOTO 110
100 PRINT "РЕШЕНИЙ БЕСКОНЕЧНО"
110 GOTO 20
120 DATA 0, 0, 0, 4, 2, 3
130 END

```

Пример 4.3.2. Составить программу решения неравенства $ax > b$ (a и b — произвольные действительные числа).

Рассмотрев всевозможные сочетания значений параметров a и b , приходим к схеме алгоритма, изображенной на рисунке 53. Программа, реализующая этот алгоритм, может иметь вид:

```

10 'НЕРАВЕНСТВО AX>B
20 INPUT "КОЭФФИЦИЕНТЫ A И B"; A, B
30 PRINT "НЕРАВЕНСТВО"; A; "* X>"; B
40 IF A=0 THEN 90
50 C=B/A
60 IF A>0 THEN 80
70 PRINT "РЕШЕНИЕ X<"; C:GOTO 120
80 PRINT "РЕШЕНИЕ X>"; C:GOTO 120
90 IF B<0 THEN 110
100 PRINT "РЕШЕНИЙ НЕТ": GOTO 120
110 PRINT "X — ЛЮБОЕ ЧИСЛО"
120 GOTO 20

```

Для иллюстрации работы программы надо набрать на клавиатуре пары значений a и b , соответствующих каждому из четырех возможных исходов, например: 0 и -5 (x — любое число), 0 и 3 (решений нет), 2 и 6 (решение $x > 3$), -2 и 6 (решение $x < -3$). Выдача результата в программе организована таким образом, что каждый раз сначала выводится само решаемое неравенство. Например, после ввода значений 0 и -5 программа выдаст текст:

НЕРАВЕНСТВО 0 * X > -5
X — ЛЮБОЕ ЧИСЛО

Использование символьных величин, в том числе и при формулировании условий для операторов IF, позволяет составлять программы, в которых диалог человека и ЭВМ происходит в форме «беседы», т. е. на уровне текстовых сообщений. Рассмотрим простейший пример такой программы.

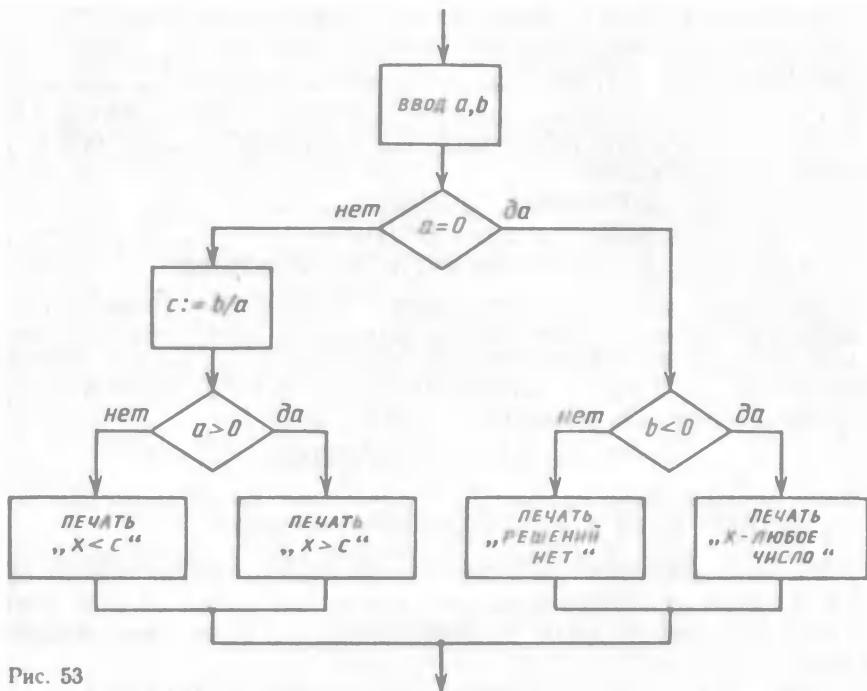


Рис. 53

Пример 4.3.3. Шуточная программа ТЕСТ, представленная ниже, сначала выясняет имя собеседника, а затем интересуется уровнем его математических познаний, требуя ответа на сакральный вопрос «Можно ли делить на нуль?»:

```

10 'ТЕСТ
20 INPUT "НЕ ВОЛНУЙТЕСЬ. ВАШЕ ИМЯ"; I$
30 PRINT "ОТЛИЧНО,"; I$; "!"
40 PRINT "ОТВЕТЬТЕ НА ВОПРОС."
50 INPUT "МОЖНО ЛИ ДЕЛИТЬ НА НУЛЬ"; A$
60 IF A$="НЕТ" THEN 100
70 IF A$="ДА" THEN 90
80 PRINT I$; ". ВАМ НЕОБХОДИМО ПОВТОРИТЬ МАТЕМАТИКУ": GOTO 110
90 PRINT I$; ". У ВАС СЕРЬЕЗНЫЕ ПРОБЕЛЫ В МАТЕМАТИКЕ": GOTO 110
100 PRINT "ВЫ,"; I$; "МОЛОДЕЦ!"
110 PRINT "КТО СЛЕДУЮЩИЙ?": PRINT
120 GOTO 20

```

Логика этой программы несложна, однако, анализируя ее работу, полезно уяснить, как происходит ввод и последующее использование символьных величин.

Оператор INPUT (строка 20) выводит текст

НЕ ВОЛНУЙТЕСЬ. ВАШЕ ИМЯ?

и ждет ввода. Допустим, что собеседник ЭВМ набирает на клавиатуре имя САША и нажимает клавишу перехода на новую строку. Литерной переменной I\$ присваивается значение САША. Вслед за этим операторами PRINT и INPUT (строки 30, 40, 50) ЭВМ выводит текст

ОТЛИЧНО, САША!

ОТВЕТЬТЕ НА ВОПРОС:

МОЖНО ЛИ ДЕЛИТЬ НА НУЛЬ?

и снова ждет ввода (оператор INPUT в строке 50). Любой ответ, вводимый теперь с клавиатуры, становится значением символьной переменной A\$. После чего происходит переход к строке 60 (оператор IF). Если ответом было слово НЕТ, то ЭВМ переходит к строке 100 и выводит текст

ВЫ, САША, МОЛОДЕЦ!

Если ответом было слово ДА, то от строки 60 ЭВМ перейдет к строке 70, а затем к строке 90 и выведет текст:

САША, У ВАС СЕРЬЕЗНЫЕ ПРОБЕЛЫ В МАТЕМАТИКЕ

Если ответ не совпадет ни со словом ДА, ни со словом НЕТ (например, НЕ ЗНАЮ), то ЭВМ перейдет к строке 80 и выдаст текст:

САША, ВАМ НЕОБХОДИМО ПОВТОРИТЬ МАТЕМАТИКУ

В каждом из трех случаев программа заканчивает работу вопросом КТО СЛЕДУЮЩИЙ? и переходит к началу.

Рассмотренные возможности Бейсика позволяют составлять серьезные диалоговые программы, в том числе и программы, организующие процесс обучения и контроль знаний.

Контрольные вопросы

1. Какова структура оператора условного перехода в Бейсике? Как действует этот оператор?
2. Как программируется на Бейсике базовая структура РАЗ-ВИЛКА в виде полной условной конструкции?
3. Как программируется на Бейсике базовая структура РАЗ-ВИЛКА в виде неполной условной конструкции?
4. С помощью каких средств Бейсика организуется программа, поддерживающая диалог человека и ЭВМ?

Упражнения

1. Составить программы вычисления значений функций:

а) $f(z)=\begin{cases} z^2, & \text{если } z \geq 1, \\ 1-z, & \text{если } z < 1; \end{cases}$ б) $f(x)=\begin{cases} 0 & \text{для целого } x, \\ \operatorname{ctg}(lx) & \text{для нецелого } x. \end{cases}$

2. Составить программу поиска большего из трех чисел (БНТ).
3. Составить программы решения квадратного уравнения (КВУР) $ax^2+bx+c=0$ ($a \neq 0$): а) в области действительных и б) в области комплексных чисел.
4. Составить программу решения неравенства $ax < 3$.
5. Составить программы вычисления значений функций:

a) $f(x) = \begin{cases} \frac{\sin x}{1-x^2}, & \text{если } x < -1, \\ \arccos x^2, & \text{если } -1 \leq x \leq 1, \\ \ln(x+0,8), & \text{если } x > 1; \end{cases}$

б) $f(x) = \begin{cases} 2u+1, & \text{если } u \leq 0, \\ \cos u, & \text{если } u > 0, \end{cases}$

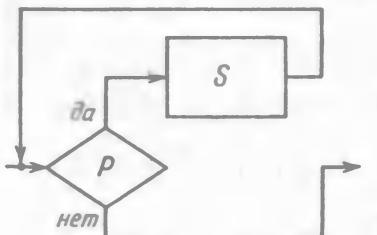
где $u = \begin{cases} \ln(-x), & \text{если } x < 0, \\ \sqrt{x}, & \text{если } x \geq 0. \end{cases}$

6. Составить программу, которая в процессе беседы со школьником ставит перед ним какой-либо вопрос и требует ответа. При этом программа должна предоставлять школьнику три попытки для формулирования правильного ответа и в каждом случае комментировать его действия.

4.4. Программирование циклов

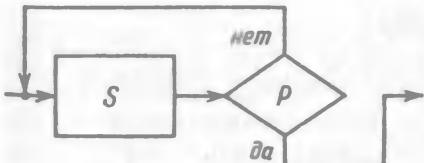
Программные шаблоны всех разновидностей базовой структуры ЦИКЛ легко составляются с использованием оператора IF языка Бейсик — по аналогии с тем, как это делалось средствами построчной алгоритмической нотации (гл. 3, п. 3.8).

Базовая структура ЦИКЛ-ПОКА:



$n \text{ IF } \bar{P} \text{ THEN } n+20$
 $n+10 \text{ S:GOTO } n$
 $n+20 \text{ выход}$

Базовая структура ЦИКЛ-ДО:



$n \text{ S}$
 $n+10 \text{ IF } \bar{P} \text{ THEN } n$
 $n+20 \text{ выход}$

Замена в записи оператора IF условия Р его отрицанием обусловлена выбором наиболее целесообразного способа программного представления структуры. Рассмотрим теперь примеры составления циклических программ.

Пример 4.4.1. Составить программу нахождения наибольшего общего делителя (НОД) двух целых положительных чисел.

Схема алгоритма нахождения НОД по методу вычитания, использующая структуру ЦИКЛ-ПОКА, изображена на рисунке 33.

Программа может иметь вид:

```
10 'ПРОГРАММА НОД (A, B)
20 INPUT A, B
30 X=A:Y=B
40 IF X=Y THEN 80
50 IF X>Y THEN 70
60 Y=Y-X:GOTO 40
70 X=X-Y:GOTO 40
80 PRINT "НОД ("; A, ";"; B, ")="; X
90 GOTO 20
```

Печать результата имеет форму $\text{НОД}(A, B)=X$, где A и B — исходные числовые значения, а X — найденный числовой результат. Так, например, если программе задать значения $A=12$ и $B=18$, то ответ будет выдан в виде

$$\text{НОД}(12, 18)=6$$

Для того чтобы сохранить для печати исходные значения данных A и B , в начале программы (строка 30) эти значения передаются рабочим переменным X и Y .

Пример 4.4.2. Составить программу вычисления (генерирования) 100 членов последовательности, заданной формулой

$$a_k = \frac{k^2}{k^2 + 1} \quad (k = 1, 2, \dots, 100).$$

В основу программы положим алгоритм, построенный по схеме ЦИКЛ-ДО (рис. 54).

```
10 'СУММА ПОСЛЕДОВАТЕЛЬНОСТИ
20 K=1
30 A=K^2/(K^2+1)
40 PRINT "A("; K, ")="; A
50 K=K+1
60 IF K<=100 THEN 30
70 END
```

Иной принцип организации имеют циклические программы суммирования числовых рядов не по заданному количеству членов, а в зависимости от значения самих членов.

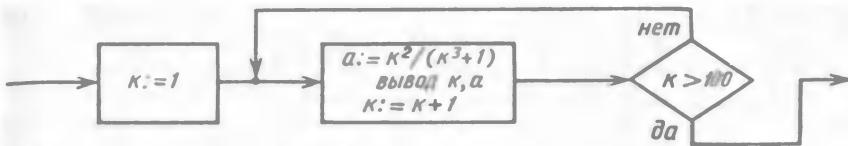


Рис. 54

Пример 4.4.3. Составить программу вычисления суммы всех членов ряда

$$\sum_{k=1}^{\infty} (-1)^{k+1} \frac{k}{x^{k+1}},$$

не меньших по абсолютной величине заданного числа ϵ .

Как следует из формулы общего члена, значение первого члена ряда равно $1/x$. Значение каждого последующего члена также может быть вычислено по формуле $a_k = k(-1/x)^{k+1}$ (при $k = 2, 3, \dots$). Примем вначале $S = \emptyset$ и будем добавлять к исключенной сумме очередной член ряда, проверяя предварительно выполнение условия. Схема алгоритма, основанного на структуре ЦИКЛ-ПОКА, изображена на рисунке 55. Пользуясь этой схемой, легко составить программу на Бейсике:

```

10 *СУММА РЯДА
20 INPUT E, X
30 K=1:S=0:A=1/X^2
40 IF ABS (A)<E THEN 70
50 S=S+A:K=K+1
60 A=K*(-1/X)^(K+1):GOTO 40
70 PRINT "S="; S
80 END

```

По поводу полученной программы можно заметить, что использование для ее составления структуры ЦИКЛ-ДО оказалось бы ошибочным, так как при определенном сочетании задаваемых программе значений E и X значение результата S может оказаться равным 0, в то время как программа ЦИКЛ-ДО и в этом случае выдаст в качестве результата значение первого члена $1/X$.

Рассмотрим сейчас особо случай, когда условием Р в структуре ЦИКЛ-ДО (см. рис. 44, б, с. 140) является неравенство вида

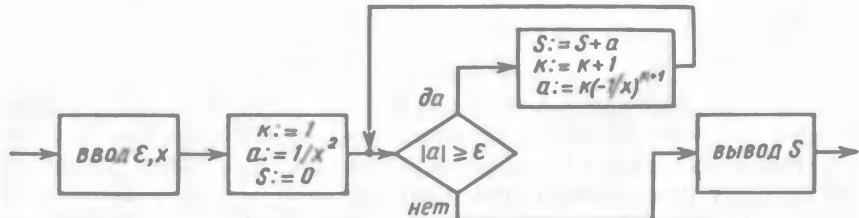


Рис. 55

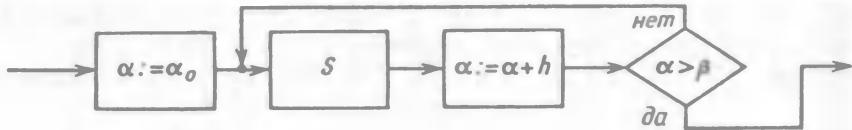


Рис. 56

$\alpha > \beta$, где параметр цикла имеет своим начальным значением α , а в каждом обращении цикла преобразуется по закону арифметической прогрессии $\alpha := \alpha + h$ (h — заданный шаг). Соответствующий циклический алгоритм описывается схемой, изображенной на рисунке 5б. Для программирования циклических алгоритмов такого вида в Бейсике имеются два специальных оператора: *оператор начала цикла* (или, коротко, оператор FOR) и *оператор конца цикла* (оператор NEXT). Операторы FOR и NEXT всегда используются совместно.

Оператор начала цикла (его называют еще *заголовком цикла*) имеет вид:

Вслед за оператором начала цикла располагают строки программы, которые должны выполняться циклически (эти строки образуют тело цикла). За последним оператором тела цикла размещается оператор конца цикла, который имеет вид:

NEXT <простая переменная>

Переменная, указываемая после NEXT, должна быть той же (простой) переменной, что и в операторе начала цикла после FOR (эту переменную называют параметром цикла). Арифметические выражения, стоящие слева и справа от ТО, задают соответственно начальное и конечное значения параметру цикла. Выражение, стоящее после слова STEP, задает шаг (число), на величину которого изменяется параметр цикла после каждого повторения тела цикла. Шаг может быть как положительным, так и отрицательным.

Условная программная запись алгоритма, изображенного на рисунке 56, с помощью операторов FOR и NEXT имеет вид:

n FOR $\alpha = \alpha_0$ TO β STEP h
 $n+10$ S
 $n+20$ NEXT α

Выполнение этой программы (как это и следует из схемы, изображенной на рисунке 56) происходит следующим образом.

Параметру цикла α присваивается его начальное значение α_0 , и один раз выполняется тело цикла S (т. е. группы операторов, размещенных между заголовком цикла и соответствую-

щим оператором NEXT). Вслед за этим оператор NEXT увеличивает значение параметра цикла α на величину шага h и проверяет, не превосходит ли значение параметра цикла значения арифметического выражения β , расположенного за словом TO в заголовке цикла. Если это не происходит, то осуществляется повторное выполнение тела цикла, в противном случае выход из цикла, т. е. переход к оператору, следующему за NEXT.

Если шаг цикла равен +1, указание STEP 1 в операторе начала цикла может опускаться. Операторы FOR и NEXT могут быть записаны в строках с несколькими операторами при условии, что FOR является первым оператором в своей строке, а NEXT — последним оператором в своей строке. Возможен досрочный выход из цикла не через его окончание, а с помощью операторов IF, GOTO, размещенных в теле цикла S. В MSX-Бейсике параметр цикла в операторе NEXT указывать не обязательно.

Приведенные выше правила выполнения операторов FOR — NEXT могут быть лаконично выражены через равносильную запись с использованием оператора IF:

$n \quad \alpha = \alpha_0$
 $n+1 \quad S: \alpha = \alpha + h$
 $n+20 \quad \text{IF } \alpha \leq \beta \text{ THEN } n+1$
 $n+30 \quad \text{вых од}$

Эта запись показывает, что все алгоритмы, программируемые с помощью операторов FOR — NEXT, могут быть запрограммированы и обычными средствами языка Бейсик. Тем не менее использование операторов цикла (там, где это возможно) позволяет более лаконично и выразительно описывать циклические алгоритмы по схеме ЦИКЛ-ДО.

Пример 4.4.4. Составить программу табулирования функции

$$f(x) = \frac{\ln(x+1)}{x+1}$$

на отрезке [0; 1] с шагом 0,05.

Схема алгоритма табулирования (ЦИКЛ-ДО) изображена на рисунке 57. Программа на Бейсике:

```
10 *ТАБУЛИРОВАНИЕ
20 PRINT "X", "F"
30 FOR X=0 TO 1 STEP .05
40 F=LOG(X+1)/(X^2+1)
50 PRINT X, F
60 NEXT X
```

Программа будет выводить две колонки числовых значений: слева — значения аргумента X, справа — значения функции F.

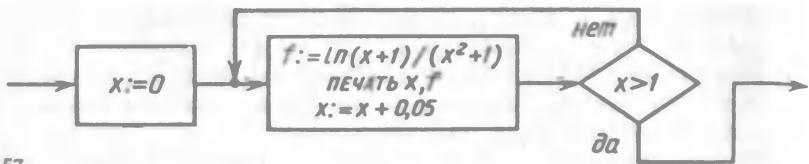


Рис. 57

Особенно эффективно операторы FOR — NEXT используются при составлении циклических программ, работающих с массивами. Массивы, используемые в программах, должны быть в этих программах обязательно описаны (объявлены). Для описания массивов в Бейсике имеется специальный оператор-описатель DIM. Оператор DIM размещается в программе раньше, чем начинается работа с соответствующим массивом. По описанию массива ЭВМ резервирует в памяти необходимое количество мест для его элементов. Описание делается так: вслед за словом DIM помещается имя массива, а за ним (в круглых скобках) — верхние границы индексов. Нижние границы индексов равны нулю. Например, описание

DIM A (100)

задает одномерный массив действительных чисел, состоящий из 101 элемента:

A (0), A (1), A (2), ..., A (100)

Одним оператором DIM можно описать несколько массивов — в этом случае описания различных массивов располагаются в строке оператора DIM через запятую.

Так, например, описание

DIM T (50), X (4), P (14,26)

резервирует в памяти места для двух одномерных и одного двухмерного массива. Как и прочие строки программы, строки описаний получают свои порядковые номера. В MSX-Бейсике допускаются массивы любой размерности и произвольного типа.

Пример 4.4.5. Вычислить сумму элементов одномерного массива целых чисел x_1, x_2, \dots, x_{15} .

Чтобы придерживаться обозначений, принятых в формулировке задачи, в программе опишем массив, состоящий из 16 элементов:

DIM X% (15),

но первый элемент (с индексом 0) не будем использовать. Программа должна предусматривать ввод значений элементов исходного массива. С целью сокращения программы ввод очередного элемента и добавление его к искомой сумме S включим в тело одного цикла. Используя операторы FOR — NEXT, получаем программу:

```
10 'СУММА МАССИВА
20 DIM X% (15)
30 S=0
40 FOR K=1 TO 15
50 READ, X% (K)
60 S=S+X% (K)
70 NEXT K
80 PRINT "СУММА="; S
90 DATA 1, 2, 3, 4, 5, 6, 7, 8
100 DATA 9, 10, 11, 12, 13, 14, 15
```

В качестве конкретных значений в списки операторов DATA включены для примера натуральные числа от 1 до 15.

Приведем пример программы, работающей с массивом личерных величин.

Пример 4.4.6. Сформировать текст, составленный из имен учащихся одной «звездочки», перечисленных через пробел.

```
10 'ЗВЕЗДОЧКА
20 DIM I$ (5)
30 T$=""
40 FOR K=1 TO 5
50 READ I$ (K)
60 T$=T$+I$ (K)
70 T$=T$+" "
80 NEXT K
90 PRINT "ЗВЕЗДОЧКА:"
100 PRINT T$
110 DATA ДИМА, ЮЛЯ, ПАВЛИК
120 DATA МАРИНА, ВАНЯ
```

Искомый текст формируется в результате поочередной конкатенации значения переменной T\$ (ее начальное значение — пустой текст "") со всеми элементами личерного массива, получающими значения имен членов «звездочки». Вслед за каждым именем в формируемом тексте появляется пробел. Закончив формирование, ЭВМ выдает результат:

ЗВЕЗДОЧКА:
ДИМА ЮЛЯ ПАВЛИК МАРИНА ВАНЯ

С помощью операторов FOR — NEXT особенно эффективно программируются алгоритмы с вложенными циклами.

Пример 4.4.7. Составить программу вычисления и печати элементов таблицы Пифагора.

Таблица Пифагора — это квадратная матрица из n строк и n столбцов. Первый столбец и первая строка состоят из натуральных чисел 1, 2, ..., n так, что левый угловой элемент $a_{1,1}$ равен 1. Каждый элемент a_{ij} определяется формулой $a_{ij} = i \times j$.

Искомый алгоритм содержит два цикла: во внутреннем цикле формируются строки таблицы Пифагора, во внешнем происходит переход от строки к строке. В программе параметром внутреннего цикла служит переменная J, а внешнего — переменная I:

```
10 *ТАБЛИЦА ПИФАГОРА
20 DIM A (10 10)
30 FOR I=1 TO 10
40 FOR J=1 TO 10
50 A (I, J)=I * J
60 NEXT J
70 NEXT I
80 MAT PRINT A.
```

Приведенная выше программа составлена для $n=10$ (10 строк и 10 столбцов). Строки 40, 50, 60 образуют тело внешнего цикла, которое выполняется 10 раз ($I=1, 2, \dots, 10$). При этом строка 50, являясь телом внутреннего цикла, выполняется 10 раз ($J=1, 2, \dots, 10$) для каждого значения I. Таким образом, строка 50 в общей сложности выполняется $10 \times 10 = 100$ раз.

Печать результата осуществляется специальным матричным оператором MAT PRINT, который построчно выводит двухмерный массив (матрицу) A в виде:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Контрольные вопросы

1. Как программируются базовые структуры ЦИКЛ-ПОКА и ЦИКЛ-ДО в Бейсике с помощью оператора IF?
2. В чем заключаются особенности циклических программ, составленных на основе алгоритмических структур ЦИКЛ-ПОКА и ЦИКЛ-ДО и как это следует учитывать в программировании?

3. Как записываются и как действуют операторы цикла FOR — NEXT? Какая из структур циклических алгоритмов реализуется этими операторами?

4. Как составляются описания массивов в программах на Бейсике?

Упражнения

1. Составить программу генерирования 50 членов последовательности, заданной формулой общего члена $b_k = \frac{k}{k+0,5}$:

а) с использованием оператора IF (по схеме ЦИКЛ-ПОКА и ЦИКЛ-ДО);

б) с использованием операторов FOR — NEXT.

2. Составить программы табулирования функций:

а) $F_1(x) = \frac{2 \sin^3 x}{1+x^3}$ на отрезке $[-1; 1]$ с шагом 0,1;

б) $F_2(x) = \begin{cases} 2x^2 + 1, & |x| < 4, \\ |x - 0,6|, & |x| \geq 4 \end{cases}$ для $x = -10, -9, \dots, 10$.

3. Дан ряд

$$\sum_{n=1}^{\infty} \frac{\sqrt{n}}{n^3 + 0,8} = \frac{1}{1,8} + \frac{\sqrt{2}}{8,8} + \dots$$

Составить программы:

а) вычисления суммы первых N членов ряда;

б) вычисления суммы всех первых членов ряда, величина которых не меньше заданного числа e . (Значения параметров N и e определяются при вводе.)

4. Составить программу вычисления суммы $\sum_{n=1}^{50} \frac{n+1}{n!}$.

Указание. Использовать рекуррентную формулу, связывающую два соседних члена ряда.

4.5. Функции пользователя и подпрограммы

В программировании довольно часто возникают случаи, когда одинаковые вычисления приходится производить в разных местах программы. Чтобы каждый раз не выписывать одинаковые операторы, в языки программирования включают специальные средства, позволяющие описывать повторяющиеся в программе вычисления только раз, а по мере необходимости использовать эти описания в тех местах, где это требуется. Для обозначения такого рода средств обычно используется общее наименование — процедуры. Имеются такие средства и в языке Бейсик, хотя в разных вер-

сиях этого языка этот вопрос решается по-разному. К таким средствам относятся определяемые функции (или функции пользователя), подпрограммы и подпрограммы с параметрами.

Функции пользователя. Составитель программы может по своему усмотрению определять часто используемые в программе математические функции, а затем использовать их так, как используются стандартные (встроенные) функции типа SIN, COS, LOG и т. п. Определяется функция пользователя с помощью оператора DEFFN (служебное слово DEFFN образовано сокращением двух английских: DEFINITION и FUNCTION), который записывается в виде

DEFFN <имя FN> (<прост. числ. перем. >) =
= <ариф. выраж. >

Здесь <имя FN> — цифра или буква латинского алфавита, выполняющая роль обозначения определяемой функции. Пример определения функции:

$$\text{DEFFN A(X)=X}^{\wedge} 2+5 \cdot X+6$$

Если в ходе выполнения программы необходимо использовать значение определенной в программе новой функции при конкретном значении ее аргумента, то достаточно вставить в нужном месте идентификатор этой функции вида FN <имя FN>, вслед за которым в круглых скобках вместо <простой числовой переменной >, выполняющей роль формального параметра, вставить нужное значение аргумента. Так, например, вызов определенной выше функции A(X) может быть осуществлен обращением FNA(3).

Действие оператора DEFFN вызывается только при упоминании функции FN, сам по себе он не является исполняемым оператором и может быть размещен в любом месте программы. Правая часть определения функции FN (т. е. <арифметическое выражение>) может включать в себя другие определяемые функции, но если, например, функция A вызывает функцию B, то B не может вызывать A (получается бесконечный цикл).

Пример 4.5.1. Составить программу вычисления значения выражения

$$Z(x)=\frac{\operatorname{th}(x+a)+\operatorname{th}(x+b)}{c\operatorname{th}(x-a)+c\operatorname{th}(x-b)}.$$

Учитывая, что тангенс гиперболический вычисляется по формуле

$$\operatorname{th} x=\frac{e^x-e^{-x}}{e^x+e^{-x}},$$

а $c\operatorname{th} x=1/\operatorname{th} x$, в программе целесообразно определить процедуру-функцию вычисления гиперболического тангенса:

```

10 'ВЫЧИСЛЕНИЕ ПО ФОРМУЛЕ Z (X)
20 DEFFN T (X)=(EXP (X)-EXP (-X))/(EXP (X)+EXP (-X))
30 INPUT X, A, B
40 P=X+A:Q=X+B
50 R=X-A:S=X-B
60 Z=(FNT (P)+FNT (Q))/(1/FNT (R)+1/FNT (S))
70 PRINT "Z (X) ="; Z
80 GOTO 30

```

Подпрограммы. Подпрограмма представляет собой логически законченный участок вычислительного процесса, завершающийся оператором RETURN (возврат). Подпрограмма располагается в произвольном месте общей программы, а ее строки нумеруются обычным порядком. Существенное значение при этом имеет номер первой строки подпрограммы, так как он используется при обращении к подпрограмме. Обращение к подпрограмме осуществляется с помощью специального оператора GOSUB (сокращение английских слов GOTO и SUBROUTINE). После служебного слова GOSUB указывается натуральное число — номер строчки, с которой начинается подпрограмма, например:

GOSUB 120

После обращения к подпрограмме выполняются все предусмотренные в ней действия, а когда очередь доходит до оператора RETURN, он осуществляет возврат в основную программу, т. е. передает управление тому оператору, который стоит непосредственно за соответствующим оператором вызова подпрограммы GOSUB.

Пример 4.5.2. Рассмотрим работу простой вычислительной программы, иллюстрирующей процесс обращения к подпрограмме с помощью оператора GOSUB:

```

10 'РАБОТА С ПОДПРОГРАММОЙ
20 FOR K=1 TO 5
30 INPUT X, Y, Z
40 GOSUB 60
50 NEXT K:END
60 'ПОДПРОГРАММА
70 A=X*Y+Z
80 IF A>20 THEN 100
90 P=0:GOTO 110
100 P=1
110 PRINT "P="; P
120 RETURN

```

Подпрограмма в данном случае располагается в строках 60—120 и предназначена для вычисления и печати значения признака P (P=0 или P=1), которое находится в зависимости от определяемых в основной программе значений X, Y, Z. После каж-

дого из пяти обращений к подпрограмме оператор RETURN осуществляет возврат в тело цикла, а после выхода из цикла содержащийся в строке 50 оператор END остановит программу.

Подпрограммы могут входить одна в другую, однако оператор GOSUB нельзя применять для входа внутрь цикла. Если подпрограмма имеет в своем составе цикл, то он должен полностью заканчиваться в этой подпрограмме (т. е. оператор NEXT цикла должен стоять раньше оператора RETURN соответствующей подпрограммы).

Как видно из рассмотренного примера, подпрограмма, организуемая операторами GOSUB и RETURN, не имеет строго обозначенного начала, так что в нее можно при желании входить в любом месте. Другие способы организации процедур в Бейсике предполагают строгое определение их начала.

Подпрограммы с параметрами. Первый оператор (заголовок) подпрограммы с параметрами имеет вид:

DEFFN' <номер подпрограммы> (<список параметров>) Обозначение DEFFN' читается как «помеченный DEFFN». Номером подпрограммы может быть целое число. Переменные из <списка параметров> оператора DEFFN' представляют собой формальные параметры, перечисленные через запятую. Как и при организации обычных подпрограмм, последним оператором помеченной подпрограммы является оператор RETURN. Операторы, расположенные между DEFFN' и RETURN, образуют тело подпрограммы.

Вызов помеченной подпрограммы осуществляется оператором GOSUB', имеющим вид:

GOSUB' <номер подпрограммы> (<список параметров>) Здесь <номер подпрограммы> равен соответствующему <номеру подпрограммы> из текста соответствующего оператора DEFFN', а <список параметров> содержит перечисленные через запятую фактические параметры. Фактическими параметрами могут быть или константы или переменные, однако в последнем случае их значения перед обращением к подпрограмме должны быть определены в основной программе. Формальные параметры после обращения принимают значения соответствующих им фактических параметров, после чего тело подпрограммы исполняется обычным порядком.

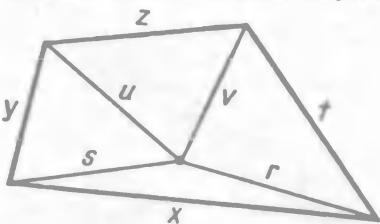


Рис. 58

Пример 4.5.3. Рассмотрим задачу вычисления площади четырехугольника, заданного сторонами x, y, z, t и длинами отрезков r, s, u, v , соединяющих вершины с внутренней точкой (рис. 58). Составим программу вычисления площади четырехугольника, ис-

пользуя подпрограмму с параметрами вычисления площади треугольника по формуле Герона. В приведенной ниже программе эта подпрограмма имеет заголовок DEFFN' 5 (A, B, C, S) (5 — номер подпрограммы, A, B, C, S — формальные параметры):

```
10 'ЧЕТЫРЕХУГОЛЬНИК
20 INPUT X, Y, Z, T, R, S, U, V
30 GOSUB' 5(X, S, R, P1)
40 GOSUB' 5(Y, U, S, P2)
50 GOSUB' 5(Z, U, V, P3)
60 GOSUB' 5(T, V, R, P4)
70 PRINT "ПЛОЩАДЬ="; P1+P2+P3+P4
80 END
90 DEFFN' 5(A, B, C, S)
100 P=(A+B+C)/2
110 S=SQR(P*(P-A)*(P-B)*(P-C))
120 RETURN
```

Контрольные вопросы

1. Какие средства языка Бейсик используются для исключения повторяющихся в программах описаний однотипных вычислений?
2. Каков порядок определения и использования в программах функций пользователя?
3. Каков порядок оформления и использования подпрограмм без параметров?
4. Как оформляются и используются подпрограммы с параметрами?

Упражнения

1. Составить программу вычисления значения выражения

$$A(x, y) = \frac{12.3 - \sqrt{|1 + 2 \sin(x+y)|}}{8.94 - \sqrt{|1 + 2 \sin(x^2 - y)|}},$$

используя определяемую функцию ENA (T)=SQR (ABS (1+
+2*SIN (T))).

2. Составить программу вычисления числа сочетаний $c_m^n = \frac{m!}{n!(m-n)!}$, используя подпрограмму с параметрами для вычисления значения факториала.

4.6. Графические средства Бейсика

Состав графических средств разных моделей микроЭВМ может сильно различаться. Это относится и к аппаратным средствам реализации машинной графики, и к программным средствам, поддерживаемым различными версиями Бейсика. Лаконичные и

в то же время достаточно эффективные и поучительные с образовательной точки зрения графические средства представлены в Бейсике, реализованном на отечественной микроЭВМ «Искра-226». С рассмотрения этих средств и начнем ознакомление с машинной графикой.

Устройствами вывода графики в микроЭВМ «Искра-226» являются: символьно-графический дисплей и графопостройтель.

В первом случае графическая информация изображается в световой форме на экране дисплея, во втором — в форме графического изображения на листе бумаги. И дисплей, и графопостройтель имеют рабочее поле в форме прямоугольника и вычерчивают графику на своем рабочем поле пишущим узлом, который перемещается дискретно. Элементарное перемещение пишущего узла называется дискретом.

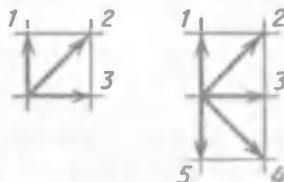
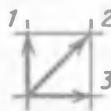
В зависимости от исходного положения за один дискрет пишущий узел может переместиться (рис. 59): а) из любой угловой точки рабочего поля в одну из трех соседних точек (четыре случая); б) из любой граничной неузловой точки в одну из пяти соседних точек; в) из любой внутренней точки рабочего поля в одну из восьми соседних точек.

Линейные размеры рабочего поля экрана дисплея — 230×105 мм, графопостройтеля — 300×200 мм. На рабочем поле экрана помещается $560 \times 256 = 143\,600$ точек, на поле графопостройтеля — $4000 \times 2666 = 10\,664\,000$ точек. Пишущим узлом экрана дисплея является световой маркер — графический курсор (имеет форму буквы «г», местоположение точки указывает вершину угла); пишущим узлом графопостройтеля является перо. И в том и в другом случае вывод графики происходит в прямоугольной декартовой системе координат с началом в левом нижнем углу рабочего поля. Исходным положением пишущего узла является начало координат; перо графопостройтеля при этом находится в приподнятом положении.

Для управления устройствами вывода графики в ряде версий Бейсика имеется специальный оператор PLOT. С некоторыми допустимыми способами записи оператора PLOT познакомимся постепенно.

В простейшем случае оператор PLOT имеет вид:

PLOT < Δx , Δy , z >



a)

b)

c)

Рис. 59

Здесь угловые скобки есть символы языка Бейсик (знаки сравнения $<$ и $>$), они являются неотъемлемым атрибутом в любой конкретной записи оператора PLOT, и их не следует в данном случае смешивать с использовавшимися раньше для определения объектов Бейсика металингвистическими угловыми скобками.

Параметры Δx и Δy задают размеры перемещения соответственно по осям OX и OY . Если $\Delta x > 0$, то перемещение осуществляется вправо; если $\Delta x < 0$, то влево. Аналогично в зависимости от знака Δy происходит перемещение или вверх ($\Delta y > 0$), или вниз ($\Delta y < 0$). В общем случае параметрами Δx и Δy могут быть произвольные арифметические выражения. В этом случае используется целая часть арифметических выражений. Если один или оба эти параметры не заданы, то соответствующее перемещение считается равным нулю; запятые внутри угловых скобок, однако, сохраняются и в этих случаях.

Параметр z используется, в частности, для определения состояния пишущего узла. Если вместо z ставится параметр D (down — вниз), то пишущий узел будет перемещаться в опущенном (рабочем) состоянии. Если вместо z поставить параметр U (up — вверх), то пишущий узел будет находиться в поднятом (нерабочем) состоянии.

Пусть устройством графического вывода является экран дисплея, а курсор находится в исходном положении (в левом нижнем углу). Тогда оператор

PLOT<280, U>

(здесь $\Delta x = 280$, $\Delta y = 0$) вызывает холостое перемещение курсора по горизонтали вправо на 280 точек, т. е. до середины экрана. А оператор

PLOT<, 240, D>

прочертит вертикальную линию в движении снизу вверх на 240 дискрет. Если эти два оператора выполнить один за другим, то на экране получится изображение в виде вертикальной линии, пересекающей экран точно посередине (рис. 60, а). Для последовательного выполнения двух указанных операторов можно разместить их в программе друг за другом, но можно объединить в один, расположив их правые части в порядке следования операторов через запятую:

PLOT<280,,U>, <,240, D>

Нетрудно составить продолжение, с помощью которого на экране будет изображена, например, прямоугольная система координат с началом (относительно исходной системы) в точке (280, 40):

PLOT<, -200, U>, <-275,, U>, <540,, D>

Первый элемент этого оператора осуществит холостое перемещение курсора по вертикали вниз на 200 дискрет, второй элемент сдвинет курсор в этом же (нерабочем) состоянии влево на 275

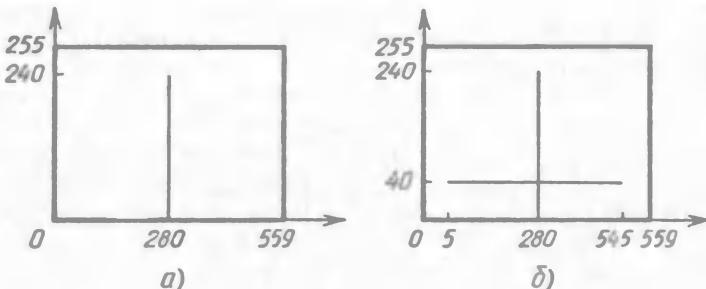


Рис. 60

дискрет, третий элемент прочертит горизонтальную прямую в движении вправо на 540 дискрет. Поскольку два элемента определяют действия курсора в одном и том же (нерабочем) состоянии U, их можно объединить:

PLOT < -275, -200, U>, <540,,D>

Изображение полученной таким образом системы координат показано на рисунке 60, б.

С помощью оператора PLOT может быть изображен любой символ алфавита или строка символов (символьная константа). Делается это заменой параметра z в операторе PLOT соответствующей символьной константой (в кавычках) или символьной переменной. Вернемся к рассмотренной выше последовательности действий по изображению прямоугольной системы координат и заставим оператор PLOT «пометить» оси координат символами X и Y. Символы, выводимые оператором PLOT, имеют размеры матрицы 5×7 шагов, причем вычерчиваются, начиная с левого нижнего узла этой матрицы. После вычерчивания символа пишущий узел устанавливается в правом нижнем углу матрицы в поднятом состоянии. Начиная теперь от положения курсора, определенного оператором

PLOT <280,,U>, <, 240, D>,

установку символа Y можно осуществить с помощью элемента <3, 5, "Y">. В элементе оператора возвращаем курсор вниз, нужно теперь учесть поправку на ширину символа Y (5 дискрет) и расстояние от оси OY до символа Y (3 дискрета):

PLOT < -283, -205, U>, <540,,D>

Установка символа X вслед за этим может быть осуществлена следующим образом:

PLOT <5, 3, "X">

Быстрое перемещение пишущего узла в поднятом положении в левый нижний угол (исходное начало координат) осуществляется оператором

PLOT <,R>

Здесь символ R (*reset* — восстановить) находится в элементе оператора PLOT на месте параметра z, а значения Δx и Δy безразличны.

Таким образом, построение координатных осей с центром в точке (280, 40) вместе с обозначениями осей символами X и Y и установкой курсора в начале исходной системы координат может быть выполнено операторами:

```
PLOT<280.,U>, <.240, D>, <3, 5, "Y">
PLOT<-280, -245, U>, <540.,D>
PLOT <5, 3, "X">, <..R>
```

Для построения графиков функций удобно иметь процедуру вычерчивания координатных осей на рабочем поле с началом в произвольно заданной точке. Ниже приведена программа, в которой описана и используется такая процедура (подпрограмма с параметрами 1000 (A, B)):

```
10 СИСТЕМА КООРДИНАТ
20 SELECT PRINT 10
30 INPUT A, B
40 GOSUB 1000 (A, B)
50 STOP: PRINT HEX (10D)
60 GOTO 30
70 DEFFN 1000 (A, B)
80 PLOT<A.,U>, <.240, D>, <3, 5, "Y">
90 PLOT<-A-3, B-245, U>, <540., D>
100 PLOT<5, 3, "X">, <..R>
110 RETURN
```

Оператором SELECT PRINT 10 устанавливается графический режим работы дисплея. В строке 50 приостановка исполнения программы (оператор STOP), а при последующем пуске произойдет гашение экрана (PRINT HEX (10D)) и переход к вводу новых значений A и B. Схема расчета смещений для элементов оператора PLOT показана на рисунке 61. Как следует из этой схемы, исходные значения параметров в приведенной выше программе следует задавать в промежутках $5 \leq A \leq 545$, $1 \leq B \leq 240$.

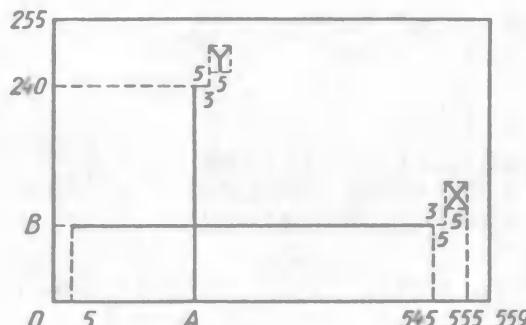


Рис. 61

П р и м е р 4.6.1. Вывести на экран изображение равностороннего треугольника.

Примем длину стороны треугольника за 100 единиц, тогда его высота равна $(100\sqrt{3})/2 \approx 87$. Если нижнюю левую вершину поместить в точку (230, 50), то построение обеспечивается оператором (рис. 62):

```
PLOT<230, 50, U>, <50, 87, D>, <50, -87, D>,  
<-100,,D>
```

Нетрудно определить процедуру с параметрами для построения равностороннего треугольника по заданной стороне А с учетом симметричного расположения фигуры на экране:

```
100 DEFFN 3(A)  
110 X=INT ((50-A)/2):T=A*SQR (3)/2  
120 IF INT(T)<256 THEN 150  
130 PRINT "УМЕНЬШИТЕ ДЛИНУ СТОРОНЫ"  
140 GOTO 180  
150 Y=INT ((256-T)/2)  
160 PLOT<X, Y, U>, <INT(A/2), INT(T), D>  
170 PLOT<INT(A/2)-INT(T), D>, <-A,,D>, <,R>  
180 RETURN
```

П р и м е р 4.6.2. Вывести на экран график функции

$$f(x) = 0.1x^2 - 2x - 150.$$

Организуя построение графика функции на рабочем поле устройства графического вывода, необходимо правильно определить положение осей координат, с тем чтобы точки графика на выбранном промежутке изменения аргумента не выходили за пределы рабочего поля. В данном случае ось ОХ необходимо поднять по меньшей мере на 160 дискрет. В приведенной ниже программе сначала строится система координат с центром в точке (280, 160). Построение точечного графика параболы ведется оператором PLOT<X, Y, ".>, который в каждом обороте цикла возбуждает движение графического курсора из начала исходной системы координат к точке с координатами (X+280, Y+160), т. е. к точке (X, Y) в новой системе координат:

```
10 SELECT PRINT 10  
20 PLOT<280,U>, <, 240, D>, <3, 5, "Y">  
30 PLOT<-283, -85, U>, <560,,D>  
40 PLOT<5, 3, "X">, <,R>  
50 FOR X=-40 TO 60  
60 Y=1*X^2-2*X-150  
70 Z=X+280:T=Y+160  
80 PLOT<Z, T, ".>, <,R>  
90 NEXT X
```

Параметр X в цикле изменяется от 40 до 60, и такие границы X в данном случае выбраны не случайно, так как при всех

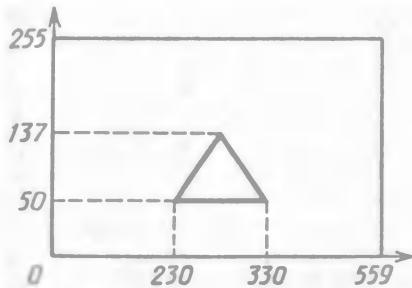


Рис. 62

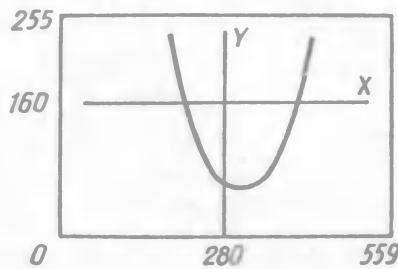


Рис. 63

значениях аргумента из этого интервала точки строящегося графика оказываются в пределах рабочего поля экрана (рис. 63).

При несоблюдении этого условия график будет изображаться в искаженном виде.

Путем незначительных усовершенствований программу можно сделать более универсальной. Для этого нужно прежде всего предусмотреть задание центра координат (A, B) путем ввода: это позволит пользователю самому определять положение новой системы координат с целью получения наиболее желательного расположения графика на поле экрана. При таком подходе, однако, в программе придется перед изображением каждой точки графика проверять прежде, не оказывается ли она за пределами рабочего поля. Универсальный характер приведенной ниже программы придает также возможность определять вид функции с помощью оператора DEFFN, не изменяя всей программы в целом.

```

10 'ГРАФИК ФУНКЦИИ
20 SELECT PRINT 10
30 INPUT A, B
40 PLOT <A,, U>, <210, D>, <3, 5, "Y">
50 PLOT <-A-3, B-245, U>, <540,D>
60 PLOT <5, 3, "X">, <,R>
70 FOR X = -A TO 560-A
80 Y=FNF(X)
90 IF Y>(256-B) OR Y<-B THEN 120
100 Z=X+A:T=Y+B
110 PLOT<Z, T, ".">, <,R>
120 NEXT X
130 STOP: PRINT HEX (D)
140 GOTO 30
150 DEFFN F(X)=.1*X^2-2*X-150

```

После приостановки и повторного пуска программы в строке 130 в результате вывода управляющего кода HEX (D) гасится содержимое экрана.

Во многих случаях оператор PLOT целесообразно помещать в тело цикла.

Пример 4.6.3. Составить программу построения окружности.

```
10 'ОКРУЖНОСТЬ
20 SELECT PRINT 10
30 INPUT A, B, R, N
40 IF A-R<0 OR A+R>560 THEN 70
50 IF B-R<0 OR B+R>256 THEN 70
60 GOTO 10
70 SELECT PRINT 05
80 PRINT "ДАННЫЕ А, В, Р НЕ СОГЛАСОВАНЫ"
90 GOTO 20
100 FOR T=0 TO 6.28 STEP N
110 X=A+R*COS(T)
120 Y=B+R*SIN(T)
130 PLOT<X, Y, ".>, <..R>:NEXT T
140 STOP:PRINT HEX (ID)
150 GO TO 30
```

Программа строит окружность радиуса R с центром в точке (A, B); параметр N задает угловой шаг, определяющий плотность расположения точек на окружности. Предполагается, что исходные значения всех параметров A, B, R, N положительны. Если при этом значении A, B и R все же таковы, что точки строящейся окружности выходят за пределы экрана, пользователю дается сообщение

ДАННЫЕ А, В, Р НЕ СОГЛАСОВАНЫ

и происходит возврат к вводу новых данных. При этом оператором SELECT PRINT 05 (строка 70) перед выводом сообщения отменяется графический режим экрана.

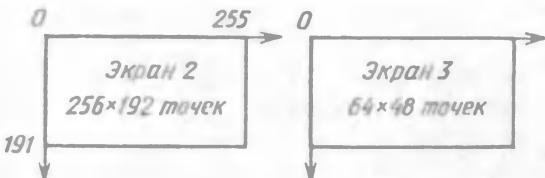
Богатые возможности работы с графикой предоставляются в Бейсике стандарта MSX.

Так, например, MSX-Бейсик, реализуемый на КУВТ «Ямаха», предоставляет в распоряжение пользователя 4 различных экрана с номерами 0, 1, 2, 3: два символьных (0, 1) и два графических (2, 3). Переключение с экрана на экран производится командой

SCREEN *n*,

где *n* ∈ {0, 1, 2, 3}. Графический экран 2 — экран высокого разрешения, имеющий 256 точек в строке и 192 точки в столбце. Графический экран 3 — экран низкого разрешения, имеющий 64 точки в строке и 48 — в столбце. При этом на экранах установлена своеобразная ориентация координатных осей, при которой начало координат оказывается в левом верхнем углу, а точка при увеличении ординаты, вместо того чтобы подниматься вверх, опускается вниз (рис. 6).

Рис. 64



Операторы ввода-вывода PRINT и INPUT можно использовать только на символьных экранах 0 и 1, а графические операторы — только на экранах 2 и 3. Рассмотрим операторы MSX-Бейсика, позволяющие строить простейшие графические объекты: точки, прямые, окружности.

Установка точки осуществляется оператором

PSET (X, Y)

Здесь PSET (*point* — точка, *set* — ставить) — ключевое слово, X, Y — произвольные арифметические выражения. Результатом выполнения оператора является установка точки, координатами которой являются целые части значений X и Y соответственно (т. е. INT (X) и INT (Y)). Пользуясь оператором PSET, можно получать точечные изображения фигур. Так, например, программа

```

10 SCREEN 2
20 FOR T=0 TO 6.28 STEP .05
30 PSET (120+50 * COS (T), 90+50 * SIN (T))
40 NEXT T
50 GOTO 50

```

по аналогии с программой примера 4.6, использующей оператор PLOT, строит точечную окружность радиуса 50 с центром в точке (120, 90). Оператор GOTO в строке 50 зацикливает программу, что позволяет полюбоваться построенным изображением.

Для работы на цветных дисплеях оператор PSET допускает расширение, связанное с управлением цветом:

PSET (X, Y) [, C]

Здесь необязательный элемент С является арифметическим выражением, целая часть значения которого лежит в интервале от 0 до 15 и определяет цвет изображаемой точки. По умолчанию цвет точки совпадает с цветом установленного ранее основного изображения.

Для вычерчивания окружностей в MSX-Бейсике имеется специальный оператор

CIRCLE (X, Y), R [, C]

Здесь CIRCLE (окружность, круг) — служебное слово, а X, Y, R и C — арифметические выражения. Оператор вычерчивает окружность, координатами центра которой являются целые части значений X и Y, а радиусом — целая часть значения выражения R. Значение C задает цвет точек окружности.

```

10 'КОНЦЕНТРИЧЕСКИЕ ОКРУЖНОСТИ
20 SCREEN2
30 FOR R=5 TO 50 STEP5
40 CIRCLE (120, 90), R
50 NEXT
60 GOTO 60

```

Программа вычерчивает концентрические окружности с центром в точке (120, 90) и радиусами 5, 10, 15, ..., 50. Путем введения дополнительных параметров оператор CIRCLE позволяет окружности превращать в эллипсы, а также строить заданные дуги этих кривых.

Изображение отрезков прямых и прямоугольников выполняется оператором

$$\text{LINE } (X_1, Y_1) - (X_2, Y_2), [C] \left\{ \begin{array}{l} [B] \\ [BF] \end{array} \right\}$$

Здесь LINE (линия) — служебное слово, X_1, X_2, Y_1, Y_2, C — арифметические выражения, B и BF — параметры, задающие режим работы оператора. Простейшая форма оператора

$$\text{LINE } (X_1, Y_1) - (X_2, Y_2)$$

обеспечивает изображение отрезка, соединяющего точки, координаты которых задаются целыми частями значений X_1, Y_1 и X_2, Y_2 соответственно (рис. 65, а). Присутствие указателя цвета C определяет цвет этого отрезка. Однако наиболее существенное влияние на выполнение оператора LINE оказывает присутствие или отсутствие параметров B и BF. Оператором

$$\text{LINE } (X_1, Y_1) - (X_2, Y_2), C, B$$

цветом C (указатель цвета может и отсутствовать) рисуется прямоугольник со сторонами, параллельными осям координат, у которого точки с координатами (INT(X_1), INT(Y_1)) и (INT(X_2), INT(Y_2)) являются противоположными вершинами (рис. 65, б). Внутренняя часть прямоугольника не закрашивается. Если вместо параметра B поставить BF, то так же, как и в предыдущем случае, нарисуется прямоугольник, но его внутренняя часть окрасится в цвет C.

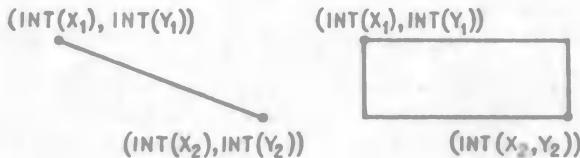


Рис. 65

а)

б)

Контрольные вопросы

1. Какие устройства вывода графической информации используются в микроЭВМ?
2. Что общего в структуре рабочих полей дисплея и графопостроителя?
3. Каковы основные возможности оператора PLOT в изображении графической и символьной информации?
4. Какими средствами обеспечивается изображение основных графических объектов (точек, прямых, окружностей, прямоугольников) в MSX-Бейсике?

Упражнения

1. Составить программу вывода на экран дисплея чертежа равнобо́чной трапеции с основаниями A и B ($A > B$) и боковой стороной C .
2. Составить подпрограмму с параметрами для построений графика функции. Подпрограмма должна строить график в произвольно задаваемой системе координат в виде сплошной (на участках непрерывности) линии, предусматривать возможность изменения масштаба и отмечать в ходе построения графика точки его пересечения с осями координат соответствующими числовыми значениями.
3. Используя графические операторы MSX-Бейсика, составить программы, рисующие схемы базовых структур алгоритмов (рис. 44, 45, 46).

4.7. Запись алгоритмов на языке Паскаль

Стремление усовершенствовать методику конструирования алгоритмов, приведшее к появлению структурного подхода, не могло не оказать влияния на разработку новых языков программирования. Потребность в разработке новых алгоритмических языков вытекает из многих факторов. Это и желание учесть недостатки и ошибки ранее разработанных языков, а также (и это наиболее существенно) учесть новые возможности вычислительных систем. Так, появившиеся в сравнительно недавнее время версии алгоритмических языков Паскаль и Ада уже учитывали потребности их использования в реальном масштабе времени, в режимах мультипрограммирования и параллельной обработки. В новых языках предусмотрена возможность описывать произвольные типы данных, в том числе литерные данные, массивы и множества. Вместе с тем язык Паскаль, а затем и построенный на его основе язык Ада не избежали влияния идей структуризации, что привело к появлению в них специальных средств, облегчающих описание структурированных алгоритмов.

Языки, специально не ориентированные на описание структурированных алгоритмов, также могут быть применены для струк-

турного программирования, а сама методология этого подхода оказывается плодотворной и в этих случаях. К тому же выгоды структурного построения алгоритмов начинают сказываться лишь в программах, достигающих определенного уровня сложности и размеров, в отличие от тех, которые в иллюстративных учебных целях рассматривались в тексте пособия. Особое неприятие сторонников структурного подхода вызывает, как известно, оператор GOTO. Было время, когда даже само структурное программирование определялось как программирование без GOTO. Действительно, неустранимая потребность частого употребления этого оператора в «неструктурированных» языках и является той неприятной их особенностью, которая способна превращать даже достаточно простые программы в нагромождения трудно улавливаемых переходов, убивающих ясность и простоту программ. Как бы там ни было, но новые («структурные») языки имеют средства, позволяющие исключать оператор GOTO во многих стандартных (для предшествующих языков) случаях, хотя и имеют этот оператор в своем арсенале.

Рассмотрим кратко эти новые черты современных языков на примере алгоритмического языка Паскаль, который был разработан более 15 лет назад Н. Виртом (Цюрих, Швейцария) и сразу получил широкое распространение как язык для публикаций и обучения программированию. Вскоре Паскаль был реализован на больших ЭВМ, а в настоящее время является составной частью математического обеспечения ряда моделей микроЭВМ.

Строки программы на языке Паскаль не требуют сквозной нумерации, как это делается, например, в Бейсике. Принципы организации программ на языке Паскаль, основываясь на строгом синтаксисе языка, допускают тем не менее достаточно свободы для наглядного изображения программ на бумаге в процессе их разработки. Эти особенности программирования на Паскале при достижении определенного навыка существенно подрывают традиционную роль схем алгоритмов как средства их предварительной записи.

В программе на Паскале выделяются следующие основные разделы:

```
<заголовок программы>;  
<описания>;  
<операторы>.
```

Разделы отделяются друг от друга точкой с запятой, в конце программы ставится точка. Каждый раздел начинается со своего ключевого слова.

Ключевым словом заголовка является слово PROGRAM*.

* В зависимости от возможностей устройств ввода-вывода изобразительные средства Паскаля в конкретных реализациях могут расширяться за счет использования различных шрифтов. В данном случае, как и при изложении Бейсика, используются лишь прописные буквы латинского и русского алфавита.

Вслед за ключевым словом указывается имя программы, а также информация об использовании устройств ввода-вывода. Пример заголовка:

PROGRAM ПОИСК (OUTPUT, INPUT);

ПОИСК — это в данном случае имя программы, а служебные слова OUTPUT и INPUT сообщают машине, что в программе будет предусмотрен вывод результатов (OUTPUT), а также ввод значений исходных данных в процессе работы программы (INPUT). Второе из этих указаний может в определенных ситуациях отсутствовать.

Раздел описаний содержит информацию для машины о перечне и свойствах величин, используемых в программе. В Паскале описания подразделяются на описания меток, описания констант, описания типов данных, описания переменных и описания функций и процедур. В каждой конкретной программе могут присутствовать лишь некоторые из указанных описаний, однако отмеченный выше порядок их следования должен выдерживаться. Обязательной частью раздела описаний является описание переменных. Описание переменной имеет вид:

VAR <имя> : Т

Здесь <имя> — имя (идентификатор) переменной, а Т — ее тип. Если описывается несколько переменных одного и того же типа, то допускается совмещенная форма:

VAR <список имен> : Т

В <списке имен> имена перечисляются через запятые. В Паскале действуют четыре стандартных типа величин (помимо тех, что могут быть дополнительно определены в каждой программе — в разделе описания типов данных): INTEGER (целый), REAL (действительный), CHAR (литерный), BOOLEAN (логический). Имена переменных одного типа могут быть указаны в одной строке описания через запятые. Пример описания переменных действительного типа:

VAR A, B, SUM, ALFA1, ALFA2:REAL;

Именем (идентификатором) может быть любая последовательность букв и цифр, начинающаяся с буквы.

Выражения в Паскале конструируются в рамках установленных типов величин и допустимых операций над этими величинами. Так, к объектам целого типа применимы операции +, -, ×, а также две особые операции DIV (целочисленное деление, вычисляется частное от деления двух натуральных чисел) и MOD (вычисление остатка от целочисленного деления). К объектам реального типа применимы арифметические операции +, -, ×, / (в Паскале нет операции возведения в степень). Над объектами символьного типа можно производить только операции сравнения (<, ≤, =, ≠, >, ≥), применение ко-

торых основывается на определенном для конкретной реализации Паскаля способе упорядочения символов алфавита языка. Символьные константы заключаются в апострофы. Для объектов логического типа определены логические операции \wedge (и), \vee (или) и \neg (не)*. Значением истинности логических выражений является одна из двух констант: TRUE (истинно) или FALSE (ложно). В Паскале установлен следующий порядок выполнения операций: наивысшую степень старшинства имеет операция \neg , затем выполняются операции типа умножения (\times , $/$, \wedge , DIV, MOD) и, наконец, операции типа сложения (+, -, \vee).

Паскаль имеет свой перечень стандартных функций. В их числе обычные для алгоритмических языков функции, такие, как SIN (синус), COS (косинус), ARCTAN (арктангенс), LN (натуральный логарифм), EXP (экспонента), SQRT (квадратный корень), ABS (абсолютная величина). Но есть и новые: SQR (вторая степень), PRED (предшествующий элемент, например $PRED(-5) = -6$, $PRED(4) = 3$, если аргумент — литера, то учитывается упорядоченность символов алфавита), SUCC (следующий элемент), TRUNC (отбрасывание дробной части; например, $TRUNC(\pi) = 3$), ROUND (целая часть числа). Используя в арифметических выражениях объекты типа REAL и INTEGER, нужно учитывать следующие условия. В выражении типа REAL любой элемент типа REAL может быть заменен на элемент типа INTEGER. Но если есть потребность использовать элемент типа REAL там, где используются только элементы типа INTEGER, то необходимо использовать явную функцию преобразования типов (например, TRUNC или ROUND).

Основным разделом программы является раздел операторов, который состоит из последовательности операторов, заключенных в операторные скобки BEGIN и END (такая конструкция в Паскале называется составным оператором). Операторы отделяются друг от друга точкой с запятой, но перед END запятая опускается. Знак присваивания в Паскале обозначается $:$ =, так что операторы присваивания имеют вид:

$<\text{переменная}> : = <\text{выражение}>$

Начальные значения величины частично задаются, как это бывает иногда, в самих программах, однако основная их часть помещается обычно после текста программы. Для ввода исходных данных применяются процедуры READ и READLN (второе имя происходит от слияния двух слов READ и LINE). После выполнения процедуры READ значение следующего данного читается из этой же строки, а при выполнении процедуры READLN — с новой строки. Вывод данных осуществляется процедурами

* В некоторых реализациях языка логические операции обозначаются словами AND (\wedge), OR (\vee), NOT (\neg).

WRITE и **WRITELN** (от слов **WRITE** и **LINE**). Во втором случае после выполнения предписанных указаний о печати происходит переход на новую строку.

Теперь приведем пример записи программы на Паскале.

Пример 4.7.1. Определить, имеется ли среди чисел *a*, *b* и *c* хотя бы одна пара взаимно обратных:

```
PROGRAM ВЗОБР (OUTPUT, INPUT)
  VAR A, B, C:REAL;
      P:BOOLEAN;
  BEGIN
    READ (A, B, C);
    P:= (A * B = 1) V (A * C = 1) V (B * C = 1);
    WRITELN (P)
  END.
```

Эта простейшая программа, помимо общих принципов организации программы на языке Паскаль, демонстрирует еще и использование величин логического типа. Результатом работы программы будет вывод одной из логических констант: **TRUE** (пара взаимно обратных чисел имеется) или **FALSE** (взаимно обратных чисел среди *a*, *b* и *c* нет).

Условный оператор в языке Паскаль имеет две формы, соответствующие вариантам базовой структуры РАЗВИЛКА (см. п. 3.11, с. 133). Одна из них является аналогом полной условной конструкции:

IF <логическое выражение> THEN <оператор> ELSE
<оператор>;

другая — неполной:

IF <логическое выражение> THEN <оператор>;

Используя условный оператор, вывод результата в предыдущей программе можно сделать более информативным. Можно, например, заменить оператор **WRITELN** (*P*) тремя операторами:

```
WRITE ('СРЕДИ ЧИСЕЛ A, B И C');
IF P
  THEN WRITE ('ИМЕЕТСЯ ПАРА')
  ELSE WRITE ('НЕ ИМЕЕТСЯ');
  WRITELN ('ВЗАЙМО НА ОБРАТНЫХ')
```

Как видно уже из этого простого примера, естественная форма оператора IF позволила отказаться от неизбежного в данном случае при использовании Бейсика оператора **GOTO**, что делает программу более лаконичной и наглядной.

Особенно эффективно идеи структуризации реализованы в операторе цикла языка Паскаль. Этот оператор имеет три формы записи, две из которых фактически совпадают с базовыми структурами ЦИКЛ-ПОКА и ЦИКЛ-ДО (см. п. 3.11, с. 134).

Оператор цикла, соответствующий структуре ЦИКЛ-ПОКА, имеет в Паскале вид:

WHILE · <условие> DO
 <тело>

Здесь WHILE (пока), DO (делать, исполнять) — ключевые слова, <условие> является логическим выражением, а <тело> состоит из одного или нескольких операторов (в последнем случае тело является составным оператором, т. е. заключается в скобки BEGIN и END). Тело цикла исполняется до тех пор, пока <условие> остается истинным.

Оператор цикла, соответствующий структуре ЦИКЛ-ДО, записывается по форме

REPEAT <тело>
 UNTIL <условие>

Здесь REPEAT (повторять), UNTIL (до) — ключевые слова, а <тело> и <условие> имеют тот же смысл, что и в ЦИКЛ-ПОКА. В данном случае выход из цикла происходит сразу после того, как <условие> станет истинным.

Если число повторений в цикле не зависит от результата работы оператора цикла, а, например, известно заранее, то для этих целей лучше всего может быть использован имеющийся в Паскале оператор цикла с параметром:

FOR · <параметр> := <выражение> TO · <выражение>
 DO <тело>

Рассмотрим примеры организации циклических программ на языке Паскаль.

Пример 4.7.2. Программа нахождения наибольшего общего делителя двух целых положительных чисел (рис. 33, с. 111 и программа примера 4.4.1, с. 170):

```
PROGRAM НОД (OUTPUT, INPUT);
  VAR A, B: INTEGER;
  BEGIN
    READ (A, B);
    X:=A; Y:=B;
    WHILE X≠Y DO
      IF X>Y
        THEN X:=X-Y
        ELSE Y:=Y-X
    WRITE ('НОД ('; A, ', ', B, ') = ', X)
  END.
```

Пример 4.7.3. Программа генерирования 100 членов последовательности с общим членом $a_k = \frac{k^3}{k^3 + 1}$:

```

PROGRAM ПОСЛ (OUTPUT);
  VAR K, N: INTEGER;
    A: REAL;
  BEGIN
    K:=1;
    REPEAT
      N:=SQR (K);
      A:=N/(N * K+1)
      WRITELN ('A (' , K, ') =', A);
      K:=K+1
    UNTIL K>100
  END.

```

Роль операторных скобок BEGIN и END для тела цикла в данном случае выполняют ключевые слова REPEAT и UNTIL.

Эта же задача может быть решена с использованием оператора цикла с параметром. Ниже приведена только операционная часть программы:

```

BEGIN
  FOR K:=1 TO 100 DO
  BEGIN
    N:=SQR (K);
    A=N/(N * K+1);
    WRITELN ('A (' , K, ') =', A)
  END
END.

```

Выше мы отмечали четыре стандартных типа (REAL, INTEGER, CHAR, BOOLEAN), которые считаются заданными для каждой Паскаль-программы. Важнейшей особенностью Паскаля является наличие мощного аппарата, позволяющего вводить произвольные типы данных.

Все типы данных в Паскале разбиваются на два больших класса — *скалярные* (перечислимые) и *структурные*. Описание скалярного типа имеет вид:

TYPE <имя>=(<список констант>)

Здесь TYPE (тип) — ключевое слово, <список констант> — перечень констант задаваемого типа, перечисленных через запятые. Пример:

TYPE ДЕНЬ=(ПОНЕДЕЛЬНИК, ВТОРНИК, СРЕДА,
ЧЕТВЕРГ, ПЯТНИЦА, СУББОТА,
ВОСКРЕСЕНЬЕ);
ФИГУРА=(КРУГ, ПРЯМОУГОЛЬНИК);

Значения скалярного типа упорядочены и получают порядковые номера. Порядковый номер первого элемента — 0, второго — 1 и т. д. Из этого следует, например, что

PRED (СУББОТА) = ПЯТНИЦА
SUCC (КРУГ) = ПРЯМОУГОЛЬНИК

Введя в программу описание новых типов, мы получаем возможность описывать переменные этих типов. Так, имея приведенные выше скалярные типы ДЕНЬ и ФИГУРА, можно составить, например, такие описания переменных:

VAR X, Y:ДЕНЬ;
A:ФИГУРА;

Согласно приведенному описанию переменные X и Y могут иметь своими значениями ПОНЕДЕЛЬНИК, ВТОРНИК, СРЕДА, ЧЕТВЕРГ, ПЯТНИЦА, СУББОТА, ВОСКРЕСЕНЬЕ, а переменная A — одно из значений КРУГ, ПРЯМОУГОЛЬНИК.

Значения переменных структурного типа (в отличие от скалярных переменных) состоят из нескольких отдельных компонент. Структурные типы данных позволяют, например, описать информацию: фамилия, имя, отчество; год рождения; номер школы; город и т. п. Среди данных структурного типа в Паскале, в частности, имеются множества и массивы.

Множество — это любой набор объектов одного типа, воспринимаемый как целое. Элементом множества является все, что может быть именем в Паскале. Множества задаются в квадратных скобках или перечислением, или указанием диапазона. Например:

[2, 8, 73]
[5, 9..14, 19]
['КРАСНЫЙ', 'ГОЛУБОЙ', 'ЗЕЛЕНЫЙ', 'БЕЛЫЙ']

Задание множества указанием диапазона включает нижнюю и верхнюю границы значений, разделенных двумя последовательными точками. Над множеством в Паскале можно производить операции присваивания (`:=`), отношения (`=, ≠, ≤, ≥`), объединения (`+`), пересечения (`*`), вычитания (`-`). Все стандартные операции над множествами выполняются по правилам, известным из теории множеств. При этом следует иметь в виду, что, как и в теории множеств, множество определяется только своими элементами, порядок их следования роли не играет. Два множества считаются равными, если они содержат одни и те же элементы. Объекты множественного типа могут быть определены в программе, например, через описание скалярного типа:

TYPE ДЕНЬ = (ПН, ВТ, СР, ЧТ, ПТ, СБ, ВС);
ДЕЖУРСТВО = SET OF ДЕНЬ;
VAR X, Y, Z: ДЕЖУРСТВО;

В приведенном перечне описаны скалярный тип ДЕНЬ, множественный тип ДЕЖУРСТВО, а также переменные X, Y и Z типа ДЕЖУРСТВО. В описании типов SET OF — ключевые слова, имеющие смысл «устанавливать принадлежность». Переменные

X, Y и Z типа ДЕЖУРСТВО могут иметь своими значениями подмножества множества ДЕНЬ, например:

[ПН, ЧТ], [ВТ, ЧТ, СБ] и т. п.

Над этими значениями могут выполняться любые разрешенные в Паскале операции над множествами. В рассмотренном примере тип ДЕЖУРСТВО — производный множественный тип для типа ДЕНЬ, а ДЕНЬ — базовый тип для типа ДЕЖУРСТВО.

Массив — это упорядоченный набор переменных одинакового типа. В Паскале массивом, в частности, может быть представлена и строка текста (массив литер); это позволяет при желании обращаться к любому символу текста (через его номер). Пример описания массива вещественных чисел:

VAR A: ARRAY [1..N] OF REAL;

Приведено описание массива вещественных чисел A[1], A[2], ..., A[N] (индексы в Паскале пишутся в квадратных скобках). ARRAY (массив), OF (из) — ключевые слова.

П р и м е р 4.7.4. Составить программу вычисления скалярного произведения двух векторов.

Координаты векторов X (X₁, X₂, ..., X₂₀) и Y (Y₁, Y₂, ..., Y₂₀) определяют два массива вещественных чисел. Скалярное произведение находится по формуле

$$S = \sum_{k=1}^{20} x_k y_k.$$

```
PROGRAM СКАЛЯР (INPUT, OUTPUT);
  VAR S: REAL; K: INTEGER;
      X, Y: ARRAY [1..20];
  BEGIN
    S := 0; K := 0;
    REPEAT
      K := K + 1;
      S := S + X[K] * Y[K]
    UNTIL K = 20;
  END.
```

Выше были рассмотрены лишь самые первоначальные сведения о некоторых конструкциях Паскаля, отражающих идеи структурного программирования и организации данных. Язык программирования Паскаль, как и последовавший за ним алгоритмический язык Ада, вобрал в себя новые глубокие идеи программирования, подробное рассмотрение которых требует значительно большего места и времени ([11], [10], [8]).

Контрольные вопросы

1. Из каких основных разделов состоит программа на Паскале?
2. Какие стандартные типы величин могут использоваться в языке Паскаль?
3. Каков перечень стандартных функций в языке Паскаль?
4. Каковы правила совместного использования величин типа REAL и INTEGER?
5. Какова структура условных операторов в языке Паскаль?
6. Какую форму имеют в Паскале операторы, реализующие структуры ЦИКЛ-ПОКА, ЦИКЛ-ДО, цикл с параметром?
7. Каким образом в Паскале задаются произвольные типы данных?

Упражнения

Составить программы на Паскале для решения следующих задач:

1. Решение уравнения $ax = b$.
2. Решение неравенства $ax > b$.
3. Табулирование функций:

a) $y = \frac{3 \ln 0,5x}{2+x}$ на отрезке $[1; 2]$ с шагом 0,02;

b) $y = \begin{cases} \sqrt{3x^2 + 8}, & |x| < 10, \\ \ln|x - 0,5|, & |x| \geq 10, \end{cases}$ для $x = -20, -19, \dots, 20$.

4. Дан ряд $\sum_{n=1}^{\infty} \frac{n}{\sqrt{n^2 + 5}}$.

- a) Вычислить сумму первых n членов ряда.
- b) Вычислить сумму всех первых членов ряда, величина которых не меньше заданного числа ε .

5. Дан массив a_1, a_2, \dots, a_{100} .

- a) Определить, является ли он упорядоченным по возрастанию.
- b) Определить суммы положительных и отрицательных элементов.
- c) Отрицательные элементы заменить нулями.
- d) Упорядочить массив по возрастанию.

6. Массивы x_1, x_2, \dots, x_m и y_1, y_2, \dots, y_n упорядочены по возрастанию. Слити эти массивы в один массив z_1, z_2, \dots, z_p ($p = m + n$) так, чтобы он тоже был упорядочен по возрастанию.

4.8. Особенности учебно-производственного языка Рапира

Многие интересные идеи программирования воплощены в языке Рапира, который разработан в начале 80-х годов в СССР как диалоговый учебно-производственный язык и получил распространение на школьных компьютерах. Приятной особенностью

языка Рапира является то, что в основу конструирования его ключевых слов положена русская транскрипция — это облегчает составление и чтение программ, написанных на этом языке.

В алфавит Рапиры вошли русские и латинские буквы, арабские цифры, а также достаточно стандартный набор специальных символов, среди которых появился символ _ (подчерк). Имена в Рапире не могут содержать пробелов, поэтому если имя составляется из нескольких слов, то эти слова соединяют подчерком, например:

СИЛА_ТОКА

Одна из особенностей Рапиры состоит в том, что в этом языке отсутствуют описания типов величин. Одни и те же величины в различных ситуациях могут принимать значения различных типов, среди которых имеются, в частности, и числовые (вещественные и целые), и литерные.

Команда присваивания в Рапире составляется по обычной схеме:

<переменная> := <выражение>

При составлении арифметических выражений, помимо знаков + (сложение), — (вычитание), * (умножение) и / (деление), используются также // (целочисленное деление) и ** (возведение в степень) и часто применяемые в вычислениях стандартные математические функции. Например, вычисление и присваивание значения дискриминанта квадратного уравнения выполняются оператором (командой)

D:=B2-4*A*C**

Кроме команд присваивания, задание переменным значений осуществляют также команды ввода значений. В Рапире используются две формы команды ввода значений:

ВВОД: <список имен>

ВВОД ДАННЫХ: <список имен>

С помощью первой команды вводятся значения литерных констант, с помощью второй — значения других величин, используемых в Рапире (среди них и числовые значения). Если <список имен> состоит из нескольких имен, то они перечисляются через запятую. После того как значения для всего списка имен набраны на клавиатуре, нажимается клавиша выполнения и машина присваивает вводимые значения указанным в списке переменным. Вывод значений на экран дисплея обеспечивается командой

ВЫВОД: <список вывода>

Здесь <список вывода> может содержать константы, переменные

* В первой версии Рапиры знак присваивания имел вид стрелки ⇒ (два символа = и >), & <переменная> и <выражение> менялись местами.

и выражения, перечисленные через запятую. Например, выполнение команд

$A := 2$, $B := 5$

ВЫВОД: "РЕШЕНИЕ:", В/А

повлечет выдачу на экран сообщения

РЕШЕНИЕ: 2.5

Если значения должны быть выведены не на экран, а на бумагу, используется другая форма команды вывода:

ВЫВОД НА ПЕЧАТЬ: <список вывода>

Команды Рапира, реализующие ветвления, в частности следуют конструкции базовой структуры РАЗВИЛКА (полной и неполной форме):

ЕСЛИ <условие>
ТО <серия>
ИНАЧЕ <серия>
ВСЕ

ЕСЛИ <условие>
ТО <серия>
ВСЕ

В данном случае <серия> — это последовательность команд, разделенных точкой с запятой. При записи условий наряду с простыми отношениями разрешается использовать логические операции И, ИЛИ, НЕ, например:

ЕСЛИ БАЛЛ > =4.5 И
РЕКОМЕНДАЦИЯ = "ЕСТЬ"
ТО ВЫВОД: "ПОСТУПАТЬ В ВУЗ"
ВСЕ

При записи команд цикла в Рапире используются ключевые слова-ограничители НЦ (начало цикла) и КЦ (конец цикла), ограничивающие с двух сторон тело цикла, например:

$X := 1$; $S := 0$
ПОКА $X < 100$
НЦ
 $S := S + 1/X$
 $X := X + 1$
КЦ

Этот же цикл может быть переписан и с помощью команды цикла с параметром:

$S := 0$
ДЛЯ X ОТ 1 ДО 100
НЦ
 $S := S + 1/X$
КЦ

Важнейшей особенностью Рапира является развитый аппарат процедур. Форму процедур имеют, по существу, все программы, составляемые на Рапире. Ниже приведен пример описания про-

цедуры нахождения наибольшего общего делителя двух целых положительных чисел:

```
ПРОЦ НОД ( $\Rightarrow$ А,  $\Rightarrow$ В, N $\Rightarrow$ )
    ИМЕНА: X, Y
    НАЧ
        X:=A; Y:=B
        ПОКА X $\neq$ Y
        НЦ
            ЕСЛИ X>Y
                ТО X:=X-Y
            ИНАЧЕ Y:=Y-X
        ВСЕ
        КЦ
        N:=X
    КОН
```

Описанная процедура является процедурой с параметрами. В заголовке процедуры после ключевого слова ПРОЦ (процедура) указано имя процедуры НОД, вслед за которым в круглых скобках через запятую перечислены формальные параметры, состоящие из аргументов и результатов процедуры (перед аргументами и после результатов ставится стрелка \Rightarrow ; если одно имя обозначает одновременно и аргумент и результат, стрелка \Rightarrow ставится дважды: и перед ним, и после него). В данном случае в списке формальных параметров три параметра: А и В являются аргументами, N — результатом. Вслед за заголовком объявляются имена (вспомогательных) переменных X и Y. Объявление имен в процедурах Рапиры выполняет важную роль локализации этих имен. Объявленные имена становятся доступными лишь в теле процедуры, в котором они объявлены. Это означает, что после исполнения процедуры значения этих имен не могут быть прочитаны. Кроме того, если в основной программе имеются величины с теми же именами, которые объявлены в теле процедуры, то значения этих величин при исполнении процедуры не изменятся (это означает, что программист, использующий процедуру, может и не интересоваться системой обозначений имен, принятой в этой процедуре). Параметры процедуры специально объявлять не нужно, так как они обладают теми же свойствами, что и объявленные имена.

Тело описанной выше процедуры НОД заключено в специальные скобки НАЧ (начало) и КОН (конец). В теле процедуры обязательно присутствие оператора, выполняющего присваивание результата (или результатов, если их несколько) их носителям, объявленным в заголовке. В данном случае таким оператором является оператор присваивания N:=X. В частном случае процедура может и не иметь параметров.

Вызов процедуры осуществляется с помощью команды, которая составляется из имени процедуры, за которым в круглых скобках указываются фактические параметры, например:

НОД (12, 18, M)

На месте фактических аргументов могут стоять произвольные арифметические выражения (в данном случае важно только, чтобы их значениями были целые положительные числа). Круглые скобки в команде вызова присутствуют и тогда, когда вызывается процедура без параметров, например:

СОРТИРОВКА ()

Описание процедур-функций в Рапире отличается от описания процедур заголовком: вместо слова ПРОЦ пишется ключевое слово ФУНК. В процедуре-функции список параметров всегда включает только аргументы, поэтому они не снабжаются стрелками. Значение функции в теле всегда присваивается специальной переменной с зарезервированным именем ЗНАЧ (значение). Описание процедуры НОД в форме функции будет иметь вид:

```
ФУНК НОД (A, B)
    ИМЕНА: X, Y
    НАЧ
        X:=A; Y:=B
        ПОКА X!=Y
            НЦ
                ЕСЛИ X>Y
                    ТО X:=X-Y
                ИНАЧЕ Y:=Y-X
                ВСЕ
            КЦ
            ЗНАЧ:=X
    КОН
```

Тело процедуры или функции в Рапире может содержать обращение к другим процедурам или функциям. Особенno интересен случай, когда процедура (или функция) вызывает сама себя. Такие процедуры (функции) называют рекурсивными. Простейший пример рекурсивной функции — вычисление значения факториала по формуле

$$n! = 1 \cdot 2 \cdot 3 \cdots \cdot n.$$

```
ФУНК ФАКТ (N)
    ИМЕНА: X
    НАЧ
        X:=N
        ЕСЛИ X=1
            ТО ЗНАЧ:=1
        ИНАЧЕ ЗНАЧ:=N * ФАКТ (N-1)
        ВСЕ
    КОН
```

Над литерными величинами в Рапире разрешены операции: конкатенация (склеивание), определение длины и вырезка. Конкатенация обозначается знаком +. Длина текста Т записывается

как ДЛИН (T) и имеет своим значением число, показывающее количество символов в T (включая пробелы). С помощью операции вырезки можно вырезать участок текста, для которого указываются номера начального и конечного символа. Например, после выполнения команд

```
P:="ПРОГРАММИРОВАНИЕ"  
Q:=P [4 : 8]; R:=P [7 : 9]
```

значением текста Q будет слово ГРАММ, а текста R — слово МИР. Ниже приведено описание функции ОБРАЩЕНИЕ, использующей операции над литерными величинами.

```
ФУНК ОБРАЩЕНИЕ (T)  
ИМЕНА: X, I  
НАЧ  
X:=T  
ЗНАЧ:=""  
ДЛЯ I ОТ 1 ДО ДЛИН (X)  
НЦ  
ЗНАЧ:=X [I : I] + ЗНАЧ  
КЦ  
КОН
```

Функция ОБРАЩЕНИЕ преобразует текст так, что его первый символ становится последним, второй — предпоследним и т. д.

Из сложных структур данных, используемых в Рапире, остановимся на кортежах. Кортеж — это упорядоченный набор элементов, каждый из которых (как литеры в тексте) имеет порядковый номер. По заданному номеру можно прочитать элемент кортежа или изменить его. Кортежи записываются в Рапире в виде списка элементов, разделенных запятыми и заключенных в угловые скобки (знаки < и >). Например, в результате присваивания

```
M:=<11, "НАТРИЙ", "NA", 22.991>
```

переменная M станет иметь своим значением кортеж, причем M [1]=11, M [2] = "НАТРИЙ", M [3] = "NA", M [4] = 22.991. Длина кортежа K обозначается ДЛИН (K). Кортеж, не содержащий ни одного элемента, называют пустым кортежем: < >; ДЛИН (< >) = 0. Конкатенация (склеивание, сложение) кортежей аналогична конкатенации литерных величин. После выполнения команд

```
A:=<2, "ФЕВРАЛЬ">  
B:=<3, "МАРТ">  
C:=A+B
```

значением C будет кортеж <2, "ФЕВРАЛЬ", 3, "МАРТ">. К кортежам применима и операция вырезки. Возьмем, к примеру, только что полученный кортеж C; тогда C [2:3] будет иметь своим значением кортеж <"ФЕВРАЛЬ", 3>, а C [4:4] — кортеж <"МАРТ">, содержащий один элемент.

Очевидно, что кортежи, состоящие из элементов одного типа, — это аналоги одномерных массивов. Но элементами кортежей в Рапире могут быть, в свою очередь, кортежи. Это позволяет при необходимости конструировать массивы произвольной размерности. Так, например, кортеж

`<<"ОН", "ОНА">, <"ОНО", "ОНИ">>`

соответствует двухмерному литерному массиву (таблице):

Выполним присваивание:

ОН	ОНА
ОНО	ОНИ

`M:=<<"ОН", "ОНА">, <"ОНО", "ОНИ">>`

Теперь значением `M [1]` будет кортеж `<"ОН", "ОНА">`, а значением `M [1] [2]` — константа `"ОНА"`. Для таких случаев в Рапире разрешено писать сокращенное обозначение: `M [1, 2]`.

Рассмотрим пример программной обработки кортежей. Пусть сведения о каждом школьнике хранятся в кортежах вида

`<ФАМИЛИЯ, ИМЯ, КЛАСС, ДЕНЬ, МЕСЯЦ, ГОД>`

Здесь `ДЕНЬ, МЕСЯЦ, ГОД` — числовые величины, а остальные — литерные, например:

`<"ВЕСЕЛОВ", "ДИМА", "9 В", 8, 3, 1971>`

Тогда кортеж таких кортежей может содержать список всех учащихся школы; присвоим ему имя `ШКОЛА`. Поместив список `ШКОЛА` в память ЭВМ, с помощью несложных процедур можно получить любую справку об учащихся. Приведенная ниже команда цикла с параметром по значениям `ДЕНЬ` и `МЕСЯЦ` выдает список (из значений величин `ФАМИЛИЯ, ИМЯ, КЛАСС`) всех учащихся кортежа `ШКОЛА`, имеющих заданный день рождения.

для НОМЕР от 1 до ДЛИН (`ШКОЛА`)

НЦ

ЕСЛИ `ШКОЛА [НОМЕР, 4]` = `ДЕНЬ` И
`ШКОЛА [НОМЕР, 5]` = `МЕСЯЦ`

ТО

ВЫВОД НА ПЕЧАТЬ: `ШКОЛА [НОМЕР, 1]`,
`ШКОЛА [НОМЕР, 2]`, `ШКОЛА [НОМЕР, 3]`

ВСЕ

КЦ

Контрольные вопросы

1. В чем особенности конструирования имен в Рапире?
2. Как записываются в Рапире команды ветвления и цикла?
3. В чем заключается принцип локализации имен в процедурах?
4. В чем суть новой структуры данных — кортежа? Как с помощью кортежей в Рапире можно создавать массивы нужной размерности?

Рекомендуемая литература

1. Основы информатики и вычислительной техники: Проб. учеб. пособие для средних учебных заведений. Ч. I, II / Под ред. А. Н. Ершова, В. М. Монахова.— М., 1985, 1986.
2. Изучение основ информатики и вычислительной техники: Метод. пособие для учителей и преподавателей сред. учеб. заведений. Ч. I, II / Под ред. А. Н. Ершова, В. М. Монахова.— М., 1985, 1986.
3. Антипов И. Н. Программирование.— М., 1976.
4. Абрамов С. А. Элементы программирования.— М., 1982.
5. Беляй Ю. А. Считывающая электроника.— М., 1983.
6. Блох А. Ш., Павловский А. Н., Пенкрагт В. В. Микрокалькулятор в школе.— Минск, 1986.
7. Брудно А. Л., Каплан Л. И. Олимпиады по программированию.— М., 1985.
8. Виленкин Н. Я., Окская В. М., Шварцбурд С. И. Микрокалькулятор — школьнику.— М., 1986.
9. Вирт Н. Систематическое программирование: Введение/Пер. с англ.— М., 1977.
10. Григас Г. Начала программирования/Пер. с лит.— М., 1987.
11. Грогоно П. Программирование на языке Паскаль/Пер. с англ.— М., 1982.
12. Гудман С., Хидетиши С. Введение в разработку и анализ алгоритмов/Пер. с англ.— М., 1981.
13. Дьяконов В. П. Справочник по расчетам на микрокалькуляторах.— М., 1985.
14. Ершов А. П. Алгоритмический язык в школьном курсе основ информатики и вычислительной техники.— М., 1985, № 2.
15. Звенигородский Г. А. Первые уроки программирования/Под ред. А. П. Ершова.— М., 1985.
16. Звенигородский Г. А. Вычислительная техника и ее применение.— М., 1987.
17. Икаунисек Е. Основы языка Бейсик // Информатика и образование.— 1986.— № 3; 1987.— № 1.
18. Ковалев М. П., Шварцбурд С. И. Электроника помогает считать.— М., 1978.
19. Криницкий Н. А. Алгоритмы вокруг нас.— М., 1977.
20. Лодатко Е. А. Школьнику о вычислениях с микрокалькулятором.— М., 1985.
21. Минакова С. С. Вычисления на уроках и внеклассных занятиях по математике.— М., 1983.
22. Монахов В. М., Лапчик М. П., Демидович Н. Б., Черночкова Л. П. Формирование алгоритмической культуры школьника при обучении математике.— М., 1978.
23. Симонс Дж. ЭВМ пятого поколения: компьютеры 90-х годов/Пер. с англ.— М., 1985.
24. Турский В. Методология программирования.— М., 1981.
25. Уздерель Ч. Этюды для программистов/Пер. с англ.— М., 1982.

Предисловие	3
Г л а в а 1. Вычисления на микрокалькуляторах и микроЭВМ	5
1.1. Назначение и устройство микрокалькуляторов	5
1.2. Арифметические расчеты на МК	9
1.3. Калькуляторы для научно-технических вычислений	21
1.4. Особенности школьного калькулятора МКШ-2	35
1.5. Программируемый микрокалькулятор	38
1.6. Вычисления на персональных микроЭВМ	53
Г л а в а 2. Анализ точности вычислений	59
2.1. Откуда берутся ошибки в вычислениях по формуле	59
2.2. Первоначальные понятия приближенных вычислений	60
2.3. Определение количества верных цифр по относительной погрешности приближенного числа	64
2.4. Подсчет погрешностей арифметических действий с приближенными данными	65
2.5. Оценка погрешностей значений функций	69
2.6. Способы приближенных вычислений	76
Г л а в а 3. Основы алгоритмизации	85
3.1. Метод алгоритмизации	85
3.2. Алгоритмы и их свойства	88
3.3. Алгоритмы для вычислений	92
3.4. Схемы алгоритмов	94
3.5. Словесная запись алгоритмов	99
3.6. Запись ветвлений	103
3.7. Циклические алгоритмы	108
3.8. Построчная алгоритмическая нотация	114
3.9. Массивы	122
3.10. Подчиненные алгоритмы	128
3.11. Базовые алгоритмические структуры	132
3.12. Структурная алгоритмическая нотация	141
Г л а в а 4. Программирование	149
4.1. Начальные понятия языка Бейсик	149
4.2. Организация программы	156
4.3. Программирование ветвлений	163
4.4. Программирование циклов	169
4.5. Функции пользователя и подпрограммы	177
4.6. Графические средства Бейсика	181
4.7. Запись алгоритмов на языке Паскаль	191
4.8. Особенности учебно-производственного языка Рапира	200
Рекомендуемая литература	207