С. Т. Усачев

ПЕРСОНАЛЬНАЯ ЭВМ

ДВК-1

Государственный комитет СССР
по народному образованию

С. Т. Усачев

ПЕРСОНАЛЬНАЯ ЭВМ ДВК-1

Москва
Издательство Университета дружбы народов
1988

ББК 32.973—01
У74

Утверждено
Редакционно-издательским советом
Университета

Рецензенты:
д-р техн. наук, проф. В. П. Колесник,
канд. физ.-мат. наук Б. В. Архипов

Усачев С. Т.

У74 Персональная ЭВМ ДВК-1: Учеб. пособие.— М.: Изд-во УДН, 1988.— 144 с., ил.

ISBN 5—209—00009—5

В простой и доступной форме описываются принципы работы диалогового вычислительного комплекса ДВК-1, представляющего собой микроЭВМ, объединенную с дисплеем. Изложение программирования на языке Бейсик построено таким образом, чтобы обеспечить возможность быстрого овладения навыками работы на ДВК. Книга написана с учетом программы курса «Спецпрактикум на микро- и миниЭВМ».

Пособие предназначено для учащихся, преподавателей и широкого круга лиц, желающих овладеть основами программирования.

У $\frac{2405000000—095}{093(02)—88}$ 22—88

ББК 32.973—01

ISBN 5—209—00009—5

ПРЕДИСЛОВИЕ

В настоящее время умение работать на ЭВМ считается второй грамотностью. Обучать этому начинают уже в школе, тем самым предполагая, что предмет изучения сравнительно прост и доступен для восприятия. В сущности это, конечно, правильно. Однако практика показывает, что первоначальное обучение вызывает много трудностей, особенно у взрослых людей гуманитарных специальностей.

Объяснить это можно различными причинами. Прежде всего, имеются объективные трудности. Вычислительная техника развивается очень бурно, непрерывно возникают новые модели электронных вычислительных машин и новые идеи, направленные на упрощение работы с ними, создание удобств и расширение возможностей их использования. Но последнее требует правильного понимания этих возможностей. Процесс обучения осложняется также существованием множества языков программирования. Все эти трудности создают определенный психологический барьер при обучении начинающих.

Однако для использования ЭВМ при решении практических задач рядовому пользователю совсем необязательно изучить программирование в том объеме, как это необходимо специалисту в этой области, а достаточно иметь некоторый минимум знаний, который в ходе дальнейшей работы с ЭВМ будет постепенно расширяться.

Настоящая работа является начальным курсом обучения программированию людей, не имеющих предварительной подготовки в области вычислительной техники. Облегчению обучения способствуют два обстоятельства. Во-первых, диалоговый вычислительный комплекс ДВК-1 несложен в обращении. Во-вторых, язык Бейсик — один из простейших языков программирования высокого уровня, сравнительно близкий к естественному языку, что вытекает из самого его названия. Русское Бейсик происходит от английского Basic, что является сокращением от Beginners' All purpose Symbolic Instruction Code — «универсальный язык символьных инструкций для начинающих».

В главе 1 описываются общие принципы программирования и работы ЭВМ. Определяются основные понятия программирования. Этот вводный материал дает начальные сведения о принципах обработки информации на ЭВМ.

В главе 2 даются те сведения о конкретной ЭВМ ДВК-1, которые необходимо использовать в процессе работы. Они сво-

дятся к правилам подготовки ЭВМ к работе и к использованию клавиатуры дисплея.

В главе 3 представлены сведения, необходимые для восприятия материала последующих глав и для работы на машине. Рекомендуется возвращаться к материалу этой главы до его полного усвоения.

В главе 4 рассмотрены средства и приемы создания простейших программ на языке Бейсик. Изложение материала позволяет быстро приступить к работе на машине и составлению программ.

Необходимость введения новых средств программирования, вызываемая недостатками уже рассмотренных, демонстрируется на примере одной простой задачи вычисления стоимости товаров по данным об их количестве и цене. Использование одной задачи облегчает понимание и сопоставление функций различных операторов.

В главе 5 рассматриваются средства и приемы создания более эффективных программ.

Использование массивов позволяет создавать эффективные программы, например, для обработки таблиц — характерной составной части экономических расчетов. Применение функций, определяемых пользователем, и подпрограмм позволяет описать какую-либо последовательность операций лишь один раз, а затем многократно обращаться к этому описанию из различных мест программы. Типовые программы, анализ которых помогает усвоить приемы программирования, могут служить готовым материалом для практической работы.

В приложении рассматриваются стандартные математические функции, программы вычисления которых входят в интерпретатор Бейсик. Наличие этих программ избавляет пользователя от необходимости самостоятельного их создания. В приложение включены также справочные материалы.

Материал, изложенный в данной работе, дает возможность неподготовленному читателю пройти курс обучения языку программирования Бейсик. В случае успешного усвоения языка читатель получает в свое распоряжение средства, достаточные для решения многих практических задач. Кроме того, этот курс является хорошей основой для дальнейшего изучения более сложных языков программирования.

Глава 1
ОБЩИЕ ПРИНЦИПЫ ПРОГРАММИРОВАНИЯ

1. ОБЩИЕ ПОНЯТИЯ О ПРОГРАММАХ

Знакомство с обработкой информации на ЭВМ проводится на примере конкретной задачи вычисления стоимости товаров. Пусть сведения о товарах представлены в виде табл. 1.

Таблица 1

Количество и цена товаров

Наименование товара	Количество, шт.	Цена, руб.
Электропроигрыватель	20	135
Магнитофон	10	265
Кассета	50	4
Грампластинка	100	2,50

Требуется вычислить стоимость товаров каждого наименования.

Для решения этой задачи человеком достаточно задать исходные данные и формулировку задачи. Результат решения он представит в виде табл. 2.

Таблица 2

Стоимость товаров

Наименование товара	Количество, шт.	Цена, руб.	Стоимость, руб.
Электропроигрыватель	20	135	2700
Магнитофон	10	265	2650
Кассета	50	4	200
Грампластинка	100	2,50	250

Для решения той же задачи на ЭВМ в ее распоряжение надо предоставить гораздо бо́льшую информацию, чем для человека.

Прежде всего ЭВМ не обладает ни способностью к мышлению, ни знаниями и потому не может вывести из формулировки задачи последовательности действий, приводящих к ее решению. Как бы ни были очевидны (для человека) отдельные действия, для ЭВМ требуются полные и точные указания о них. ЭВМ ничего не может додумать, а может только очень быстро выполнить то, что ей указано. При этом все указания ЭВМ воспринимает буквально, без их расширенного толкования и без скрытого подтекста или переносного смысла. Указание, определяющее действие ЭВМ, называется **предписанием**.

Примерный набор предписаний для решения рассматриваемой задачи на ЭВМ можно представить в следующих понятных для человека словесных формулировках.

Предписания для вычисления стоимости товаров

1. Ввести в память ЭВМ данные о количестве и цене товаров одного наименования.
2. Получить стоимость умножением количества на цену.
3. Выдать на экран дисплея сведения о количестве, цене и стоимости товаров одного наименования, располагая эти сведения в одной строке.
4. Повторить действия, приведенные в пунктах с 1 по 3 для товаров всех наименований.

В этих предписаниях уже встречаются некоторые характерные для программирования термины. Пока достаточно воспринимать их интуитивно, не отвлекаясь от уяснения главного: ЭВМ выполняет только последовательность предписаний, и поэтому они должны быть абсолютно полны и точны.

В соответствии с указанными предписаниями ЭВМ должна была бы выдать ответ в следующем виде.

Результаты работы ЭВМ по приведенным предписаниям

20	135	2700
10	265	2650
50	4	200
100	2,50	250

А где же наименования таблицы и граф, где сетка таблицы? Их нет, так как не было предписаний получить их. Этот пример ярко показывает, что ЭВМ делает только то, что предписано.

Следует также отметить, что последнее предписание (п. 4), неточно, так как не указано, каким образом ЭВМ может определить окончание вычислений. Окончание данных в исходной таблице является достаточным сигналом для человека об окончании работы, так как он делает ее осмысленно. ЭВМ выполняет работу механически, переходя от предписания к предписанию. Обработав данные об очередном товаре, ЭВМ должна вернуться к первому предписанию. Если не будет специальных

указаний, то она сделает то же самое и после обработки данных о последнем товаре. Но выполнить первое предписание она уже не сможет ввиду исчерпания всех данных. Переходить к последующему предписанию, не выполнив предыдущего, ЭВМ также не имеет права. В случае невозможности выполнения предписания (по любой причине) происходит аварийный останов ЭВМ с выдачей сообщения о его причинах.

Итак, для работы ЭВМ ей необходимо задать полные и точные предписания, определяющие ее действия. Теперь встает вопрос о форме представления этих предписаний.

Разговорный язык с его сложной грамматикой, огромным словарем и многообразием построений фраз для передачи одной и той же мысли недоступен ЭВМ. Ей доступны лишь сравнительно простые, специально для нее разработанные языки. Каждый из них, как и человеческие языки, имеет свой алфавит, словарь и грамматику.

Следовательно, предписания должны составляться на доступном для машины языке. Предписания, определяющие действия ЭВМ, составленные на языке, доступном для машины, называются **программой**. Отдельное предписание программы называется **командой программы**.

В общем случае доступность языка для машины означает лишь возможность восприятия машиной программы на данном языке. Но это не означает возможности непосредственного выполнения команд программы без предварительного изменения формы их представления. Язык описания действий машины в форме, используемой для их непосредственного выполнения, называется **машинным языком**, а в форме, требующей предварительного перед выполнением преобразования,— **языком программирования.**

Алфавит машинного языка состоит всего лишь из двух символов: нуля и единицы — цифр двоичной системы счисления. Выбор этой системы для работы ЭВМ объясняется простотой получения всего лишь двух различных сигналов, соответствующих ее двум символам. Использование же различных комбинаций из нулей и единиц позволяет закодировать любые символы аналогично азбуке Морзе, в которой также используются только 2 символа: точка и тире.

Действия на машинном языке описываются в виде команд. **Машинной командой** называется описание на машинном языке элементарной операции, которую должна выполнить ЭВМ. Машинная команда определяет вид операции (например, сложение или пересылка данных) и местонахождение операндов (т. е. объектов операции) в памяти машины. Машинная команда имеет вид последовательности нулей и единиц определенной длины и структуры. Длина и структура машинной команды одного содержания могут отличаться в зависимости от типа ЭВМ. Это означает, что, несмотря на использование одного и того же двоичного алфавита, машинные языки этих ЭВМ не иден-

тичны из-за различия слов и грамматики. Например, в ДВК-1 используются в основном машинные команды длиной в 16 двоичных разрядов, а в машинах общего назначения ЕС ЭВМ — команды длиной от 16 до 48 двоичных разрядов.

Очевидно, что человеку неудобно составлять программы для ЭВМ на машинном языке по разным причинам.

1. Запись машинной команды в виде длинной последовательности нулей и единиц требует много времени и большóго внимания. Этот процесс порождает множество ошибок. Анализ смыслового содержания машинной команды весьма затруднен формой ее представления.

2. Машинная команда определяет элементарную операцию ЭВМ, а потому является слишком мелкой операцией. Даже для выполнения одного из действий арифметики может потребоваться больше одной машинной команды, например, из-за необходимости предварительной пересылки данных между различными запоминающими устройствами ЭВМ. Вычисление же по длинной формуле приведет, соответственно, к длинной последовательности машинных команд.

3. Использование программы на машинном языке возможно лишь на ЭВМ данного типа и невозможно на ЭВМ других типов с другими машинными языками.

Возникает противоречие: ЭВМ может выполнять команды только на машинном языке, а для человека этот язык неудобен. Противоречие разрешается использованием языков программирования. Язык программирования — это язык, удобный для составления программы человеком, но требующий последующего перевода с него на машинный язык.

Перевод программы с языка программирования на машинный язык осуществляется самой машиной с помощью специальной программы-переводчика, называемой **транслятором** (англ. translator — переводчик).

Составление программ на языках программирования дает человеку следующие преимущества:

1. Языки программирования значительно ближе к человеческим языкам, чем машинные.

2. Языки программирования позволяют более укрупненно записывать действия ЭВМ. Например, вычисление по формуле — одна команда на языке программирования.

3. По степени зависимости от машинного языка языки программирования делятся на две группы: машинно-зависимые, или машинно-ориентированные, и машинно-независимые. Последние позволяют обрабатывать одну и ту же программу на ЭВМ с различными машинными языками при наличии соответствующих трансляторов.

У языков программирования имеются также и недостатки.

1. Каждый из языков программирования эффективен лишь в своей области применения, поэтому существует множество таких языков.

2. Программа на машинном языке, получаемая в результате трансляции с машинно-независимого языка программирования, оказывается менее эффективной, чем программа, непосредственно составленная на машинном языке. Именно этим объясняется создание машинно-ориентированных языков программирования, занимающих промежуточное положение между машинным языком и машинно-независимыми языками программирования.

На примере программы-транслятора видна большая роль специальных программ в повышении удобства использования ЭВМ. В современных ЭВМ роль таких программ непрерывно возрастает. Совокупность специальных программ, предназначенных для повышения удобства и эффективности использования ЭВМ, называется **программным обеспечением.** В целях повышения эффективности этих программ они составляются на машинном или на машинно-ориентированных языках.

Процесс составления программы называется **программированием**, а человек, создающий их, — **программистом**. В настоящее время этот термин применяется в основном для обозначения должности работника, занимающегося программированием профессионально. Высококвалифицированный программист, разрабатывающий специальные программы для системы программного обеспечения на машинном или на машинно-ориентированных языках, называется **системным программистом**, а человек, использующий ЭВМ для решения задач по своим или чужим программам,— **пользователем** (англ. user — пользователь).

В соответствии с приведенными определениями можно произвести и дальнейшую классификацию программ. **Программой пользователя** называется программа решения некоторой его конкретной задачи. **Системной программой** называется специальная программа, предназначенная для повышения удобства использования ЭВМ или повышения эффективности ее работы и применяемая при выполнении различных программ пользователя.

2. УСТРОЙСТВО И РАБОТА ЭВМ

Упрощенная функциональная схема ЭВМ представлена на рис. 1.

Центральным устройством ЭВМ является **арифметико-логическое устройство (АЛУ)**. АЛУ называется устройство ЭВМ, непосредственно осуществляющее обработку информации, например, операции вычисления, сравнения, пересылки.

Операции АЛУ определяются выполняемой в данный момент программой. В этом заключается **принцип программного управления** работой ЭВМ. Это может быть программа пользователя или системная программа. Обычно они работают попеременно: в процессе выполнения программы пользователя вызывается выполнение той или другой системной программы (например,

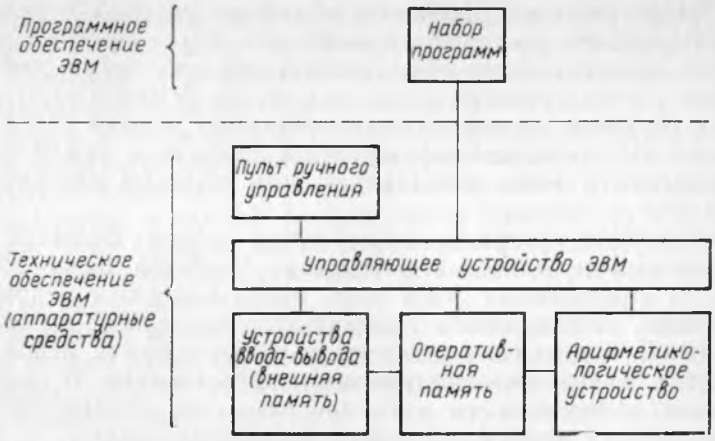


Рис. 1. Упрощенная функциональная схема ЭВМ

программы-транслятора). В свою очередь, системная программа, окончив свою работу, вызывает продолжение работы программы-пользователя (или, как говорят, передает ей управление) и т. д. Таким образом, программное обеспечение является таким же необходимым для работы ЭВМ средством, как и ее техническое обеспечение (аппаратурные средства).

Выполняемая программа должна находиться в **оперативной памяти ЭВМ**. В этом заключается принцип **хранимой в памяти программы**. Операнды, необходимые для выполнения текущей операции, также должны быть расположены в оперативной памяти. АЛУ может обрабатывать лишь информацию, содержащуюся в оперативной памяти, туда же оно отправляет результаты обработки. Оперативная память — это устройство для хранения информации на время ее обработки.

Длительное хранение информации осуществляется на машинных носителях, составляющих **внешнюю память ЭВМ. Машинными носителями** информации называются средства хранения информации, с которыми может работать ЭВМ. К ним относятся перфокарты, перфоленты, магнитные ленты, магнитные диски, бумага для печатающего устройства, экран дисплея. Обмен информацией между внешней и оперативной памятью осуществляется с помощью **устройств ввода — вывода**. Сюда относятся карточный и ленточный перфораторы для работы с соответствующими перфоносителями, накопитель на магнитной ленте (магнитофон), накопитель на магнитных дисках (дисковод), печатающее устройство, дисплей. В ДВК-1 имеется лишь единственное устройство ввода — вывода информации — дисплей.

Пульт ручного управления используется для включения и выключения ЭВМ, изменения режимов ее работы и для оперативного вмешательства в процесс обработки информации.

Управляющее устройство координирует работу всех устройств ЭВМ.

В процессе работы ЭВМ программа и данные с помощью устройств ввода помещаются в оперативную память машины. По командам программы арифметико-логическое устройство производит обработку находящейся в оперативной памяти информации. Результаты обработки также помещаются в оперативную память. Затем с помощью устройств вывода производится выдача информации из оперативной памяти на машинные носители информации.

3. АЛГОРИТМЫ И БЛОК-СХЕМЫ

Точное описание вычислительного процесса, направленного на получение результата из возможных значений исходных данных, называется **алгоритмом**. Алгоритм, записанный на доступном для машины языке, называется **программой**.

Для облегчения процесса составления программы алгоритм предварительно представляется в некоторой описательной форме, например, в текстовой или графической. Форма описательного представления алгоритма и степень его детализации должны помогать программисту в составлении программы. Следовательно, они зависят от сложности задачи, а также от квалификации и стиля работы программиста.

В п. 1 описание алгоритма решения задачи о стоимости товаров представлено в виде текста, состоящего из предписаний. Для такой простой задачи можно было бы ограничиться укрупненным описанием без излишней детализации: чтобы получить стоимость, необходимо количество каждого товара умножить на его цену.

При решении сложной задачи, требующей составления длинной программы с разветвлениями и циклами, используется графическая форма представления алгоритма. Наглядное графическое изображение алгоритма с помощью стандартизованных геометрических фигур называется **блок-схемой** алгоритма.

Блок-схема, соответствующая словесным предписаниям для вычисления стоимости товаров, представлена на рис. 2.

На блок-схеме каждая фигура соответствует одному блоку программы. Степень детализации блоков определяется программистом. Содержание блока описывается внутри фигуры словами или математическими выражениями. Вид фигуры соответствует виду описываемых в ней операций.

Между параллельными линиями, соединенными полуокружностями, указывается начало программы, останов вычислений и конец программы. В параллелограммах указываются операции ввода — вывода, в прямоугольниках — операции обработки информации, в ромбах — проверка условий с последующими разветвлениями в программе. Блоки можно нумеровать для удобства ссылок в текстовом описании блок-схемы и программы. Связи между блоками указываются стрелками. При разры-

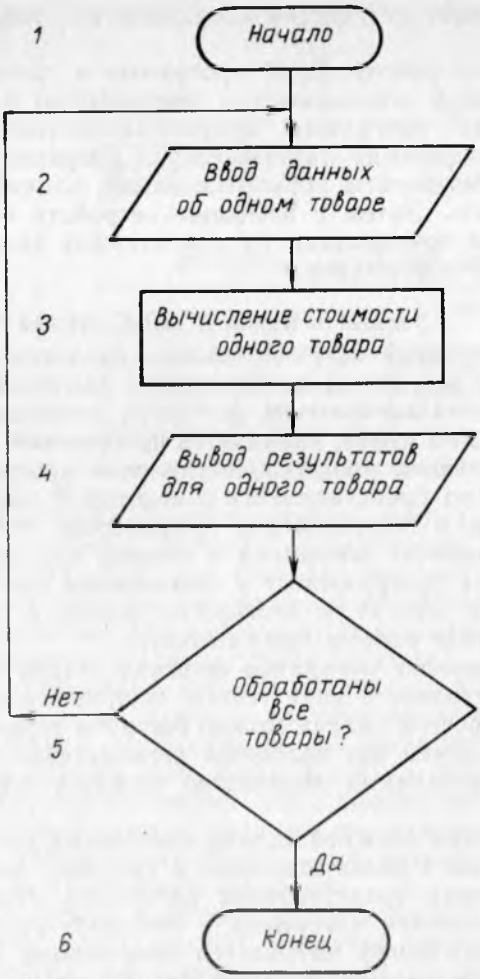


Рис. 2. Блок-схема для вычисления стоимости товаров

ве стрелки она заканчивается, а затем начинается небольшим кружочком — **коннектором**. Внутри коннектора ставится метка. Линии с одинаковыми метками коннекторов считаются соединенными.

Блок-схема дает наглядное и четкое представление об алгоритме. Она избавляет от необходимости удерживать в голове весь алгоритм со всеми взаимосвязями его частей. Она позволяет программировать каждый блок отдельно, причем в произвольном порядке. Требуется лишь обеспечить установление указанных между блоками связей. В сложных задачах блок-схемы позволяют производить программирование отдельных крупных блоков различными программистами. И, наконец, блок-схема облегчает чтение уже составленной программы как своей так и чужой.

Глава 2
ДИАЛОГОВЫЙ ВЫЧИСЛИТЕЛЬНЫЙ КОМПЛЕКС ДВК-1

1. ОБЩЕЕ УСТРОЙСТВО ДВК-1

Внешний вид диалогового вычислительного комплекса ДВК-1 представлен на рис. 3. Монитор *1*, входящий в состав дисплея, служит для представления на экране *2* алфавитно-цифровой и графической информации. Информация отображается на экране электронно-лучевой трубки с помощью системы телевизионной развертки, входящей в состав монитора.

На экране *2* представляются 25 строк информации. Верхняя 25-я строка является служебной, определяющей режим функционирования дисплея. Последние три пары цифр в строке указывают десятки и единицы часов, минут и секунд, прошедших со времени включения дисплея. Остальные 24 строки (нумеруемые сверху вниз от 1) по 80 символов в каждой предоставляются в распоряжение пользователя. Дисплей имеет запоминающее устройство «ЗУ» на 48 строк по 80 символов, т. е. емкостью в 2 экрана. Яркость и четкость изображения на экране регулируются ручками яркости *4* и фокусировки *3*.

На задней панели блока сопряжения *5* находится тумблер «СЕТЬ» *6*, а на передней панели — группа индикаторов и клавиш *7*. Совместно с клавишей «СЕТЬ» *9* блока логики *8* они используются для включения ДВК-1, т. е. реализуют часть функций пульта ручного управления, представленного в общей схеме ЭВМ на рис. 1.

Блок клавиатуры дисплея *10* с индикаторами *11* и клавишами *12* реализует остальные функции пульта ручного управления, а также функции устройства ввода — вывода. Внешний вид блока клавиатуры приведен на рис. 4, а расположение на нем индикаторов и клавиш — на рис. 5. Индикаторы расположены в верхнем ряду.

Длинная клавиша без обозначения в нижнем ряду между клавишами «РУС» и «ЛАТ» является клавишей пробела. Остальные клавиши, приведенные на рис. 5 без обозначений, использовать в ДВК-1 не рекомендуется, так как их нажатие может привести к сообщению об ошибке при последующем правильном наборе текста. Для нейтрализации последствий нажатия этих клавиш нажимается клавиша «ВК» (возврат каретки).

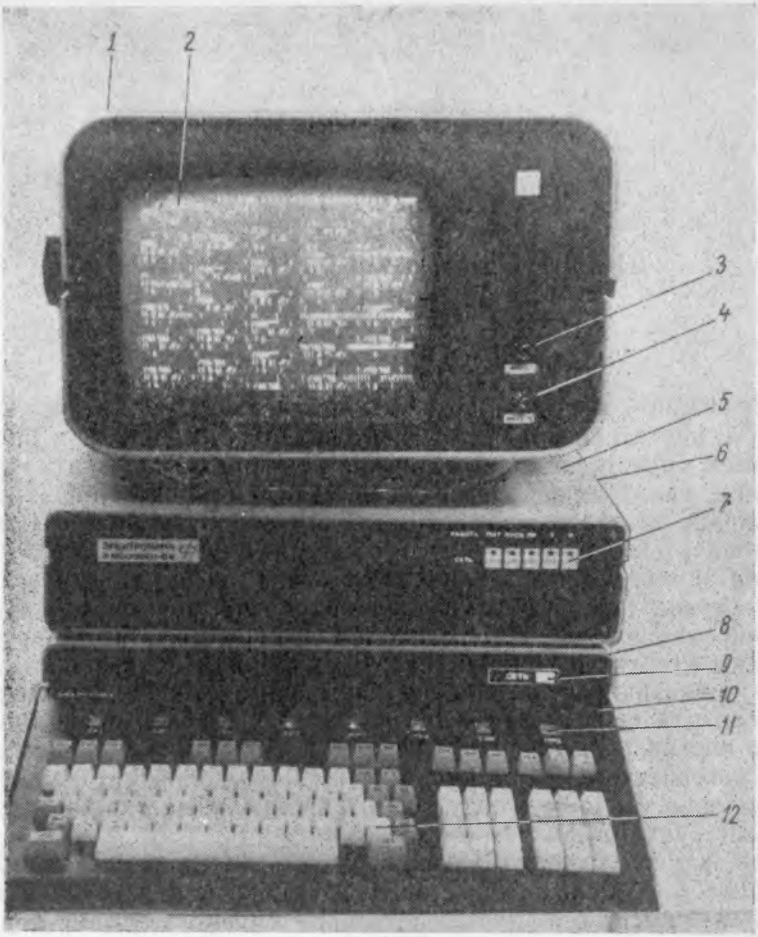


Рис. 3. Внешний вид ДВК-1:

1 — монитор; *2* — экран; *3* — ручка «ФОКУС»; *4* — ручка «ЯРКОСТЬ»; *5* — блок сопряжения; *6* — тумблер «СЕТЬ» на задней панели блока сопряжения; *7* — индикаторы и клавиши управления на передней панели блока сопряжения; *8* — блок логики; *9* — индикатор и клавиша «СЕТЬ» на передней панели блока логики; *10* — блок клавиатуры дисплея; *11* — индикаторы блока клавиатуры; *12* — клавиши блока клавиатуры

Пользователи работают на ДВК-1 в режиме, задаваемом нажатием клавиш: «ДУП», «ЛИН», «РЕД», расположенных вверху клавиатуры под индикаторами. Эти клавиши фиксируются в утопленном положении. При этом включаются соответствующие им индикаторы. Обычно эти клавиши на ДВК-1 всегда утоплены и перед работой надо просто в этом убедиться.

Цифровые клавиши и клавиша запятой, расположенные отдельной группой в правой части клавиатуры, а также клавиши

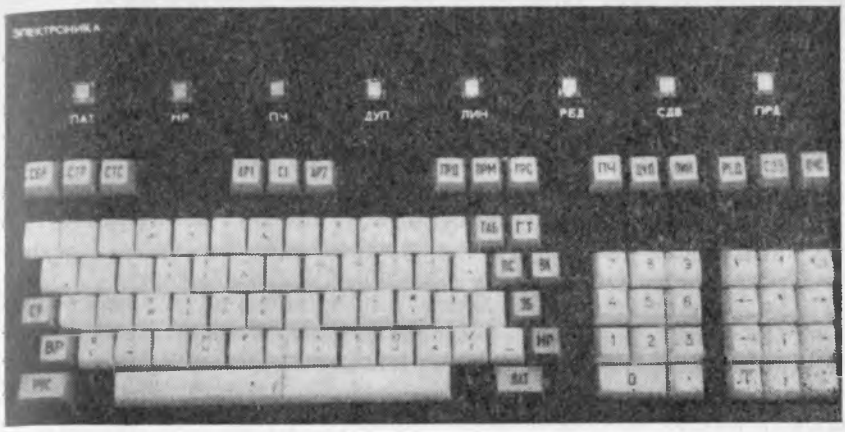


Рис. 4. Внешний вид блока клавиатуры

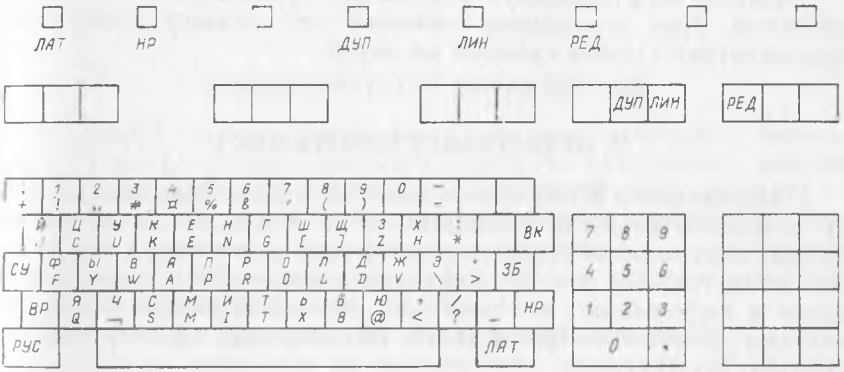


Рис. 5. Расположение индикаторов и клавиш на блоке клавиатуры

пробела, нуля и «ЗБ» (забой) не зависят от установки регистров работы клавиатуры (хотя клавиша «ЗБ» отметку о забое на латинском регистре производит символом « \ », а на русском — символом «э»).

Латинский регистр включается клавишей «ЛАТ», русский — клавишей «РУС», при этом индикатор «ЛАТ» соответственно светится или нет. Нижний регистр включается клавишей «НР» (нижний регистр), верхний — клавишей «ВР» (верхний регистр); при этом индикатор «НР», соответственно, светится или нет.

Алфавитные клавиши вырабатывают код буквы русского или латинского алфавита в соответствии с включенным алфа-

ределенном месте оперативной памяти (см. гл. 2, п. 4). Размещение программы в оперативной памяти с целью ее выполнения называется **загрузкой**. Загрузка интерпретатора производится после каждого включения ДВК-1, так как содержимое оперативной памяти после выключения ДВК-1 пропадает.

4. ПОДГОТОВКА ДВК-1 К РАБОТЕ

Подготовка ДВК-1 к работе заключается не только во включении аппаратуры, но и в загрузке интерпретатора Бейсик в оперативную память. Подготовка проводится в следующей последовательности (см. рис. 3, 5).

1. Тумблер «СЕТЬ» на задней панели блока сопряжения перевести в верхнее положение.

В результате включается индикатор «СЕТЬ» на передней панели блока сопряжения и вентилятор.

2. Нажать клавишу «СЕТЬ» на передней панели блока логики.

В результате на передней панели блока логики и индикатора клавиатуры включается индикатор «СЕТЬ». На экране появляется верхняя 25-я служебная строка. Начинается отсчет времени работы дисплея. В первой позиции 1-й строки появляется **курсор** — светящийся указатель места дальнейшего появления информации на экране.

3. Проверить, что клавиши «ДУП», «ЛИН», «РЕД» нажаты (клавиши должны быть утоплены, а их индикаторы должны светиться). Если это не так, то нажать их.

4. Нажать клавиши «ПИТ», «ПУСК», «ПР» на передней панели блока сопряжения.

В результате включаются их индикаторы и индикатор «РАБОТА» на передней панели блока сопряжения. На 1-й и 2-й строках экрана появляется сообщение:

160000
@ . . . .

Курсор устанавливается вслед за символом @ — коммерческое «at».

5. Загрузить интерпретатор Бейсик в оперативную память с адреса 140000_8 (в восьмеричной системе счисления). Для этого набрать на клавиатуре дисплея без пробелов команду 140000G.

При наборе команды она будет появляться на экране на 2-й строке вслед за символом @.

После набора команды на 3-й и 4-й строках экрана появляется сообщение

БЕЙСИК ДВК НЦ
@0

Если набор команды произведен с ошибкой, то на экране снова появится символ @ и следует правильно повторить набор.

6. Нажать клавишу «ВК».

В результате на 5-й строке экрана появится сообщение

ЖДУ

и курсор установится в первой позиции 6-й строки.

Это сообщение означает, что интерпретатор Бейсик загружен в оперативную память, готов к работе и ожидает информацию от пользователя.

Далее до самого окончания работы машину выключать нельзя, так как при выключении пропадает содержимое оперативной памяти. Ошибки при работе следует исправлять указанными ниже способами, а не уничтожать всю работу выключением машины.

Выключение ДВК-1 производится в последовательности пунктов 4, 2, 1 выполнением действий, противоположных указанным для включения.

Глава 3
ВВЕДЕНИЕ В БЕЙСИК

1. СПОСОБ ОПИСАНИЯ ЯЗЫКА ПРОГРАММИРОВАНИЯ

Каждый язык программирования так же, как и любой естественный язык, имеет свой алфавит, словарь и грамматику.

Алфавит состоит из основных символов, используемых в программах.

В словарь входят слова данного языка программирования, образуемые по определенным правилам.

Грамматика языка программирования устанавливает правила построения инструкций, определяющих действия машины. Словесное описание этих правил громоздко и не наглядно. Вместо него можно применять некоторое формализованное описание, называемое **форматом. Форматом инструкции** языка программирования называется правило построения данной инструкции, выраженное в краткой и наглядной форме с использованием условных обозначений.

В данной работе используются широко применяемые в форматах условные обозначения. Они заключаются в следующем.

1. Элементы формата, записанные прописными латинскими буквами, математическими знаками и знаками пунктуации, должны использоваться без изменений.

2. Элементы формата, записанные строчными буквами, должны заменяться на фактические элементы данного типа. Если смысл одного элемента формата передается несколькими словами, то они записываются через дефис.

3. Элементы формата, заключенные в квадратные скобки, не являются обязательными и могут опускаться.

4. Из элементов формата, записанных в несколько строк друг под другом и заключенных в общие фигурные скобки, необходимо выбрать один элемент (одну строку).

5. Многоточие после элемента формата означает возможность многократного повторения элемента данного типа.

6. Квадратные и фигурные скобки, а также многоточия используются только в формате в качестве указанных условных обозначений и не записываются в конкретной программе.

2. АЛФАВИТ

Алфавит Бейсика составляют следующие символы.

1. Прописные латинские буквы от A до Z.
2. Арабские цифры от 0 до 9.
3. Знаки, используемые в математических выражениях:

↑ — возведение в степень,
* — умножение,
/ — деление,
\+ — сложение,
— — вычитание,
. — десятичная точка для разделения целой и дробной части (использование запятой запрещено),
() — скобки,
< — меньше,
<= — меньше или равно (не больше),
= — равно,
> — больше,
>= — больше или равно (не меньше),
<> — не равно.

4. Знаки препинания:

, — запятая,
; — точка с запятой,
: — двоеточие,
— пробел.

Пробелы в Бейсике используются только для удобства чтения текста человеком. Интерпретатор игнорирует пробелы, за исключением специальной выдачи их на экран, поэтому допустимо записывать весь текст слитно или разрывать пробелами слова и числа.

Остальные приведенные на клавиатуре символы являются вспомогательными и используются в программе на Бейсике только для выдачи их на экран или в примечаниях.

3. КОНСТАНТЫ

Язык Бейсик в общем случае позволяет работать как с числовыми, так и с символьными константами и переменными. Однако интерпретатор ДВК-1 оперирует только с числовыми константами и переменными. Символы можно использовать только в примечаниях и для выдачи на экран.

Константы делятся на **целые** и **вещественные**. Первые записываются без десятичной точки, вторые — с десятичной точкой. Вещественные константы имеют две формы представления: **основную** и **экспоненциальную**.

Основная форма отличается от обычной записи дробных де-

сятичных чисел только тем, что целая и дробная части отделяются десятичной точкой, а не запятой.

Кроме того, нуль целых при выводе для экономии опускается и число представляется, начиная с десятичной точки (при вводе можно записывать и нуль целых).

Примеры записи констант в основной форме.

Обычная запись	Запись на Бейсике
1234567	1234567
—1,23	—1.23
0,123	.123
—0,123	—.123

При выводе целых и вещественных чисел в основной форме выделяются 9 позиций: одна для знака числа (вместо «+» используется пробел), одна для десятичной точки (в вещественных числах) и семь позиций для цифр числа. Следовательно, числа с большой целой частью или маленькие правильные дроби оказываются непредставимыми в этих формах, например, числа 12345678 и 0,000000012.

Для расширения диапазона и сокращения записи чисел используется экспоненциальная форма вещественного числа. В этой форме любое число можно представить умноженным на степень десяти. Например, $1{,}2=1{,}2\times10^0=12\times10^{-1}=120\times\times10^{-2}=0{,}12\times10^1=0{,}012\times10^2$.

Следовательно, существует множество представлений одного и того же числа в экспоненциальной форме. Число с нулевой целой частью и ненулевой первой цифрой дроби называется **нормализованным.**

Дробная часть числа называется мантиссой числа, показатель степени десяти — порядком числа. В рассмотренном примере нормализованным является число 0,12. При этом порядок равен 1.

Ввод чисел в экспоненциальной форме можно производить как в нормализованном, так и в ненормализованном виде, вывод производится только в нормализованном виде без печати нуля целых.

В программировании нельзя записать надстрочный показатель степени, подстрочный индекс или многоэтажную дробь. Все записывается на уровне одной строки и с максимальной экономией используемых позиций экрана. Поэтому при записи числа в экспоненциальной форме на Бейсике знак умножения на 10 опускается, двузначное число 10 заменяется на букву E (англ. exponent — показатель степени), а далее в той же строке записывается порядок числа со знаком (знак «+» при выводе опускается, при вводе может присутствовать). Порядок должен находиться в интервале от —9809 до +9809.

При выводе числа в экспоненциальной форме выделяется 15 позиций: одна для знака числа (вместо «+» используется

пробел), одна для десятичной точки, семь позиций для цифр мантиссы, одна для буквы Е, одна для знака порядка (вместо «+» используется пробел) и четыре позиции для цифр порядка.

Примеры записи константы в экспоненциальной форме.

Обычная запись	Вывод на Бейсике
12345678	.1234568E 8
0,000000012	.12E—7

При отбрасывании лишних разрядов производится округление числа. Максимальное количество цифр в числе в основной форме или в мантиссе числа в экспоненциальной форме равно семи. При этом седьмая цифра может быть увеличена на единицу, если производилось округление.

4. ПЕРЕМЕННЫЕ

Константы в программе указываются своими значениями. Переменные представляют значения обобщенно, именами. Для каждой переменной выделяется место в оперативной памяти машины, где должны появляться все последовательные значения этой переменной.

Место отдельной структурной единицы памяти в общем объеме памяти определяется адресом этой единицы, т. е. ее порядковым номером. Пользоваться номером неудобно по разным причинам, в частности потому, что он не отражает смысловое содержание переменной величины. Удобнее использовать описательное задание адреса, как это принято в языках программирования.

В Бейсике описательное задание адреса определяется **именем переменной**. Имя переменной — это название места этой переменной в оперативной памяти, используемое в программе вместо указания точного адреса, определяемого номером. Роль имен переменных в программах подобна роли обозначений величин в математических формулах или названий граф в таблицах.

При трансляции программы интерпретатор составляет таблицу, в которой каждому имени переменной ставится в соответствие конкретный числовой адрес. Этот адрес заменяет имя переменной в программе на машинном языке. Таким образом, происходит переход от описательного задания адреса к его конкретному указанию.

К сожалению, в интерпретаторе ДВК-1 возможности образования имени переменной весьма ограничены.

Правила образования имени переменной. Имя переменной образуется из одного или двух символов. Пер-

вым символом может быть только прописная латинская буква, вторым — только цифра.

В соответствии с этими правилами для числовых переменных табл. 2 можно предложить следующие варианты назначения имен.

Первый вариант:

K — для количества (по первой букве названия графы),
C — для цены (используется латинский эквивалент русской буквы Ц),
S — для стоимости (латинский эквивалент русской буквы С).

Использовать здесь латинскую букву C, совпадающую с русской буквой С, уже нельзя, так как буква С выбрана ранее для имени цены.

Второй вариант:

D1 — для количества (данное первое),
D2 — для цены (данное второе),
P — для стоимости (первая буква слова «результат»).

Третий вариант:

A — для количества,
B — для цены,
C — для стоимости.

Здесь просто используются латинские буквы в алфавитном порядке.

При выборе имен переменных можно использовать любую удобную для программирования систему их назначения. Обязательным является лишь требование соблюдения правил их образования. Кроме того, необходимо следить, чтобы разным переменным присваивались разные имена.

Назначение имени переменной означает лишь выделение для переменной места в оперативной памяти. Однако никакого конкретного значения переменной в этом месте памяти не будет до тех пор, пока оно не будет туда занесено. Использование в качестве известной величины имени переменной, значение которой еще не определено, является типичной ошибкой начинающих программистов.

Константы необязательно представлять в программе непосредственным написанием их значений. Им так же, как и переменным, можно назначить имя. Это удобно в тех случаях, когда константа представляется большим числом знаков и часто используется в программе. Например, константе $\pi=3{,}141593$ можно назначить имя P, или P1, или любое другое неиспользованное имя. Естественно, что значение константы должно быть присвоено ее имени до использования имени константы в вычислениях.

5. СИМВОЛЫ

В общем случае символом называется любой типографский знак. В программировании под символом понимают знак, не используемый в вычислениях. Так, например, использование цифр в качестве второго символа имени переменной не предусматривает учета значения этой цифры.

В современных ЭВМ и в языках программирования обработка символьной информации предусмотрена наряду с обработкой числовой информации. Предусматривается использование символьных констант и символьных переменных. Примером символьных констант могут служить названия граф в табл. 1, а примером символьных переменных — конкретные наименования товаров. С символьными константами и переменными предусматриваются логические операции сравнения. Равенство двух символьных переменных означает последовательное совпадение всех их символов (например, можно было бы выбирать из таблицы товары с одинаковыми наименованиями). Соотношение меньше или больше для двух символов связано с соотношением представляющих их кодов в двоичной системе счисления. Оно аналогично старшинству букв в алфавитах естественных языков, которое используется человеком при работе со словарями и энциклопедиями. Так, в латинском и русском алфавитах буква А «меньше» буквы В в том смысле, что она расположена раньше.

К сожалению, в интерпретаторе ДВК-1 (но не в языке программирования Бейсик) не предусмотрена работа с символьными константами и переменными. Интерпретатор ДВК-1 позволяет использовать символы лишь в примечаниях и для выдачи их на экран.

6. АРИФМЕТИЧЕСКИЕ ВЫРАЖЕНИЯ

Арифметическим выражением называется составной элемент конструкции фразы языка программирования, построенный из констант, имен переменных и обозначений математических функций Бейсика с использованием знаков математических действий ↑, *, /, +, — и круглых скобок.

В частном случае арифметическое выражение может состоять только из одной константы или из одного имени переменной.

Примеры арифметических выражений

2	(2—A)/B
2+5	A+B+C
A	EXP(A)
2—A	(2—A)/EXP((2—A)/B)

Обратите внимание, что в арифметическом выражении не используется знак равенства!

В двух последних примерах используется стандартная функция EXP, вычисляющая значение e=2,718282, возведенное в степень, определяемую значением заключенного в скобках за EXP аргумента.

Стандартной функцией называется функция, программа вычисления которой входит в состав программного обеспечения ЭВМ. Подробно эти функции рассматриваются в приложении 1. В общем случае аргументом стандартной функции может быть любое арифметическое выражение, содержащее в свою очередь, стандартные функции.

При вычислении арифметических выражений в Бейсике используются правила приоритета (старшинства) операций.

В порядке убывания приоритета операции располагаются в следующей последовательности:

действия в скобках,
вычисления математических функций Бейсика,
возведение в степень,
умножение и деление,
сложение и вычитание.

При равных приоритетах операции выполняются в порядке их следования, а внутри скобок — в соответствии со всеми предыдущими правилами.

7. УСЛОВИЯ

Условием называется соединение двух арифметических выражений знаками логических операций <, <=, =, >, >=, <>. Если условие выполняется, то оно считается **истинным**, в противном случае — **ложным.**

Проверка выполнения условий используется для организации разветвлений в программе. При истинном условии выполняется одна ветвь программы, при ложном — другая. В каждой из этих ветвей может производиться проверка других условий, что приведет к дальнейшим разветвлениям в программе.

Примеры записи условий

2<A	B>C
A<=B	B+4>=C—A
(2+A)/EXP(B)=C	B<>C

8. СТРОКИ

Инструкции, используемые при работе на ДВК-1, по терминологии руководства к этой машине делятся на две группы: операторы языка Бейсик и команды оператора.

Здесь встречается характерная для программирования многозначность терминологии. В первом случае слово «оператор»

означает «инструкция языка программирования», во втором случае — «человек, задающий информацию с клавиатуры». В дальнейшем описании термины «оператор» и «команда оператора» используются только в указанных здесь смыслах.

Команда оператора — это команда оперативного управления работой машины. Обычно команды оператора задаются с клавиатуры и выполняются в непосредственном режиме. Команды DELETE (удалить), LIST (выдать программу на экран) и RUN (выполнить программу) могут использоваться и в программе на Бейсике, хотя практически это применяется крайне редко.

Форматы команд оператора и соответствующие им действия ЭВМ приведены в табл. 3.

Операторами языка Бейсик называются инструкции машине, используемые в программах на этом языке. Один оператор должен занимать не более одной строки, но несколько операторов могут располагаться на одной строке, отделяясь друг от

Таблица 3

Команды оператора

Формат команды	Действия ЭВМ
DELETE n[,m]	При нажатии клавиши «ВК» или при выполнении этой команды в программе на экране выдается текст мером n или строки с n до m(n<m). В частном случае DELETE 1,8191 удаляются все строки во всей отведенной пользователю области оперативной памяти.
LIST [n[,m]]	При нажатии клавиши «ВК» или при выполнении этой команды в программе на экран выдается текст программы: либо весь (при отсутствии n и m), либо строка n, либо строки с n до m (n<m).
RUN	При нажатии клавиши «ВК» или при выполнении этой команды в программе вызывается выполнение программы с оператора с наименьшим номером. При повторном выполнении ЭВМ устанавливается в исходное состояние.
СУ/Р	При одновременном нажатии клавиш «СУ» и «Р» (знак «/» условно обозначает одновременность нажатия) прерывается выполнение программы (прерывание отмечается на экране символами «¬|Р»), и ДВК-1 переводится в режим ожидания информации от пользователя.
СУ/U	При одновременном нажатии клавиш «СУ» и «U» отменяется передача интерпретатору данной строки (отмена строки отмечается на экране символами «¬|U»), и ДВК-1 переводится в режим ожидания информации от пользователя.
ЗБ	При каждом нажатии клавиши «ЗБ» забивается (т. е. удаляется) один последний символ в строке. Отметка на экране о забое при работе на латинском регистре производится символом «/», при работе на русском регистре — символом «э».

друга двоеточием. Во всех случаях под запись операторов отводится не более 72 позиций строки из 80. При заполнении 72-й позиции раздается звуковой сигнал.

На расположение некоторых операторов в одной строке с другими операторами накладываются определенные ограничения. Эти ограничения указываются в описании каждого оператора, если они существуют. Для облегчения чтения и исправления программы рекомендуется располагать по одному оператору на строке.

В дальнейшем операторы Бейсика и команды оператора объединяются общим названием **строка.**

9. РЕЖИМЫ РАБОТЫ ИНТЕРПРЕТАТОРА

Интерпретатор ДВК-1 работает с пользователем в форме диалога. На первой свободной от предыдущей информации строке он высвечивает слово «ЖДУ» и в первой позиции следующей строки устанавливает курсор. Это означает, что интерпретатор ожидает информацию от пользователя.

Основной информационной единицей для ответа пользователя является одна строка экрана. **Информация строки принимается интерпретатором только после нажатия клавиши «ВК»,** что является сигналом об окончании набора строки. Начинающие пользователи постоянно забывают нажимать клавишу «ВК». В этом случае машина терпеливо ждет их, и мерцание курсора за последним набранным в строке символом сигнализирует о том, что набор строки не окончен. В дальнейшем необходимость нажатия клавиши «ВК» в конце набора строки специально не оговаривается, но всегда подразумевается.

Приняв информацию строки, интерпретатор либо сразу выполняет указанные в ней действия, либо запоминает информацию этой строки в оперативной памяти. Первый режим работы интерпретатора в руководстве к ДВК-1 называется **непосредственным**, а второй — **косвенным**. В данном пособии второй режим далее называется **программным**.

Интерпретатор отличает программный режим от непосредственного соответственно по наличию или отсутствию номера строки. **Номером строки** называется число в диапазоне от 1 до 8191 в начале строки.

Номер строки выполняет следующие функции:

1. Указывает на программный режим работы интерпретатора.

2. Определяет дальнейший порядок выполнения строк по возрастанию их номеров, а не в порядке ввода. Поэтому первоначальную нумерацию строк рекомендуется задавать с шагом 10, чтобы в дальнейшем иметь возможность вставлять между строками дополнительные строки.

3. Выделяет строку среди других строк при обращениях и ссылках.

4. При расположении в одной строке нескольких операторов номер строки может предшествовать только первому из них. При передаче управления на такую строку оно передается первому оператору строки.

В непосредственном режиме строка выполняется сразу и только один раз. Для повторного ее выполнения надо повторить набор строки. Работа в этом режиме похожа на работу микрокалькулятора. В этом режиме целесообразно задавать команды оператора (которые надо выполнять сразу и, как правило, один раз), передавать с клавиатуры управление определенным строкам программы, выводить на экран нужные величины при поиске ошибок в программе и производить другие оперативные действия.

В программном режиме строки записываются в оперативную память и хранятся в ней до тех пор, пока не будут заменены или удалены или пока не будет выключена ЭВМ. Для выполнения программы команда оператора RUN набирается в непосредственном режиме (без номера). Повторный набор RUN приводит к повторному выполнению программы. Программу можно выполнять многократно. Программу можно просмотреть на экране полностью или по частям, задав в непосредственном режиме соответствующую команду LIST. В программе можно изменять текст в строках, удалять ненужные строки и добавлять новые.

Программный режим обеспечивает большую гибкость в работе и позволяет производить достаточно сложные вычисления.

10. ИСПРАВЛЕНИЕ ТЕКСТА

Если ошибки в наборе текста обнаружены до нажатия клавиши «ВК», то возможны два способа их исправления.

1. Если первый ошибочный символ находится близко к концу строки, то надо сосчитать количество забиваемых символов в конце строки вместе с ошибочным (включая и пробелы) и соответствующее число раз нажать клавишу «ЗБ». После этого набрать правильные символы вместо забитых.

2. Если первый ошибочный символ находится далеко от конца строки, то точно сосчитать количество забиваемых символов сложно. В этом случае целесообразно отменить всю строку одновременным нажатием клавиш «СУ» и «U». После этого набрать всю строку с начала.

Если строка с ошибками уже введена нажатием клавиши «ВК», то интерпретатор обнаруживает ошибку либо непосредственно при вводе строки, либо при ее последующем выполнении. Обнаружив ошибку, интерпретатор выдает сообщение о ней в следующем формате:

ОШИБКА код-ошибки СТРОКЕ номер-строки

Код ошибки — это число, определяющее тип ошибки. Соответствие кодов и типов ошибок устанавливается разработчиками системы. Таблица кодов ошибок с указанием характера ошибок приведена в приложении 3.

Номер строки в сообщении указывает ошибочную строку. Если приводится нулевой номер, то это означает, что ошибка обнаружена в последней введенной строке.

Исправление текста в строках программы основано на правиле: при вводе строки с имеющимся в оперативной памяти номером старый текст строки с этим номером заменяется на новый. Следовательно, для исправления текста строки надо набрать ее номер и весь текст в правильном виде.

Для удаления из оперативной памяти отдельной строки используются два способа.

Первый и самый простой — это следствие из правила исправления текста в строке программы. Если вводится только номер строки без текста, то из оперативной памяти удаляются эта строка и ее номер.

Второй способ состоит в наборе в непосредственном режиме команды оператора DELETE с номером удаляемой строки. Для удаления одной строки этот способ длиннее первого, но он эффективен, когда необходимо убрать несколько последовательных строк. В этом случае набирается в непосредственном режиме команда DELETE, в которой указывается номер первой удаляемой строки (меньший номер), а затем через запятую — номер последней удаляемой строки (больший номер). Команда DELETE 1,8191 с запасом удаляет всю программу, так как перекрывает весь возможный диапазон номеров строк.

Для вставки новой строки необходимо, чтобы номер вставляемой строки был больше номера предыдущей и меньше номера последующей строки. Набор вставляемой строки можно произвести в любое удобное время, а ее выполнение будет производиться в соответствии с ее номером. Именно поэтому рекомендуется первоначально нумеровать строки с шагом 10.

Наращивание программы осуществляется добавлением новых строк в конце программы.

Исправление текста приводит к затруднению его чтения на экране. При забое в строках остаются забиваемые символы и символы забоя. При отмене строки командой «СУ/U» отмененная строка с отметкой « ¬ U» остается на экране. При вставке строки ее первоначальное расположение среди других строк определяется моментом вставки.

Все это не мешает работе машины, но затрудняет чтение текста. Поэтому при желании в любой момент можно получить новое исправленное изображение текста. Для этого в непосредственном режиме задается команда оператора LIST.

Глава 4
СРЕДСТВА И ПРИЕМЫ НАЧАЛЬНОГО ПРОГРАММИРОВАНИЯ

1. ОПЕРАТОР REM

REM — сокращ. от англ. REMARK — примечание, замечание, комментарий

Это самый простой оператор Бейсика.

Формат оператора:

REM символ ...

При расположении в одной строке с другими операторами оператор REM должен быть последним, так как весь текст от REM до конца строки воспринимается интерпретатором как примечание.

В соответствии с форматом после REM могут использоваться любые символы клавиатуры при любых сочетаниях регистров: «ЛАТ», «РУС», «ВР», «НР». Следовательно, REM можно дополнять до полной формы REMARK.

Напоминания.

1. Пробелы в операторах Бейсика используются только для удобства чтения. Вообще же можно записывать весь текст слитно или разрывать пробелами слова и числа.

2. Клавиша «ВК» должна нажиматься только при латинском регистре (верхнем или нижнем).

Оператор REM не вызывает никаких действий. Строки с оператором REM используются в программе в качестве заголовков и пояснительного текста к различным ее частям.

Пример оператора REM

```
10 REM Вычисления для одного товара.
```

Оператор REM не производит никаких вычислительных действий, а также не производит ни ввода исходных данных, ни вывода результатов решения задачи. С утилитарной точки зрения он является лишним в программе. Он не влияет на выполнение программы и всегда может быть из нее исключен. Однако его присутствие облегчает чтение программы, комментируя смысловое содержание как всей программы, так и отдельных ее частей.

LET в переводе с англ.— пусть, положим.

Формат оператора:

LET имя-переменной=арифметическое-выражение

Оператор LET называется оператором присвоения. Его действие заключается в том, что он вычисляет значение арифметического выражения, расположенного за знаком равенства, и посылает это значение в место оперативной памяти, соответствующее имени переменной, указанному перед знаком равенства.

В зависимости от вида арифметического выражения возможны следующие частные случаи.

1. Арифметическое выражение представляется константой. В этом случае производится засылка константы в память. Например:

```
20 LET A=25
```

Число 25 отсылается в место оперативной памяти, соответствующее имени переменной A, или кратко: переменной A присваивается значение 25.

2. Арифметическое выражение представляется именем переменной. В этом случае происходит пересылка значения из одного места оперативной памяти (определяемого именем переменной в арифметическом выражении) в другое место (определяемое именем переменной перед знаком равно) или пересылка из **передающего** адреса оперативной памяти в **принимающий.** В передающем адресе значение величины всегда сохраняется, в принимающем адресе новая величина стирает старую. Фактически пересылка приводит к появлению второго экземпляра пересылаемой величины в другом месте оперативной памяти.

Например:

```
30 LET B=A
```

Выполнение этого оператора после строки 20 приведет к появлению по адресу B числа 25 в результате его пересылки из адреса A. По адресу A число 25 сохранится, а прежнее значение, хранившееся по адресу B, пропадет.

3. Арифметическое выражение содержит математические действия. В этом случае действие оператора производится в соответствии с общей формулировкой, приведенной за описанием его формата.

Например:

```
40 LET C=A+B
```

Сначала производится вычисление арифметического выражения A+B, которое равно 25+25, т. е. 50. Затем полученное значение 50 посылается по адресу C, стирая прежнее значение в C.

Внимание начинающих программистов необходимо обратить

на частный случай, когда в арифметическом выражении используется то же имя переменной, что и перед знаком равно.

Например:

```
50 LET A=A+1
```

В этом случае сначала вычисляется арифметическое выражение A+1, которое равно 25+1, т. е. 26. Затем число 26 отсылается по адресу A, стирая там прежнее число 25.

Работу ЭВМ при выполнении четырех операторов LET с номерами 20—50 можно пояснить следующим образом.

В этих операторах обрабатываются три переменные с именами A, B и C. Эти имена выбраны пользователем в соответствии с правилами образования имен. Кроме того, желательно, чтобы имя переменной соответствовало ее смысловому значению (см. гл. 3, п. 4).

Каждой из переменных интерпретатор выделяет место в оперативной памяти машины. При работе ЭВМ оно определяется числовым адресом в двоичной системе счисления. Пользователю же предоставляется возможность применять в программе вместо числового двоичного адреса его описательное обозначение с помощью имени переменной. Интерпретатор составляет таблицу имен, в которой каждому имени ставит в соответствие конкретный двоичный адрес. Этот двоичный адрес подставляется вместо имени переменной в машинную команду, получаемую после трансляции составленного пользователем оператора на языке Бейсик.

Таким образом, ЭВМ и пользователь применяют различные обозначения для адресов оперативной памяти: ЭВМ — двоичные числа, пользователь — более удобные для него имена переменных. Соответствие этих двух видов обозначений друг другу устанавливает программа-интерпретатор.

Важно уяснить следующее. Каждый из этих видов обозначений определяет лишь место для переменной в оперативной памяти и не определяет ее значение. Значение переменной будет определяться лишь теми операциями, которые заносят информацию в оперативную память. В частности, при выполнении оператора LET информация заносится в оперативную память. Ее адрес соответствует имени переменной, стоящему перед знаком равенства.

Размещение переменных в оперативной памяти машины и их адресация с помощью имен переменных и двоичных чисел схематически представлены на рис. 6.

На этом рисунке прямоугольниками обозначены места для переменных в оперативной памяти. Сверху над прямоугольниками приведена адресация этих мест пользователем с помощью имен переменных, снизу — адресация этих мест машиной с помощью двоичных чисел. Конкретные значения двоичных чисел не приводятся, так как их выбор определяется программой-ин-

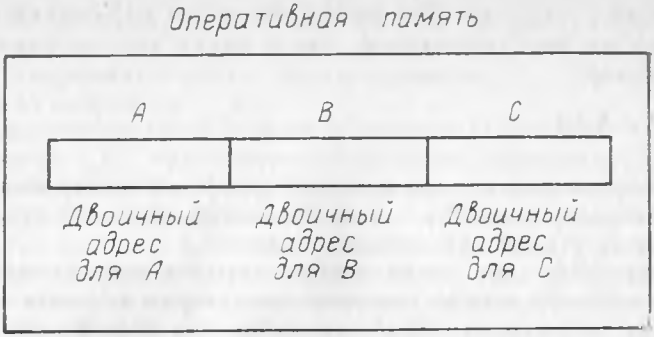


Рис. 6. Схематическое представление переменных в оперативной памяти

терпретатором и не влияет на составление пользователем программы на языке Бейсик.

Работа ЭВМ при выполнении операторов LET с номерами 20—50 сводится к изменению и использованию значений переменных A, B и C. Проследить за этим процессом удобно с помощью табл. 4.

Таблица 4

Изменение значений переменных A, B, C

Номер строки	Значение переменной		
	A	B	C
	?	?	?
20	25		
30		25	
40			50
50	26		

Предполагается, что до строки 20 в программе нет никаких строк, определяющих значения переменных A, B и C. Следовательно, первоначальные значения этих переменных являются неопределенными, и они условно представлены в табл. 4 знаком «?». Значение каждой переменной остается постоянным до момента очередного ее изменения. Этот момент отмечается появлением нового значения переменной в табл. 4.

Из таблицы видно, что значение переменной A последовательно изменяется от неопределенности на числа 25 и 26. Значения переменных B и C изменяются от неопределенности на числа 25 и 50, соответственно.

Попытка дополнить рассматриваемые операторы еще одной строкой в виде

60 LET D=A+E

приведет (при выполнении по команде RUN) к сообщению об ошибке.

ОШИБКА 123 СТРОКЕ 60

В соответствии с классификатором ошибок код 123 означает использование несуществующей переменной. Ошибка состоит в том, что переменной E еще не присвоено значение (ее значение еще не определено), поэтому ЭВМ не может вычислить, сколько будет A плюс неопределенное значение E.

Начинающим пользователям рекомендуется проводить разбор всех трудных для понимания мест в программах с помощью таблиц, аналогичных табл. 4. Такой способ разбора является довольно громоздким и трудоемким, но он дает понятное и наглядное представление о процессе работы ЭВМ. При определенном навыке такой разбор производится в уме.

В заключение можно сделать следующие выводы о назначении оператора LET.

1. Оператор LET используется для засылки константы в оперативную память (например, оператор 20).

2. Оператор LET используется для пересылки значения из одного места оперативной памяти в другое (например, оператор 30).

3. Оператор LET используется для арифметических вычислений (например, операторы 40 и 50).

Важно отметить, что результаты выполнения операторов непосредственно на экран не выводятся. Они спрятаны от пользователя в оперативной памяти. Это объясняется тем, что среди результатов вычислений может быть много промежуточных, не нужных пользователю, поэтому только сам пользователь определяет, что надо выводить на экран. Для этого служит оператор PRINT, рассматриваемый ниже.

3. ОПЕРАТОР PRINT

PRINT — в переводе с англ.— печатать, печать

Формат оператора:

PRINT [«символ...» [{; ,}]]...

[имя-переменной [{; ,}]]...

[арифметическое-выражение [{; ,}]]...

Оператор PRINT выводит на экран информацию в заданном пользователем расположении.

Формат оператора выглядит достаточно сложным, однако он в кратчайшем виде представляет все возможные конкретные конструкции оператора PRINT. Расшифровка условных обозна-

чений формата производится с помощью правил, приведенных в п. 1, гл. 3.

Разбор возможных конкретных конструкций оператора проводится на примерах некоторых частных случаев, вытекающих из его формата.

1. Все элементы, заключенные в квадратные скобки, опущены. Получается вариант

PRINT

Оператор представлен без параметров, т. е. без его элементов. Такой оператор выводит «пустую строку» (пропускает строку на экране), что используется для отделения друг от друга строк с различным характером информации.

Другое назначение оператора без параметров — дополнять незаконченную строку выводимой информации пробелами, чтобы обеспечить дальнейший вывод с новой строки. Эта способность незаменима никакими другими средствами при выводе таблиц или графиков, что будет продемонстрировано ниже.

2. В общем случае оператор может содержать элементы трех типов: символы, имена переменных и арифметические выражения. Эти элементы могут многократно повторяться и использоваться в операторе в различных сочетаниях. **Разделителями** между элементами являются запятая или точка с запятой (после символов разделитель может вообще отсутствовать):

а) запятая в качестве разделителя.

Использование запятой в качестве разделителя является указанием для интерпретатора о делении строки на зоны, т. е. на части строки по 14 позиций в каждой части или зоне. На строке выделяется всего 5 зон по 14 позиций в каждой. В каждой зоне представляется один элемент оператора PRINT.

Запятая за последним элементом в операторе PRINT означает, что последующий вывод информации по данному или другому оператору PRINT производится на той же строке, если не заполнены все зоны. Незаполненную строку можно «закрыть», т. е. обеспечить последующий вывод с новой строки, выполнив оператор PRINT без параметров. Отсутствие разделителя за последним элементом означает, что последующий вывод по любому из операторов PRINT начинается с новой строки.

Включение дополнительной запятой между двумя элементами оператора PRINT приводит к пропуску одной зоны;

б) точка с запятой в качестве разделителя.

В отличие от запятой точка с запятой приводит к более компактному выводу информации и размещению большего числа элементов информации на одной строке.

Точка с запятой после числового элемента обеспечивает один пробел после него. Если далее следует положительный числовой элемент, то вместо знака плюс появляется второй пробел.

Многократное повторение точки с запятой не приводит к увеличению числа пробелов.

После символьного элемента точка с запятой не обеспечивает пробела и равносильна отсутствию разделителя. Она используется только для удобства чтения оператора PRINT.

Введение нужного числа пробелов между элементами при использовании точки с запятой достигается использованием соответствующего числа пробелов в качестве символьных элементов.

Точка с запятой за последним элементом любого типа также, как и запятая, обеспечивает продолжение вывода на той же строке.

3. Символы в операторе PRINT.

Использование символов в операторе PRINT аналогично их использованию в операторе REM. Однако в соответствии с форматом PRINT их следует заключать в кавычки. Кавычки являются указанием на то, что содержащаяся внутри них информация должна выводиться на экран. Пробелы внутри кавычек выводятся на экран, как и всякий другой символ. Сами кавычки на экран не выводятся.

Напоминание.

Кавычки и клавишу «ВК» нажимать только при латинском регистре.

4. Имена переменных в операторе PRINT.

Использование имен переменных приводит к выводу на экран соответствующих им значений.

5. Арифметические выражения в операторе PRINT.

Использование арифметических выражений приводит к их вычислению и выводу полученных значений на экран.

Приведенные правила поясняются на примере простейшей учебной программы, которую рекомендуется выполнить на машине.

4. ПЕРВЫЕ УЧЕБНЫЕ ПРОГРАММЫ

Простейшие программы можно создавать с использованием всего двух операторов: LET и PRINT. Для пояснений к ним можно добавлять оператор REM.

Основное назначение первой программы — пояснение работы оператора PRINT на конкретных примерах и овладение приемами работы на машине в процессе ввода программы, ее корректировки и запуска на выполнение. Текст программы 1 приводится вместе с результатами ее выполнения машиной.

Программа 1 и ее выполнение

```
10  REM РАБОТА ОПЕРАТОРА PRINT
20  LET A=10
30  LET B=20
```

```
40  LET C=A+B
50  PRINT
60  PRINT A, B, C; A; B; —C
70  PRINT «A», «B»; «C» «D»; « EF»; « » «G»; « H  I   J»
80  PRINT —A, , —B; ; —C; «  » —C
90  PRINT A, B, C,
100 PRINT A+C, B+C, C+C
110 PRINT A, B, C,
120 PRINT
130 PRINT A+C, B+C, C+C
140 PRINT «A==», A, «B==»; B,«C=» C
150 PRINT «Конец !-!-!»
RUN
                                                    (50)
  10        20              30  10   20  —30        (60)
A       BCD EF   G H  I    J                        (70)
—10                      —20 —30   —30              (80)
  10        20              30        40      50    (90, 100)
  60
  10        20              30                      (110, 120)
  40        50              60                      (130)
A=          10          B=20        C=30            (140)
Конец !-!-!                                         (150)
ОСТ СТРОКЕ 150
ЖДУ
```

Текст программы составляют операторы с номерами 10 — 150. Для ввода текста в ЭВМ она предварительно подготавливается к работе в соответствии с п. 4 гл. 2. Ошибки, возникающие при наборе текста, исправляются в соответствии с п. 10 гл. 3.

После ввода правильного текста задается команда оператора RUN, вызывающая выполнение программы. Команда RUN задается в непосредственном режиме (без номера). Результаты выполнения выводятся в следующих за RUN строках. Числа в скобках, приведенные в конце этих строк, не выводятся машиной. Они приведены для указания номеров операторов PRINT, выводящих эти строки.

Оператор 50 выводит пустую строку, отделяющую вывод результатов от команды RUN. Таким приемом разделяются различные по характеру информации строки.

Оператор 60 показывает действие запятой и точки с запятой в качестве разделителей имен переменных и арифметических выражений. Он начинает вывод с новой строки, так как предыдущий оператор PRINT не имел разделителя в конце строки (он вообще не имел параметров). Первые три переменные выводятся в последовательных зонах, так как они разделены запятыми. Числовые значения этих переменных выводятся со второй позиции каждой зоны, а первые позиции зон заняты пробелами,

заменяющими знак плюс. Далее в качестве разделителя используется точка с запятой. Она обеспечивает разделение одним пробелом, как показано перед значением арифметического выражения —С. Второй пробел перед значениями переменных А и В возникает из замены пробелом знака плюс.

Оператор 70 показывает вывод символов на экран и действие разделителей после символов. Любые символы, заключенные в кавычки в операторе PRINT, выводятся на экран в том же виде, в каком они представлены внутри кавычек. Запятая после символов действует так же, как после имен переменных и арифметических выражений, т. е. обеспечивает переход в следующую зону. Точка с запятой после символов не обеспечивает пробела и равносильна отсутствию разделителя. Она может использоваться лишь для удобства чтения оператора PRINT. Пробелы после символов можно обеспечить использованием символа пробела в качестве самостоятельного элемента в операторе PRINT или в качестве составной части символьных элементов.

Оператор 80 показывает, что удвоение запятой приводит к пропуску зоны, а удвоение точки с запятой не приводит к появлению второго пробела. Введение нужного числа пробелов можно обеспечить использованием соответствующего числа символов пробела в качестве символьного элемента в операторе PRINT.

Оператор 90 заканчивается разделителем в виде запятой. Поэтому оператор 100 обязан начать вывод на той же строке, что и оператор 90.

Операторы 110 и 130 совпадают с операторами 90 и 100 соответственно. Но оператор 120 дополняет пробелами строку вывода оператора 110, «закрывая» ее. Поэтому оператор 130 начинает вывод с новой строки.

Оператор 140 показывает возможность вывода пояснительной текстовой информации перед значениями переменных или арифметических выражений. В операторе 140 символ «А=» отделен от имени переменной запятой, а символы «В=» и «С=» — точкой с запятой и равносильным ей отсутствием разделителя. В первом случае происходит отрыв обозначения переменной от ее значения, что плохо смотрится на экране. Рекомендуется использовать второй или третий варианты.

Оператор 150 показывает возможность вывода на экран любых имеющихся на клавиатуре символов. Можно использовать прописные и строчные буквы латинского и русского алфавитов, цифры и другие символы.

За выводом результатов сообщается об останове после выполнения оператора 150. В ДВК-1 можно использовать специальные операторы останова и окончания программы. При их отсутствии останов происходит после выполнения операторов в строке с наибольшим номером. В последней строке сообщается, что интерпретатор ожидает новую информацию от пользователя.

Выполнение программы можно повторить, снова задав команду RUN. Но для программы 1 в этом нет смысла, так как исходные данные остаются неизменными, и результат повторного выполнения будет тем же. Можно вызвать текст программы на экран командой LIST с целью просмотра или изменения программы.

В общем случае возможна ситуация, когда ни текст программы, ни результаты ее выполнения не умещаются на экране. Если весь текст программы не умещается на экране, то он вызывается по частям командой в формате LIST n, m, (где n и m — номер первой и последней вызываемой строки, соответственно). Если результаты не умещаются на экране, то их вывод прерывается использованием в программе оператора STOP (см. п. 11).

Введенную в ЭВМ программу можно изменить, нарастить или полностью заменить на новую. В случае замены новую программу можно набирать так, как будто старой не было. При этом новый текст замещает старый в строках с совпадающими номерами. Но если в новой программе не все номера совпадают со старой, то строки с несовпадающими номерами из старой программы войдут в новую. Такими обычно являются номера, не кратные 10. Кроме того, если новая программа окажется короче старой, то «хвост» старой программы станет «хвостом» новой. Поэтому целесообразно перед набором новой программы удалить из памяти старую программу командой DELETE. Для удаления программы 1 эта команда имеет следующий вид:

```
DELETE 10, 150
```

По аналогии с программой 1, используя только операторы LET и PRINT, можно составить программу вычисления стоимости только одного товара по данным табл. 1. Программа 2 приводится с результатами ее выполнения. Сообщения машины об останове и ожидании пользователя здесь и в последующих программах, как правило, не приводятся.

Программа 2 и ее выполнение

```
10 REM Вычисление стоимости только одного товара
20 LET K=20
30 LET C=135
40 LET S=K*C
50 PRINT «Кол. =»K, «Цена=»C, «Стоим.=»S
RUN
Кол.= 20        Цена= 135        Стоим.= 2700
```

Программа 2 является принципиально плохой, так как она составлена для конкретных значений данных (а именно: количество=20, цена=135), и в ней нет операторов, соответствующих блоку 5 блок-схемы, представленной на рис. 2. Для вычисления стоимости другого товара придется изменять в программе 2 операторы 20 и 30. Это значит, что при изменении исход-

ных данных требуется изменять операторы программы, т. е. программу. А программа, так же как и математическая формула, должна оставаться неизменной при изменении исходных данных.

Недостаток программы 2 состоит в задании исходных данных в виде констант. Оператором LET удобно задавать константы, пересылать значения и вычислять арифметические выражения, однако не рекомендуется использовать его для ввода изменяющихся значений исходных данных. Для этой цели существуют два способа, рассматриваемые ниже.

5. ОПЕРАТОР INPUT

INPUT **в переводе с англ. — ввести, ввод, входное значение**

Формат оператора:

INPUT имя-переменной-1 [, имя-переменной-2] ...

При выполнении оператора INPUT на экран выдается знак вопроса (один знак независимо от количества представленных в операторе имен переменных). В ответ пользователь **обязан** вводить только числовые значения переменных, указанных в операторе INPUT. Значения различных переменных разделяются запятыми. После ввода всех значений нажимается клавиша «ВК», и выполнение программы продолжается.

Если введено меньше значений, чем требуется оператором INPUT, то выдается сообщение:

ОШИБКА 121 СТРОКЕ n

(где n — номер строки с оператором INPUT); если введено больше значений, то выдается сообщение:

ОШИБКА 122 СТРОКЕ n

В обоих случаях после сообщения об ошибке снова выводится на экран знак вопроса, затем следует правильно повторить набор вводимых значений.

Предупреждение: в ответ на вопросительный знак пользователь обязан либо ввести требуемые значения, либо прекратить выполнение программы одновременным нажатием клавиш «СУ» и «Р». Других возможностей нет. Любые иные действия воспринимаются как ошибка. Начинающие пользователи, не понимая смысла знака вопроса, обычно набирают команду LIST. Машина отвергает этот неправильный ответ и снова предлагает им знак вопроса. Не находя выхода из создавшегося положения, начинающие пользователи просто выключают машину (что приводит к потере их программы), повторяют все сначала и снова приходят к тому же вопросительному знаку.

Появление на экране знака вопроса при выполнении оператора INPUT является не очень удобной для пользователя фор-

мой запроса от машины, так как не указано, какую именно величину надо вводить (в программе может быть несколько операторов INPUT), а также количество величин и порядок ввода.

Пользователь может облегчить работу, выдав на экран перед знаком вопроса имена вводимых переменных, что осуществляется с помощью оператора PRINT.

Использование оператора INPUT и комментирующего его оператора PRINT приводится в программе 3 для вычисления стоимости всех товаров по данным табл. 1.

Программа 3 и ее выполнение

```
10 REM Вычисление стоимости товаров с использованием
11 REM оператора INPUT
20 PRINT «K, C=»;
30 INPUT K, C
40 LET S =K *C
50 PRINT «Кол.=»K, «Цена=»C, «Стоим.»=S
RUN
K, C=?20, 135
Кол.= 20      Цена= 135      Стоим.= 2700
ОСТ СТРОКЕ 50
ЖДУ
RUN
K, C=?10, 265
Кол.= 10      Цена= 265      Стоим.= 2650
ОСТ СТРОКЕ 50
ЖДУ
RUN
K, C=?50, 4
Кол.= 50      Цена= 4        Стоим.= 200
ОСТ СТРОКЕ 50
ЖДУ
RUN
K, C=?100, 2.5
Кол.= 100     Цена= 2.5      Стоим.= 250
ОСТ СТРОКЕ 50
ЖДУ
```

При выполнении программы по команде RUN на экране возникает информация «K, C=?». Информация «K, C=» выдается на экран оператором 20. Она подсказывает пользователю, какие величины вводить и в каком порядке. Порядок ввода величин должен соответствовать порядку их указания в операторе INPUT. Оператор 20 заканчивается разделителем в виде точки с запятой. Она обеспечивает появление вопросительно-

го знака на одной строке с «К, С=» и рядом с этими символами.

Вопросительный знак выдается на экран оператором INPUT. В ответ пользователь обязан набрать на той же строке значения количества и цены, разделенные запятой. При нажатии клавиши «ВК» машина принимает эти значения и продолжает вычисления по программе.

В этой программе функции блока *5*, представленной на рис. 2 блок-схемы, выполняет сам пользователь. Он запускает программу на очередное выполнение командой RUN до тех пор, пока не будут обработаны все товары. Принципиальное преимущество программы 3 перед программой 2 состоит в том, что в программе 3 не требуется изменять ее операторы при изменении значений исходных данных.

Для характеристики работы оператора INPUT (см. программу 3) справедливы следующие выводы.

1. Оператор INPUT удобен для работы в оперативном режиме, так как выбор конкретных значений вводимых переменных не ограничивается никакими указаниями в программе. Его удобно использовать для подбора значений переменных, дающих желаемый результат, поиска возможного диапазона изменения переменных и в других случаях, требующих предварительной проверки. Достоинством оператора INPUT является оперативность работы.

2. Выдача информации при повторном выполнении программы отделяется от предыдущей выдачи четырьмя строками служебных сообщений:

```
ОСТ СТРОКЕ n
(где n — номер последней строки программы)
ЖДУ
RUN
[имена переменных]?
```

Введение служебного текста в выводимые результаты — недостаток оператора INPUT.

3. При использовании оператора INPUT быстродействующая ЭВМ выдает запрос и может продолжить работу только после правильного ответа на него. Но человек медленно и не всегда правильно реагирует на запросы ЭВМ, допускает ошибки при вводе значений и тем самым снижает производительность ЭВМ. В этом состоит второй недостаток оператора INPUT.

Применение оператора INPUT для ввода исходных данных требует учета всех его достоинств и недостатков.

Другим способом ввода исходных данных является применение оператора READ.

6. ОПЕРАТОРЫ READ, DATA И RESTORE

READ в переводе с англ.— читать;
DATA — данные и RESTORE — восстановить содержимое памяти

Форматы операторов:

READ имя-переменной-1 [, имя-переменной-2] ...
DATA значение-переменной-1 [, значение-переменной-2]...
RESTORE

Эти операторы используются для ввода данных в оперативную память. Операторы READ и DATA являются основными, оператор RESTORE — вспомогательным.

Оператор DATA должен быть единственным в строке. Оператор RESTORE состоит только из одного слова.

Операторы READ и DATA работают только в паре друг с другом. Оператор DATA, встретившийся раньше оператора READ, сам по себе не выполняется и пропускается. Его выполнение производится только при выполнении оператора READ. В программе может быть несколько операторов READ и DATA. Взаимное расположение операторов READ и DATA не имеет значения.

При выполнении операторов READ указанным в них именам переменных присваиваются последовательные значения из операторов DATA, начиная с оператора DATA с наименьшим номером. Каждое значение из DATA используется только один раз. Количество имен переменных во всех операторах READ должно быть не больше количества значений во всех операторах DATA (при отсутствии операторов RESTORE). В противном случае выдается сообщение:

ОШИБКА 20 СТРОКЕ n

(где n — номер строки с оператором READ, для имен переменных которого не хватило значений переменных в операторах DATA).

Работу операторов READ и DATA можно пояснить на конкретном примере. Числа в скобках на строках вывода указывают номера операторов PRINT, по которым выводятся эти строки.

Пример работы операторов READ и DATA

```
10 DATA 1, 2, 3
20 READ A, B
30 PRINT A, B
40 READ C, D
50 PRINT C, D
60 DATA 4, 5
70 READ E, F
80 PRINT E, F
RUN
```

```
 1         2                                        (30)
 3         4                                        (50)
ОШИБКА 20 СТРОКЕ 70
ЖДУ
```

Переменным A и B из оператора 20 присваиваются соответственно значения 1 и 2 из оператора 10. Переменным C и D из оператора 40 присваиваются соответственно значение 3 из оператора 10 и значение 4 из оператора 60. Переменной E из оператора 70 присваивается значение 5 из оператора 60, а переменной F из оператора 70 уже не хватает значений. Это приводит к сообщению об ошибке.

Оператор RESTORE при каждом своем выполнении восстанавливает все значения в операторах DATA, начиная с оператора DATA с наименьшим номером. Следующий за RESTORE оператор READ начинает считывание значений с самого начала (с первого значения первого оператора DATA). Работа оператора RESTORE показана в следующем примере.

Числа в скобках на строках вывода указывают номера операторов PRINT, по которым выводятся эти строки.

Пример работы оператора RESTORE

```
10  DATA 1, 2, 3
20  READ A, B
30  PRINT A, B
35  RESTORE
40  READ C, D
50  PRINT C, D
60  DATA 4, 5
70  READ E, F
80  PRINT E, F
90  RESTORE
100 READ G, H, I, J
110 PRINT G, H, I, J
RUN
 1         2                                       (30)
 1         2                                       (50)
 3         4                                       (80)
 1         2         3         4                   (110)
```

Переменным A и B из оператора 20 присваиваются соответственно значения 1 и 2 из оператора 10. Оператор RESTORE в строке 35 позволяет снова использовать эти значения. Переменным C и D из оператора 40 присваиваются соответственно восстановленные значения 1 и 2 из оператора 10. Переменным E и F из оператора 70 присваивается соответственно значение 3 из оператора 10 и значение 4 из оператора 60. Оператор RESTORE в строке 90 позволяет снова использовать все значения в операторах DATA, начиная с оператора DATA с наименьшим номером. Оператор 100 присваивает переменным G, H, I, J зна-

чения 1, 2, 3 из оператора 10 и значение 4 из оператора 60 соответственно. Значение 5 из оператора 60 осталось неиспользованным. Окончание выполнения программы завершается выполнением оператора в строке с наибольшим номером.

Сравнение операторов READ и INPUT показывает, что их преимущества и недостатки меняются местами. Для характеристики работы оператора READ справедливы следующие выводы.

1. Все вводимые оператором READ значения заранее предопределены оператором DATA и не могут быть изменены без изменения оператора DATA. Следовательно, использование оператора READ удобно только в тех случаях, когда точно известны последовательные значения переменных (что чаще всего и встречается на практике) и неудобно для задания произвольных значений переменных.

2. При использовании оператора READ служебные сообщения не выводятся (кроме сообщения об ошибке при недостаче значений переменных для оператора READ). Это позволяет выводить результаты вычислений, не «засоренные» служебными сообщениями.

3. ЭВМ выполняет свою работу непрерывно, не обращаясь за информацией к человеку. Это обеспечивает высокую эффективность ее использования.

Предупреждение: между операторами READ и DATA нет другого соответствия, кроме последовательного считывания операторами READ данных из операторов DATA. Для операторов READ не существует «своих» операторов DATA. Операторы DATA выполняют всегда в порядке возрастания их номеров, даже если они находятся в невыполняемой в каком-то случае ветви программы. В подобной ситуации потребуется считывание «лишних» данных для сохранения необходимого соответствия между операторами READ и DATA. Для облегчения контроля за соответствием операторов READ и DATA рекомендуется все операторы DATA собирать вместе в начале программы.

Приведенное сравнение работы операторов INPUT и READ указывает на предпочтительность использования второго оператора в задаче вычисления стоимости товаров. Однако попытка прямолинейной замены в программе 3 оператора INPUT и комментирующего его оператора PRINT на операторы READ и DATA приводит к выполнению вычислений лишь для одного товара, как показано в программе 4.

Программа 4 и ее выполнение

```
10 REM Попытка 1 использования оператора READ
20 DATA  20, 135, 10, 265, 50, 4, 100, 2.5
30 READ K, C
40 LET S=K*C
50 PRINT «Кол.=»K, «Цена=»C, «Стоим.=»S
RUN
Кол.= 20       Цена= 135       Стоим.= 2700
```

ОСТ СТРОКЕ 50
ЖДУ

Здесь ЭВМ выдала ответ для одного товара и остановилась, хотя в операторе DATA представлены данные для четырех товаров. Это объясняется тем, что операторы программы выполняются последовательно друг за другом, и нет никакого указания, что после оператора PRINT в строке 50 надо еще что-то делать. При выполнении программы 3 такое указание давал пользователь командой RUN, вызывающей новое выполнение программы. При этом замена исходных данных производилась введением новых их значений с клавиатуры в ответ на запрос машины, выдаваемый знаком вопроса. В программе 4 повторный запуск программы командой RUN приведет к повторению первоначального ответа. Получилось, что в программе 4 пользователь лишился возможности выполнения функций блока *5* представленной на рис. 2 блок-схемы и не компенсировал эту потерю использованием каких-либо других средств программирования. Выход из создавшегося положения состоит в том, чтобы не повторяя заново выполнение оператора DATA, обеспечить многократное выполнение оператора READ и следующих за ним операторов. Для этого после выполнения оператора PRINT в строке 50 надо обеспечить продолжение вычислений с оператора READ в строке 30. Этого можно достигнуть средствами, которые рассматриваются ниже.

7. ОПЕРАТОР GO TO

GO TO **в переводе с англ.— перейти к**

Формат оператора:

GO TO номер-строки

Оператор GO TO должен быть последним в строке, потому что расположенные за ним операторы принципиально не могут быть выполнены.

Так как пробелы в Бейсике не имеют никакого значения, то слитное написание оператора равносильно его раздельному написанию.

Оператор GO TO изменяет последовательное построчное выполнение программы, передавая управление оператору с указанным в GO TO номером строки. **Передачей управления** называется указание следующего выполняемого оператора. Оператор GO TO называется оператором **безусловной передачи управления**, или **безусловного перехода**, потому что он всегда (независимо ни от каких условий) передает управление указанной в нем строке.

Оператор GO TO используется либо для ухода на новый участок программы, либо для возврата к пройденному участку

программы. Дальнейшее продвижение по программе будет определяться уже операторами этих участков.

Многократное выполнение одних и тех же участков программы называется **циклом.** Если программа из этих участков будет всегда возвращаться к тому же оператору GO TO, то это приведет к бесконечному повторению одних и тех же команд (если не будет переполнения разрядной сетки машины или исчерпания значений переменных в операторах DATA). Бесконечное повторение одних и тех же команд называется **зацикливанием.**

Следовательно, при использовании оператора GO TO надо заботиться о нормальном окончании процесса вычислений.

Простейший пример зацикливания можно представить в следующем виде:

```
10 LET A=25
20 LET B=A
30 GO TO 10
```

Операторы с 10 по 30 должны повторяться бесконечно. Значения A и B остаются постоянными и находятся в пределах разрядной сетки машины, так что переполнения быть не может. Других операций в цикле нет.

Прервать зацикливание можно одновременным нажатием клавиш «СУ» и «Р». При этом выполнение программы прекращается, на строке экрана появляются символы « ┐Р», а на следующей строке — сообщение «ЖДУ». После этого можно задавать новую информацию. Иначе прервать зацикливание невозможно. Любые другие попытки вмешаться в работу машины отвергаются.

Пример выхода из зацикливания по сообщению об ошибке приведен в программе 5. Здесь сделана попытка исправления программы 4. Эта попытка приводит к вычислению стоимости не одного, а всех товаров.

Программа 5 и ее выполнение

```
10 REM Попытка 2 с введением оператора GO TO
20 DATA 20, 135, 10, 265, 50, 4, 100, 2.50
30 READ K, C
40 LET S=K*C
50 PRINT «Кол.=»K, «Цена=»C, «Стоим.=»S
60 GO TO 30
RUN
Кол.=  20      Цена=  135     Стоим.=  2700
Кол.=  10      Цена=  265     Стоим.=  2650
Кол.=  50      Цена=  4       Стоим.=  200
Кол.=  100     Цена=  2,5     Стоим.=  250
ОШИБКА 20 СТРОКЕ 30
ЖДУ
```

Формально цель задачи достигнута — вся необходимая информация выдана. Но выполнение программы закончилось сообщением об ошибке, и если бы в ней требовались дальнейшие вычисления, то они не проводились бы. Причина ошибки в том, что оператор GO TO в строке 60 требует бесконечного повторения ввода данных, а этот процесс всегда должен быть конечным. Если обратиться к блок-схеме, представленной на рис. 2, то можно сказать, что вместо блока *5* программа 5 реализует другой блок. Этот блок не содержит проверки обработки всех товаров и не предусматривает возможности выхода на окончание вычислений. Он постоянно требует возобновления ввода данных.

Кроме того, повторение на всех строках названий выдаваемых величин плохо смотрится на экране. Целесообразнее было бы представить эти названия один раз в виде названий граф, как это принято в таблицах.

Следовательно, программа 5 нуждается в доработке и применении новых средств языка Бейсик.

8. ОПЕРАТОР IF

IF **в переводе с англ.— если**

Формат оператора:

IF условие [THEN] { номер-строки | GO TO номер-строки | оператор }

Условие составляется по правилам, приведенным в п. 7 гл. 3.

Слово THEN (в переводе с англ.— то, тогда, затем) может опускаться только в варианте с GO TO (употребление не запрещено).

На месте оператора за словом THEN может использоваться любой оператор, в том числе и IF (вложенный IF). Оператор за словом THEN является только частью оператора IF, а не самостоятельным оператором.

Оператор IF называется оператором **условной передачи управления**, или **условного перехода.** При выполнении поставленного в нем условия управление передается на часть оператора IF, стоящую за словом THEN. При невыполнении условия выполняется следующий за IF оператор, т. е. в этом случае действия оператора IF ограничиваются лишь проверкой условия.

Первый и второй варианты за словом THEN равнозначны. В обоих случаях при выполнении условия управление передается на строку с указанным номером. Эта строка может быть расположена до оператора IF или на любом отдалении после него. Третий вариант за словом THEN позволяет использовать в операторе IF другие операторы, отличные от операторов безусловной передачи управления (например, LET, PRINT или

снова IF). Он применяется в программах в тех случаях, когда при невыполнении условия ничего не надо делать, а при выполнении условия требуется дополнительно выполнить лишь один оператор, а затем перейти к следующему за IF оператору.

Оператор IF используется для организации разветвлений в программе. При выполнении условия вычисления идут по одной ветви программы, при невыполнении — по другой. Действие оператора IF можно пояснить с помощью блок-схемы, представленной на рис. 7.

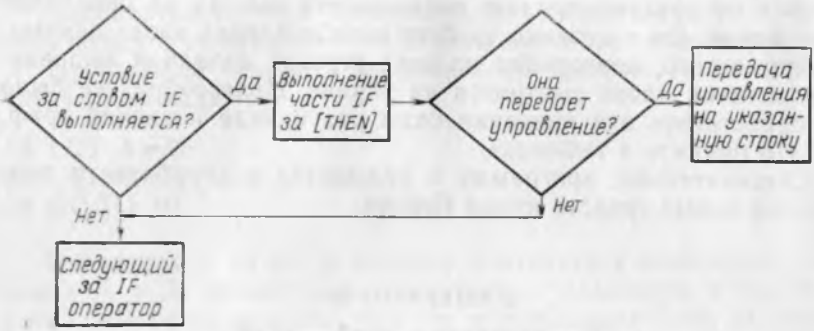


Рис. 7. Блок-схема работы оператора IF

Как следует из рис. 7, после проверки условия возникают две ветви программы. Одна соответствует ответу «Да», другая — «Нет». Если оператор за словом THEN не передает управления, то эти ветви снова сливаются в одну, в противном случае все будет определяться дальнейшими операторами программы. Наличие в программе нескольких операторов IF может создать сложную разветвленную структуру программы.

Работа оператора IF поясняется на конкретных примерах. Числа в скобках на строках вывода указывают номера операторов PRINT, по которым выводятся эти строки.

```
10  LET A=25
20  LET B=A
30  IF A=B THEN 50
40  PRINT «Не равно»
45  GO TO 60
50  PRINT «Равно»
60  LET A=A+1
70  IF A>B GO TO 90
80  PRINT «Не больше»
90  PRINT A
100 IF A—1=B THEN PRINT A; B; A—B
110 LET C=A+B
120 IF C<A THEN PRINT C
130 PRINT «Конец»
```

```
RUN
Равно                                    (50)
 26                                      (90)
 26   25   1                             (100)
Конец                                    (130)
```

Условие A=B в операторе 30 выполняется, поэтому управление передается на оператор 50, а операторы 40 и 45 пропускаются. Условие A>B в операторе 70 выполняется (в операторе 60 значение A увеличено на 1), поэтому управление передается оператору 90, а оператор 80 пропускается. Условие A—1=B в операторе 100 выполняется, поэтому выполняется стоящий в нем за словом THEN оператор PRINT, выдающий на экран значения A, B и A—B. Далее выполняется следующий за IF оператор 110. Условие C<A в операторе 120 не выполняется, поэтому не выполняется стоящий за THEN оператор PRINT C, а управление передается следующему за IF оператору 130. После выполнения оператора 130 работа программы заканчивается.

Оператор IF можно использовать для исправления программы 5. С его помощью можно организовать проверку окончания вычислений и уход либо на ветвь продолжения вычислений, либо на ветвь их окончания в зависимости от результатов проверки.

Можно предложить два способа проверки окончания вычислений. Первый можно назвать окончанием вычислений по условному значению, второй — по количеству проделанных циклов вычислений.

Для проверки первым способом можно ввести в оператор DATA за данными для последнего товара какое-либо условное значение, которое принципиально не может встретиться среди значений исходных данных по смыслу задачи. Например, цена не может быть нулевой, поэтому каждую вводимую цену надо проверять на равенство нулю. При выполнении равенства необходимо уйти на окончание вычислений. Следует заметить, что оператор READ вводит сразу две величины, поэтому количество тоже должно быть представлено каким-либо значением (например, также нулевым).

Приведенных словесных описаний пути решения проблемы вполне достаточно для знающего программиста. Начинающему программисту лучше пользоваться блок-схемой, дающей наглядное представление о последовательности действий, которые должна выполнить ЭВМ.

В предлагаемом способе проверка на окончание вычислений должна проводиться после ввода данных о товаре, а не после вывода результатов вычислений для него. Поэтому используемая до сих пор блок-схема, представленная на рис. 2, должна измениться. Кроме того, следует устранить отмеченный недостаток программы 5 в представлении названий выдаваемых ве-

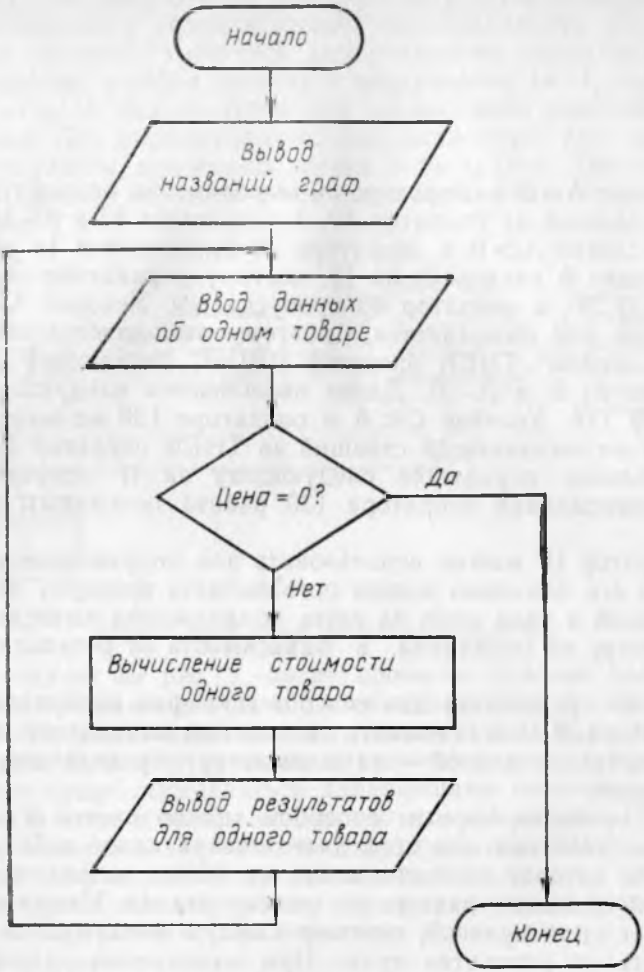


Рис. 8. Блок-схема для вычисления стоимости товаров с окончанием вычислений по условному значению

личин и организовать их вывод в виде названий граф таблицы. Это требуется сделать один раз до начала цикла обработки данных о товарах. Следовательно, потребуется ввести новый блок в блок-схему. В результате получается новая блок-схема, представленная на рис. 8.

В соответствии с этой блок-схемой составлена программа 6. В ней устранены недостатки программы 5. Все операторы программы 5 в неизменном или измененном виде вошли в программу 6, где за ними сохранены прежние номера строк. Это сделано для удобства превращения введенной в машину программы 5 в программу 6. Поэтому включение в программу 6 дополнитель-

ных операторов привело к появлению номеров строк, не кратных 10.

Программа 6 и ее выполнение

```
10 REM Окончание вычислений по условному значению
15 PRINT «Кол.», «Цена», «Стоим.»
20 DATA 20, 135, 10, 265, 50, 4, 100, 2.5, 0, 0
30 READ K, C
35 IF C=0 GO TO 70
40 LET S=K * C
50 PRINT K, C, S
60 GO TO 30
70 PRINT
RUN
Кол.        Цена        Стоим.
 20          135         2700
 10          265         2650
 50          4           200
 100         2.5         250
```

Операторы 15 и 20 можно поменять местами, соответственно изменив номера их строк. Ход вычислений от этого не меняется. Для окончания вычислений в этой программе следовало бы использовать оператор останова. Но поскольку он еще не рассмотрен (см. п. 11), вместо него используется оператор вывода пустой строки, не портящий программу.

Окончание вычислений по условному значению в операторе DATA позволяет использовать программу 6 для списка товаров произвольной длины. При этом изменяется лишь оператор DATA, в котором приводятся конкретные исходные данные о товарах.

Второй способ проверки на окончание вычислений состоит в подсчете проведенных циклов вычислений и в сравнении подсчитанного значения с заданным, вытекающим из условия задачи. Заданное число циклов вычислений указывается в операторе DATA в качестве первого значения, так как оно должно быть считано оператором READ до начала обработки данных о товарах. Для подсчета произведенных циклов вычислений выделяется место в оперативной памяти машины, используемое в качестве счетчика. Начальное значение счетчика задается равным нулю. Перед обработкой каждого очередного товара в счетчик добавляется единица. Таким образом, значение в счетчике соответствует порядковому номеру обрабатываемого товара. После обработки очередного товара значение в счетчике сравнивается с заданным. Если значение в счетчике меньше заданного, то значение счетчика увеличивается на единицу и производится обработка очередного товара, т. е. цикл повторяется. В противном случае происходит окончание вычислений. Блок-схема, соответствующая данному алгоритму вычислений, представлена на рис. 9.

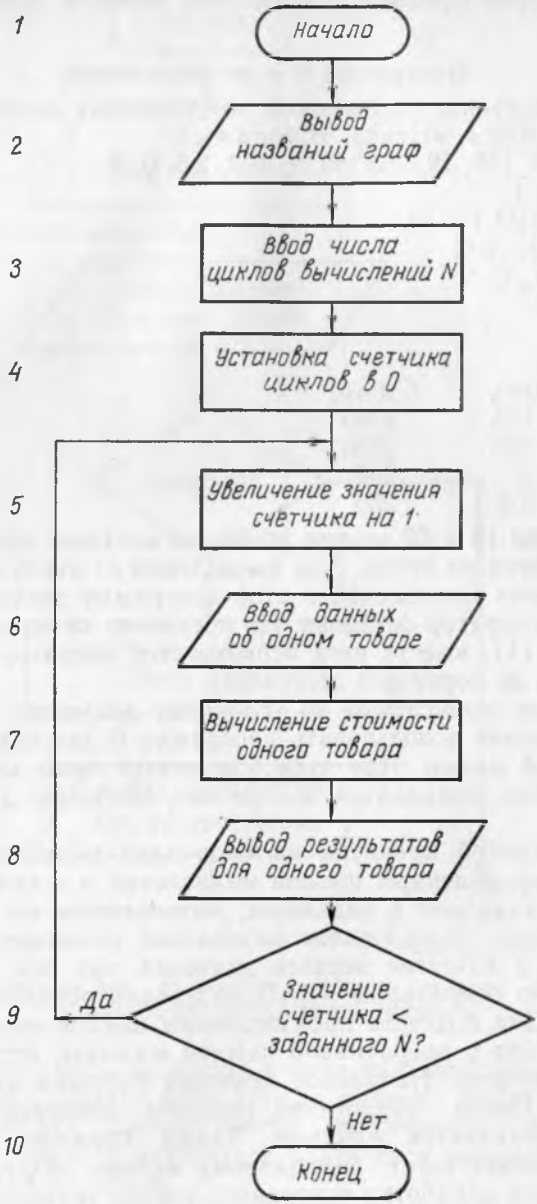


Рис. 9. Блок-схема для вычисления стоимости товаров с подсчетом циклов проведенных вычислений

По этой блок-схеме составлена программа 7.

Программа 7 и ее выполнение

```
10 REM Окончание вычислений по подсчету циклов
15 PRINT «Ном.», «Кол.», «Цена», «Стоим.»
20 DATA 4, 20, 135, 10, 265, 50, 4, 100, 2.5
23 READ N
26 LET  I=0
28 LET I=I+1
30 READ K, C
35
40 LET S=K*C
50 PRINT I, K, C, S
60 IF I<N GO TO 28
70
RUN
```

Ном.	Кол.	Цена	Стоим.
1	20	135	2700
2	10	265	2650
3	50	4	200
4	100	2.5	250

В программе 7 для операторов, сходных с операторами программы 6, сохранена нумерация их строк. Это сделано для удобства превращения введенной в машину программы 6 в программу 7. При этом нельзя забывать о необходимости удаления из оперативной памяти тех операторов программы 6, которые не входят в программу 7 и номера строк которых не используются в программе 7. Для удаления таких операторов в программе 7 указываются номера их строк без последующего текста (см. п. 10 гл. 3). Дополнительные операторы, включенные в программу 7, получают номера строк, не кратные 10.

Значение в счетчике, соответствующее порядковому номеру обрабатываемого товара, можно использовать в качестве порядкового номера выдаваемой строки. Поэтому в оператор 15 введен символьный элемент «Ном.», а в оператор 50 — имя переменной I. Оператор DATA должен начинаться с задания числа циклов вычислений N. В данном случае выбрано N=4 (как и в предыдущих примерах). Операторы 15, 20, 23, и 26 могут произвольно меняться местами при соответствующем изменении номеров их строк. Оператор 28 может занимать любое место после операторов 23 и 26 и до оператора 50. При этом надо учитывать, что номер строки 28 используется в операторе 60 для передачи управления. Операторы 35 и 70 из программы 6 оказываются лишними и исключаются из программы 7.

Задание числа циклов вычислений в операторе DATA позволяет использовать программу 7 для списка товаров произвольной длины. При этом изменяется лишь оператор DATA, в кото-

ром представляются конкретные исходные данные о товарах. В этом отношении программы 6 и 7 равноценны.

В программе 7 окончание вычислений по подсчету произведенных циклов обеспечивается тремя операторами: 26, 28 и 60. Эти операторы соответствуют блокам 4, 5 и 9 блок-схемы, представленной на рис. 9. Они производят установку начального значения счетчика, его увеличение и проверку конечного значения. Операторы, реализующие эти функции, оказываются разбросанными по программе, а соответствующие им блоки — по блок-схеме. Все это усложняет логику построения и блок-схемы, и программы. Поэтому данный способ организации цикла используется только в том случае, когда не подходит другой способ. Там же, где это возможно, предпочтительнее использовать операторы цикла FOR и NEXT, рассматриваемые ниже.

9. ОПЕРАТОРЫ FOR и NEXT

FOR **в переводе с англ.— для;** NEXT — **следующее значение**

Форматы операторов:

FOR имя-переменной=арифметическое-выражение-1
TO арифметическое-выражение-2
[STEP арифметическое-выражение-3]
NEXT имя-переменной

В переводе с англ. TO означает до, STEP — шаг.

При расположении в одной строке с другими операторами FOR должен быть первым в своей строке, а NEXT — последним в своей.

Операторы FOR и NEXT называются операторами цикла, так как используются для его организации. Они работают только в паре. Оператор FOR называется **заголовком цикла**, оператор NEXT — **концом цикла.** Имя переменной в операторе NEXT должно быть тем же самым, что и в операторе FOR. Эта переменная называется **управляющей переменной цикла.** Арифметическое выражение 1 называется начальным значением управляющей переменной, арифметическое выражение 2 — ее граничным значением. Они определяют область действия управляющей переменной. Арифметическое выражение 3 называется шагом изменения управляющей переменной. Если оно опущено, то по умолчанию принимается шаг, равный +1. При положительном шаге граничное значение должно быть больше начального, при отрицательном — меньше; в противном случае цикл не выполняется. Арифметические выражения могут принимать целые, дробные, положительные и отрицательные значения.

Между операторами FOR и NEXT располагаются операторы, подлежащие многократному выполнению. Они называются **телом цикла.**

Пример организации цикла

```
10 FOR I=1 TO 10
20 PRINT I;
30 NEXT I
40 PRINT I
```

Здесь шаг изменения управляющей переменной опущен и по умолчанию принимается равным +1. Тело цикла составляет единственный оператор 20. Оператор 40 является следующим за циклом оператором.

В общем случае выполнение цикла начинается с вычисления арифметических выражений 1, 2, 3 в операторе FOR и с присвоения начального значения управляющей переменной. Если оно находится в области действия управляющей переменной, то выполняются операторы тела цикла. Затем оператор NEXT возвращает управление оператору FOR. Управляющая переменная изменяется на величину шага. Если она остается в ее области действия, то снова выполняются операторы тела цикла, и описанный процесс повторяется. Происходит циклическое выполнение операторов тела цикла. При последнем выполнении цикла управляющая переменная либо точно равна ее граничному значению, либо отличается от него меньше чем на шаг (по модулю). Это значение управляющей переменной сохраняется, и происходит выход из цикла на следующий за NEXT оператор.

Замечание. Значение управляющей переменной нельзя изменять в операторах тела цикла.

В рассмотренном примере оператор 20 будет циклически выполняться при значениях управляющей переменной равных 1, 2, 3, ..., 10. После этого выполняется следующий за NEXT оператор 40, который выводит значение 10 (на той же строке, так как оператор 20 заканчивается точкой с запятой). В этом примере управляющая переменная точно вышла на ее граничное значение. Если бы в операторе 10 был указан шаг 2, то выполнение цикла происходило бы при значениях управляющей переменной равных 1, 3, 5, 7, 9. В этом случае конечное значение управляющей переменной отличалось бы от ее граничного значения меньше, чем на шаг.

Описанный процесс работы операторов цикла в наглядном виде представлен на блок-схеме, изображенной на рис. 10. Для сокращения словесных описаний на схеме не указано, что первоначально производится вычисление арифметических выражений в операторе FOR (если они представлены в общем виде, а не в виде констант или имен переменных). Не указано также, что конечное значение управляющей переменной при выходе из цикла сохраняется (в интерпретаторах, отличных от интерпретатора ДВК-1, может и не сохраняться).

Операторы FOR и NEXT выполняют все необходимое для организации цикла: устанавливают начальное значение управляющей переменной, изменяют ее на величину шага, цикличе-

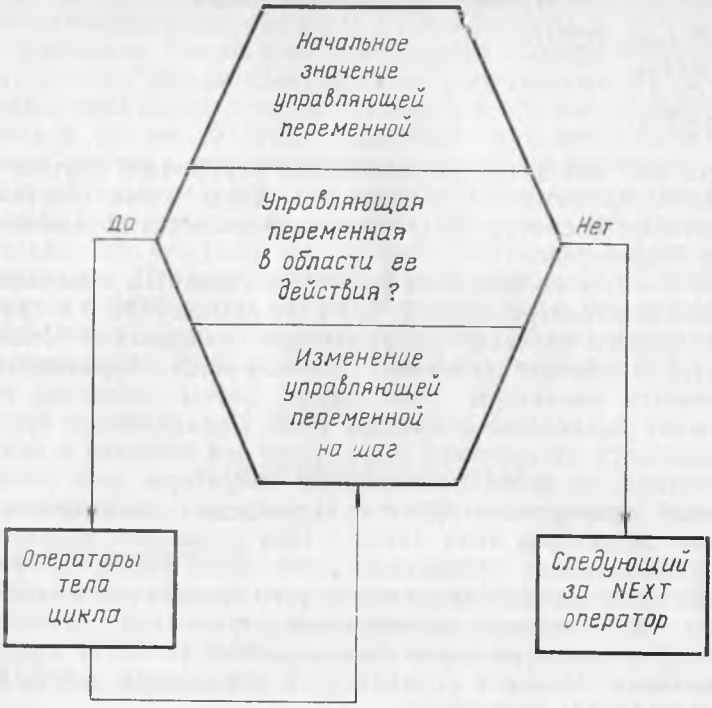


Рис. 10. Блок-схема работы операторов цикла FOR и NEXT

ски выполняют операторы тела цикла и осуществляют выход из цикла. При этом все переменные цикла записываются в удобной форме в одном операторе FOR.

Например, в блок-схеме, представленной на рис. 9, организацию цикла обеспечивают блоки 4, 5 и 9, рассредоточенные по блок-схеме. При использовании операторов FOR и NEXT эта блок-схема преобразуется в блок-схему, представленную на рис. 11. Ее изменение приводит к соответствующему изменению программы 7 в программу 8.

Операторы, изменяющие программу 7 в программу 8

```
10 REM Использование операторов цикла FOR и NEXT
26
28 FOR I=1 TO N
60 NEXT I
```

Программа становится на один оператор короче, так как отпадает необходимость в предначальной установке счетчика. Но главное, оператор FOR показывает, что далее последует цикл с приведенными значениями управляющей переменной в этом цикле.

В данной главе подробно рассматривалось решение одной и

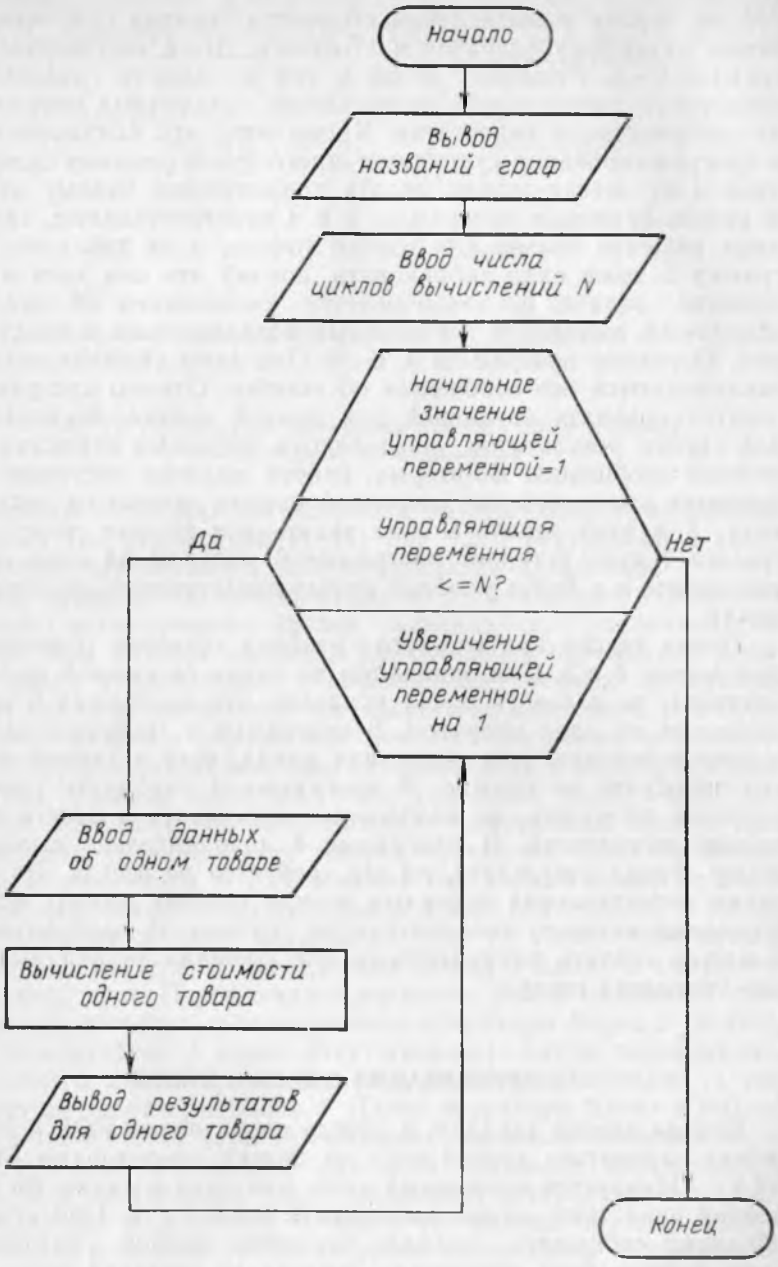


Рис. 11. Блок-схема для вычисления стоимости товаров с использованием операторов FOR и NEXT

той же задачи о вычислении стоимости товаров при использовании различных операторов Бейсика. Были составлены программы 2—8. Решение одной и той же задачи различными средствами сопоставляет возможности различных операторов, их достоинства и недостатки. Кроме того, это показывает, что в программировании существует много путей решения одной задачи и не всегда можно отдать предпочтение одному из них. В наших примерах программы 2 и 4 просто отпадают, так как дают решение только для одного товара, а не для всех. Программу 5 тоже надо забраковать, потому что она хотя и дает решение задачи, но заканчивается сообщением об ошибке в программе, вызванной бесконечным возвращением к вводу данных. Остаются программы 3, 6—8. Они дают решение задачи и заканчиваются без сообщения об ошибке. Однако программу 3 следует признать неудачной для данной задачи. Во-первых, в ней строки результатов разделяются четырьмя строками служебных сообщений. Во-вторых, работа машины постоянно прерывается для получения очередной порции данных от пользователя. А в этой задаче можно задать все данные сразу. Программа 7 явно уступает программе 8, которая на один оператор короче и в более удобной форме представляет организацию цикла.

После такого критического разбора остается сопоставить программы 6 и 8. Они одинаковы по длине (в каждой по 9 операторов), но далее (в п. 11) показано, что программу 6 можно сократить на один оператор. В программе 6 требуется вводить условное значение для окончания цикла, хотя в данной задаче его подобрать не сложно. В программе 8 подбирать условное значение не нужно, но необходимо подсчитать и ввести число циклов вычислений. В программе 8 дополнительно выводится номер строки результата, но это требуется не всегда. На основании сопоставления программ можно сделать вывод: если не оговорены какие-то дополнительные условия, то программы 6 и 8 можно считать равноценными для решения задачи вычисления стоимости товара.

10. ОРГАНИЗАЦИЯ ЦИКЛА В ЦИКЛЕ

Внутри одного цикла, т. е. между операторами FOR и NEXT, может находиться другой цикл со своими операторами FOR и NEXT. Образуется вложенный цикл, или цикл в цикле. Во внутренний цикл тоже можно вкладывать цикл и т. д. При этом необходимо соблюдать правило вложения циклов: внутренний цикл должен быть полностью вложен во внешний цикл, т. е. FOR и NEXT внутреннего цикла должны находиться обязательно между FOR и NEXT внешнего цикла, как показано на рис. 12. На рис. 12, *а* представлена правильная организация вложенных циклов. При этом операторы внутреннего цикла (по

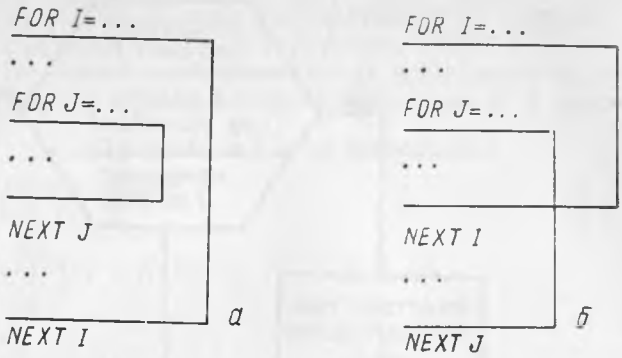


Рис. 12. Организация вложенных циклов (цикла в цикле):
а — правильная; *б* — неправильная

переменной J) находятся полностью внутри операторов внешнего цикла (по переменной I). На рис. 12, *б* внутренний и внешний циклы пересекаются, что запрещается правилами.

При организации цикла в цикле управляющие переменные внешнего и внутреннего циклов изменяются с различной скоростью. При каждом значении управляющей переменной внешнего цикла управляющая переменная внутреннего цикла пробегает все свои значения от начального до конечного. Это означает, что внешняя переменная меняется медленно, а внутренняя — быстро. Аналогично двигаются часовая и минутная стрелки. При каждом значении часа (внешний цикл) минутная стрелка пробегает все свои значения (внутренний цикл). Для следующего значения часа минутная стрелка снова пробегает все свои значения и т. д. Вложение еще одного цикла во внутренний цикл аналогично движению секундной стрелки в часах.

Описанный процесс показан на рис. 13 в виде блок-схемы, которая соответствует организации цикла в цикле в соответствии с рис. 12, *а*. Переменная I внешнего цикла принимает свое начальное значение, и выполняются операторы блока *2* (в частных случаях блок *2* может отсутствовать). Затем переменная J внутреннего цикла принимает свое начальное значение, и выполняются операторы блока *4*. Далее операторы блока *4* выполняются циклически (при неизменном начальном значении переменной I внешнего цикла) при всех значениях переменной J из области ее действия. При завершении внутреннего цикла по J выполняются операторы блока *5* (в частных случаях блок *5* может отсутствовать). После этого переменная I принимает свое следующее значение, и снова выполняются операторы блока *2*. Блок *3* снова определяет изменение переменной J во всей области ее действия, как это было и при начальном значении переменной I. Далее процесс повторяется в описанной последовательности до завершения внешнего цикла по переменной I.

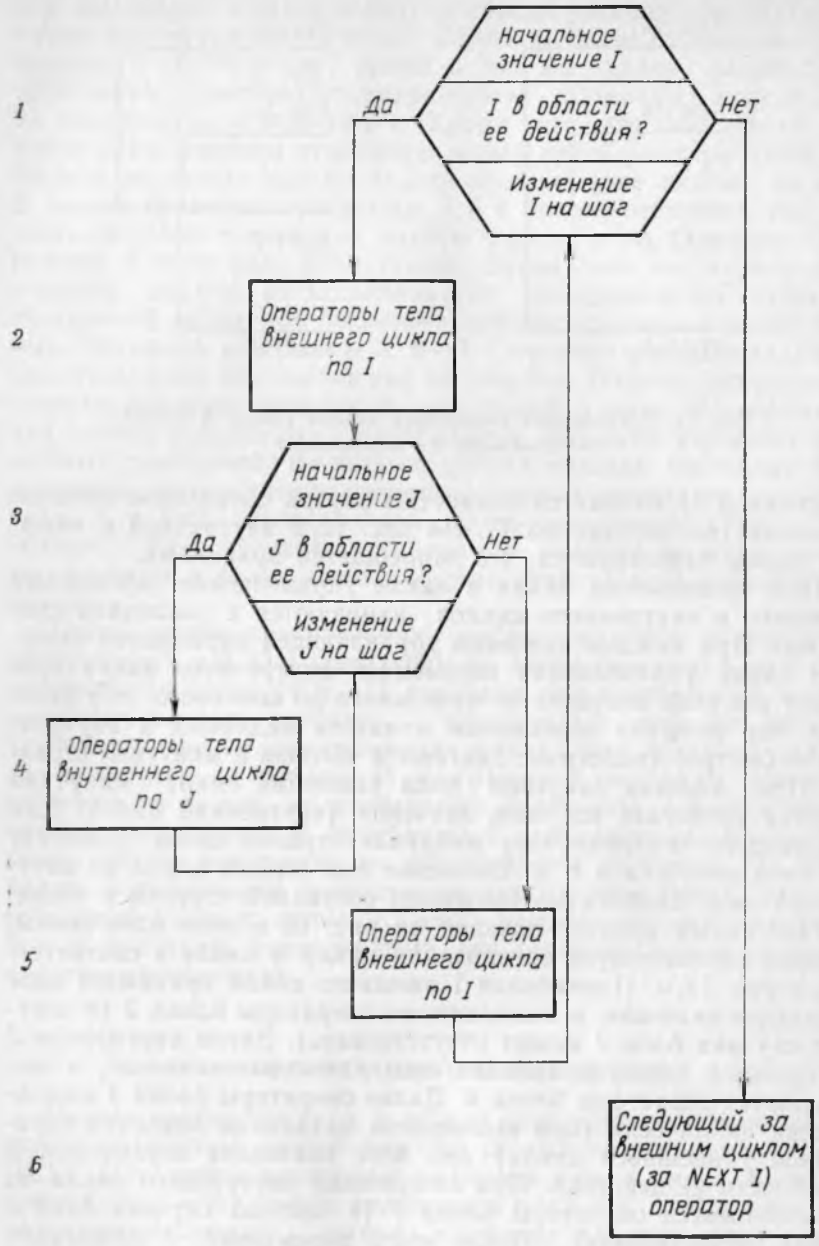


Рис. 13. Блок-схема работы цикла в цикле

Выход из цикла происходит на следующий за внешним циклом оператор, расположенный за оператором NEXT I.

В соответствии с этой блок-схемой составлена программа 9, которая является примером организации цикла в цикле.

Программа 9 и ее выполнение

```
10 REM Цикл в цикле
20 FOR I=1 TO 3
30 PRINT I,
40 FOR J=1 TO 5 STEP 2
50 PRINT J;
60 NEXT J
70 PRINT
80 NEXT I
RUN
 1          1   3   5
 2          1   3   5
 3          1   3   5
```

Оператор 30 заканчивается разделителем в виде запятой, что дает возможность выводить значения J на той же строке, что и значения I, но со следующей зоны. Оператор 50 заканчивается разделителем в виде точки с запятой. Это позволяет выводить во внутреннем цикле все значения J на одной строке в плотном расположении. При выходе из внутреннего цикла во внешний оператор 70 «закрывает» строку вывода значений J, что позволяет оператору 30 выводить новые значения I с новой строки.

Начинающим работу программы 9 рекомендуется анализировать с помощью табл. 5, аналогичной табл. 4.

Таблица 5

Изменение значений переменных I и J по шагам циклов

Шаг цикла по I	Значение I	Шаг цикла по J	Значение J
1	1		
		1	1
		2	3
		3	5
2	2		
		1	1
		2	3
		3	5
3	3		
		1	1
		2	3
		3	5

Эта таблица показывает асинхронность изменения переменных I и J. При каждом фиксированном значении переменной I переменная J изменяется во всей ее области действия. Начинающие программисты, познакомившись с циклом в цикле, забывают об этой асинхронности и помнят только, что при этом изменяются две переменные. В результате они применяют цикл в цикле во всех случаях, требующих изменения двух переменных, не обращая внимания на соотношение скоростей их изменения.

Примером может служить следующая задача. Требуется, чтобы переменная I принимала последовательные значения 1, 2, 3, а переменная J **одновременно** с ней принимала нечетные значения 1, 3, 5. Попытка применить для решения этой задачи цикл в цикле приводит к программе 9, результат выполнения которой представлен в табл. 5 и не соответствует поставленной задаче. Одновременность изменения обеих переменных исключает применение цикла в цикле.

Для решения этой задачи только одна переменная может изменяться с помощью операторов цикла FOR и NEXT. Другая переменная должна изменяться методом ее наращивания с помощью оператора LET, как это делалось в блок-схеме рис. 9 и в соответствующей ей программе 7. Решение задачи представлено в программе 10.

Программа 10 и ее выполнение

```
10 REM Одновременное изменение двух переменных
20 LET J=—1
30 FOR I=1 TO 3
40 LET J=J+2
50 PRINT I, J
60 NEXT I
RUN
 1          1
 2          3
 3          5
```

С проблемой учета скорости изменения переменных и применения соответствующих методов организации циклов особенно часто приходится сталкиваться при работе с массивами. Это будет показано в следующей главе на примерах решения типовых задач.

11. ОПЕРАТОРЫ STOP И END

STOP **в переводе с англ.— останов;** END **— конец**

Формат каждого из этих операторов состоит из единственного слова.

Оператор END означает физический конец текста программы и должен записываться в строке с наибольшим номером. В ДВК-1 его можно опускать.

Оператор STOP может использоваться в различных местах программы для останова вычислений. В сообщении об останове указывается номер строки с оператором STOP, по которому произошел останов. Для продолжения вычислений в непосредственном режиме выполняется оператор GO TO с номером строки, следующей за оператором STOP.

Останов по STOP применяется для различных целей.

Во-первых, там, где по смыслу задачи нужен оператор останова. Примером может служить программа 6. Она составлялась до рассмотрения оператора останова и поэтому вместо него в строке 70 использовался оператор PRINT. Логичнее было бы записать строку 70 в виде

```
70 STOP
```

Еще лучше вообще отказаться от строки с номером 70 и лишней передачи управления, а использовать оператор STOP непосредственно в операторе 35.

```
35 IF C=0 THEN STOP
```

Во-вторых, он применяется для целей откладки программы, т. е. поиска в ней ошибок при неправильной выдаче ответов. В этом случае в различных местах программы вставляются операторы STOP. При останове вычислений можно в непосредственном режиме выдать на экран с помощью оператора PRINT значения различных величин, которые помогут установить причину ошибки. По окончании отладки операторы STOP удаляются.

В-третьих, оператор STOP применяется для прерывания вывода результатов, когда все строки не умещаются на экране. Если такой вывод производится в цикле, то оператор STOP удобно использовать в качестве составной части оператора IF, как это сделано в следующем примере.

```
10 REM Использование оператора STOP в цикле
20 FOR I=I TO 25
30 PRINT I
40 IF I=10 THEN STOP
50 NEXT I
```

ЭВМ выдает 10 строк и сообщение об останове в строке 40. Для продолжения вычислений набирается GO TO 50 и нажимается клавиша «ВК».

В-четвертых, оператор STOP используется для отделения подпрограмм от основной программы (см. оператор GO SUB).

Оператор STOP можно также использовать для искусственного разделения одной программы на несколько обособленных частей. Каждая из таких частей отгораживается от предыдущих и последующих частей операторами STOP.

Следует учитывать, что оператор RUN всегда начинает выполнение программы с оператора с наименьшим номером во всей программе. Поэтому для выполнения не первой обособленной части вместо RUN используется оператор GO TO (также

в непосредственном режиме) с номером первого оператора в данной обособленной части.

При создании обособленных частей надо убедиться в действительной их независимости от других частей. Особенное внимание надо уделять операторам READ и DATA. Если в предыдущих частях присутствуют операторы DATA, значения которых еще не считаны, а в данной обособленной части имеется оператор READ, то он начнет считывать данные из этих предыдущих частей, так как сначала используются данные операторов DATA с наименьшими номерами. Поэтому придется вставить вспомогательный оператор READ, считывающий все «лишние» данные из операторов DATA предыдущих обособленных частей.

Глава 5
СРЕДСТВА И ПРИЕМЫ ЭФФЕКТИВНОГО ПРОГРАММИРОВАНИЯ

1. ОПЕРАТОР DIM

DIM — **сокращ. от англ.** DIMENSION — **размерность**

Этот оператор резервирует места в оперативной памяти машины для указанных в нем массивов.

Массивом называется упорядоченная последовательность переменных величин с одним и тем же именем. Одна переменная величина последовательности называется элементом массива. Упорядочение элементов массива производится их нумерацией.

Нумерацию можно провести двумя способами. При первом способе все элементы нумеруются по порядку, т. е. применяется последовательная линейная нумерация. При втором способе можно считать, что массив организован в виде таблицы или матрицы и пронумеровать все элементы с использованием номеров строк и столбцов. Число координат, требуемых для определения любого элемента массива, называется размерностью массива. При первом способе для определения элемента массива требуется указать только одну координату (порядковый номер), и массив называется *одномерным*. При втором способе требуется указать две координаты (номер строки и номер столбца), в этом случае массив называется *двумерным*. Массивы больших размерностей в Бейсике не применяются.

Оператор DIM имеет следующий формат:

DIM имя-переменной-1 (n_1 [, m_1])
[, имя-переменной-2 (n_2 [, m_2])] ...

Здесь n и m — наибольшие номера элементов массива по каждому измерению. Нумерация в Бейсике по каждому измерению начинается с нуля. При этом наибольший номер оказывается на единицу меньше количества элементов по данному измерению. Наибольшие номера в операторе DIM задаются только числами, не превышающими 255.

Следует заметить, что в двумерном массиве нельзя задать максимально возможный номер 255 по обоим измерениям сразу. В таком массиве должно быть

$$(255+1)\times(255+1)=65536$$

элементов. Но после размещения программы в оперативной па-

мяти останется место меньше чем на 8000 элементов массива (чем длиннее программа, тем меньше места для элементов массива).

Пусть требуется выделить в оперативной памяти место для 12 элементов одномерного массива с именами элементов V и место для 12 элементов двумерного массива, состоящего из 3 строк и 4 столбцов, с именами элементов W. В соответствии с форматом оператора DIM этого можно достигнуть с помощью оператора

```
10 DIM V (11), W (2, 3)
```

Оператор DIM вводит новую систему адресации оперативной памяти, как показано на рис. 14.

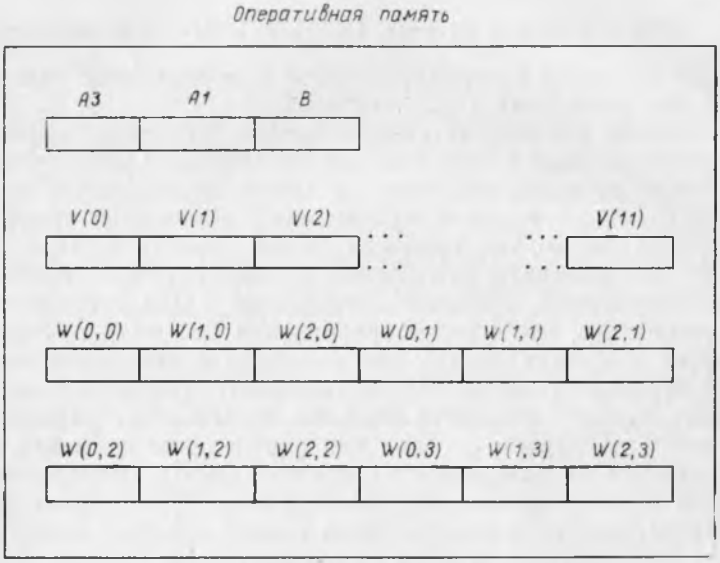


Рис. 14. Схематическое представление простых переменных и массивов в оперативной памяти

До сих пор адресация проводилась с использованием имен переменных, не требующих ни их особого описания в программе, ни каких-либо уточнений имен при обращении с их помощью к оперативной памяти. Соответствующие этим условиям переменные называются **простыми переменными.** Такие переменные с именами A3, A1, B показаны на рис. 14.

Место в оперативной памяти для простой переменной интерпретатор Бейсик выделяет при первом указании ее имени в каком-либо операторе программы. Все имена простых переменных должны быть различными, и они никак не связаны между собой. Наличие цифры в имени не имеет никакого порядкового значения. Цифры, так же как и буквы, служат лишь для созда-

ния различий в именах. Они могут использоваться в произвольном порядке, с пропуском любых цифр или вообще могут не употребляться в имени.

Характеристика простой переменной состоит в том, что каждая простая переменная определяет место в оперативной памяти для одновременного хранения только одной величины. Этой величине соответствует свое собственное имя, никак не связанное с именами других переменных. Несколько простых переменных позволяют одновременно хранить несколько величин, но всем этим величинам соответствуют различные имена.

Совершенно другая картина возникает при использовании в программе массивов. Место в памяти для массива выделяется только по специальному указанию, которым является оператор DIM. Это объясняется тем, что в оперативной памяти отводится непрерывная область сразу для всех элементов массива, поэтому необходимо знать, сколько элементов в массиве и как они нумеруются.

Каждый из двух массивов, приведенных на рис. 14, состоит из 12 элементов. Но массив V является одномерным с порядковой нумерацией его элементов от 0 до 11, а массив W — двумерным с нумерацией по двум измерениям. По первому измерению дается нумерация строк от 0 до 2, а по второму — нумерация столбцов от 0 до 3.

Основное отличие массива от простой переменной состоит в том, что массив позволяет одновременно хранить в оперативной памяти сразу несколько величин — все его элементы. При этом все элементы массива имеют одно и то же имя, определенное в операторе DIM. Все элементы массива упорядочены по номерам и по этим номерам они различаются между собой. Следовательно, для обращения к элементу массива мало указать имя элемента. Требуется еще уточнить это имя с помощью номеров. Номера, используемые для уточнения имени элемента массива, называются **индексами**, а имя элемента вместе с уточняющими его индексами называется **переменной с индексами.**

При обращении к элементу массива индексы указываются в круглых скобках вслед за его именем. Количество индексов соответствует размерности массива, указанной в операторе DIM: при одномерных массивах требуется один индекс, при двумерных — два. В последнем случае индексы разделяются запятой. Отсутствие индексов равнозначно указанию на начальный элемент массива с нулевыми индексами. Так, V равнозначно V(0), а W равнозначно W(0, 0). Значения индексов должны находиться в диапазоне от 0 до наибольшего номера, указанного в операторе DIM по данному измерению.

Индексы могут представляться константами, именами простых переменных и арифметическими выражениями.

Например:

V(2), V(I), V(I+1).

Использование константы в качестве индекса означает обращение к элементу массива с жестко зафиксированным номером. Например, V(2) означает обращение к третьему элементу массива (нумерация начинается с нуля).

Использование имени переменной предоставляет большую гибкость в выборе конкретного числового значения индекса. Это значение должно быть присвоено индексу до обращения к переменной с этим индексом. В данном примере сначала надо присвоить конкретное числовое значение индексу I, а затем уже обращаться к переменной V(I). Индексу I можно присвоить любое значение из допустимого диапазона изменения индекса, указанного в операторе DIM для массива V (в данном примере от 0 до 11). Следовательно, форма V(I), использующая имя переменной, допускает обращение к любому элементу массива. Здесь проявляется основное преимущество использования массивов в программе — возможность обращения к различным местам оперативной памяти при одной и той же для всех мест форме обращения.

Использование арифметического выражения в качестве индекса удобно при обращении к элементам массива с определенным взаимным расположением. Например, оператор

LET A=V(I)+V(I+1)

получает сумму двух соседних элементов массива.

Все имена массивов должны отличаться между собой и от имен простых переменных. Обращение к переменной с индексами должно производиться только после описания соответствующего массива в операторе DIM. Поэтому рекомендуется располагать все операторы DIM в начале программы.

Напоминание.

Отсутствие индексов у переменной с индексами (описанной в операторе DIM) равносильно указанию нулевых индексов, определяющих начальный элемент массива. Забывчивость начинающих программистов в простановке индексов приводит к тому, что вместо перебора различных элементов массива они работают только с одним его начальным элементом.

2. ТИПОВЫЕ ПРОГРАММЫ ОБРАБОТКИ ДАННЫХ ОДНОМЕРНОГО МАССИВА

Использование массивов позволяет одновременно размещать в памяти большое количество величин и обращаться к элементам одного массива по одному имени. Это приводит к резкому сокращению в программе числа необходимых операторов в сравнении с программой решения той же задачи, использующей простые переменные. Количество элементов в массиве не влияет на число операторов в программе, оно остается неизменным. Изменяется лишь количество выполнений этих операторов ма-

шиной в цикле. При использовании же простых переменных увеличение их числа обычно приводит к росту числа операторов в программе. Начинающим программистам рекомендуется убедиться в этом, составив программы с использованием простых переменных вместо приводимых ниже программ с массивами.

При работе с массивами используются определенные типовые приемы. Можно было бы провести классификацию программ только по этим приемам. Но с другой стороны, каждый массив представляет собой набор чисел с определенным смысловым содержанием. Так, одномерный массив может представлять собой строку экономической таблицы, определяющую значения какого-либо показателя в различные моменты времени (например, по месяцам или по годам). И для экономиста оказывается важной классификация программ по содержанию решаемых задач (например, задача определения суммарного или среднеарифметического значения показателя, минимального или максимального значения показателя и т. д.). Поэтому представляется целесообразным отмечать как приемы программирования, так и содержание решаемой задачи.

При составлении программ всегда встает вопрос о степени их универсальности. В программах с массивами этот вопрос сводится к способу задания числа используемых элементов массива (или значения наибольшего номера элемента, который на единицу меньше этого числа). В программах, рассчитанных на частный случай, число элементов в массиве считается постоянным и задается константой во всех операторах, использующих сведения о числе элементов (например, в операторе цикла FOR). Использование константы сокращает программу, но требует изменения всех операторов, в которых она употребляется, в случае изменения числа элементов в массиве. В длинной программе какой-нибудь из этих подлежащих изменению операторов можно и не заметить. Такой ошибочный оператор приведет к неверному результату вычислений, что тоже трудно обнаружить, если результат будет казаться правдоподобным.

В программах, рассчитанных на общий случай, число элементов в массиве считается переменным (но не превосходящим указанного в DIM значения). Оно задается в виде имени простой переменной, значение которой определяется в начале программы (например, считывается оператором READ из оператора DATA). В составленных ниже программах используется именно этот способ. Это несколько удлиняет программу, но делает ее независимой от числа элементов массива. Начинающим программистам рекомендуется переделать некоторые из этих программ на частный случай задания константой постоянного числа элементов в массиве.

При работе с массивами широко используется прием последовательного перебора всех элементов массива.

Задача 1. Ввод и вывод данных одномерного массива и вы-

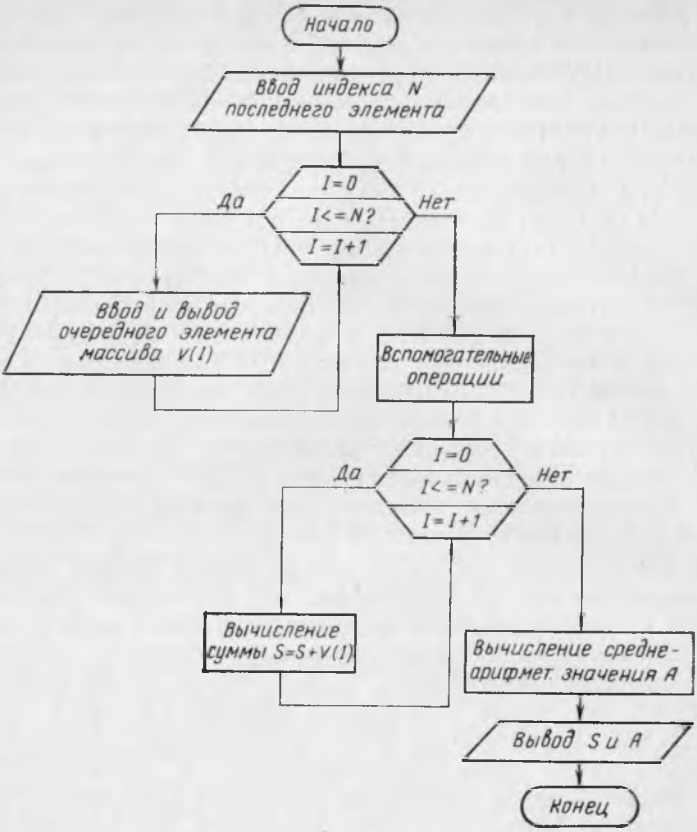


Рис. 15. Блок-схема для ввода и вывода данных одномерного массива и вычисления суммы и среднеарифметического значения элементов массива

числение суммы и среднеарифметического значения элементов массива.

Постановка задачи состоит в том, что в оперативной памяти машины выделяется место для одномерного массива, рассчитанного на некоторое максимально возможное в задаче число элементов (в данном случае 12). Затем данные некоторого конкретного варианта вводятся в качестве значений элементов массива (в данном случае вводятся 10 цифр от 0 до 9 в произвольном порядке). Введенные данные требуется вывести на экран в порядке их ввода (например, для целей контроля за правильностью набора цифр). После этого требуется вычислить сумму элементов массива, а затем их среднеарифметическое значение, равное частному от деления суммы на число элементов массива. Результаты вычислений необходимо вывести на экран.

Блок-схема решения этой задачи представлена на рис. 15. В соответствии с этой блок-схемой составлена программа 11.

Программа 11 и ее выполнение

```
10  REM Ввод, вывод, сумма, среднеарифм. значение
20  DIM V(11)
30  DATA 9, 7, 4, 1, 0, 8, 5, 2, 9, 6, 3
40  READ N
50  FOR I= TO N
60  READ V(I)
70  PRINT V(I);
80  NEXT I
90  PRINT
100 LET S=0
110 FOR I=0 TO N
120 LET S=S+V(I)
130 NEXT I
140 LET A=S/ (N+1)
150 PRINT «Сумма=»S, «Среднеарифм.=»A
RUN
 7   4   1   0   8   5   2   9   6   3

Сумма=  45        Среднеарифм.=  4.5
```

Оператор DIM резервирует в оперативной памяти место для 12 последовательно расположенных элементов массива V с номерами от 0 до 11.

Для составления программы удобнее задавать не число вводимых элементов массива, а индекс N последнего элемента. Этот индекс на единицу меньше числа элементов, так как нумерация элементов в Бейсике начинается с нуля. Поэтому в операторе DATA сначала указан этот индекс N, равный 9, а затем — значения 10 цифр от 0 до 9 в произвольном порядке.

Оператор 40 производит считывание значения N. Оно в дальнейшем используется в качестве граничного значения управляющей переменной при организации цикла, а также для вычисления среднеарифметического значения.

Вводимые данные должны стать значениями элементов массива с индексами от 0 до N. Следовательно, индекс элемента должен представляться именем простой переменной, значения которой будут последовательно изменяться от 0 до N. Это изменение индекса удобнее всего получить, сделав его управляющей переменной цикла с операторами FOR и NEXT. Действительно, оператор 50 последовательно устанавливает значения I, равные 0, 1, 2, ..., N. При этом оператор 60 последовательно вводит данные в элементы V(0), V(1), V(2), ..., V(N), соответственно. Попутно с вводом данных осуществляется их вывод.

После окончания цикла ввода-вывода оператор 90 выполняет вспомогательную операцию, «закрывая» строку вывода оператора 70, что обеспечивает дальнейший вывод информации с

новой строки. Оператор 100 выполняет вторую вспомогательную операцию — обнуление переменной, в которой получается результат суммирования. Суммирование элементов массива осуществляется в данной программе характерным для программирования приемом накопления. Он состоит в последовательном прибавлении очередного элемента суммирования к значению одной и той же переменной величины. Другими словами, этот прием можно описать так: новое значение переменной, в которой получается сумма, равно ее старому значению плюс значение очередного суммируемого элемента. Для правильного хода этого процесса начальное значение переменной, в которой получается сумма, должно равняться нулю.

В Бейсике ДВК-1 нет необходимости в предварительном обнулении переменной, в которой получается сумма, если эта переменная ранее не использовалась в программе. Но на других машинах и в других языках программирования это требование может быть обязательным. Программист должен выработать навык не использовать переменные до присвоения им значений. Даже на ДВК-1 попытка вывести на экран элементы массива, которым еще не присвоены значения, приводит к длительной работе машины и к выводу в экспоненциальной форме очень маленьких чисел.

Цикл суммирования, состоящий из операторов 110—130, совпадает с циклом ввода-вывода из операторов 50—80. В обоих циклах задан один и тот же закон изменения управляющей переменной цикла. Эти циклы записаны раздельно только для выделения решаемых в них задач по смысловому содержанию, однако их можно объединить в один. При этом программа станет на 2 оператора короче, так как исключатся вторые экземпляры операторов заголовка и конца цикла, т. е. операторы 110 и 130. При этом оператор 120 включается в состав операторов тела первого цикла и должен получить номер строки между 50 и 80. Оператор 100 должен выполняться один раз до начала цикла и потому получить номер строки меньше 50. Операторы 140 и 150 могут сохранить свои номера, так как разрывы в нумерации не имеют никакого значения. Начинающим рекомендуется провести эти изменения в программе с проверкой на машине.

В знаменателе среднеарифметического выражения используется сумма N+1. Это связано с тем, что число элементов массива на единицу больше индекса последнего элемента массива, так как нумерация элементов в Бейсике начинается не с единицы, а с нуля. Начинающим следует постоянно помнить об этом, в том числе и при организации циклов, чтобы по ошибке не исключить из обработки элементы массива с нулевыми индексами.

На примере этой программы видны преимущества использования массивов. Ввод всех элементов массива осуществляется в цикле одним оператором 60, вывод — оператором 70, а суммирование — оператором 120. Ясно, что увеличение числа элемен-

тов в массиве никак не скажется на программе (возможно лишь изменение индекса последнего элемента массива в операторе DIM), потому что вырастет только количество циклов, выполняемых машиной, т. е. объем работы машины. Основной принцип программирования состоит в том, что программист описывает обработку лишь одного элемента массива, а затем дает указание применить ее ко всем другим элементам, требующим той же обработки.

Задача 2. Ввод и вывод данных одномерного массива с вычислением сумм отдельно для элементов массива с четными и нечетными индексами.

Эта задача является модификацией задачи 1. Она показывает, что, меняя шаг в операторе цикла FOR, можно осуществлять перебор элементов массива по любому закону. Новая программа 12 получается из программы 11 заменой в последней операторов 10, 100—150 на операторы, представленные ниже.

Операторы, превращающие программу 11 в программу 12

```
10  REM Ввод, вывод, сумма элементов с четными
11  REM и нечетными индексами
100 LET S0=0
110 LET S1=0
120 FOR I=0 TO N STEP 2
130 LET S0=S0+V(I)
140 IF I<N THEN LET S1=S1+V(I+1)
150 NEXT I
160 PRINT «Сумма чет.=» S0, «Сумма нечет.=» S1
```

В программе 12 получаются две суммы и потому введены две переменные S0 и S1. В операторе 120 управляющая переменная цикла I изменяется по четным значениям 0, 2, 4, ... В соответствии с этим в операторе 130 суммируются элементы массива с четными индексами, а в операторе 140 — с нечетными индексами (1, 3, 5, ...). Использование в строке 140 оператора IF обязательно в случае четного значения N. При этом предотвращается попытка суммирования элемента с индексом I+1 бóльшим, чем N. Индекс больше, чем N, быть не может, так как по условию N является номером последнего элемента массива. Этот пример показывает, что при организации циклов надо внимательно следить за значениями используемых индексов.

Задача 3. Определение минимального и максимального значений элементов одномерного массива и значений индексов этих экстремальных элементов.

Блок-схема решения этой задачи представлена на рис. 16. Данной блок-схеме соответствует программа 13.

Программа 13 и ее выполнение

```
10  REM Мин. и макс. и их индексы в одномерном массиве
20  DIM V(11)
```

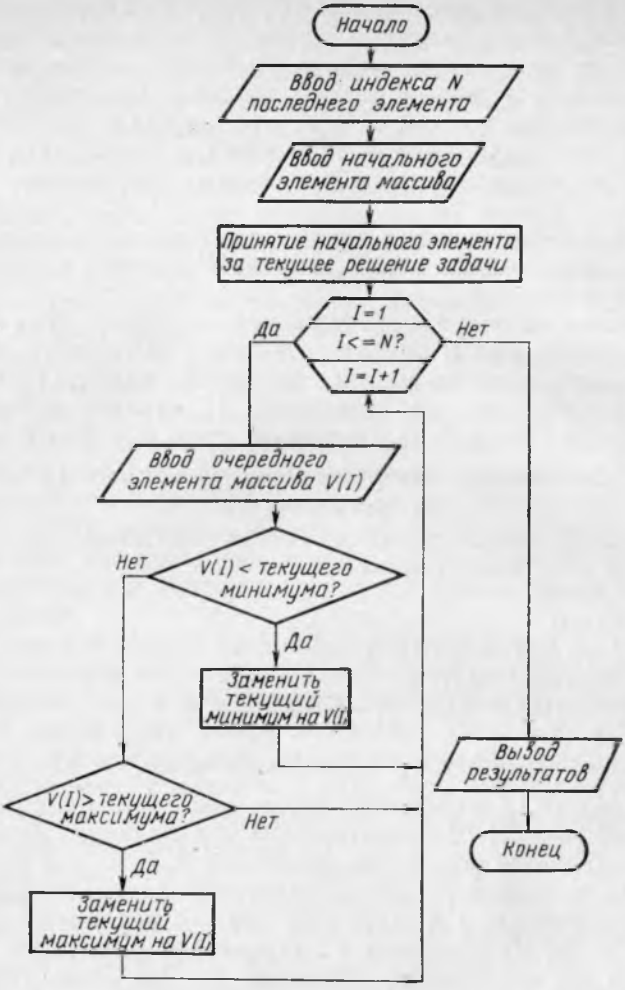


Рис. 16. Блок-схема для определения минимального и максимального значений элементов одномерного массива и индексов экстремальных элементов

```
30  DATA 9, 7, 4, 1, 0, 8, 5, 2, 9, 6, 3
40  READ N
50  READ V(0)
60  LET MI=V(0)
70  LET I1=0
80  LET M2=V(0)
90  LET I2=0
100 FOR I=1 TO N
110 READ V(I)
```

```
120 IF V(I)>=M1 GO TO 160
130 LET M1=V(I)
140 LET I1=I
150 GO TO 190
160 IF V(I)<=M2 GO TO 190
170 LET M2=V(I)
180 LET I2=I
190 NEXT I
200 PRINT «Мин.=»M1; «Инд.=»I1, «Макс.=»M2; «Инд.=»I2
RUN

Мин.= 0     Инд.= 3     Макс.= 9     Инд.= 7
```

Программа 13 имеет много общего с программой 11. Здесь тоже осуществляется ввод данных одномерного массива, заданного с запасом, и производится последовательный перебор всех элементов массива. Однако ввод начального элемента массива производится отдельно от ввода остальных элементов. Кроме того, в программе 13 используются два оператора IF, которые создают разветвления.

Оператор 50 производит ввод начального элемента массива. Значение этого элемента принимается в качестве первоначального решения задачи, т. е. за значение минимума, обозначенное M1, и за значение максимума, обозначенное M2. Соответственно индекс начального элемента принимается за значение индекса для минимума, обозначенное I1, и за значение индекса для максимума, обозначенное I2.

Операторы 100 и 190 организуют обработку в цикле остальных элементов массива. Оператор 110 считывает значение очередного элемента массива. Если это значение претендует на минимум (т. е. оказывается меньше значения M1), то оно запоминается в качестве нового значения M1, а индекс этого элемента — в качестве нового значения I1. Проверять это значение на максимум нет смысла. Управление передается оператору 190 конца цикла для организации ввода значения следующего элемента массива.

Если же значение элемента массива не претендует на минимум, то оператор 120 передает управление оператору 160 для проверки этого значения на максимум. В случае, когда значение претендует на максимум (т. е. оказывается больше значения M2), оно запоминается в качестве нового значения M2, а индекс этого элемента — в качестве нового значения I2. После этого выполняется оператор 190 конца цикла для организации ввода значения следующего элемента массива. Управление оператору 190 передается также оператором 160 в случае, когда элемент массива не претендует на максимум.

Таким образом, значение M1 соответствует текущему минимальному значению, а значение M2 — текущему максимальному значению. Значения I1 и I2 представляют текущие значения индексов минимального и максимального элементов массива со-

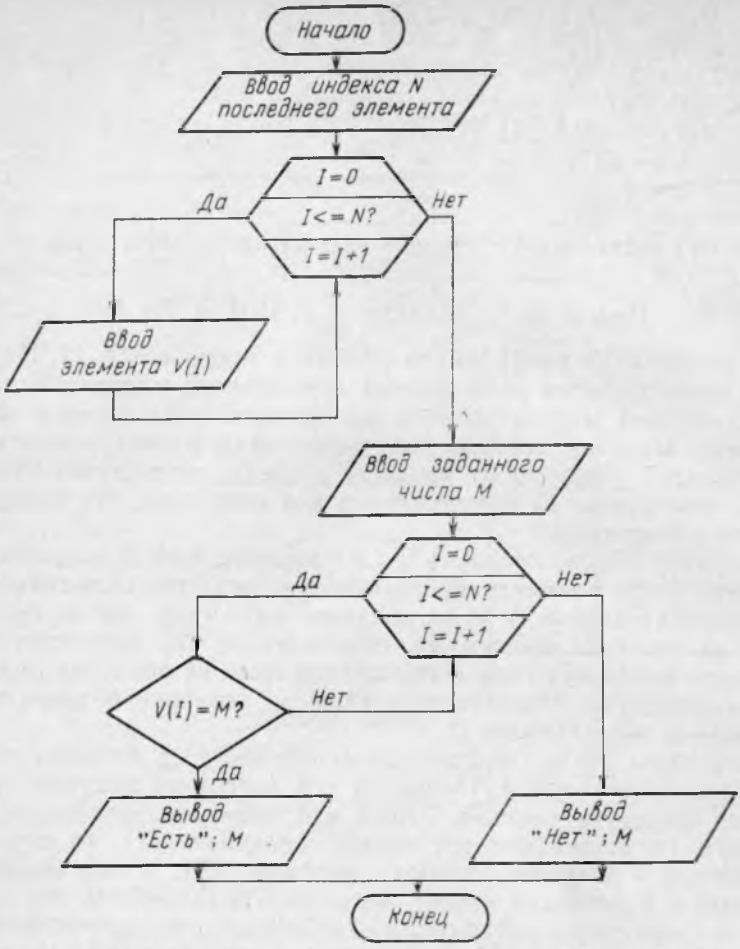


Рис. 17. Блок-схема для поиска заданного элемента в одномерном массиве

ответственно. После обработки всех данных текущие экстремальные значения становятся окончательным решением задачи. Результаты решения выводятся на экран оператором 200.

Следует отметить, что в массиве может быть несколько элементов с минимальными и максимальными значениями. Так, в данном примере среди исходных значений оператора DATA может быть несколько нулей и несколько девяток. Программа 13 обнаруживает лишь первые из минимальных и максимальных элементов, так как замена старых экстремальных значений на новые в случае их равенства не производится.

Читателям рекомендуется самостоятельно составить программу, выводящую индексы всех минимальных и всех максимальных элементов массива с подсчетом их соответствующего чи-

сла. Следует учитывать, что суждение об экстремальности элемента можно составить лишь после обработки всех элементов массива. А до этого момента значения индексов элементов, претендующих на экстремум, необходимо хранить в двух дополнительных массивах. При появлении нового экстремального значения требуется удалять все значения индексов, относящиеся к старому экстремальному значению.

Задача 4. Поиск заданного элемента в одномерном массиве.

Постановка задачи состоит в том, чтобы определить, находится ли среди значений элементов массива некоторое заданное число M. При положительном ответе требуется вывести на экран слово «Есть» и значение M, при отрицательном — слово «Нет» и значение M.

Блок-схема решения задачи представлена на рис. 17. Данной блок-схеме соответствует программа 14.

Программа 14 и ее выполнение

```
10 REM Поиск заданного элемента
20 DIM V(11)
30 DATA 9, 7, 4, 1, 0, 8, 5, 2, 9, 6, 3
40 READ N
50 FOR I=0 TO N
60 READ V(I)
70 NEXT I
80 PRINT «M=»;
90 INPUT M
100 FOR I=0 TO N
110 IF V(I)=M GO TO 150
120  NEXT I
130 PRINT «Нет » M
140 STOP
150 PRINT «Есть » M
RUN
M=?15
Нет   15
ОСТ СТРОКЕ 140
ЖДУ
RUN
M=?7
Есть   7
ОСТ СТРОКЕ 150
ЖДУ
```

Ввод данных массива производится так же, как в программе 11. Ввод заданного значения M в данной программе удобно осуществлять с помощью оператора INPUT, чтобы иметь возможность оперативно задавать различные значения. Оператор 80 служит подсказкой пользователю о необходимости ввода значения M. Использование в программе оператора INPUT и

комментирующего его оператора PRINT описано ранее в п. 5 гл. 4 и в пояснениях к программе 3.

Если очередной элемент массива равен M, то дальнейший поиск прерывается. Управление передается оператору 150, который выводит на экран сообщение «Есть» и значение M. Выполнение программы заканчивается.

Если очередной элемент массива не равен M, то это не дает оснований для отрицательного ответа. Требуется провести проверку остальных элементов массива. Если же среди всех элементов массива не оказалось элемента, равного M, то по завершении цикла проверки управление передается оператору 130, который выводит на экран сообщение «Нет» и значение M. После этого требуется прекратить дальнейшее выполнение программы. Для этого служит оператор 140. При отсутствии оператора 140 после оператора 130 выполнился бы оператор 150. Это привело бы к появлению на экране двух противоречивых сообщений.

Задача 5. Определение количества элементов одномерного массива со значениями в заданном диапазоне.

Задача 5 является модификацией задачи 4. Задание диапазона вместо одного значения требует проверки двух условий: значение элемента массива должно быть не меньше значения левой границы диапазона и не больше значения правой его границы. В задаче предполагается, что граничные значения включаются в диапазон.

Далее вместо ответов «Есть» и значения M или «Нет» и значения M требуется подсчитать количество элементов, попадающих в диапазон. Подсчет в программировании осуществляется следующим приемом. В оперативной памяти выделяется место, выполняющее функции счетчика. Предварительно в счетчик заносится нулевое значение. Затем при каждом появлении подсчитываемого объекта в счетчик добавляется единица. От рассмотренного в задаче 1 и программе 11 приема суммирования этот прием отличается лишь добавлением единицы вместо суммируемого значения.

Блок-схема решения задачи представлена на рис. 18. Данной блок-схеме соответствует программа 15.

Программа 15 и ее выполнение

```
10 REM Кол. элементов в заданном диапазоне
20 DIM V(11)
30 DATA 9, 7, 4, 1, 0, 8, 5, 2, 9, 6, 3
40 READ N
50 FOR I=0 TO N
60 READ V(I)
70 NEXT I
80 PRINT «L, P=»;
90 INPUT L, P
100 LET K=0
```

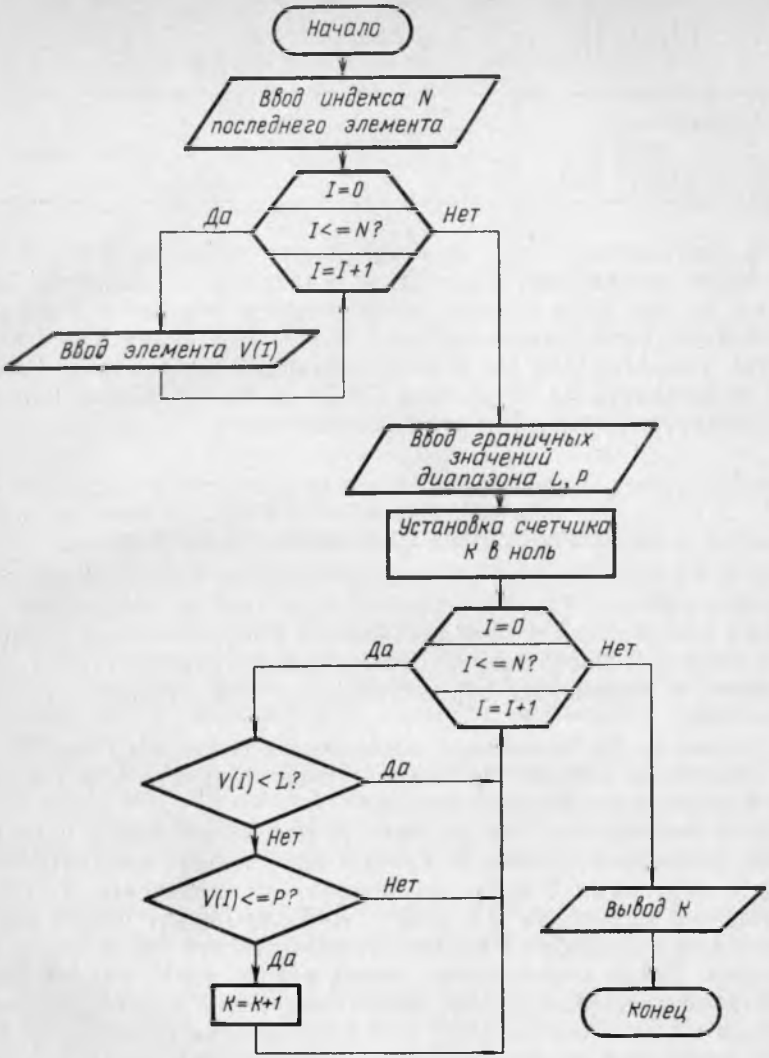


Рис. 18. Блок-схема для определения количества элементов одномерного массива со значениями в заданном диапазоне

```
110 FOR I=0 TO N
120 IF V(I)<L GO TO 140
130 IF V(I)<=P THEN LET K=K+1
140 NEXT I
150 PRINT K
RUN
L, P=?4, 6
```

```
 3
ОСТ СТРОКЕ 150
ЖДУ
RUN
L, P=?0, 9
 10
ОСТ СТРОКЕ 150
ЖДУ
```

В программах 14 и 15 используется оператор INPUT. Он позволяет оперативно задавать с клавиатуры различные значения, но при этом выводимые результаты теряются в строках служебных сообщений машины. Для устранения этого недостатка оператор INPUT с сопутствующим оператором PRINT можно заменить на операторы DATA и READ. Пользователям рекомендуется проделать это самостоятельно.

3. ТИПОВЫЕ ПРОГРАММЫ ПРЕОБРАЗОВАНИЯ ОДНОМЕРНОГО МАССИВА

В п. 2 производились различные операции над данными одномерного массива. Однако расположение данных в массиве оставалось неизменным и определялось их расположением в операторе DATA. В настоящем пункте рассматриваются задачи, приводящие к изменению первоначального расположения данных в массиве.

Задача 6. Расположение элементов одномерного массива в неубывающем порядке (в возрастающем порядке при отсутствии в массиве одинаковых чисел).

Для решения задачи рассматриваются все пары соседних чисел, начиная с первой. В первую пару входят элементы массива с индексами 0 и 1, во вторую — с индексами 1 и 2, в третью — с индексами 2 и 3 и т. д. Если первое число какой-либо пары оказывается больше второго, то эти числа меняются местами. После перестановки новое второе число данной пары рассматривается как первое число следующей пары. Сравнение чисел в парах продолжается для всех остальных пар.

В результате первого прохода по всем парам самое большое число будет обязательно поставлено в конец массива. После этого процесс повторяется сначала, с первой пары. Но при втором проходе процесс заканчивается сравнением с предпоследним элементом преобразованного массива, так как последний элемент уже выставлен при первом проходе. При втором проходе будет выставлен уже и предпоследний элемент. Аналогично при третьем проходе процесс заканчивается сравнением с предпредпоследним элементом и т. д. При последнем проходе процесс заканчивается сравнением со вторым элементом (с индексом 1). В результате все числа в массиве оказываются упорядоченными.

Описанный процесс поясняется с помощью табл. 6.

Перестановка чисел в массиве в общем случае

Номер прохода	Расположение чисел в массиве	Количество перестановок за проход
	4 3 2 1	
1	3 4 2	1
	2 4 1	2
	1 4	3
2	2 3 1	1
	1 3	2
3	1 2	1

В верхней строке таблицы приведено исходное расположение чисел в массиве.

При первом проходе сначала переставляются числа в первой паре. После этого устанавливается, что числа во вновь полученной второй паре также стоят неправильно. Если последующее число оказывается меньше предыдущего, то запись чисел в строке табл. 6 обрывается и продолжается на следующей строке с перестановки этого числа. В результате первого прохода самое большое число 4 выставляется на место последнего элемента массива.

При втором проходе снова сравниваются числа, начиная с первой пары, но последний элемент массива уже не рассматривается. При втором проходе на место предпоследнего элемента массива выставляется следующее максимальное число из оставшихся.

При третьем проходе сравнивается и переставляется первое число со вторым. На этом процесс заканчивается. Числа в массиве упорядочены.

В табл. 6 показан общий случай процесса упорядочения чисел в массиве. При этом процесс заканчивается сравнением чисел в одной оставшейся первой паре.

В частных случаях процесс упорядочения может закончиться значительно раньше. Например, числа сразу могут стоять правильно или могут быстро выставиться правильно. В этих случаях нет необходимости доводить процесс до конца, т. е. до сравнения чисел в одной оставшейся первой паре. Вывод о том, что числа уже выставлены правильно, можно сделать по отсутствию перестановок при очередном проходе. Этот случай поясняется с помощью табл. 7.

Как следует из табл. 7, при втором проходе не сделано ни одной перестановки. Это означает, что числа выставились правильно уже при предыдущем проходе и процесс упорядочения

Таблица 7

Перестановка чисел в массиве в частном случае

Номер прохода	Расположение чисел в массиве	Количество перестановок за проход
	4 1 2 3	
1	1 4 2	1
	2 4 3	2
	3 4	3
2	1 2 3	0

следует прекратить. Такая возможность окончания процесса используется в рассматриваемой ниже программе.

Блок-схема, соответствующая описанному алгоритму, представлена на рис. 19. По данной блок-схеме составлена программа 16.

Программа 16 и ее выполнение

```
10 REM Расположение элементов одномерного массива
11 REM в неубывающем порядке
20 DIM V (11)
30 DATA 9, 7, 4, 1, 0, 8, 5, 2, 9, 6, 3
40 READ N
50 FOR I=0 TO N
60 READ V(I)
70 NEXT I
80 LET M=N
90 LET M=M—1
100 LET K=0
110 FOR I=0 TO M
120 IF V(I)<=V(I+1) GO TO 170
130 LET C=V(I)
140 LET V(I)=V(I+1)
150 LET V(I+1)=C
160 LET K=K+1
170 NEXT I
180 IF K=0 GO TO 200
190 IF M>0 GO TO 90
200 FOR I=0 TO N
210 PRINT V(I);
220 NEXT I
RUN
 0   1   2   3   4   5   6   7   8   9
```

Ввод данных массива производится так же, как в программе 11. После ввода данных определяется значение вспомогательной переменной M. Эта переменная указывает наибольшее значение индекса левого числа в последней проверяемой паре

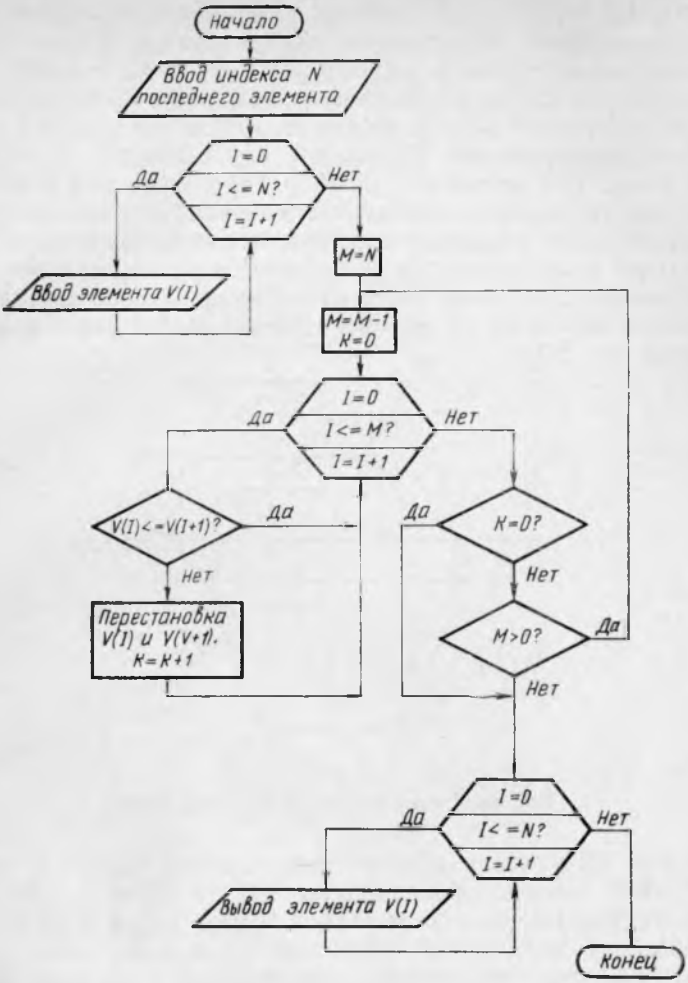


Рис. 19. Блок-схема для расположения элементов массива в неубывающем порядке

чисел при каждом проходе. Это значение последовательно уменьшается от N—1 до нуля. В программе это достигается с помощью операторов 80 и 90. Оператор 80 предварительно устанавливает значение M, равное N. Затем оператор 90 в цикле (который организуется оператором 190) последовательно уменьшает значение M на единицу.

Оператор 100 восстанавливает нулевое значение счетчика количества перестановок K перед каждым новым проходом. Если во время прохода производятся перестановки, то при каждой перестановке оператор 160 добавляет единицу к значению счетчика K. Если за время прохода перестановок не было, то

оператор 180 прекращает процесс упорядочения массива, передавая управление оператору 200, организующему цикл вывода значений упорядоченного массива на экран. Если же перестановки были, то оператор 190 проверяет, не закончился ли процесс упорядочения перестановкой в одной оставшейся первой паре (см. пример в табл. 6).

Оператор 120 проверяет числа в паре. Если они стоят правильно, то управление передается оператору конца цикла 170 для организации перехода к проверке следующей пары. Если числа стоят неправильно, то организуется их перестановка. Для перестановки двух чисел одно из этих чисел необходимо предварительно переслать во вспомогательное место памяти, как показано на рис. 20.

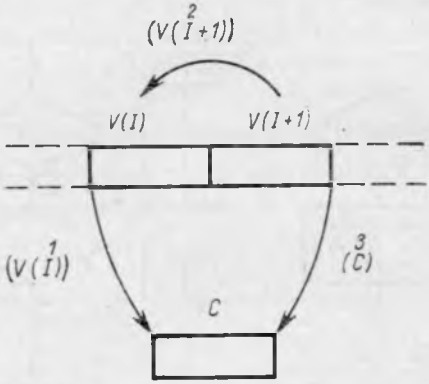


Рис. 20. Схема перестановки двух чисел

На рис. 20 показан процесс перестановки чисел в местах оперативной памяти, обозначенных V(I) и V(I+1). Чтобы отличать на рисунке имя переменной (имя места в памяти) от значения этой переменной (хранящегося в этом месте памяти числа), значение переменной заключается в круглые скобки. Так, V(I) означает имя переменной, а (V(I)) — значение этой переменной.

При перестановке одно из переставляемых значений необходимо предварительно переслать для сохранения в какое-либо вспомогательное место памяти. На рисунке сначала значение (V(I)) пересылается в место памяти для переменной C. После этого переменная V(I) считается свободной, и туда можно переслать значение (V(I+1)). Наконец, переменная V(I+1) становится свободной, и туда можно переслать значение (C), равное прежнему значению (V(I)). Попытка сразу переслать значение (V(I+1)) в место для переменной V(I) приведет к потере значения (V(I)). Указанный прием перестановки довольно широко используется в программировании.

Расположение элементов одномерного массива в невозра-

стающем порядке (в убывающем порядке при отсутствии в массиве одинаковых чисел) производится по программе, аналогичной программе 16. Только в этом случае в операторе 120 условие «меньше или равно» заменяется на условие «больше или равно».

Задача 7. Перестановка элементов одномерного массива в обратном порядке.

Требуется первый элемент массива поставить на место последнего, второй — на место предпоследнего и т. д., а послед-

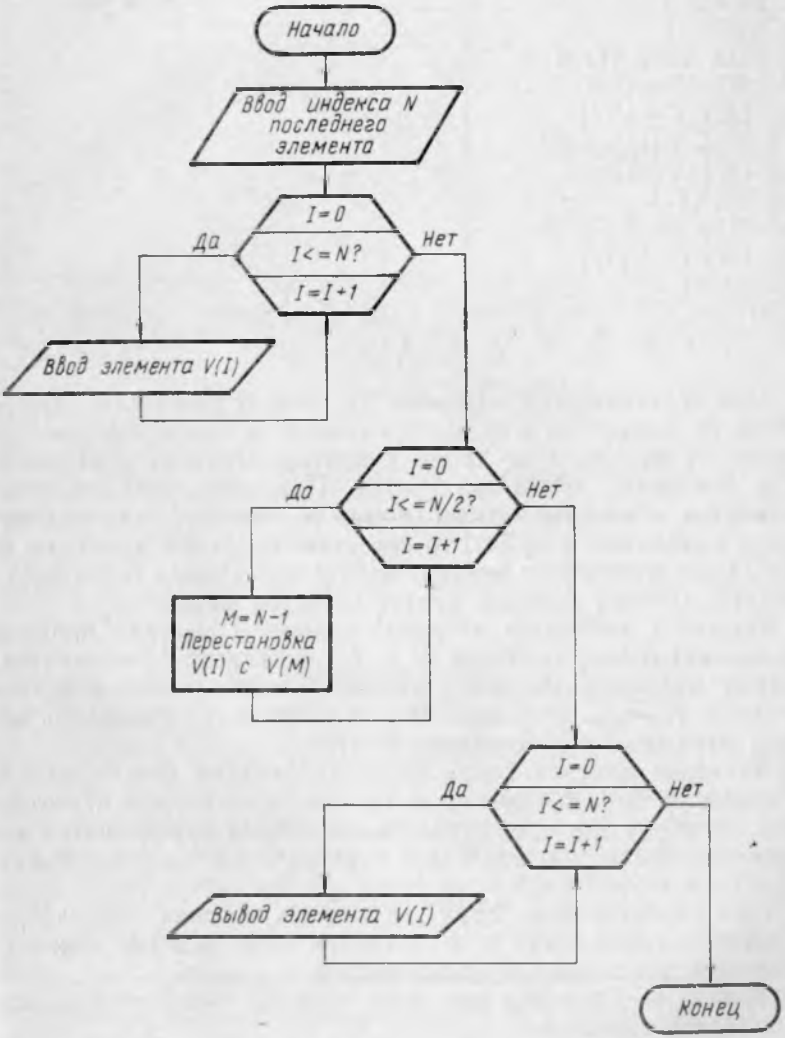


Рис. 21. Блок-схема для перестановки элементов одномерного массива в обратном порядке

ний — на место первого. Блок-схема решения задачи представлена на рис. 21. Ей соответствует программа 17.

Программа 17 и ее выполнение

```
10 REM Перестановка в обратном порядке
20 DIM V(11)
30 DATA 9, 7, 4, 1, 0, 8, 5, 2, 9, 6, 3
40 READ N
50 FOR I=0 TO N
60 READ V(I)
70 NEXT I
80 FOR I=0 TO N/2
90 LET M=N—I
100 LET C=V(I)
110 LET V(I)=V(M)
120 LET V(M)=C
130 NEXT I
140 FOR I=0 TO N
150 PRINT V(I);
160 NEXT I
RUN
 3   6   9   2   5   8   0   1   4   7
```

При составлении программы 17 следует учитывать, что движение по элементам массива производится одновременно с двух сторон от начала и от конца к центру. Поэтому перестановки надо закончить, достигнув центра. При этом, если количество элементов в массиве четное (номер N нечетный, так как нумерация начинается с нуля), то переставляются все элементы массива. Если количество элементов нечетное (номер N четный), то средний элемент массива просто остается на месте.

Индекс I элементов из левой половины массива принимает последовательные значения 0, 1, 2, ... Индекс M элементов из правой половины массива принимает соответственно значения N, N—1, N—2, ... Это означает, что значения индекса M могут быть выражены соотношением M=N—I.

Значение индекса I при перестановках не может выходить за значение N/2. В противном случае после первой перестановки в обратном порядке началась бы вторая перестановка в обратном порядке, приводящая к первоначальному расположению элементов в массиве.

При перестановке двух элементов массива используется вспомогательное место в оперативной памяти. Этот прием рассмотрен в пояснениях к программе 16.

Задача 8. Перестановка всех нулевых элементов в конец одномерного массива.

Блок-схема решения этой задачи представлена на рис. 22. Ей соответствует программа 18.

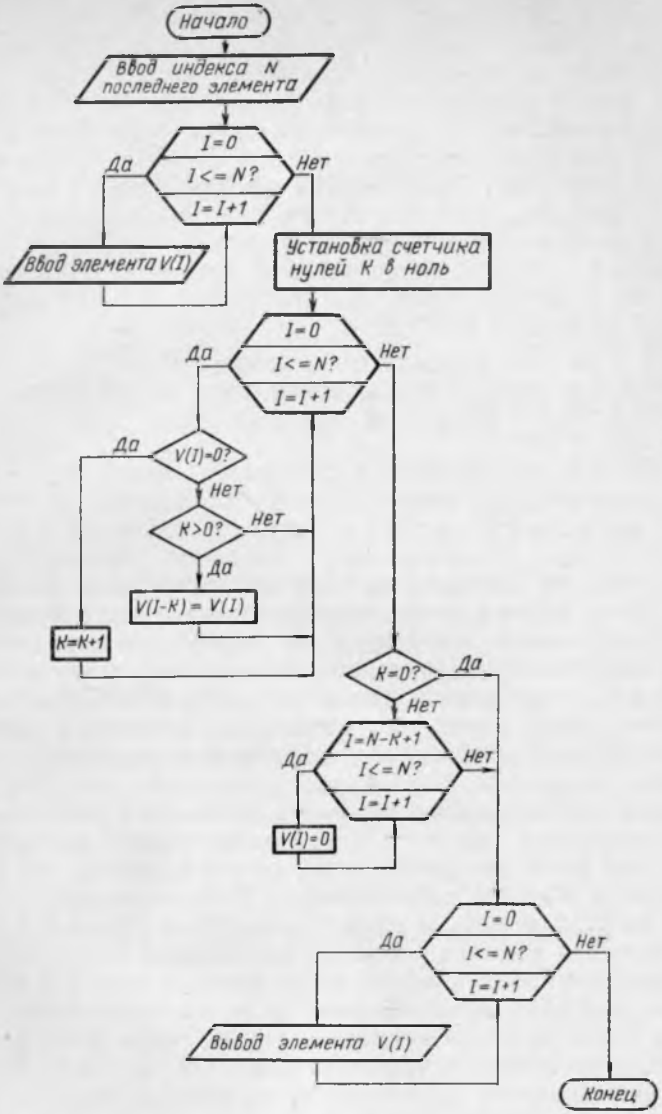


Рис. 22. Блок-схема для перестановки нулевых элементов в конец одномерного массива

Программа 18 и ее выполнение

```
10 REM Перестановка нулевых элементов в конец
20 DIM V(11)
30 DATA 9, 7, 4, 1, 0, 8, 5, 2, 9, 6, 3
40 READ N
```

```
50 FOR I=0 TO N
60 READ V(I)
70 NEXT I
80 LET K=0
90 FOR I=0 TO N
100 IF V(I)=0 GO TO 130
110 IF K>0 THEN LET V(I—K)=V(I)
120 GO TO 140
130 LET K=K+1
140 NEXT I
150 IF K=0 GO TO 190
160 FOR I=N—K+1 TO N
170 LET V(I)=0
180 NEXT I
190 FOR I=0 TO N
200 PRINT V(I);
210 NEXT I
RUN
 7  4  1  8  5  2  9  6  3  0
```

Оператор 80 устанавливает счетчик нулей K в нулевое значение. После этого в цикле обрабатываются все элементы массива. Если элемент равен нулю, то управление передается оператору 130. Этот оператор прибавляет единицу к значению счетчика нулей, подсчитывая таким образом количество нулевых элементов. Затем управление передается оператору конца циклов 140, который обеспечивает обработку следующего элемента массива.

Если элемент массива не равен нулю, то управление передается оператору 110. Этот оператор проверяет значение счетчика нулей. Если оно равно нулю, то это означает, что нулевых элементов в массиве еще не было. Следовательно, в массиве ничего не надо менять, а надо переходить к обработке следующего элемента массива. Это обеспечивается оператором 120, передающим управление оператору конца цикла 140. Если же значение счетчика нулей больше нуля, то это означает, что в массиве были нулевые элементы. На их место надо поставить ненулевые элементы, осуществив сдвиг в массиве. Величина сдвига определяется количеством нулевых элементов. Сдвиг осуществляет оператор 110. После сдвига следует перейти к обработке следующего элемента массива, что осуществляет оператор 120.

После обработки всех элементов массива возможны две ситуации. Одна соответствует отсутствию нулевых элементов в массиве. В этом случае в массиве ничего менять не надо и можно просто переходить к выводу элементов массива на экран.

Другая ситуация соответствует наличию нулевых элементов в массиве. В процессе сдвига они были удалены из своих мест, но их требуется восстановить в конце массива. Если был один

нулевой элемент, то он должен быть поставлен на место последнего элемента с индексом N, если было два нулевых элемента, то они должны быть поставлены на место элементов с индексами N—1 и N, если было K нулевых элементов, то они должны быть поставлены на место элементов с индексами N—(K—1), ..., N. Индекс N—(K—1) можно записать в виде N—K+1. Установку нулей в конце массива обеспечивают операторы 160—180.

Без операторов 160—180 программа 18 осуществляет просто удаление нулей из массива и «сжатие» массива. При этом индекс последнего элемента массива после «сжатия» равен N—K.

4. ТИПОВЫЕ ПРОГРАММЫ ПОЛУЧЕНИЯ ДОПОЛНИТЕЛЬНЫХ ПРОИЗВОДНЫХ ОДНОМЕРНЫХ МАССИВОВ

В п. 3 проводились различные преобразования данных исходного массива. Результаты преобразований представлялись в том же массиве на месте исходных данных. В настоящем пункте исходные массивы остаются неизменными, а результаты преобразований представляются в новых массивах.

Задача 9. Получение процентных соотношений.

Пусть данные исходного массива представляют собой значения какого-либо суммируемого показателя по месяцам года. Таким суммируемым показателем может быть, например, выпуск продукции в денежном выражении. Требуется получить новый массив, элементами которого являются значения этого показателя по месяцам в процентах к суммарному годовому значению.

Блок-схема решения задачи представлена на рис. 23. Ей соответствует программа 19.

Программа 19

```
10 REM Получение массива процентных соотношений
20 DIM V(11), X(11)
30 DATA ...
40 LET S=0
50 FOR I=0 TO 11
60 READ V(I)
70 LET S=S+V(I)
80 NEXT I
90 FOR I=0 TO 11
100 LET X(I)=V(I)/S *100
110 PRINT X(I);
120 NEXT I
```

В данной программе в операторе DIM описаны уже два массива. Массив V является массивом исходных данных, массив X — массивом результатов. Длина этих массивов определена точно, так как в году 12 месяцев. В соответствии с этим в дан-

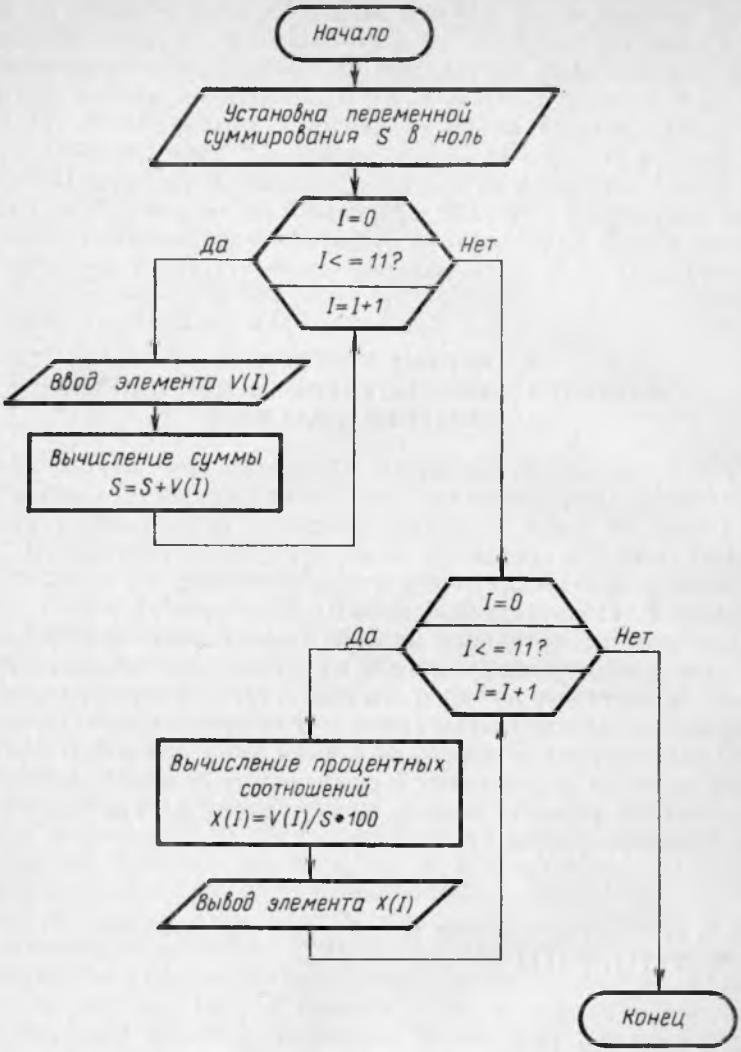


Рис. 23. Блок-схема для получения процентных соотношений

ной программе в отличие от предыдущих исключен ввод индекса последнего элемента массива в виде переменной N. Этот индекс указывается явно константой 11 в операторах цикла FOR 50 и 90. Естественно, что в операторе DATA должны быть указаны 12 конкретных значений показателя для правильной работы указанных циклов.

Задача 10. Разделение одного массива на два массива.

Задача состоит в том, чтобы из исходного массива V получить два новых массива X и Y. Элементами массивов X и Y

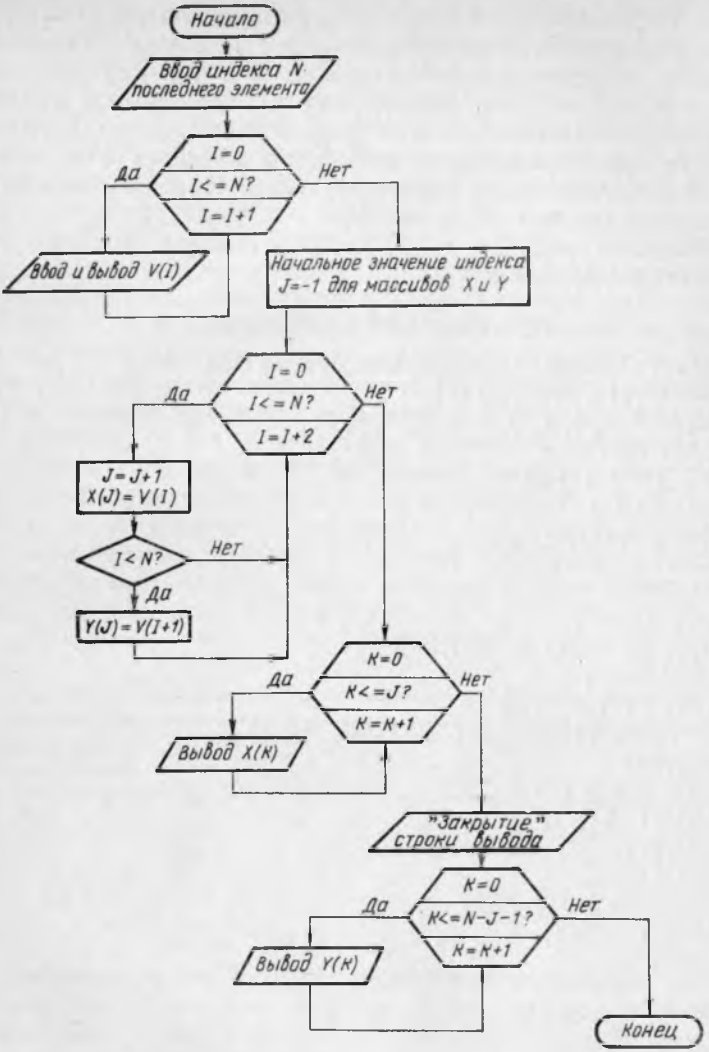


Рис. 24. Блок-схема для разделения одного массива на два массива

являются элементы массива V с четными и нечетными индексами, соответственно.

Эта задача сходна с задачей вычисления сумм отдельно для элементов массива V с четными и нечетными индексами, решение которой обеспечивается программой 12. Однако принципиальное различие этих задач состоит в том, что в программе 12 вычисления производились без образования новых массивов. Впервые новый массив получался в программе 19. Но там оба массива были одной и той же длины и с одним и тем же по-

рядком расположения элементов в массивах. При этом для обращения к элементам обоих массивов использовался один и тот же индекс. В данной задаче массивы X и Y в два раза короче массива V. Кроме того, элементы, расположенные в массиве V через один, в массивах X и Y расположены подряд. Это означает, что при обращении к элементам массивов V и X или Y должны использоваться разные индексы. При этом каждый индекс изменяется по своему закону.

Блок-схема решения задачи представлена на рис. 24. Ей соответствует программа 20.

Программа 20 и ее выполнение

```
10 REM Разделение одного массива на два
20 DIM V(11), X(5), Y(5)
30 DATA 9, 7, 4, 1, 0, 8, 5, 2, 9, 6, 3
40 READ N
50 FOR I=0 TO N
60 READ V(I)
70 PRINT V(I);
80 NEXT I
90 PRINT
100 LET J=—1
110 FOR 1=0 TO N STEP 2
120 LET J=J+1
130 LET X(J)=V(I)
140 IF I<N THEN LET Y(J)=V(I+1)
150 NEXT I
160 FOR K=0 TO J
170 PRINT X(K);
180 NEXT K
190 PRINT
200 FOR K=0 TO N—J—1
210 PRINT Y(K);
220 NEXT K
RUN
 7   4   1   0   8   5   2   9   6   3
 7   1   8   2   6
 4   0   5   9   3
```

Индексом для элементов массива V служит переменная I, а для элементов массивов X и Y (при их образовании) — переменная J. Индекс I в операторе 110 изменяется по четным значениям 0, 2, 4, ... Это позволяет обращаться с помощью I к элементам массива V с четными индексами. Для обращения к элементам с нечетными индексами используется арифметическое выражение I+1, которое изменяется по нечетным значениям 1, 3, 5, ... Индекс J должен принимать последовательные значения 0, 1, 2, ..., так как в массивах X и Y элементы расположены друг за другом.

При каждом изменении индекса I на 2 индекс J меняется на 1. Шаги изменения индексов различны, но индексы изменяются синхронно, одновременно. В этом случае для изменения индекса J нельзя использовать оператор FOR. Вместе с оператором FOR для изменения индекса I он создал бы конструкцию «цикла в цикле», приводящую к асинхронному изменению переменных (см. п. 10 гл. 4 и пояснения к программам 9 и 10). Поэтому изменение индекса J осуществляется его наращиванием с помощью оператора LET с номером 120. Оператор 100 устанавливает начальное значение J, которое позволяет после первой операции наращивания получить требуемое значение J=0.

Оператор 140, получающий массив Y из элементов массива V с нечетными индексами, содержит условие I<N. Это связано с четностью или нечетностью количества элементов в массиве V. От этого зависит равенство или неравенство количества элементов в массивах X и Y. При четном количестве элементов в массиве V (при этом индекс N последнего элемента нечетный, так как нумерация начинается с нуля) в массивах X и Y одинаковое количество элементов, равное (N+1)/2. При нечетном количестве элементов в массиве V (индекс N четный) в массиве X на один элемент больше, чем в массиве Y. Это поясняется на частных случаях N=3 и N=4 в табл. 8.

Таблица 8

Влияние четности и нечетности количества элементов в массиве V

Арифметические выражения	N=3	N=4
	0, 1, 2, 3	0, 1, 2, 3, 4
I	0, 2	0, 2, 4
I+1	1,3	1, 3, 5
J	0,1	0, 1, 2
N—J	2	2

В верхней строке таблицы приводятся возможные значения индексов для элементов массива V при каждом значении N. В следующих строках приводятся значения индекса I, которые определяются оператором 110, и соответствующие значения арифметического выражения I+1. Из табл. 8 видно, что при нечетном количестве элементов в массиве V (при четном N) последнее значение арифметического выражения I+1 оказывается больше N (в данном случае 5 и 4, соответственно). Но это невозможно, так как N по условию является индексом последнего элемента массива. Для устранения вероятности такой ошибки в оператор 140 вводится условие I<N.

При выводе на экран значений элементов массива X используется индекс K. Граничным значением для индекса K является последнее значение индекса J, полученное при образовании массивов X и Y. Как следует из табл. 8, количество зна-

чений J всегда соответствует количеству четных значений индекса I. Это означает, что для массива X последнее значение индекса J является точной верхней границей. Оператор 190 «закрывает» строку вывода элементов массива X, чтобы обеспечить вывод элементов массива Y с новой строки.

При выводе на экран значений элементов массива Y также используется индекс K. Однако граничным значением для индекса K в этом случае является арифметическое выражение N—J—1. Как следует из табл. 8, при нечетном количестве элементов в массиве V (при N четном) количество значений J на единицу больше количества допустимых нечетных значений арифметического выражения I+1. При четном количестве элементов в массиве V (при N нечетном) обе эти величины совпадают друг с другом. Это означает, что для массива Y последнее значение индекса J не является точной верхней границей. Эту границу можно получить из того условия, что количество элементов в массиве Y равно количеству элементов в массиве V минус количество элементов в массиве X. Эта разность равна

$$(N+1)-(J+1)=N+1-J-1=N-J.$$

При переходе от количества элементов к индексу последнего элемента для массива Y, который на единицу меньше количества, получается выражение N — J — 1.

Задача 11. Слияние двух массивов в один массив.

Задача состоит в том, чтобы из исходных массивов X и Y получить новый массив Z. Элементами массива Z являются сначала элементы массива X, а затем элементы массива Y.

Блок-схема решения задачи представлена на рис. 25. Ей соответствует программа 21.

Программа 21 и ее выполнение

```
10 REM Слияние двух массивов в один
20 DIM X(5), Y(5), Z(11)
30 DATA 3, 0, 1, 2, 3
35 DATA 2, 5, 6, 7
40 READ N1
50 FOR I=0 TO N1
60 READ X(I)
70 PRINT X(I);
80 NEXT I
90 PRINT
100 READ N2
110 FOR I=0 TO N2
120 READ Y(I)
130 PRINT Y(I);
140 NEXT I
150 PRINT
160 LET J=-1
170 FOR I=0 TO N1
```

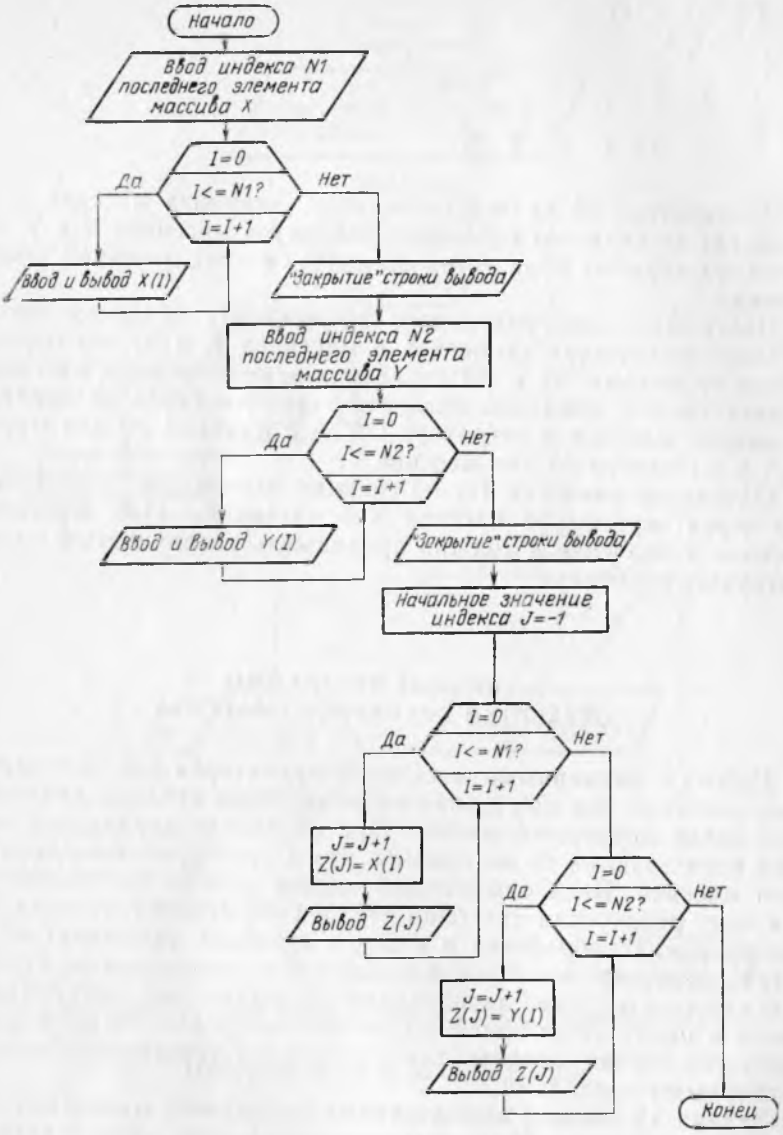


Рис. 25. Блок-схема для слияния двух массивов в один массив

```
180 LET J=J+1
190 LET Z(J)=X(I)
200 PRINT Z(J);
210 NEXT I
220 FOR I=0 TO N2
230 LET J=J+1
240 LET Z(J)=Y(I)
```

```
250 PRINT Z(J);
260 NEXT I
RUN
 0  1  2  3
 5  6  7
 0  1  2  3  4  5  6  7
```

В операторе 20 индекс последнего элемента массива Z не равен сумме индексов последних элементов массивов X и Y. Он равен суммарному количеству элементов в этих массивах, минус единица.

Программа предусматривает возможность задания произвольного количества элементов в массивах X и Y, что определяется индексами N1 и N2 последних элементов этих массивов, соответственно. Значения индексов задаются перед значениями элементов массива в операторе DATA с номером 30 для массива X и с номером 35 для массива Y.

Начальное значение J=—1 должно задаваться только один раз перед пересылкой массива X в массив Z. При пересылке массива Y значение J должно продолжать увеличиваться от достигнутого значения.

5. ТИПОВЫЕ ПРОГРАММЫ ОБРАБОТКИ ДВУМЕРНОГО МАССИВА

Работа с двумерными массивами характерна для экономических расчетов, так как любая экономическая таблица представляет собой двумерный массив. При обработке двумерного массива используются те же приемы, что и при обработке одномерного массива. Ведь одномерный массив можно рассматривать как одну строку или как один столбец двумерного массива. Но одновременная обработка и строк и столбцов усложняет обработку двумерных массивов в сравнении с одномерными. Основную сложность для начинающих представляет организация цикла в цикле. Этот прием подробно описан в п. 10 гл. 4 и его требуется хорошо освоить для обеспечения уверенной работы с двумерными массивами.

Задача 12. Ввод и вывод данных двумерного массива.

В оперативной памяти машины выделяется место для двумерного массива, рассчитанное на некоторое максимально возможное в задаче число строк и столбцов. Затем данные некоторого конкретного варианта вводятся в качестве значений элементов массива. Введенные данные требуется вывести на экран в виде таблицы с учетом их расположения по строкам и столбцам.

Блок-схема решения этой задачи представлена на рис. 26. Ей соответствует программа 22.

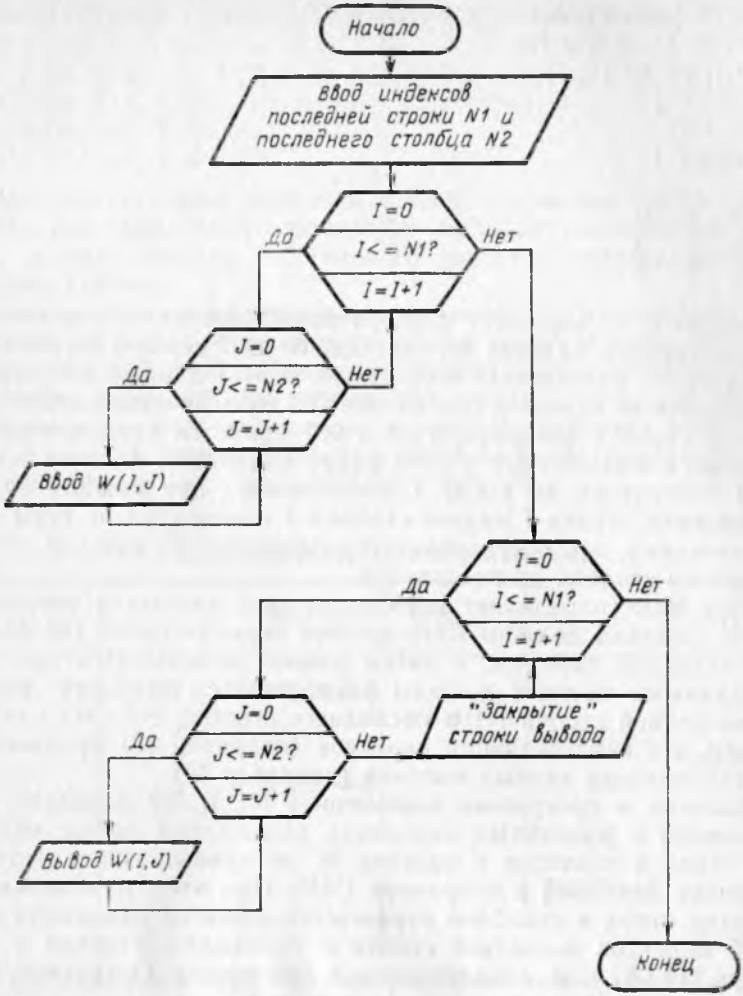


Рис. 26. Блок-схема для ввода — вывода данных двумерного массива

Программа 22 и ее выполнение

```
10 REM Ввод и вывод данных двумерного массива
20 DIM W(6,7)
25 REM Ввод данных по строкам
30 DATA 2, 3, 7, 4, 1, 0, 8, 5, 2, 9, 6, 3, 7, 4
40 READ N1, N2
50 FOR I=0 TO N1
60 FOR J=0 TO N2
70 READ W(I, J)
80 NEXT J
90 NEXT I
```

```
100 FOR I=0 TO N1
110 FOR J=0 TO N2
120 PRINT W (I, J);
130 NEXT J
140 PRINT
150 NEXT I
RUN
 7   4   1   0
 8   5   2   9
 6   3   7   4
```

Оператор 20 выделяет в оперативной памяти машины место для двумерного массива W, состоящего из 7 строк и 8 столбцов. Оператор 25 напоминает пользователю, что данные в программе вводятся по строкам. Это следует из того, что цикл изменения индекса строк I (операторы 50 и 90) является внешним по отношению к вложенному в него циклу изменения индекса столбцов J (операторы 60 и 80). Следовательно, при каждом значении индекса строки I индекс столбца J изменяется по всем своим значениям, это обеспечивает продвижение на каждой строке от первого столбца до последнего.

Этот факт определяет порядок записи данных в операторе DATA. Сначала должны идти данные первой строки (не столбца!) исходной таблицы, а затем данные остальных строк. Перед данными массива должны располагаться значения индексов последней строки N1 и последнего столбца N2. Это следует из того, что ввод значений индексов (оператор 40) производится раньше ввода данных массива (оператор 70).

Наличие в программе переменных N1 и N2 позволяет использовать в различных вариантах вычислений любое количество строк и столбцов в массиве W, не превышающее соответствующих значений в операторе DIM. При этом изменение количества строк и столбцов отражается лишь на изменении значений индексов последней строки и последнего столбца в операторе DATA, т. е. локализовано в программе. Остальные операторы программы остаются неизменными.

В программе 22 все исходные данные приведены в одном операторе DATA с номером 30. Однако их можно записывать в различных операторах DATA, разделив по смысловому содержанию. Например, значения индексов можно записать в отдельном операторе DATA перед оператором DATA с данными массива. Тогда первоначальный оператор 30 заменится на два новых оператора.

```
29 DATA 2, 3
30 DATA 7, 4, 1, 0, 8, 5, 2, 9, 6, 3, 7, 4
```

Данные массива можно записывать не все сразу в одном операторе, а по отдельным строкам исходной таблицы, записывая каждую строку в виде отдельного оператора DATA. Тогда

первоначальный оператор 30 заменится на четыре новых оператора.

```
29 DATA 2, 3
30 DATA 7, 4, 1, 0
31 DATA 8, 5, 2, 9
32 DATA 6, 3, 7, 4
```

Все приведенные варианты записи оператора DATA равноценны для машины. Пользователь выбирает конкретный вариант, исходя лишь из собственного удобства представления исходных данных.

Начинающим пользователям рекомендуется анализировать выполнение цикла в цикле письменно в виде таблицы, имитируя работу ЭВМ. В этой таблице представляются последовательные значения управляющих переменных циклов, индексы обрабатываемых в цикле величин и, возможно, значения этих величин. Анализ действий ЭВМ при вводе данных (операторы 50—90) по этому способу представлен в табл. 9.

Таблица 9

Анализ действий ЭВМ по шагам цикла

I	J	W(I, 0)	W(I, 1)	W(I, 2)	W(I, 3)
0	0	W(0,0) = 7			
	1		W(0,1) = 4		
	2			W(0,2) = 1	
	3				W(0,3) = 0
1	0	W(1,0) = 8			
	1		W(1,1) = 5		
. . . . .					
2	2			W(2,2) = 7	
	3				W(2,3) = 4

Проставлять в этой таблице действительные значения величин, как правило, нет необходимости. Обычно бывает достаточно проследить за порядком изменения индексов. При этом рассматриваются начало и конец цикла по **каждой** переменной. При известном опыте программисты проделывают эту работу в уме.

При выводе данных требуется располагать каждую строку исходной таблицы на отдельной строке экрана. Это достигается «закрытием» строки вывода оператором 140 после окончания внутреннего цикла по переменной J и до перехода к выводу новой строки массива.

Если по смыслу задачи удобнее вводить данные массива не по строкам, а по столбцам, то они записываются по столбцам в операторе DATA. При этом меняется взаимное расположение циклов изменения индекса строки I и индекса столбца J. Это

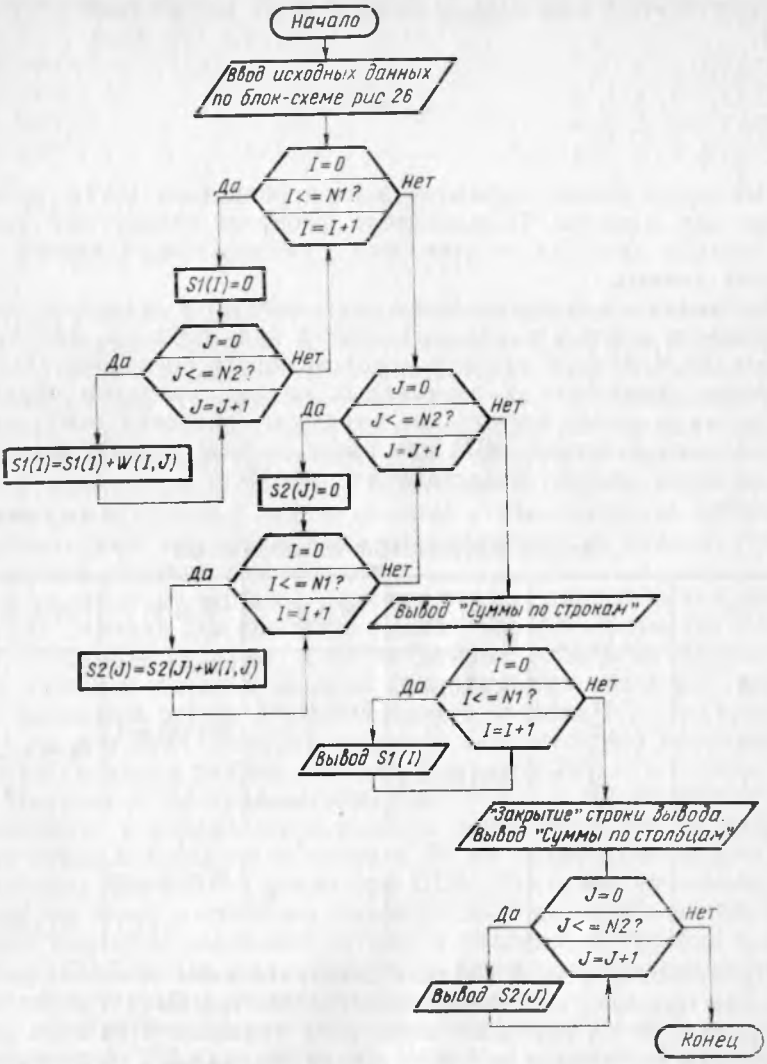


Рис. 27. Блок-схема для вычисления сумм отдельных строк и столбцов двумерного массива

достигается взаимной перестановкой с заменой номеров операторов в парах 50 — 60 и 80 — 90.

Задача 13. Вычисление сумм отдельных строк и столбцов двумерного массива.

Предполагается, что данные двумерного массива вводятся в оперативную память так же, как в программе 22. Следовательно, в новой программе будут использоваться операторы 25 — 90 из программы 22. Для сохранения в памяти всех получаемых сумм необходимо завести одномерные массивы сумм по строкам

S1 и сумм по столбцам S2. Длины этих массивов должны равняться количеству строк и столбцов в массиве W, соответственно.

Блок-схема решения этой задачи представлена на рис. 27. Ей соответствует программа 23.

Программа 23 и ее выполнение

```
10 REM Суммы отдельных строк и столбцов
20 DIM W(6, 7), S1(6), S2(7)
(Операторы 25 — 90 из программы 22).
100 FOR I=0 TO N1
110 LET S1(I)=0
120 FOR J=0 TO N2
130 LET S1(I)= S1(I)+W(I, J)
140 NEXT J
150 NEXT I
160 FOR J=0 TO N2
170 LET S2(J)=0
180 FOR I=0 TO N1
190 LET S2(J)=S2(J)+W(I, J)
200 NEXT I
210 NEXT J
220 PRINT «Суммы по строкам»;
230 FOR I=0 TO N1
240 PRINT S1(I);
250 NEXT I
260 PRINT
270 PRINT «Суммы по столбцам»;
280 FOR J=0 TO N2
290 PRINT S2(J);
300 NEXT J
RUN
Суммы по строкам   12   24   20
Суммы по столбцам   21   12   10   13
```

Программа 23 составлена по тем же принципам, что и программа 11, в которой получается сумма элементов одномерного массива. Но повышение размерности переменных в программе 23 приводит к усложнению логики ее построения и к увеличению длины программы.

Программа 23 составлена из отдельных независимых блоков суммирования и блоков вывода результатов. Ее можно сократить на четыре оператора, включив вывод результатов в блоки суммирования. Для этого надо изменить номера строк операторов 220, 240, 260, 270, 290 на номера строк 95, 145, 153, 157, 205, соответственно. При этом операторы с номерами строк 230, 250, 280, 300 исключаются из программы.

Задача 14. Получение квартально-годовой сводки по месячным данным.

Эта задача — типично экономическая. Исходными данными являются значения некоторых показателей по месяцам года, которые сведены в таблицу. Количество строк равно количеству показателей и может быть произвольным. Количество столбцов равно 12 по числу месяцев в году. Требуется получить сводную таблицу, в которой представлены суммарные и среднеарифметические значения этих показателей по каждому кварталу и за год. Количество строк в сводной таблице равно количеству строк исходной таблицы. Количество столбцов в сводной таблице равняется 10 (по два столбца на каждый квартал и два столбца на годовые значения).

Эта задача характерна тем, что здесь внутрь цикла в цикле вкладывается еще один цикл. Возникает многоуровневая структура вложенных циклов.

Блок-схема решения задачи представлена на рис. 28. Ей соответствует программа 24.

Программа 24 и ее выполнение

```
10 REM Квартально-годовая сводка
20 DIM W(6, 11), T(6, 9)
25 REM Ввод данных по строкам
29 DATA 1
30 DATA 10, 11, 12, 11, 12, 13, 12, 13, 14, 13, 14, 15
31 DATA 20, 21, 22, 21, 22, 23, 22, 23, 24, 23, 24, 25
40 READ N
50 FOR I=0 TO N
60 FOR J=0 TO 11
70 READ W(I, J)
80 IF J<=9 THEN LET T(I, J)=0
90 NEXT J
100 NEXT I
110 FOR I=0 TO N
120 LET H=0
130 FOR K=0 TO 6 STEP 2
140 FOR J=H TO H+2
150 LET T(I, K)=T(I, K)+W(I, J)
160 NEXT J
170 LET T (I, K+1)=T(I, K)/3
180 PRINT T(I, K); T(I, K+1);
190 LET T(I, 8)=T(I, 8)+T(I, K)
200 LET H=H+3
210 NEXT K
220 LET T(I, 9)=T(I, 8)/12
230 PRINT T(I, 8); T(I, 9)
240 NEXT I
RUN
 33   11   36   12   39   13   42   14   150   12.5
 63   21   66   22   69   23   72   24   270   22.5
```

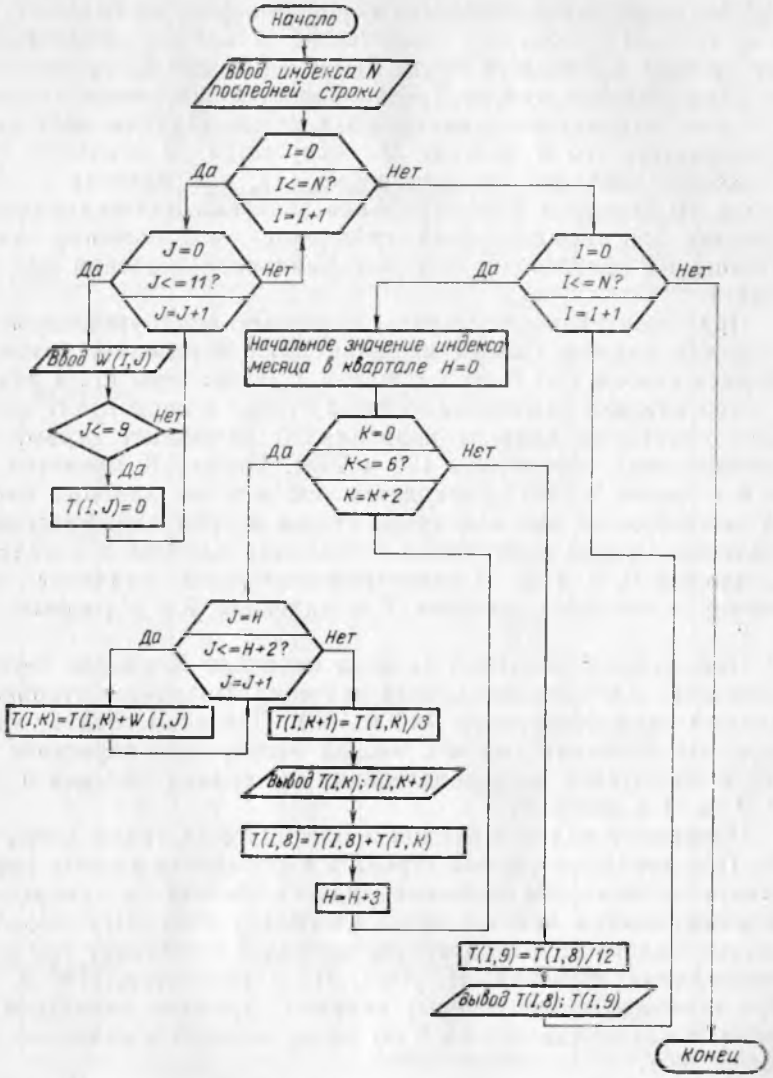


Рис. 28. Блок-схема для получения квартально-годовой сводки

Максимально возможное количество строк в массиве исходных данных W и в массиве результатов T может быть любым (в данной программе принято равным 7). Количество столбцов в этих массивах точно определено условиями задачи: 12 и 10, соответственно.

В качестве примера рассматривается ввод двух строк массива W. Конкретные значения данных этих строк представлены в операторах 30 и 31.

Ввод данных производится по принципу программы 22. Разница состоит в том, что количество столбцов фиксировано. Кроме того, в процессе ввода данных массива W производится засылка нулей в массив T. Это является подготовкой к дальнейшему накоплению квартальных сумм. При засылке нулей учитывается, что в массиве W содержится 12 столбцов (максимальное значение индекса равно 11), а в массиве T содержится 10 столбцов (максимальное значение индекса равно 9). Поэтому при засылке нулей проверяется, что значение индекса столбца не превосходит его максимального значения для массива T.

При получении результата возникает многоуровневая вложенность циклов. Самый внешний цикл определяет изменение индекса строки I от 0 до N с шагом 1 (операторы 110 и 240).

При каждом изменении индекса строки I происходит полный цикл изменения индекса квартала K. Возникает первый вложенный цикл (операторы 130 и 210). Индекс K меняется от 0 до 6 с шагом 2. Это объясняется тем, что для каждого квартала вычисляются две величины: сумма и среднеарифметическое значение. Суммы получаются в столбцах массива T с индексом K, равным 0, 2, 4, 6. Среднеарифметические значения получаются в столбцах массива T с индексом K+1, равным 1, 3, 5, 7.

При каждом значении индекса квартала K индекс месяца J принимает 3 последовательных значения. Возникает второй вложенный цикл (операторы 140 и 160). При этом следует учитывать, что значение индекса месяца непрерывно нарастает (так как используется последовательная нумерация месяцев в году) от 0 до 11 с шагом 1.

Изменение индекса месяца J организуется тремя операторами. При переходе к новой строке и к обработке данных первого квартала начальное значение индекса месяца в квартале H устанавливается равным нулю (оператор 120). При обработке данных одного квартала индекс месяца J принимает три последовательных значения: H, H+1, H+2 (операторы 140 и 160). При переходе к следующему кварталу прежнее начальное значение H увеличивается на 3 по числу месяцев в квартале (оператор 200).

После вычисления суммы за квартал (в цикле 140 — 160) до перехода к новому кварталу (до оператора 210) выполняется ряд действий. Вычисляется среднеарифметическое значение за квартал (оператор 170) и вместе с суммой за квартал выводится на экран (оператор 180). Сумма за квартал прибавляется к накапливаемой сумме за год (оператор 190). Наконец, вычисляется начальное значение индекса месяца для следующего квартала (оператор 200).

После вычислений для всех кварталов (после оператора 210) вычисляется среднеарифметическое значение за год (оператор 220) и вместе с суммой за год выводится на экран (оператор

230). Затем происходит переход к обработке следующей строки (оператор 240).

Приведенное описание показывает, что логика работы программы достаточно сложна. Проследить за выполнением операторов 110 — 240 рекомендуется с помощью табл. 10.

Таблица 10

Выполнение операторов 110 — 240

I	H	K	J	T(I, K)	T(I, K+1)	W(I, J)	T(I, 8)	T(I, 9)
0	0	0	0	(0,0)		(0,0)		
			1	(0,0)		(0,1)		
			2	(0,0)		(0,2)		
					(0,1)			
Вывод на экран				T(0,0);	T(0,1);			
							(0,8)	
	3	2	3	(0,2)		(0,3)		
			4	(0,2)		(0,4)		
			5	(0,2)		(0,5)		
					(0,3)			
Вывод на экран				T(0,2);	T(0,3);			
							(0,8)	
.	.	.	.	. . . .	. . . .	. . .	. . .	. . . .
	9	6	9	(0,6)		(0,9)		
			10	(0,6)		(0,10)		
			11	(0,6)		(0,11)		
					(0,7)			
Вывод на экран				T(0,6);	T(0,7);			
							(0,8)	
	12							(0,9)
					Вывод на экран		T(0,8);	T(0,9)
1	0	0	0	(1,0)		(1,0)		
			1	(1,0)		(1,1)		
.	.	.	.	. . . .	. . . .	. . . .	. . . .	. . . .
	9	6	9	(1,6)		(1,9)		
			10	(1,6)		(1,10)		
			11	(1,6)		(1,11)		
					(1,7)			
Вывод на экран				T(1,6);	T(1,7);			
	12						(1,8)	
								(1,9)
					Вывод на экран		T(1,8);	T(1,9)

В таблице показано последовательное изменение индексов и отмечены моменты вывода результатов на экран. Здесь хорошо видна организация вложенных циклов. При каждом значении индекса строки I индекс квартала K принимает четыре значения: 0, 2, 4, 6, а при каждом значении индекса квартала индекс месяца J принимает три последовательных значения. Вспомогательная переменная H позволяет осуществлять последовательное изменение индекса J от 0 до 11. Значение H=12, появляющееся после обработки последнего квартала, оказывается лишним и заменяется на нуль при переходе к обработке следующей строки.

6. ОПЕРАТОР DEF

DEF сокращ. от англ. DEFINE — определить

Формат оператора:

DEF FNI (параметр) = арифметическое-выражение

Здесь FN — две обязательные для написания латинские буквы, с которых должно начинаться имя функции, определяемой пользователем. Вместо строчной буквы I подставляется прописная латинская буква, определяющая конкретное имя этой функции. Параметр в этом операторе представляется именем переменной.

Часто в различных местах одной программы требуется вычислять одно и то же арифметическое выражение при различных значениях какой-либо переменной величины. В этом случае удобно определить это арифметическое выражение каким-нибудь именем один раз с помощью оператора DEF, а затем обращаться к имени этого выражения, указывая конкретное значение переменной величины в качестве параметра.

Арифметическое выражение, определяемое таким образом, по терминологии инструкции к ДВК-1 называется **функцией, определяемой пользователем.** Определение этой функции должно производиться раньше ее использования в операторах Бейсика. Рекомендуется определять все функции в начале программы. Например, по аналогии с правилами языка программирования ФОРТРАН можно сначала описать все операторы DIM, затем DATA, затем DEF.

В качестве примера использования функции, определяемой пользователем, можно рассмотреть задачу вычисления суммы заработной платы, подлежащей выплате работнику. В этой задаче сначала определяется общая начисленная сумма заработной платы, состоящая из оклада, премии, работы по совместительству и пр. Затем из начисленной суммы вычитаются подоходный налог, профсоюзный взнос (по желанию работника), плата за покупку товаров в кредит и пр.

Сумма Z, остающаяся после вычитания налога и профсоюзного взноса, для всех работников с начисленной заработной платой S больше 100 рублей вычисляется одинаково по формуле

$$Z = S - 8{,}20 - (S - 100) \cdot 0{,}13 - 0{,}01 \cdot S.$$

Эта формула довольно длинна, а используется в вычислениях многократно. Поэтому целесообразно определить ее один раз с помощью оператора DEF, а затем обращаться к ней, указывая конкретное значение начисленной заработной платы.

Определение функции можно представить в следующем виде: номер-строки DEF FNZ(S) = S — 8.2 — (S — 100) * .13 — — .01 * S

Параметр в определении функции (здесь S) называется **формальным параметром** (формальным в том смысле, что он не определяет никакого конкретного значения). Формальный параметр — условное название действительной величины, которая будет подставлена вместо формального параметра и над которой будут производиться действия, указанные для формального параметра.

Действительная величина, подставляемая вместо формального параметра, называется **фактическим параметром.** Фактический параметр может представляться константой, именем переменной, арифметическим выражением. Допускается рекурсивное использование этой же самой функции NFl в параметре.

При обращении к функции, определяемой пользователем, результат вычисления можно использовать в операторе PRINT или в других операторах, в том числе и в составе арифметического выражения. Различные варианты обращения приведены на примере функции FNZ.

Пример функции, определяемой пользователем

```
10 REM Функция, определяемая пользователем
20 DEF FNZ(S)=S — 8.2— (S — 100) ∗.13 — .01∗ S
30 PRINT FNZ(320);
40 LET S=320
50 LET A=FNZ(S)
60 LET T=112
70 LET B=FNZ(T)
80 LET C=FNZ(S+T)
90 LET D=FNZ(S)/FNZ(T)
100 PRINT A; B; C; D
RUN
 280   280   101.12   376.32   2.768987
```

В операторе 30 фактическим параметром функции FNZ является константа 320, а сама функция используется в операторе PRINT.

В операторах 50 и 70 фактическим параметром функции FNZ является имя переменной. В первом случае — имя S, во втором — имя T. Совпадение или несовпадение имени фактического параметра с именем формального параметра не имеет никакого значения, так как реальные действия будут выполняться над фактическим параметром. Сама функция FNZ используется в операторе LET.

В операторе 80 фактическим параметром является арифметическое выражение S+T.

В операторе 90 сама функция FNZ используется в арифметическом выражении.

7. ОПЕРАТОРЫ GO SUB И RETURN

SUB — **сокращ. от англ.** SUBROUTINE — **подпрограмма;** GO SUB — **перейти к подпрограмме,** RETURN — **вернуться, возврат**

Так как пробелы в Бейсике не имеют никакого значения, то слитное написание оператора GO SUB равносильно его раздельному написанию.

Формат оператора GO SUB:

GO SUB номер-строки

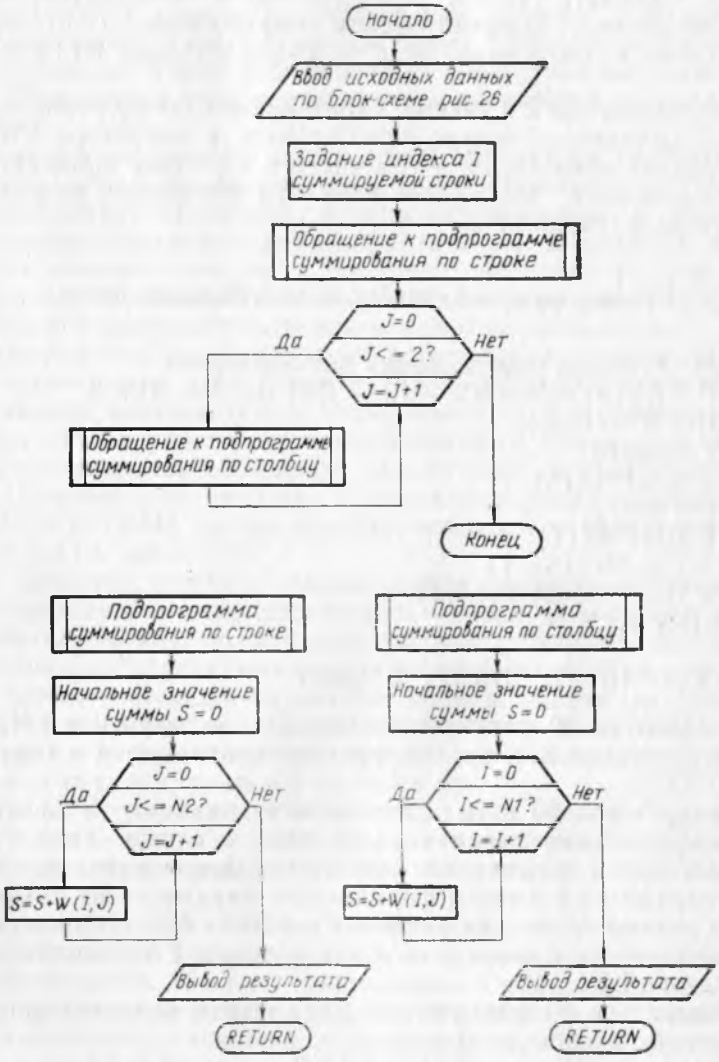


Рис. 29. Блок-схема для суммирования по строке или по столбцу с использованием подпрограмм

Оператор RETURN записывается одним этим словом.

Оператор GO SUB используется для обращения к подпрограмме. **Подпрограммой** называется обособленная часть программы, заканчивающаяся оператором RETURN. Подобно функциям, определяемым пользователем, подпрограммы используются для проведения часто встречающихся в различных местах программы вычислений по одним и тем же соотношениям при различных значениях переменных.

В отличие от функции подпрограмма может состоять из множества операторов. Выход на подпрограмму разрешается только оператором GO SUB с номером строки первого оператора подпрограммы. Выход из подпрограммы происходит на следующий за GO SUB оператор. Выход обеспечивается оператором RETURN, который должен быть последним в подпрограмме.

Подпрограммы рекомендуется располагать в конце программы, например, с номеров 1000. Для отделения их от основной программы перед ними располагается оператор STOP.

В программе 23 вычисляются суммы для **всех** отдельных строк и столбцов. Однако в практических задачах это требуется не всегда. Поэтому можно составить отдельные подпрограммы, вычисляющие суммы для одной строки и для одного столбца. Затем задать требуемый номер строки или столбца и обратиться к соответствующей подпрограмме.

Блок-схема программы, использующей подпрограммы для вычисления суммы по строке или по столбцу двумерного массива, представлена на рис. 29. Данной блок-схеме соответствует программа 25.

Програма 25 и ее выполнение

```
10 REM Суммир. по строке и по столбцу в подпрограммах
(Операторы 20 — 90 из программы 22)
100 LET I=1
110 GO SUB 1010
120 FOR J=0 TO 2
130 GO SUB 1110
140 NEXT J
1000 STOP
1010 REM Подпрограмма суммирования по строке
1020 LET S=0
1030 FOR J=0 TO N2
1040 LET S=S+W(I, J)
1050 NEXT J
1060 PRINT «Сумма по строке»; «I=»I; «   S=»S
1070 RETURN
1110 REM Подпрограмма суммирования по столбцу
1120 LET S=0
```

```
1130 FOR I=0 TO N1
1140 LET S=S+W(I, J)
1150 NEXT I
1160 PRINT «Сумма по столбцу»; «J=» J; «   S=»S
1170 RETURN
RUN
Сумма по строке I=  1   S=  24
Сумма по столбцу J=  0   S=  21
Сумма по столбцу J=  1   S=  12
Сумма по столбцу J=  2   S=  10
```

В данной программе вычисляется сумма одной второй строки (индекс I=1), а также суммы для первых трех столбцов из четырех (индекс J=0, 1, 2).

В операторе 110 происходит обращение к подпрограмме, начинающейся со строки с номером 1010. После выполнения всех операторов подпрограммы ее последний оператор RETURN с номером 1070 передает управление следующему за оператором 110 оператору 120. Обращение к подпрограмме может производиться в цикле, как показано операторами 120—140.

При обращении к подпрограмме предварительно должны быть определены значения всех тех переменных, которые в подпрограмме используются, но не определяются. Так, в подпрограмме суммирования по строке используются имя элемента массива W и индекс номера строки I. Однако в самой подпрограмме значения этих переменных не определяются. Поэтому они предварительно определяются в основной программе. В частности, значение индекса I задается оператором 100 до обращения к подпрограмме.

В каждой подпрограмме переменная суммирования S предварительно обнуляется. Иначе при повторном обращении к подпрограмме новая сумма будет складываться со старой.

Вывод результатов может производиться как в подпрограмме, так и в основной программе. Это зависит от конкретной задачи. В данном случае операторы 1060 и 1160 можно вывести из подпрограмм и включить в основную программу под номерами 115 и 135, соответственно.

Подпрограммы должны располагаться за оператором STOP, отделяющим их от основной программы. Это объясняется тем, что подпрограммы должны выполняться только при обращении к ним с помощью оператора GO SUB. В программе 25 после оператора 140 оператор STOP прекращает выполнение программы, не допуская последовательного выхода на подпрограммы. При отсутствии оператора STOP такой выход произошел бы. В этом случае при выполнении оператора RETURN в строке с номером 1070 произошло бы сообщение об ошибке, вызванной выходом на оператор RETURN без обращения по GO SUB.

8. ТАБЛИЧНОЕ ОФОРМЛЕНИЕ РЕЗУЛЬТАТОВ РАСЧЕТОВ

Результаты расчетов на ЭВМ можно выводить в привычном табличном виде с названием таблицы и с названиями граф. К сожалению, в Бейсике ДВК-1 эта задача сильно осложняется ограниченной возможностью управления количеством позиций, предоставляемых для вывода числа.

Наиболее просто поставленная задача решается в случае вывода информации по зонам. В этом случае строка вывода автоматически делится на 5 зон по 14 позиций в каждой зоне. Каждая зона представляет собой столбец таблицы. Разделение выводимой информации на столбцы достигается использованием запятой в качестве разделителя в операторе PRINT. Однако такое разделение возможно только тогда, когда количество столбцов не превышает пяти.

Вывод названия таблицы обеспечивается представлением его в качестве символьного элемента в операторе PRINT. Для расположения названия по центру таблицы используется дополнительная запятая в качестве разделителя, что приводит к пропуску зоны, а также дополнительные необходимые символы пробелов.

После названия таблицы следует горизонтальная линия шапки таблицы. Ее можно получить выводом в цикле какого-либо символа, например, знака равенства или минуса (дефиса). Затем следуют названия граф таблицы. Они используются в качестве символьных элементов в операторе PRINT, разделяемых запятыми. Внутри шапки таблицы горизонтальная линия может отделять общее для группы столбцов название. Кроме того, она используется в конце шапки таблицы, а также в конце всей таблицы. Горизонтальная линия используется многократно, поэтому ее вывод целесообразно организовать в виде подпрограммы.

Приемы, используемые при оформлении результатов расчета, разбираются на различных задачах вывода данных двумерного массива. Конкретные значения данных представлены положительными и отрицательными числами с различными длинами целых и дробных частей.

Задача 1. Вывод информации по зонам.

Блок-схема решения этой задачи представлена на рис. 30. Ей соответствует программа 26.

Программа 26 и ее выполнение

```
10 REM Вывод информации по зонам
20 DIM W(6, 4)
25 REM Ввод данных по строкам
29 DATA 2, 2
30 DATA .1, 2, 3.4
31 DATA 23.4, 56.78, .901
32 DATA —23.4, —56.78, —.901
```

```
38 DATA 49, 35
(Операторы 40 — 90 из программы 22)
100 READ K1, K2
120 PRINT, «Исходные данные»
130 LET K=K1
140 GO SUB 1210
150 PRINT «Номер», , «Названия граф»
160 PRINT «строки»,
170 LET K=K2
180 GO SUB 1210
190 PRINT, «Графа 1», «Графа 2», «Графа 3»
200 LET K=K1
210 GO SUB 1210
220 FOR I=0 TO N1
230 PRINT I+1,
300 FOR J=0 TO N2
320 PRINT W(I, J),
350 NEXT J
360 PRINT
370 NEXT I
380 GO SUB 1210
1000 STOP
1210 REM Вывод горизонтальной линии
1220 FOR J=1 TO K
1230 PRINT «—»;
1240 NEXT J
1250 PRINT
1260 RETURN
RUN
```

Исходные данные

Номер строки	Названия граф		
	Графа 1	Графа 2	Графа 3
1	.1	2	3.4
2	23.4	56.78	.901
3	—23.4	—56.78	—.901

Массив W может иметь произвольное количество строк, но не более пяти столбцов, так как на строке экрана всего пять зон. При пяти столбцах в массиве W придется отказаться от вывода столбца с номером строки. При этом в операторе 150 исключается элемент «Номер», а также исключается оператор 160, первая запятая в операторе 190 и операторы 230 и 360.

В операторе 100 вводятся номера конечных позиций: К1 — для горизонтальной линии всей таблицы и К2 — для горизонтальной линии внутри шапки таблицы. Последняя линия отделяет общее для нескольких граф название. Значения К1 и К2 определяются либо расчетным, либо пробным путем. При расчете учитывается, что каждая полная зона занимает 14 позиций, а последняя зона может быть неполной. Количество позиций в последней зоне определяется либо ее самым длинным числом, либо названием последней графы. При изменении значений К1 и К2 изменяется только оператор 38, представляющий эти значения. Остальные операторы программы остаются неизменными. В программе К1 и К2 используются в операторах 130, 170 и 200, но уже в виде имен переменных, а не в виде конкретных значений.

Оператор 120 выводит заголовок таблицы. Для размещения заголовка в центре таблицы перед его названием в операторе 120 используется запятая для пропуска одной зоны. Удвоение запятой в операторе 150 также используется для пропуска одной зоны. Для этой же цели поставлена запятая перед названием первой графы в операторе 190.

Нумерация операторов после оператора 100 ведется с пропуском номеров, кратных 10. Это сделано для сохранения одинаковых номеров в общих для программ 26 — 28 операторах.

Нумерация выводимых строк в операторе 230 начинается с 1, поэтому она на единицу больше значения индекса строки.

Вывод горизонтальной линии производится подпрограммой. Подсчет выводимых в линии символов ведется от 1. Граничное значение для числа выводимых символов задается в основной программе с помощью переменной с именем К.

Задача 2. Компактный вывод информации без выравнивания по столбцам чисел различной длины.

При количестве столбцов таблицы больше 5 переход к компактному выводу информации оказывается неизбежным. При меньшем количестве столбцов он может оказаться просто желательным. Компактный вывод обеспечивается использованием в операторе PRINT точки с запятой в качестве разделителя. При этом отрицательные числа разделяются одним пробелом, а положительные — двумя. Для увеличения количества пробелов можно ввести пробелы в качестве символьных элементов в операторе PRINT. Таким образом достигается разделение чисел между собой на любой желаемый интервал.

Более сложной проблемой является выравнивание данных по столбцам. Если числа в столбцах оказываются разной длины, то при выводе таблицы возникают сдвиги чисел по столбцам. Это демонстрируется на примере программы 27, которая является модификацией программы 26. Изменения связаны с увеличением количества столбцов в массиве и с переходом к компактному выводу чисел с введением одного дополнительного пробела между числами.

Программа 27 и ее выполнение

```
10 REM Компактный вывод без выравнивания
20 DIM W(6, 7)
25 REM Ввод данных по строкам
29 DATA 2, 4
30 DATA .1, 2, 3.4, .56, 78
31 DATA 23.4, 56.78, .901, 234, 5.678
32 DATA  —23.4, —56.78, —.901, —234, —5.678
38 DATA 41, 34
(Операторы 40 — 90 из программы 22)
100 READ K1, K2
120 PRINT, «Исходные данные»
130 LET K=K1
140 GO SUB 1210
150 PRINT «Номер», «Названия граф»
160 PRINT «строки»;
170 LET K=K2
180 GO SUB 1210
190 PRINT «          Графа 1   Графа 2   Графа 3   »;
191 PRINT «Графа 4   »
200 LET K=K1
210 GO SUB 1210
220 FOR I=0 TO N1
230 PRINT I+1; « »;
300 FOR J=0 TO N2
320 PRINT W(I, J); « »;
350 NEXT J
360 PRINT
370 NEXT I
380 GO SUB 1210
1000 STOP
(Операторы 1210 — 1260 из программы 26)
RUN
```

Исходные данные

Номер строки	Названия граф				
	Графа 1	Графа 2	Графа 3	Графа 4	
1	.1	2	3.4	.56	78
2	23.4	56.78	.901	234	5.678
3	—23.4	—56.78	—.901	—234	—5.678

Результаты выполнения программы 27 показывают, что при наличии в столбцах чисел различной длины происходит существенный сдвиг чисел по столбцам. Этот сдвиг может быть настолько значительным, что приведет к полному искажению табличной формы. В этом случае требуется применение специальных мер для выравнивания чисел по столбцам, как показано ниже.

Задача 3. Компактный вывод информации с выравниванием по столбцам чисел различной длины.

Количество позиций, которое занимает число в основной форме, равно сумме его целых и дробных разрядов плюс одна позиция знака, плюс одна позиция десятичной точки для дробных чисел. К этому добавляется еще одна позиция пробела при использовании за числом точки с запятой в качестве разделителя в операторе PRINT.

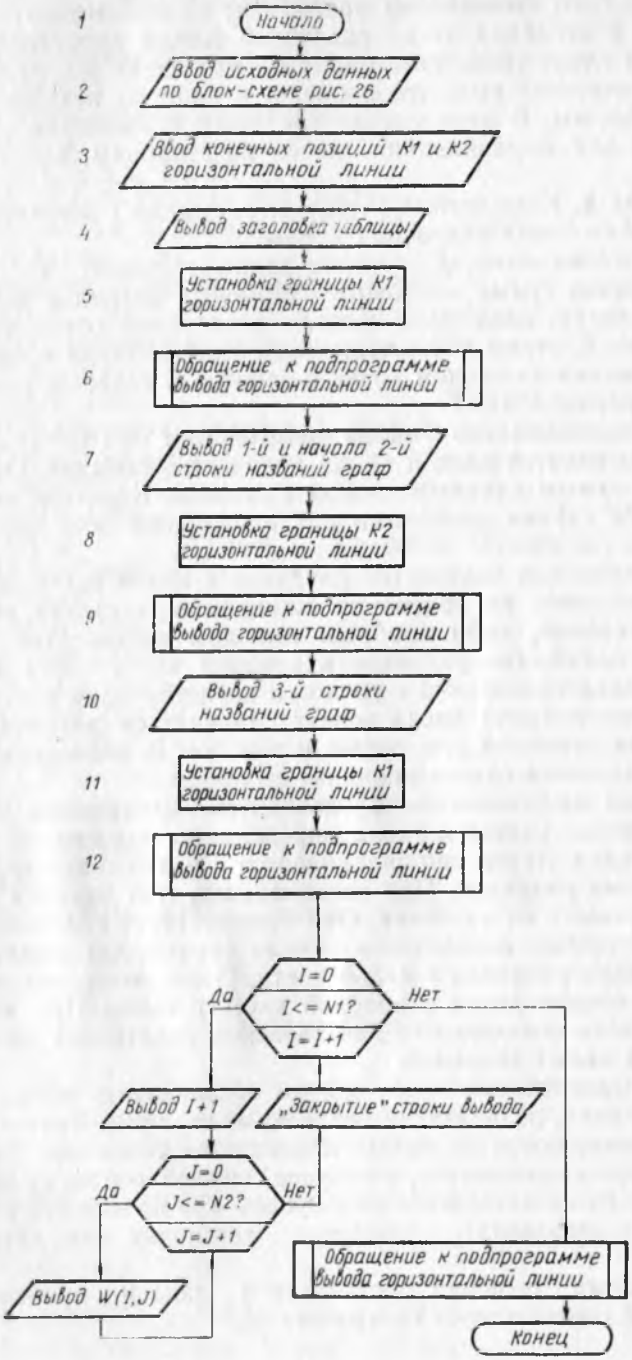
Для выравнивания чисел по столбцам каждому числу в столбце отводится одно и то же количество позиций. Оно определяется самым длинным числом в столбце. Короткие числа дополняются справа пробелами для заполнения всех отведенных позиций.

Максимальное количество разрядов в целой части числа определяется либо на основании ожидаемых значений результатов вычислений, либо пробным выводом результатов. Максимальное количество разрядов в дробной части числа задается на основании требований к точности представления результатов. Количество позиций числа может задаваться различным для различных столбцов или одним и тем же. В последнем случае оно определяется самым широким столбцом.

Процесс выравнивания производится следующим образом. После вывода каждого числа определяется количество его целых разрядов. Затем оно сравнивается с максимальным количеством целых разрядов. При несовпадении этих величин определяется разность их значений. Она соответствует количеству пробелов, которыми дополняется число справа для компенсации недостающих разрядов в целой части. После этого аналогичные действия производятся с дробной частью числа. При этом надо учитывать возможность исчезновения десятичной точки при получении целых значений.

При выравнивании чисел можно организовать вывод вертикальных линий, разделяющих отдельные столбцы. Вертикальные линии формируются из любых подходящих символов. Для этой цели можно использовать, например, символ вертикальной черты (вырабатывается клавишей с буквой «Э» на нижнем регистре латинского алфавита), двоеточие, звездочку или латинскую букву I.

Блок-схема решения поставленной задачи приведена на рис. 31. Ей соответствует программа 28.


1
Начало
2
Ввод исходных данных по блок-схеме рис. 26
3
Ввод конечных позиций K1 и K2 горизонтальной линии
4
Вывод заголовка таблицы
5
Установка границы K1 горизонтальной линии
6
Обращение к подпрограмме вывода горизонтальной линии
7
Вывод 1-й и начало 2-й строки названий граф
8
Установка границы K2 горизонтальной линии
9
Обращение к подпрограмме вывода горизонтальной линии
10
Вывод 3-й строки названий граф
11
Установка границы K1 горизонтальной линии
12
Обращение к подпрограмме вывода горизонтальной линии
I=0
I<=N1?
I=I+1
Да
Нет
Вывод I+1
„Закрытие" строки вывода
J=0
J<=N2?
J=J+1
Да
Нет
Вывод W(I,J)
Обращение к подпрограмме вывода горизонтальной линии
Конец

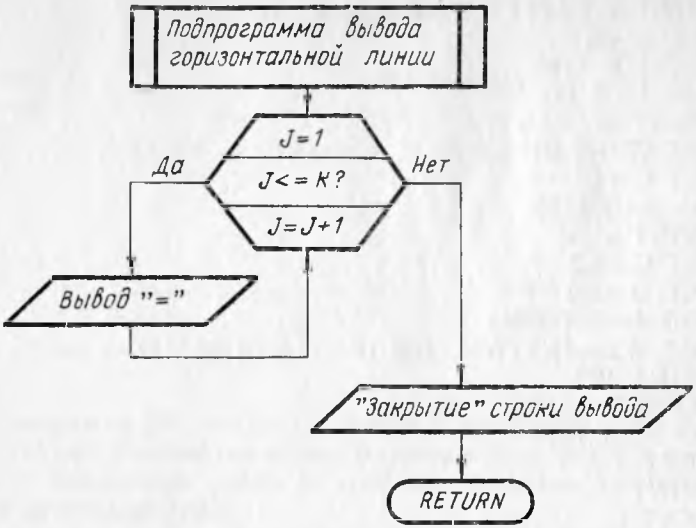


Рис. 30. Блок-схема для вывода информации по зонам

Программа 28 и ее выполнение

```
10 REM Компактный вывод информации с выравниванием
20 DIM W (6, 7)
25 REM Ввод данных по строкам
29 DATA 2, 3
30 DATA .1, 2, 3.4, .56
31 DATA 23.4, 56.78, .901, 234
32 DATA —23.4, —56.78, —.901, — 234
38 DATA 52, 44
39 DATA 3, 3, 3

(Операторы 40 — 90 из программы 22)

100 READ K1, K2
110 READ C1, C2, D2
120 PRINT, «Исходные данные»
130 LET K=K1
140 GO SUB 1210
150 PRINT «:Номер :»,
151 PRINT «Названия граф»,, «          :»
160 PRINT «:строки:»;
170 LET K=K2
180 GO SUB 1210
190 PRINT «:          : Графа 1  : Графа 2  :»;
```

```
191 PRINT « Графа 3   : Графа 4   :»
200 LET K=K1
210 GO SUB 1210
220 FOR I=0 TO N1
230 PRINT «:  »;I+1;
240 LET W2=I+1
250 LET C=C1
260 GO SUB 1310
270 PRINT «: »;
280 LET C=C2
290 LET D=D2
300 FOR J=0 TO N2
310 LET W2=INT(W(I, J)* 10 ↑D+.5)/10 ↑D
320 PRINT W2;
330 GO SUB 1310
340 GO SUB 1510
350 NEXT J
360 PRINT
370 NEXT I
380 GO SUB 1210
1000 STOP
```

(Операторы 1210 — 1260 из программы 26)

```
1310 REM Проверка целой части числа
1320 FOR M=0 TO C—1
1330 IF ABS(W2)<10↑M GO TO 1360
1340 NEXT M
1350 GO TO 1400
1360 LET P=C—M
1370 FOR L=1 TO P
1380 PRINT « »;
1390 NEXT L
1400 RETURN
1510 REM Проверка дробной части числа
1520 FOR M=D—1 TO 0 STEP —1
1530 LET A1=INT(W2* 10 ↑M)/10 ↑M
1540 IF A1<>W2 GO TO 1580
1550 NEXT M
1560 LET P=D+1
1570 GO TO 1590
1580 LET P=D—1—M
1590 FOR L=1 TO P
1600 PRINT « »;
1610 NEXT L
1620 PRINT «: »;
1630 RETURN
RUN
```

Исходные данные

Номер строки	Названия граф			
	Графа 1	Графа 2	Графа 3	Графа 4
1	.1	2	3.4	.56
2	23.4	56.78	.901	234
3	—23.4	—56.78	—.901	—234

В программе 28 появляется новый оператор 39. Числа в нем представляют количество целых разрядов для графы с номером строки и количество целых и дробных разрядов, соответственно, для остальных граф.

Количество целых разрядов для графы с номером строки определяется следующим образом. Внутри этой графы (между двоеточиями, создающими вертикальные линии) отведено шесть позиций. Они определены самым длинным словом в названии этой графы — словом «строки» в операторе 160. В программе принято, что числа будут отделяться одним пробелом от левого и правого двоеточия, соответственно. Еще одна позиция требуется под знак числа. Следовательно, на целую часть числа (дробной части у номера не может быть) остается 6—3=3 позиции. Количество целых и дробных разрядов для остальных граф принято равным 3, исходя из значений, представленных в операторах 30 — 32. При изменении количества разрядов изменяется только оператор 39. Конкретные значения из этого оператора присваиваются переменным C1, C2, D2 в операторе 110 и в дальнейшем в программе используются в виде этих переменных.

Оператор 180 выводит горизонтальную линию под общим для нескольких граф названием. Следует отметить, что в данной программе эта линия не может быть закончена двоеточием, создающим вертикальную линию. Это связано с тем, что оператор 1250 в подпрограмме вывода горизонтальной линии «закрывает» строку вывода и больше на ней уже ничего нельзя вывести. Если двоеточие необходимо, то можно либо вывести оператор 1250 из подпрограммы, выполняя его функции в основной программе, либо завести вторую подпрограмму без этого оператора.

Во всех графах, кроме графы с номером строки, между двоеточиями отводится по 10 позиций: 2 на отступление от двоеточий, 1 на знак числа, 1 на десятичную точку и по 3 на целую и дробную части числа. Это следует учитывать при размещении заголовков этих граф в операторах 190 — 191. Заголовки записаны в двух операторах для сокращения длины строки каждо-

го оператора. При таком способе записи нельзя забывать разделитель в конце первой строки. В противном случае вывод заголовка на экран будет осуществлен не в виде единой строки, а в виде двух последовательных строк.

При выводе чисел вертикальные линии и отступления от них на один пробел для графы с номером строки обеспечиваются символьным элементом в операторах 230 и 270. Для остальных граф потребуются только правые вертикальные линии (левая создается предыдущей графой). Правая вертикальная линия создается оператором 1620 в подпрограмме проверки дробной части числа. У номера строки дробная часть не проверяется, и правая вертикальная линия этой графы создается оператором 270.

Выводимые строки нумеруются с 1. Поэтому в операторе 230 значение индекса строки увеличено на 1.

В подпрограмме проверки целой части числа проверяемая величина имеет имя W2 (оператор 1330), а количество целых разрядов — имя C (операторы 1320 и 1360). Поэтому для проверки номера строки его значение присваивается переменной W2 в операторе 240. Количество целых разрядов для номера строки C1 присваивается переменной C в операторе 250. После этого оператор 260 обращается к подпрограмме проверки целой части числа. Эта подпрограмма компенсирует недостающие разряды целой части пробелами за числом. Подробнее работа этой подпрограммы рассматривается ниже. Оператор 270 выводит правую вертикальную линию. Далее следует обработка данных массива W.

Количество целых разрядов C2 и дробных разрядов D2 для чисел массива W присваивается, соответственно, переменным C и D в операторах 280 и 290. Операторы 300 и 350 изменяют индекс J столбца массива W.

В данной программе количество дробных разрядов для чисел массива W известно из исходных данных. В общем же случае массив W может получаться в результате вычислений, и количество дробных разрядов может быть любым. Поэтому в программе предусматривается ограничение количества дробных разрядов значением D с округлением числа в операторе 310. Округленное значение присваивается переменной W2 для дальнейшей проверки количества целых и дробных разрядов в соответствующих подпрограммах. Естественно, что влияние оператора 310 сказывается только в тех случаях, когда длина дробной части больше допустимой. В операторе 310 используется стандартная функция INT, описанная в приложении.

Оператор 320 выводит округленное значение W2. Операторы 330 и 340 последовательно обращаются к подпрограммам проверки целой и дробной части числа, соответственно. Эти подпрограммы компенсируют недостающие разряды пробелами за числом. Последняя подпрограмма выводит также правую вертикальную линию. Подробнее работа подпрограмм рассматри-

вается ниже. Оператор 350 обеспечивает повторение рассмотренной обработки данных массива W при всех значениях индекса столбца, т. е. для всех элементов одной строки.

После окончания обработки одной строки оператор 360 «закрывает» строку вывода. Оператор 370 обеспечивает повторение обработки для всех строк. После этого оператор 380 выводит горизонтальную линию, которой заканчивается таблица.

Работа подпрограммы проверки целой части числа поясняется с помощью табл. 11.

Таблица 11

Проверка разрядов в целой части числа

C	W2	M	10^M	W2<10^M?	C—M=P
	0,1	0	1	Да	3 — 0=3
	1	0	1	Нет	
		1	10	Да	3 — 1=2
	10	0	1	Нет	
		1	10	Нет	
		2	100	Да	3 — 2=1
	100	0	1	Нет	
		1	10	Нет	
		2	100	Нет	

В табл. 11 C — количество разрядов в целой части числа, установленное для проверяемых чисел; W2 — проверяемое число; M — управляющая переменная цикла, изменяющаяся от 0 до C—1; C—M=P — количество пробелов, компенсирующих недостающие разряды. Значение C=3 показывает, что в целой части числа максимально может быть 3 разряда.

Пусть проверяемое число W2 меньше 1, т. е. вообще не имеет разрядов в целой части. Следовательно, три отсутствующих разряда целой части должны компенсироваться тремя пробелами. Это и покажут результаты проверки. Проверка начинается сравнением проверяемого числа с 1 — наименьшим числом с одним разрядом в целой части. Сама 1 есть 10^M при M=0. Проверяемое число оказывается меньше 1. Это возможно только в том случае, если оно вообще не имеет разрядов в целой части. Разность C—M=P, определяемая в случае ответа «Да» при сравнении, равняется 3, т. е. точно определяет необходимое число пробелов.

Пусть проверяемое число W2 равно 1, т. е. является наименьшим числом с одним разрядом в целой части. Его проверка также начинается сравнением с 1. Сравнение дает ответ «Нет». Тогда сравнение продолжается с числом 10 — наименьшим числом с двумя разрядами в целой части. Теперь уже сравнение дает ответ «Да». Разность C—M=P=2 определяет правильное количество пробелов.

Аналогично проводится проверка при W2=10. Но следует

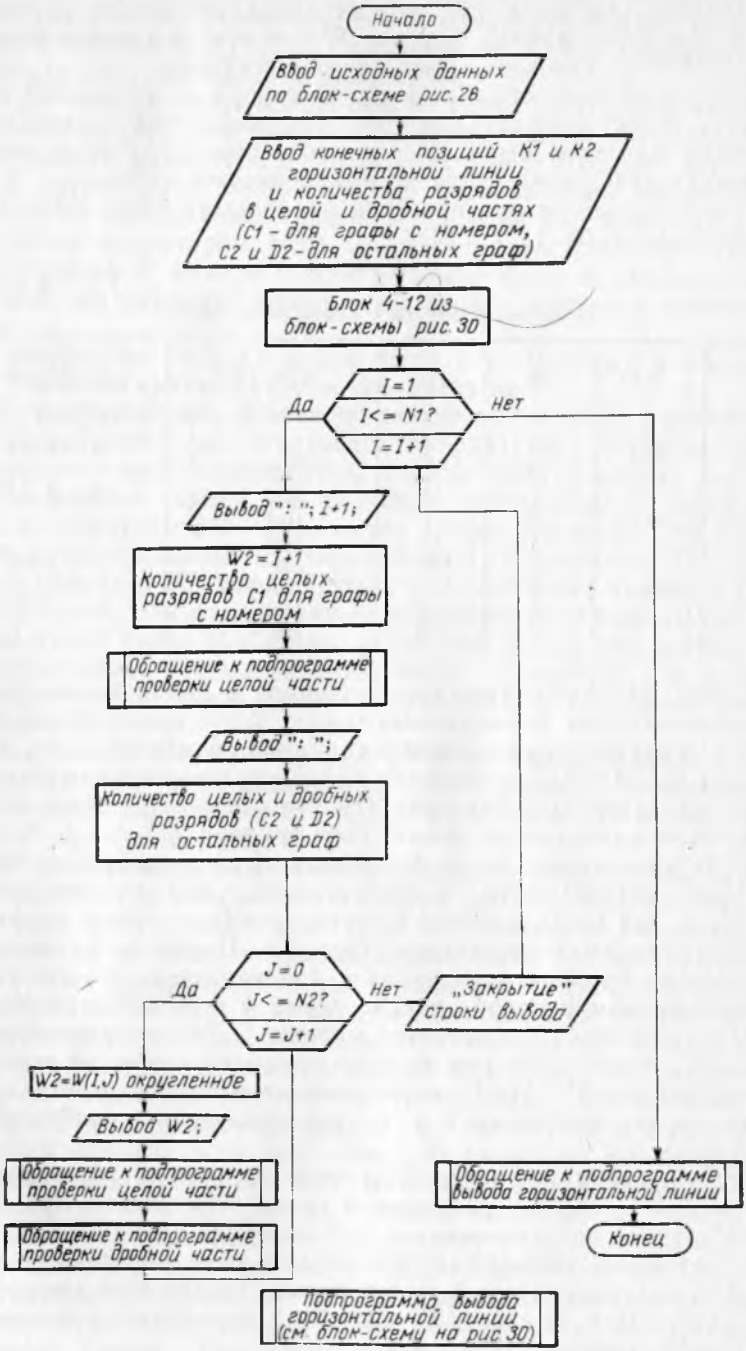
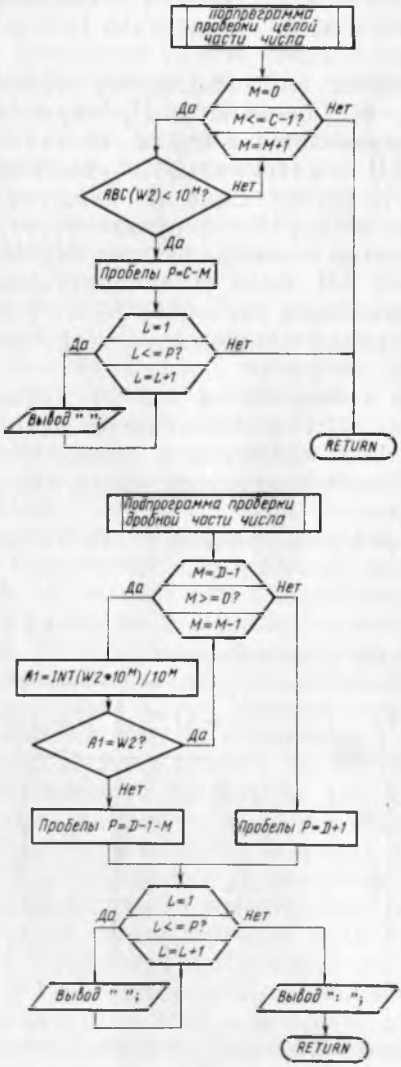


Рис. 31. Блок-схема для компактного вывода информации с



выравниванием чисел по столбцам

отметить, что здесь ответ «Да» получается при конечном значении управляющей переменной M (M не может быть больше C—1=3—1=2). И при ответе «Да» правильно определяется количество пробелов.

Пусть W2=100, т. е. является наименьшим числом с тремя разрядами в целой части. В этом случае при проверке все время возникает ответ «Нет». Выхода из цикла не происходит, и цикл заканчивается по конечному значению управляющей переменной. Это происходит при проверке числа с тремя разрядами в целой части. А для такого числа не требуются компенсирующие пробелы. Следовательно, в этом случае надо просто выходить из подпрограммы.

В соответствии с изложенным алгоритм проверки целой части числа формулируется в следующем виде. Проверяемое число W2 последовательно сравнивается в цикле со значениями 10^M, где M=0, 1, ..., C—1 (C — максимально возможное количество разрядов в целой части числа). Если при этих значениях M выполняется неравенство $W2<10^M$, то производится выход из цикла. При этом количество компенсирующих пробелов P определяется разностью P=C—M. Если указанное неравенство не выполняется при всех возможных значениях M, то количество целых разрядов в числе равно максимально возможному, и компенсирующие пробелы не требуются.

Подпрограмма позволяет проверять не только положительные, но и отрицательные числа. Это обеспечивается использованием стандартной функции ABS, описанной в приложении. Эта функция вырабатывает абсолютное значение числа, используемого в качестве его аргумента.

Работа подпрограммы проверки дробной части числа поясняется с помощью табл. 12.

Таблица 12

Проверка разрядов в дробной части числа

D	W2	M	A1	W2=A1	D — 1 — M=P?
3	1,001	2	1	Нет	3 — 1 — 2=0
	1,010	2	1,01	Да	
		1	1	Нет	3 — 1 — 1=1
	1,100	2	1,1	Да	
		1	1,1	Да	
		0	1	Нет	3 — 1 — 0=2
	1,000	2	1	Да	
		1	1	Да	
		0	1	Да	

В табл. 12 D — количество разрядов в дробной части числа, установленное для проверяемых чисел; W2 — проверяемое число; M — управляющая переменная цикла, изменяющаяся от

D—1 до 0 в убывающем порядке; A1=INT(W2*10↑M)/10↑M— значение W2 при M разрядах в дробной части; D—1—M=P — количество пробелов, компенсирующих недостающие разряды. Значение D=3 показывает, что в дробной части числа максимально может быть 3 разряда.

Проверка производится на совпадение двух чисел. Второе число получается из проверяемого последовательным отбрасыванием по одному дробному разряду. Проверка начинается с отбрасывания максимально возможного разряда и заканчивается отбрасыванием разряда десятых долей. Если в отброшенном разряде был не нуль, то второе число не будет равно исходному. В противном случае числа будут равны, и требуется такая же проверка для оставшихся дробных разрядов.

Пусть проверяемое число W2=1,001, т. е. имеет наименьшую дробную часть с тремя разрядами. При отбрасывании последнего разряда получается A1=1. Сравнение чисел на равенство дает ответ «Нет». Следовательно, отброшен значащий разряд. Исходное число имеет все дробные разряды, и компенсирующие нули не нужны. Это показывается значением P=0.

Пусть проверяемое число W2=1,010, т. е. содержит нуль в последнем разряде. При отбрасывании одного разряда получается A1=1,01. Сравнение чисел на равенство дает ответ «Да». Следовательно, отброшен нулевой разряд. Требуется дальнейшая проверка оставшихся дробных разрядов. При отбрасывании еще одного дробного разряда получается A1=1. Сравнение чисел на равенство дает ответ «Нет». Следовательно, теперь уже отброшен значащий разряд. Компенсирующий пробел должен быть всего один, что определяется значением P=1.

Аналогично проводится проверка при W2=1,100. Здесь ответ «Нет» получается при конечном значении управляющей переменной. Но это все равно позволяет вычислить количество пробелов по тому же выражению и получить ответ P=2.

При W2=1,000, т. е. в случае отсутствия дробной части, отбрасывание всех дробных разрядов при сравнении на равенство дает ответ «Да». Выхода из цикла не происходит, и цикл заканчивается просто исчерпанием значений управляющей переменной. В этом случае требуется компенсация пробелами всего возможного количества дробных разрядов, которое равно D, а также компенсация отсутствующей десятичной точки. Следовательно, при исчерпании цикла количество компенсирующих пробелов определяется выражением P=D+1.

В соответствии с изложенным алгоритм проверки дробной части числа формулируется в следующем виде. Проверяемое число W2 сравнивается в цикле со значениями A1, получаемыми из W2 последовательным отбрасыванием по одному дробному разряду. При этом количество дробных разрядов M, сохраняемых в A1, последовательно уменьшается от D—1 до 0. Если при этих значениях M значение A1 становится не равным W2, то, следовательно, был отброшен разряд с ненулевой цифрой.

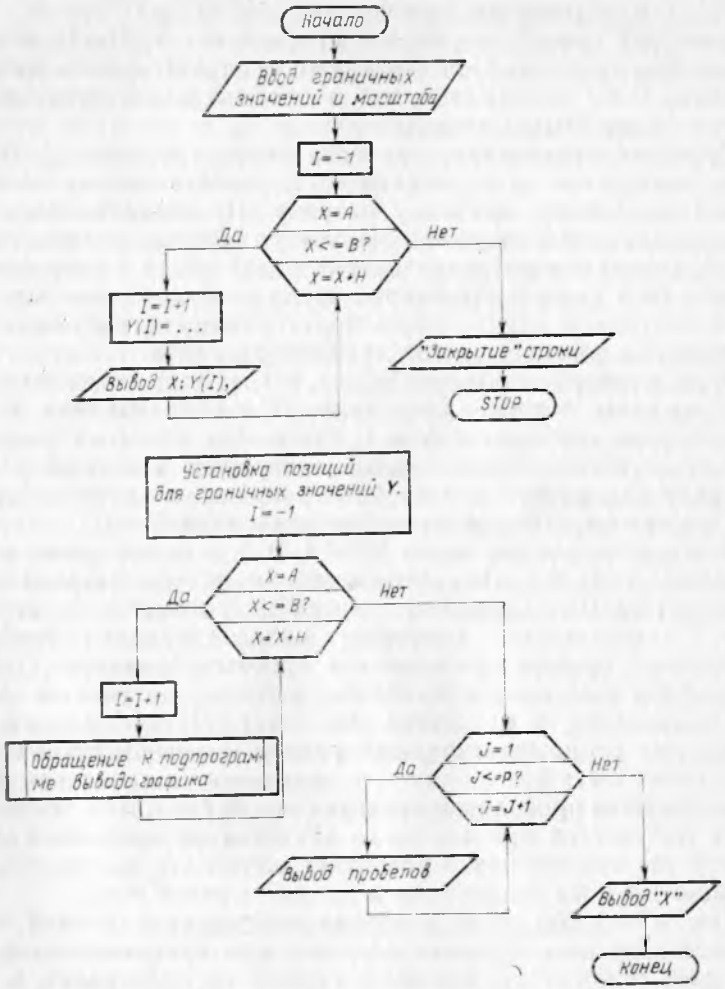
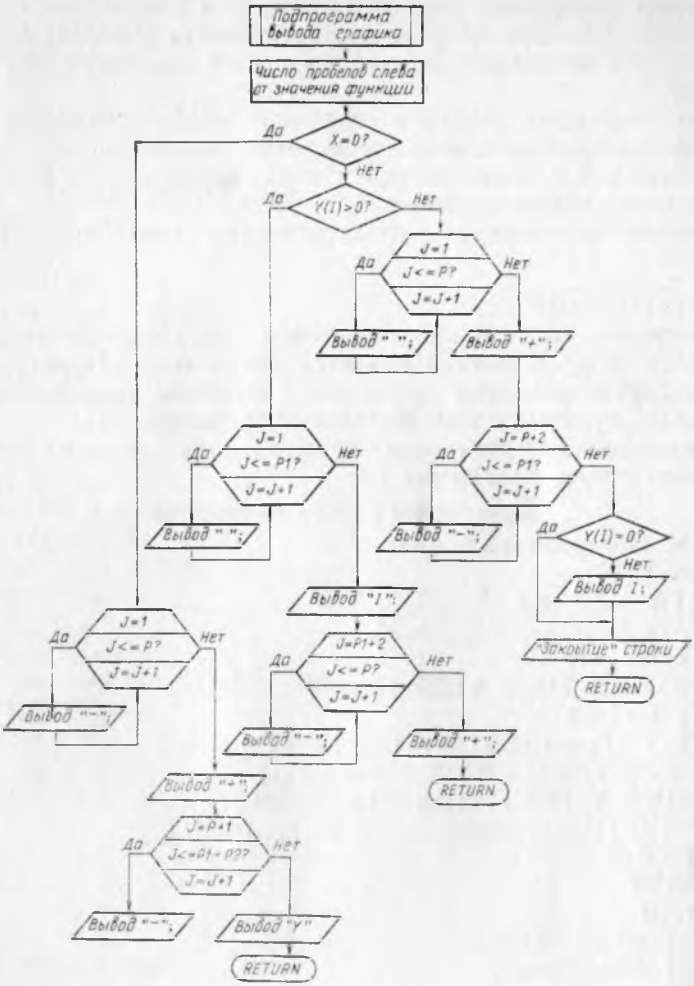


Рис. 32. Блок-схема

Тогда производится выход из цикла. При этом количество компенсирующих пробелов определяется выражением P=D—1—M. Если же цикл заканчивается исчерпанием значений управляющей переменной (т. е. отбрасыванием всех дробных разрядов) и при этом A1 остается равным W2, то это означает отсутствие у числа W2 дробных разрядов. В этом случае отсутствует и десятичная точка, а количество компенсирующих пробелов определяется выражением P=D+1.

9. ПОСТРОЕНИЕ ГРАФИКОВ

При построении графиков следует учитывать размеры экрана и способ вывода машинной информации по строкам. Обычно



для построения графика

графики строятся при равномерном изменении аргумента x с шагом h на некотором заданном интервале значений x от a по b. В этом случае удобно положительную полуось аргумента x направлять вертикально вниз, а положительную полуось функции y — горизонтально вправо. Полуоси оказываются повернутыми на 90 градусов по часовой стрелке относительно их обычного расположения.

Масштабы изображений по осям координат выбираются такими, чтобы диапазон изменения x укладывался примерно в 20 строк экрана, а диапазон изменения y — в 50 — 70 позиций строки. Следует учитывать, что расстояние между позициями в строке меньше, чем расстояние между строками. Поэтому для

получения одинаковой шкалы по осям x и y расстоянию между соседними строками должно соответствовать расстояние между несколькими позициями (в зависимости от конкретных размеров экрана).

Для отдельных участков графика можно создавать более крупные изображения. Это достигается заменой граничных значений для x и y, шага по оси x и масштаба по оси y на соответствующие новые значения.

В качестве примера рассматривается построение графика функции

$$y=\sqrt[5]{x(x^2-n)^2}$$

на интервале $a \leqslant x \leqslant b$ при изменении аргумента x на шаг h. При $a<0$ и $b>0$ пример является достаточно общим, так как в этом случае значения аргумента и функции принимают отрицательные, нулевые и положительные значения.

Блок-схема для построения графика представлена на рис. 32. Ей соответствует программа 29.

Программа 29 и ее выполнение

```
10 REM Построение графика
20 DIM Y(20)
30 DATA —3.5, 3.5, .5, —3, 3, 5
40 READ A, B, H, Y1, Y2, M
50 LET I=—1
60 FOR X=A TO B STEP H
70 LET I=I+1
80 LET Y (I)=(ABS(X)*(X*X—4)↑2)↑.2
90 IF X<0 THEN LET Y (I)=—Y(I)
100 PRINT X; INT (Y(I)*100+.5)/100,
110 IF INT((I+1)/4)*4=I+1 THEN PRINT
120 NEXT X
130 PRINT
140 STOP
150 LET P1=—Y1*M
160 LET P2=Y2*M
170 LET I=—1
180 FOR X=A TO B STEP H
190 LET I=I+1
200 GO SUB 1710
210 NEXT X
220 FOR J=1 TO P1
230 PRINT « »;
240 NEXT J
250 PRINT «X»
1000 STOP
1710 REM Вывод графика
1720 LET P=INT((Y(I)—Y1)*M+.5)
1730 IF X=0 GO TO 1950
1740 IF Y(I)>0 GO TO 1850
```

```
1750 FOR J=1 TO P
1760 PRINT «  »;
1770 NEXT J
1780 PRINT «+»;
1790 FOR J=P+2 TO P1
1800 PRINT «—»;
1810 NEXT J
1820 IF Y(I)=0 GO TO 1840
1830 PRINT «I»;
1840 PRINT
1845 RETURN
1850 FOR J=1 TO P1
1860 PRINT «  »;
1870 NEXT J
1880 PRINT «I»;
1890 FOR J=P1+2 TO P
1900 PRINT «—»;
1910 NEXT J
1920 PRINT «+»
1930 RETURN
1950 FOR J=1 TO P
1960 PRINT «—»;
1970 NEXT J
1980 PRINT «+»;
1990 FOR J=P+1 TO P1+P2
2000 PRINT «—»;
2010 NEXT J
2020 PRINT «Y»
2030 RETURN
RUN
   —3.5 —2.9     —3  —2.37    —2. 5  —1.66     —2  0
   —1.5 —1.36    —1  —1.55    —. 5   —1.48      0  0
     .5   1.48     1    1.55     1. 5   1.36     2  0
    2.5   1.66     3    2.37     3. 5   2.99
ОСТ СТРОКЕ 140
ЖДУ
GO TO 150
```

Далее следует график, представленный на рис. 33.

В данной программе вычисления проводятся при значении n=4. Граничные значения для X задаются в виде переменных A и B. Величина изменения переменной X определяется шагом H. Граничные значения для Y задаются в виде переменных Y1 и Y2. Масштаб по оси Y определяется переменной M. Массив значений Y предусматривает возможность хранения 21 величины, что обычно более чем достаточно.

Построение графика проводится для значений X, изменяющихся от A до B с шагом H. Логично организовать цикл со значением X в качестве управляющей переменной этого цикла

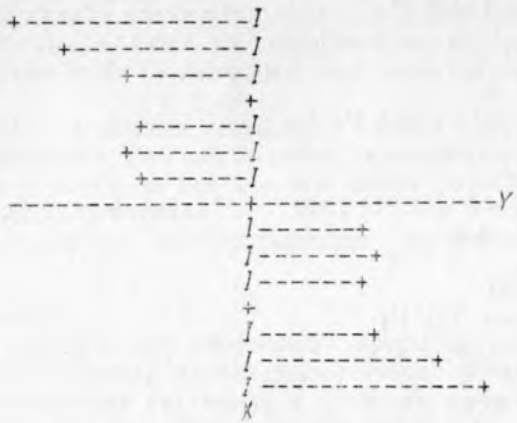


Рис. 33. График функции

(операторы 60 и 120). При этом значение индекса I для функции Y должно изменяться одновременно с изменением значения X, что достигается наращиванием предыдущего значения индекса на единицу. Начальное значение индекса I должно равняться —1, чтобы первое значение индекса было равно нулю (операторы 50 и 70).

В данном примере функция Y вычисляется с использованием операции возведения в дробную степень. Эта операция производится только над положительными числами. Поэтому здесь используется абсолютное значение аргумента X (оператор 80), а знак числа учитывается в последней операции, определяющей значение функции (оператор 90). Используемая здесь стандартная функция ABS описана в приложении.

Результаты вычислений выводятся с двумя знаками после запятой (с округлением), что определяется оператором 100. При этом используется стандартная функция INT, описанная в приложении.

В программе поставлено произвольное условие вывода только четырех пар чисел в отдельных зонах. Для «закрытия» пятой зоны используется оператор 110. Его работа основана на выявлении чисел, кратных четырем (желаемому количеству выводимых на строке пар). Это количество на единицу больше индекса для Y, так как индексация начинается с нуля.

Оператор 140 останавливает вычисления. Это позволяет выписать значения аргумента X и функции Y, выводимые оператором 100. Для продолжения вычислений задается в непосредственном режиме команда GO TO 150.

Перед выводом графика устанавливается количество позиций P1 и P2, выделяемых с учетом масштаба для представления граничных значений Y (операторы 150 и 160). После этого производится задание конкретных значений X и Y и обращение

(по оператору 200) к подпрограмме вывода этих значений на экран в виде графика.

После вывода всего графика выводится обозначение оси X (операторы 220 — 250). На этом работа программы заканчивается (оператор 1000).

Работа подпрограммы вывода графика (операторы 1710—2030) начинается с определения числа позиций P, предшествующих значению функции (оператор 1720). Если X=0, то это соответствует положению оси Y. В этом случае операции вывода определяются операторами 1950—2030. В противном случае выводятся пробелы, предшествующие значению Y (операторы 1750—1770).

Значение Y рекомендуется представлять каким-либо символом, имеющим точно определенную точку в центре позиции вывода. В данном случае используется знак плюс (оператор 1780).

В данной программе от значения Y до оси X проводится горизонтальная линия с помощью оператора 1800. Эту линию можно отменить, заменив в операторе 1800 символ минуса на символ пробела. В операторе 1790 начальное значение для J выбрано равным P+2. Это объясняется тем, что P позиций занимают пробелы перед значением функции и еще одну позицию занимает символ, представляющий значение функции. Следующая за значением функции позиция имеет индекс P+2.

Если Y=0, то символ, представляющий значение Y, попадает в позицию символа, представляющего положение оси X. В этом случае символ оси X опускается, что определяется оператором 1820.

Работа подпрограммы при Y>0 и при X=0 аналогична разобранному случаю ее работы при Y, меньшем или равном нулю. Разница состоит лишь в граничных значениях управляющей переменной цикла и в виде символа, выводимого на экран. По логике же действий участки программы, начинающиеся с номеров строк 1750, 1850 и 1950, аналогичны между собой. Именно для подчеркивания этого обстоятельства сохранено соответствие в двух последних разрядах номеров строк между аналогичными операторами путем введения номера 1845 (не кратного десяти) и исключения номера 1940.

ПРИЛОЖЕНИЯ

1. СТАНДАРТНЫЕ ФУНКЦИИ БЕЙСИКА

Стандартной называется функция, вычисление которой входит в программное обеспечение ЭВМ. Список таких функций приведен в табл. 13.

Использование стандартных функций аналогично использованию функций, определяемых пользователем (см. п. 6 гл. 5).

Таблица 13

Стандартные функции

Функция	Назначение функции
SIN(X)	Синус X, X в радианах (sin x)
COS(X)	Косинус X, X в радианах (cos x)
ATN(X)	Арктангенс X (arc tg x)
SQR(X)	Корень квадратный из X, $X \geqslant 0$ ($\sqrt{x}$ — Square root)
EXP(X)	Показательная функция e^x (Exponential function)
LOG(X)	Натуральный логарифм $\log_e x = \ln x$, $x > 0$
ABS(X)	Абсолютное значение (модуль) X ($\|x\|$)
INT(X)	Целочисленная функция от X (Integer function)
RND(X)	Случайное число между 0 и 1 (Random function)
SGN(X)	Сигнум X, $\text{Sign } x = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$

Фактический параметр, подставляемый на место формального параметра или аргумента X, может представляться константой, именем переменной, функцией или арифметическим выражением. Сами функции тоже могут использоваться в арифметических выражениях. Функции вычисляются с точностью до $\pm 10^{-8}$ (кроме четырех последних, к которым это понятие неприменимо).

Первые четыре функции (SIN, COS, ATN, SQR) называются **расширенными функциями**. Они могут стираться в памяти после загрузки интерпретатора Бейсик, что освобождает для программы пользователя дополнительно 300 слов памяти. Отказ от функций EXP и LOG освобождает дополнительно еще 250 слов памяти.

Отказ от этих стандартных функций осуществляется при подготовке ДВК-1 к работе. После набора 140000G в ответ на при-

глашение @ 0 набирается знак вопроса и нажимается клавиша «ВК». После этого ЭВМ последовательно выдает вопросы, на которые надо отвечать набором ответа «Да» или «Нет» и нажатием клавиши «ВК» или просто нажатием клавиши «ВК» без всякого ответа. В последнем случае производятся стандартные действия ЭВМ, сохраняющие все функции.

1.1. Функции SIN(X) и COS(X)

При вычислении функции необходимо обратить внимание, что аргумент этих функций задается в радианах, а не в градусах. В том случае, когда аргумент задан в градусах, можно использовать подпрограмму перевода из градусов в радианы, приводимую ниже.

Для вычисления других прямых тригонометрических функций используются соотношения, определяющие их через значения данных стандартных функций

$$\operatorname{tg} x = \frac{\sin x}{\cos x}$$

$$\operatorname{ctg} x = \frac{\cos x}{\sin x}$$

$$\sec x = \frac{1}{\cos x}$$

$$\operatorname{cosec} x = \frac{1}{\sin x}$$

При задании аргумента в градусной мере (с минутами и секундами) требуется предварительный перевод этого аргумента в радианы. При этом удобно представлять такой аргумент в искусственной десятичной форме: градусы представлять в виде целой части аргумента, минуты — в виде первого и второго дробных разрядов, секунды — в виде третьего и четвертого дробных разрядов, а десятые, сотые и тысячные доли секунды — последующими дробными разрядами, соответственно.

Пример подпрограммы, осуществляющей такой перевод из градусов в радианы, приведен в программе 30.

Программа 30 и ее выполнение

```
10 REM Вычисление тангенса при угле в градусах
20 DATA 0, 12.34567, 30, 45, 60
30 FOR I=1 TO 5
40 READ G1
50 GO SUB 2110
60 PRINT G1, R, SIN(R)/COS(R)
70 NEXT I
1000 STOP
2110 REM Перевод из градусов в радианы
```

```
2120 LET G2=ABS(G1)
2130 LET G=INT(G2)
2140 LET M1=(G2—G) ∗100
2150 LET M=INT(M1)
2160 LET S=(M1—M)∗100
2170 LET R=(G∗3600+M∗60+S)∗ 3.141593/648000
2180 IF G1<0 THEN LET R=—R
2190 RETURN
RUN
 0            0            0
 12.34567     .2196046     .2232043
 30           .5235988     .5773503
 45           .7853982     1
 60           1.047198     1.732051
```

В первом столбце приводится искусственное десятичное представление угла в градусах, во втором — значение угла в радианах и в третьем — значение тангенса угла. Второе значение угла в градусах расшифровывается так: 12 градусов, 34 минуты, 56 секунд и 7 десятых секунды. Учет того, что один градус равен $\pi/180$ радиан, а одна минута равна 1/60 градуса и 1 секунда равна 1/60 минуты производится в операторе 2170 после того, как в операторах 2130 — 2160 искусственное десятичное представление исходного значения разделяется на градусы, минуты и секунды.

1.2. Функция ATN(X)

Эта функция вычисляет для значений аргумента X величину угла в радианах в интервале от $-\pi/2$ до $+\pi/2$.

Для вычисления других обратных тригонометрических функций используются соотношения, определяющие их через значения арктангенса X

$$\text{arc sin } x = \text{arc tg } \frac{x}{\sqrt{1-x^2}}$$

$$\text{arc cos } x = \text{arc tg } \frac{\sqrt{1-x^2}}{x}$$

$$\text{arc ctg } x = \text{arc tg } \frac{1}{x}$$

$$\text{arc sec } x = \text{arc tg } \sqrt{x^2-1}$$

$$\text{arc cosec } x = \text{arc tg } \frac{1}{\sqrt{x^2-1}}$$

Для получения величины угла в градусах в искусственном десятичном представлении (как в предыдущем пункте) можно использовать подпрограмму перевода из радиан в градусы. Пример такой подпрограммы приводится в программе 31.

Программа 31 и ее выполнение

```
10 REM Вычисление арктангенса с переводом ответа в градусы
20 DATA 0, .2232043, .5773503, 1, 1.732051
30 FOR I=1 TO 5
40 READ X
50 LET R=ATN(X)
60 GO SUB 2210
70 PRINT X, R, G
80 NEXT I
1000 STOP
2210 REM Перевод из радиан в градусы
2220 LET G1=ABS(R)*180/3.141593
2230 LET G=INT(G1)
2240 LET M1=(G1—G)*60
2250 LET M=INT(M1)
2260 LET S=(M1—M)*60
2270 LET G=G+M/100+S/10000
2280 IF R<0 THEN LET G=—G
2290 RETURN
RUN
0           0           0
.2232043    .2196046    12.34567
.5773503    .5235988    29.596
1           .7853982    44.596
1.732051    1.047198    59.596
```

В первом столбце приводится значение тангенса угла, во втором — величина угла в радианах и в третьем — искусственное десятичное представление величины угла в градусах. Три последних числа в третьем столбце в дробной части представляют значение 59 минут и 60 секунд. Это эквивалентно 1 градусу. Следовательно, три последних числа представляют углы в 30, 45 и 60 градусов, соответственно.

1.3. Функции EXP(X) и LOG(X)

Показательная функция EXP(X) возводит число e=2,718282 в степень аргумента X.

Логарифмическая функция LOG(X) определяет натуральный логарифм аргумента X при основании e. Для перехода к логарифмам с другим основанием используется формула

$$\log_a x = \frac{\log_e x}{\log_e a},$$

где a — основание, к которому происходит переход.

1.4. Функция INT(X)

Эта функция определяет наибольшее целое значение, не превышающее алгебраически значение аргумента. Для положитель-

ных чисел это равносильно отбрасыванию дробной части аргумента, а для отрицательных — увеличению модуля дробного значения аргумента до ближайшего целого значения при движении влево по числовой оси. Другими словами, эта функция дает ближайшее целое число, лежащее на числовой оси слева от аргумента X.

Например:

```
PRINT INT(1.2), INT(1.9), INT(—1.2), INT(—1.9).
        1         1         —2         —2
```

Для округления числа до ближайшего целого аргумент функции задается в виде (X+.5).

Например:

```
PRINT INT(1.2+.5), INT(1.9+.5)INT(—1.2+.5), INT(—1.9+.5).
        1            2           —1             —2
```

В общем случае для округления до любого дробного или целого разряда аргумент функции задается в виде

(X*10↑D+.5)/10↑D,

где D>0 определяет номер округляемого дробного разряда,
D=0 определяет округление целого разряда единиц,
D=—1 определяет округление разряда десятков,
D=—2 определяет округление разряда сотен и т. д.

Например:

```
10 FOR A=87.65 TO 0 STEP —75.31
20 PRINT A;
30 FOR D=1 TO — 3 STEP —1
40 LET B=INT (A*10↑D+.5)/10↑D
50 PRINT «  »D; B;
60 NEXT D
70 PRINT
80 NEXT A
RUN
87.65  1 87.7  0 88  —1 90  —2 100  —3 0
12.34  1 12.3  0 12  —1 10  —2 0    —3 0
```

1.5. Функция RND(X)

Эта функция вырабатывает случайное число в интервале между 0 и 1. Аргумент X не используется и может представляться любым числом. Сама функция RND(X) выбирает одно и то же начальное значение для случайных чисел и воспроизводит их в одном и том же порядке. Применение перед этой функцией оператора RANDOMIZE (англ. — рандомизировать, перемешивать), состоящего только из одного этого слова, вызывает выбор нового начального значения для случайных чисел.

Простейший пример использования функции RND(X).

```
10 FOR I=1 TO 5
20 PRINT RND(0);
30 NEXT I
RUN
.1002502 .9648132 .8866272 .6364441 .8390198
```

Применение функции INT и масштабного множителя позволяет получать округленные значения и в расширенном диапазоне чисел.

Для получения целых случайных чисел в диапазоне от 0 до K обращение к функции RND(X) задается в виде:

INT((K+1)*RND(X))

В следующем примере получаются случайные цифры от 0 до 9.

```
20 FOR I=0 TO 2
40 FOR J=1 TO 15
60 PRINT INT (10*RND(0));
70 NEXT J
80 PRINT
90 NEXT I
RUN

 1  9  8  6  8  3  2  9  1  4  0  5  2  7  4
 7  1  6  4  8  0  4  2  4  0  5  9  6  2  3
```

Продолжение вычислений по команде GO TO 20 приведет к новой последовательности цифр, а выполнение программы заново по команде RUN — к повторению той же самой исходной последовательности. Применение оператора RANDOMIZE вызывает изменение начального значения последовательности и получение новых последовательностей случайных чисел, в том числе и по команде RUN. Рекомендуется просмотреть действие этого оператора, последовательно вводя его в строки 10, 30 и 50 этого примера.

Для получения случайных чисел в интервале между A и B используется выражение

(B—A)*RND(0)+A,

а для получения целых положительных чисел в этом интервале (без верхнего граничного значения) — выражение

INT ((B—A)*RND(0)+A).

В качестве примера можно получить 20 различных комбинаций по 5 целых чисел в диапазоне от 1 до 36, для спортлото 5 из 36. Они получаются в программе 32.

Программа 32 и ее выполнение

```
10 REM Спортлото 5 из 36
20 DIM S(4)
30 FOR I=0 TO 20
40 RANDOMIZE
50 LET J=0
60 LET S(0)=INT (36*RND(0)+1)
70  PRINT S(0);
80 LET A=INT (36*RND(0)+1)
90 FOR K=0 TO J
100 IF A=S(K) GO TO 80
110 NEXT K
120 LET J=J+1
130 LET S(J)=A
140 PRINT S(J);
150 IF J<4 GO TO 80
160 PRINT
170 NEXT I
180 PRINT «Продолжать? A=»;
190 INPUT A
200 IF A=1 GO TO 30
210 STOP
RUN
 11   25   17   23   21
 7   34   35   13   1
  .  .  .  .  .  .  .  .
 5  34   23   10   1
Продолжать? A=?1
 25   12   27   20   23
 22   25   35   26   24
  .  .  .  .  .  .  .  .
 19   25   12   32   15
Продолжать? A=?0
ОСТ СТРОКЕ 210
ЖДУ
```

После вывода партии из 20 комбинаций на экране появляется фраза

Продолжать? A=?

Для продолжения вычислений, т. е. для вывода новой партии из 20 комбинаций, необходимо ввести 1 и нажать клавишу «ВК». В соответствии с оператором 200 в этом случае управление передается оператору 30, что приводит к выводу новой партии. После этого повторяется та же фраза машины.

Для прекращения вычислений можно ввести любую цифру, например 0, и нажать клавишу «ВК». В этом случае управление передается на оператор останова.

Формат оператора	Назначение оператора
DATA значение-переменной-1 [, значение-переменной-2]...	Представление значений данных, вводимых оператором READ.
DEF FNI (параметр) = арифметическое-выражение	Функция, определяемая пользователем.
DIM имя-переменной-1 (n_1[, m_1]) [, имя-переменной-2 (n_2[,m_2])]...	Выделение места в оперативной памяти для элементов массивов.
END	Обозначение физического конца программы.
FOR имя-переменной = арифметическое-выражение-1 TO арифметическое-выражение-2 [STEP арифметическое-выражение-3]	Определение заголовка цикла.
NEXT имя-переменной	Обозначение конца цикла.
GO TO номер-строки	Передача управления первому оператору в строке с указанным номером.
GO SUB номер-строки	Передача управления подпрограмме, начинающейся со строки с указанным номером.
RETURN	Возврат из подпрограммы в основную программу на следующий за GO SUB оператор.
IF условие [THEN] { номер-строки / GO TO номер-строки / оператор }	При выполнении условия, стоящего за словом, IF выполняет то, что указано за словом THEN. В противном случае выполняется следующий за IF оператор.
INPUT имя-переменной-1 [, имя-переменной-2]...	Ввод в указанные переменные данных с клавиатуры в процессе счета.
LET имя-переменной = арифметическое-выражение	Вычисление арифметического выражения и присвоение его значения указанной переменной.
NEXT...	См. после оператора FOR.

Продолжение

Формат оператора	Назначение оператора
PRINT [«символ»... [{ ; ··· }]]... [имя-переменной [{ ; ··· }]]... [арифметическое-выражение [{ ; ··· }]]...	Вывод информации на экран.
RANDOMIZE	Рандомизация (перемешивание) случайных величин (см. «Стандартная функция RND(X)» в приложении).
READ имя-переменной-1 [, имя-переменной-2]...	Ввод в указанные переменные значений данных из операторов DATA.
REM символ...	Разрешение на вставку в программу комментариев.
RESTORE	Восстановление для считывания оператором READ значений во всех предшествующих RESTORE операторах DATA.
RETURN	См. за оператором GO SUB.
STOP	Останов вычислений.

3. КОДЫ ОШИБОК

Код	Вид ошибки
0	Переполнение памяти, отведенной пользователю.
1	Нераспознаваемый оператор.
2	Недопустимый оператор GO TO или GO SUB.
3	Недопустимый знак, ограничивающий оператор и вызывающий преждевременное окончание его действия.
4	Выход на оператор RETURN без оператора GO SUB.
5	Неправильно сформирован индекс.
6	Индекс не попадает в интервал от 0 до 255 или превышает максимум, установленный в программе.
7	Несоответствие скобок в операторе.
8	Недопустимый оператор LET.
9	Недопустимый знак операции отношения в операторе IF.
10	Недопустимый оператор IF.
11	Недопустимый оператор PRINT.
12	Вводимая строка слишком длинна (превышает 80 знаков).
13	Неправильная размерность в операторе.
14	В памяти недостаточно места для массива.
15	Неправильно сформирован оператор DEF.
16	Недопустимый номер строки или значение размерности.
17	Оператор DIM для ранее описанного или использованного элемента.
18	Неправильная переменная в списке оператора INPUT.
19	Неправильная переменная в списке оператора READ.
20	Данные для оператора READ исчерпаны.
21	Неправильный формат оператора DATA.
22	Недопустимый оператор FOR.
23	Оператор FOR не сопровождается соответствующим оператором NEXT.
24	Оператор NEXT без оператора FOR.
25	Несоответствие кавычек в операторе.
26	Неправильно установлена функция EXF.
27	Неправильно сформировано выражение (пропущен порядок числа в формате E).
120	Недопустимые знаки при вводе.
121	Введено недостаточно данных по оператору INPUT.
122	Введено слишком много данных по оператору INPUT.
123	Несуществующая переменная.
124	Число слишком велико для фиксации (вероятно, комбинация индексов превышает диапазон).
125	Переполнение или заем при делении или умножении.
126	Квадратный корень из отрицательного числа.
127	Логарифмы отрицательного числа или нуля; переполнение при вычислении EXP.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Станислав Тихонович Усачев

Персональная ЭВМ ДВК-1

Учебное пособие

Редактор *Е. Е. Митина*
Художественный редактор *В. Ю. Яковлев*
Технический редактор *Ю. В. Михалева*
Корректоры *В. Ф. Демидова, Н. В. Коврижных, И. В. Волкова*

ИБ № 575

Сдано в набор 25.05.88. Подписано к печати 23.12.88. Л-67862. Формат $60\times90^{1}/_{16}$. Бумага оберточная. Гарнитура литературная. Печать высокая. Усл. печ. л. 9,5. Усл. кр.-отт. 9,875. Уч.-изд. л. 9,23. Тираж 60 000 экз. Изд. № 1118. Заказ 153. Цена 40 коп.

Издательство Университета дружбы народов
117923, Москва, ул. Орджоникидзе, 3.

ПО «Чертановская типография» Управления издательств, полиграфии и книжной торговли Мосгорисполкома
113545, Москва, Варшавское ш., 129а.

40 к.