ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ПРОБНОЕ УЧЕБНОЕ ПОСОБИЕ
для средних учебных заведений
В двух частях

•

ЧАСТЬ ПЕРВАЯ

Под редакцией
А. П. Ершова и В. М. Монахова

Рекомендовано
Главным управлением школ
Министерства просвещения СССР

МОСКВА
«ПРОСВЕЩЕНИЕ» 1985

ББК 73я72
О-75

А. П. ЕРШОВ, В. М. МОНАХОВ, С. А. БЕШЕНКОВ,
Я. Э. ГОЛЬЦ, А. А. КУЗНЕЦОВ, Э. И. КУЗНЕЦОВ,
М. П. ЛАПЧИК, Д. О. СМЕКАЛИН

Основы информатики и вычислительной техники: Проб. О-75 учеб. пособие для сред. учеб. заведений. В 2-х ч. Ч. 1/ А. П. Ершов, В. М. Монахов, С. А. Бешенков и др.; Под ред. А. П. Ершова, В. М. Монахова.— М.: Просвещение, 1985.— 96 с., ил.

В этом учебном пособии даны первоначальные сведения об ЭВМ, их устройстве и применении, о подготовке решения задач на ЭВМ с помощью алгоритмов. В конце каждого параграфа приводятся вопросы для повторения теории и упражнения. Пособие рассчитано для использования учащимися средних специальных учебных заведений, профессионально-технических училищ, средней общеобразовательной школы.

О $\frac{4306020400-516}{103(03)-85}$ инф. письмо

ББК 73я72+32.973я72
001.8(075)+6Ф7.3(075)

ВВЕДЕНИЕ

РОЛЬ ЭВМ В СОВРЕМЕННОМ ОБЩЕСТВЕ

Мы с вами начинаем изучать основы информатики и вычислительной техники. Информатика исследует законы и методы переработки и накопления информации. Эта наука появилась недавно — во второй половине XX в. Ее развитие связано с появлением электронно-вычислительных машин (ЭВМ), мощных универсальных устройств для хранения и переработки информации. ЭВМ называют также компьютерами (от английского слова computer — вычислитель).

Потребность выразить и запомнить информацию привела к появлению речи, письменности и изобразительного искусства. В дальнейшем необходимость передачи и распространения информации вызвала к жизни книгопечатание, почтовую связь. Появление телеграфа, телефона, радио и телевидения позволило передавать огромные потоки текстовой информации и изображений со скоростью света. Однако целенаправленная обработка этой информации до самого последнего времени проводилась только человеком.

Использование ЭВМ позволяет теперь переложить часть этой обработки на автоматические устройства, способные достаточно долго работать без участия человека и со скоростью, в несколько миллионов раз превышающей скорость обработки информации человеком.

Универсальность ЭВМ, ее способность к целенаправленной переработке различных видов информации и объясняют происходящий сейчас стремительный процесс внедрения ЭВМ в самые разные сферы деятельности человека в современном обществе.

Внедрение ЭВМ приводит к коренной перестройке технологии производства во многих отраслях народного хозяйства, повышению производительности и улучшению условий труда людей. Разумеется, внедрение вычислительной техники невозможно без обучения людей, которые будут ее использовать (их часто называют пользователями ЭВМ). Это обучение начинается в школе. В начальной школе вы научились читать и писать — стали грамотными. В IX и X классах вы постигнете азы второй грамотности — компьютерной.

Область применений ЭВМ чрезвычайно широка.

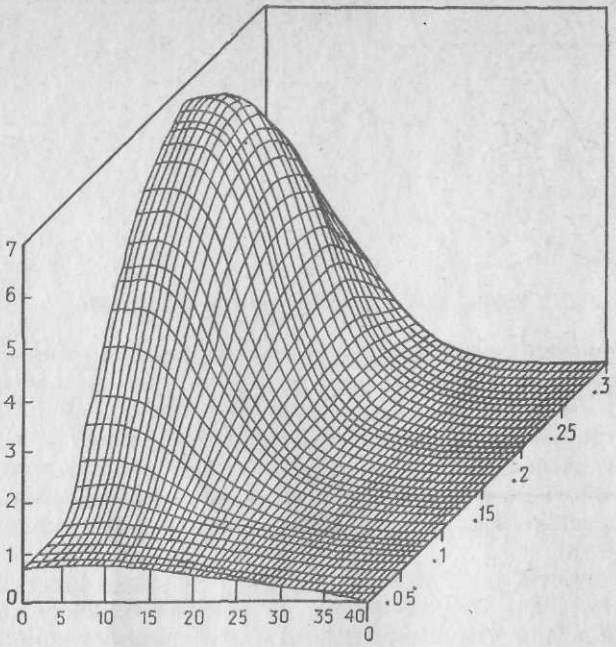


Рис. 1. Результаты математического моделирования плазмы в термоядерном реакторе

Важнейшим средством современного научного исследования является математическое моделирование физических явлений и исследование этих моделей с помощью ЭВМ. Современные компьютеры позволяют, например, проводить сложнейшие расчеты при создании установок термоядерного синтеза или сверхзвуковых самолетов (рис. 1).

Без экспериментов невозможен научно-технический прогресс. Эксперименты с термоядерными реакторами, аэродинамическими трубами, ускорителями очень сложны и дорогостоящи. ЭВМ позволяют заменить реальные эксперименты в тысячи раз более дешевыми машинными, вычислительными экспериментами. Кроме того, такие вычислительные эксперименты часто можно проводить во много раз быстрее, чем настоящие. Наконец, в некоторых областях науки, например в астрофизике, проведение реальных экспериментов и вовсе невозможно. Здесь на помощь исследователю приходит вычислительный эксперимент.

Одной из наиболее трудоемких вычислительных задач является задача прогноза погоды. Для ее решения необходимо собрать и проанализировать информацию, получаемую со спутников и метеостанций, выполнить огромный объем вычислений, необходимых для решения уравнений, возникающих при математическом моделировании процессов в атмосфере и океане, наконец, представить

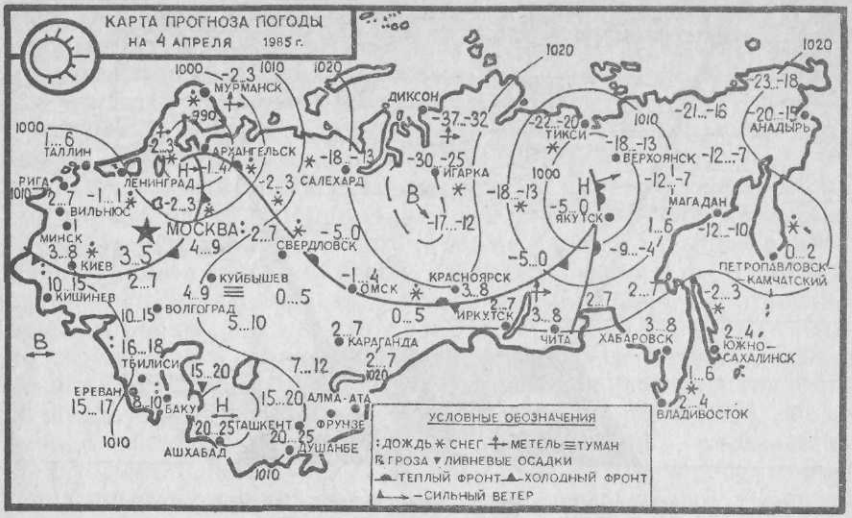


Рис. 2. Сообщает Гидрометцентр СССР

полученные результаты в удобной для человека форме. Все эти этапы немыслимы без применения ЭВМ (рис. 2).

Компьютеры нужны не только для проведения машинных экспериментов, но и для обработки результатов реальных экспериментов. Современный физический или биологический эксперимент часто дает столько информации, что обработать ее без ЭВМ практически невозможно (рис. 3).

Использование ЭВМ позволило в последние годы создать новый метод получения изображения внутренних частей непрозрачных тел. Этот метод называется томографией. Он позволяет получать изображение гораздо лучшего качества, чем рентгеноскопия. Томография основана на способности ЭВМ быстро выполнить сотни миллионов арифметических действий над числами. Именно

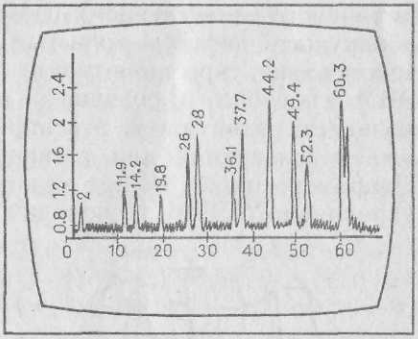


Рис. 3. Результаты математической обработки физического эксперимента на экране графического дисплея

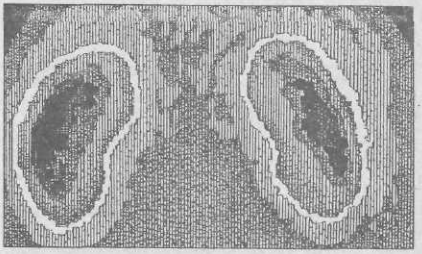


Рис. 4. Компьютерная томограмма, используемая при медицинской диагностике

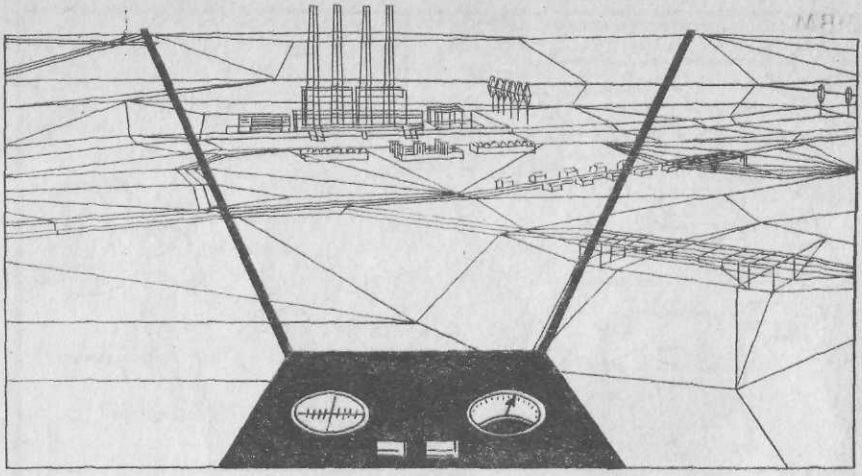


Рис. 5. В системе, обучающей пилота посадке самолета, используется изображение, созданное компьютером

такое количество действий требуется в томографии для получения одного-единственного изображения. Томография позволяет обнаружить дефекты, скрытые в глубине детали, или признаки заболевания, скрытые в тканях человеческого организма (рис. 4). ЭВМ способна и создавать изображения, в том числе движущиеся, «живущие». Мультфильмы, созданные компьютерами, уже используются для тренировки пилотов, водителей и т. д. Например, пилот может с помощью специального тренажера, управляемого ЭВМ, не покидая земли, отработать навыки управления новым самолетом или тренироваться в посадке на полосу нового аэропорта (рис. 5).

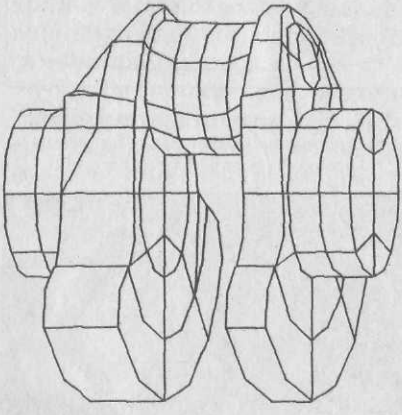


Рис. 6. Часть модели коленчатого вала автомобиля, построенная системой автоматизированного проектирования автомобилей

Графические построения и расчеты с помощью ЭВМ различных машиностроительных, авиационных, автомобильных деталей и конструкций являются составной частью систем автоматизированного проектирования (сокращенно САПР). Такие системы во много раз повышают производительность труда конструктора и сокращают сроки разработки. Сидя перед подключенным к ЭВМ экраном (подобным телевизионному), конструктор автомобиля, например, может скомандовать

ЭВМ изобразить отдельные части и узлы автомобиля, что позволяет оценить внешний вид и компоновку (рис. 6). ЭВМ может также произвести моделирование работы этих узлов и показать их части, где наиболее вероятна поломка при эксплуатации.

Одной из важных областей применения компьютеров являются справочно-информационные системы. Возможно, вы видели на вокзале автоматическую справочную. Она имеет несколько десятков кнопок, около каждой из которых написан вопрос. Нажав на любую из них, можно прочесть ответ на соответствующий вопрос. Такая система может отвечать только на вопросы из заданного списка с помощью заранее подготовленных табличек с ответами. Бывают, однако, и более сложные задачи, которые система такого рода решить не в состоянии. Одна из таких задач, которую нельзя решить без компьютера, возникает при продаже авиабилетов. Эта задача может быть решена с помощью мощной ЭВМ, соединенной с большим числом специальных устройств — терминалов, за которыми работают кассиры. Информация о всех рейсах самолетов, об уже занятых и еще свободных местах на все эти рейсы хранится в памяти ЭВМ. Эта информация должна оперативно меняться при каждой покупке билета, при каждом назначении дополнительного рейса и т. д. С другой стороны, эта информация или некоторый результат ее переработки должны по требованию кассира быстро выдаваться на экран терминала. Наконец, когда пассажир выбрал нужный рейс или маршрут с пересадками, печатание соответствующего авиационного билета тоже должно проводиться автоматически (рис. 7). Со всеми этими задачами успешно справляется автоматизированная система продажи авиабилетов «Сирена-2», услугами которой вам еще доведется воспользоваться.

Информационные системы, построенные на базе ЭВМ, могут оказаться полезными не только при покупке авиабилетов, но и при решении многих других задач. Они могут помочь в разных иссле-

Рис. 7. Билет, выданный автоматизированной системой управления продажи авиабилетов «Сирена-2»

дованиях, например в химии. Если химик прошлого века мог помнить свойства интересующего его химического соединения или найти их в справочнике, то сегодня это немыслимо: исследовано громадное число различных химических веществ, описания которых заняли бы сотни томов. Найти описание конкретного вещества в этих томах еще можно, но найти описание вещества, обладающего заданными свойствами, или решить какую-нибудь другую задачу поиска нужной нам информации вручную уже невозможно. На помощь приходят компьютеры. С их помощью можно создать специальным образом организованное хранилище информации, так называемую «базу данных», содержащую сведения об известных к настоящему времени химических веществах. После организации «базы данных» компьютер может работать в качестве так называемой информационно-поисковой системы. Эта система способна практически мгновенно ответить химику на интересующие его вопросы: известно ли данное химическое вещество, какими свойствами оно обладает, какие другие химические вещества по свойствам наиболее близки к данному и т. д.

Информационно-поисковые системы находят широкое применение во многих областях науки. Например, они интенсивно используются в молекулярной биохимии при анализе структур генов, входящих в ДНК.

Выше мы уже говорили о системах автоматизированного проектирования, многократно повышающих производительность труда конструктора. Но после успешного окончания проектирования детали ее нужно еще изготовить. Процесс изготовления деталей на станках также можно автоматизировать с помощью ЭВМ. Станок, управляемый ЭВМ, называется станком с числовым программным управлением (ЧПУ). Для того чтобы обработать деталь на станке с ЧПУ, человек должен прежде всего написать программу обработки этой детали, т. е. описать последовательность элементарных операций, в результате выполнения которых деталь будет обработана. После того как эта программа будет написана и введена в память ЭВМ, станок с ЧПУ может обработать деталь автоматически, без участия человека. По одной и той же программе можно обработать серию одинаковых деталей. Для перехода к обработке другой детали нужно будет лишь сменить программу в памяти ЭВМ, управляющей станком.

В наше время значительная часть предназначенной для человека информации создается, передается и воспринимается в виде текстов. Тексты обычно пишутся или печатаются на бумаге, таковы книги, газеты, ваши классные и домашние задания, письма, деловая документация и т. д. Работы по составлению, написанию и оформлению текстов весьма трудоемки. Хорошо подготовленный и оформленный текст легко воспринимается человеком, плохо подготовленный текст трудно понять. ЭВМ успешно используются для подготовки и корректирования (оформления) текстов. Стоит ввести текст в память ЭВМ, и дальше исправление опечаток,

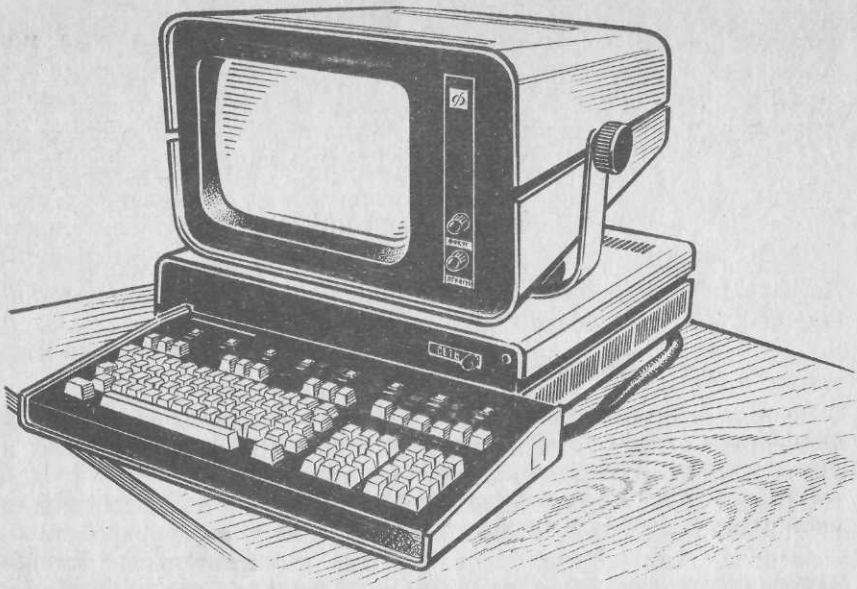


Рис. 8. Персональный компьютер

деление на страницы, многократную перепечатку текста, составление указателей ЭВМ может взять на себя. Круг задач по обработке текстов, которые ЭВМ может помочь решить, весьма велик. Применение ЭВМ в этой области существенно повышает производительность труда.

Работа ЭВМ может происходить без непосредственного контакта с человеком. Специальные компьютеры — микропроцессоры могут встраиваться в самое различное оборудование и заменять человека при управлении этим оборудованием. Такой микропроцессор может, например, автоматически переключать скорости при движении автомобиля и выбирать наиболее экономичный режим работы двигателя.

Компьютер может быть соединен с другими компьютерами, тогда они образуют сеть ЭВМ. Такая сеть может охватывать компьютеры в одном здании, а может связывать компьютер с компьютерами других городов. Сети ЭВМ позволяют использовать информацию, накопленную в одном месте, сразу во многих местах, где она нужна.

Прогресс в развитии ЭВМ сегодня привел к появлению новых ЭВМ — персональных компьютеров (рис. 8). Такие компьютеры помогут врачу поставить диагноз, школьнику выполнить уроки, учителю проверить домашние работы учащихся, архитектору спроектировать новые дома. Присоединенные к компьютерной сети персональные ЭВМ свяжут человека со справочными служ-

бами и библиотеками, позволяя ему быстро получить нужные сведения.

Поручая компьютерам механическую, рутинную работу, мы освобождаем человека для творческой деятельности. Для того чтобы ЭВМ могли решать нужные задачи, люди должны постоянно передавать компьютерам свои знания в виде точной информации, строгих правил, безошибочных алгоритмов и эффективных программ. Вот почему знание основ информатики и вычислительной техники, понимание их роли в жизни общества, деятельности людей становятся элементом человеческой культуры, составной частью общего образования, учебным предметом.

ПЕРВОНАЧАЛЬНЫЕ СВЕДЕНИЯ ОБ ЭВМ

Огромное разнообразие областей применения вычислительной техники потребовало создания ЭВМ разных типов — больших и малых, универсальных и специализированных.

Любая ЭВМ — сложнейшая техническая система, состоящая из миллионов и даже сотен миллионов простейших устройств — элементов. При создании современных вычислительных машин используется технология, опирающаяся на последние достижения многих отраслей науки и техники — информатики, математики, физики, химии, а также материаловедения, микроэлектроники и др.

Несмотря на разницу в размерах, внешнем виде, выполняемых функциях, различные ЭВМ имеют общую структуру и принципы работы. Эти принципы достаточно просты. Для того чтобы понять их, нужно познакомиться с назначением памяти, процессора и устройств ввода и вывода информации — основных компонентов, из которых состоит любая ЭВМ (рис. 9).

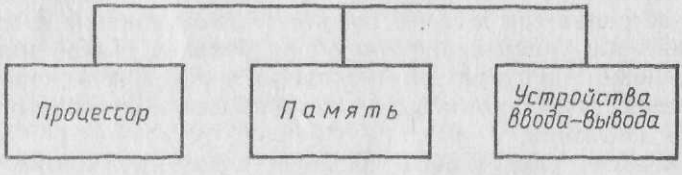


Рис. 9. Схема ЭВМ

Форма представления информации в ЭВМ. Сначала познакомимся с тем, в какой форме представляется информация в ЭВМ. ЭВМ может «хранить» и обрабатывать информацию в виде комбинации электрических сигналов двух типов, которые принято обозначать цифрами 0 и 1.

Любая информация представляется в ЭВМ последовательностью этих двух цифр 0 и 1, такие последовательности называются двойчными кодами. В большинстве ЭВМ один символ (т. е. буква, цифра, знак препинания или специальный знак §, % и т. д.)

записывается кодом из 8 цифр (такой код называется восьмиразрядным). Например, буква *А* кодируется как 01000001, а буква *М* — как 01001101.

Используя двоичные коды, можно закодировать (записать с помощью кода) любую информацию. Скажем, слово *мама* кодируется последовательностью из 32 цифр:

01001101 01000001 01001101 01000001.

Поскольку многие важнейшие применения ЭВМ связаны с обработкой числовой информации, то и числа кодируются последовательностями двоичных цифр, например число 1985 — последовательностью из 16 цифр:

0000 0111 1100 0001

Изображение, которое мы видим на экране телевизора, также может быть закодировано последовательностью нулей и единиц, состоящей из сотен тысяч и даже миллионов цифр. Любое преобразование информации внутри ЭВМ сводится к работе с двоичными кодами.

Для измерения количества информации используется единица измерения бит. Один бит — это количество информации, содержащееся в сообщении типа «да — нет» (0 или 1 в двоичном коде). Восемь бит образуют более крупную единицу информации — байт.

Память ЭВМ состоит из отдельных ячеек памяти. В большинстве ЭВМ каждая такая ячейка способна хранить один байт информации. Для измерения объема памяти ЭВМ байт — слишком мелкая единица, практичнее использовать производные единицы — килобайт (1 килобайт $= 2^{10}$ байт $= 1024$ байт), мегабайт (1 мегабайт $= 2^{10}$ килобайт) и гигабайт (1 гигабайт $= 2^{10}$ мегабайт). Эти производные единицы обозначаются кбайт, Мбайт и Гбайт соответственно.

Программный принцип работы ЭВМ. Процессор — центральное устройство ЭВМ, обрабатывающее информацию. Процессор может выполнять фиксированный набор операций над информацией, хранящейся в памяти ЭВМ. Каков этот набор, определяется устройством процессора. Количество таких операций не очень велико. Среди них арифметические действия (сложение, умножение, вычитание, деление) над числами, содержащимися в памяти, перемещение информации из одной ячейки памяти в другую и др.

Работа ЭВМ состоит в выполнении процессором заданной последовательности операций. Это выполнение происходит под управлением программы. Программа состоит из отдельных команд, предписывающих процессору выполнить то или иное действие над информацией, хранящейся в памяти. В этом и состоит программный принцип работы ЭВМ.

В каждой команде указывается, где именно в памяти находится нужная информация, какую именно следует выполнить операцию, в какое место памяти нужно поместить результат операции.

Важнейшими особенностями работы ЭВМ являются следующие:

1. Решение задачи по заданной программе происходит автоматически, без вмешательства человека.

2. Программа записывается в память машины и может быть заменена на другую программу.

Итак, задача процессора состоит в выполнении операций, задача памяти — в хранении обрабатываемой процессором информации и программы работы ЭВМ.

Устройства ввода-вывода. Информация, обрабатываемая процессором и хранящаяся в памяти, в некоторый момент должна быть помещена или, как говорят, введена в память. Результаты работы ЭВМ должны быть переданы человеку (другой ЭВМ, управляемому станку и т. д.), другими словами — выведены. Эти операции осуществляются устройствами ввода-вывода.

Основным устройством ввода, используемым человеком, является клавиатура (аналогичная клавиатуре пишущей машинки, кассового аппарата и калькулятора). На клавиатуре имеются буквы русского и латинского алфавитов, цифры, знаки препинания и специальные символы. В память ЭВМ они передаются закодированными с помощью электрических сигналов так, как это объяснялось выше.

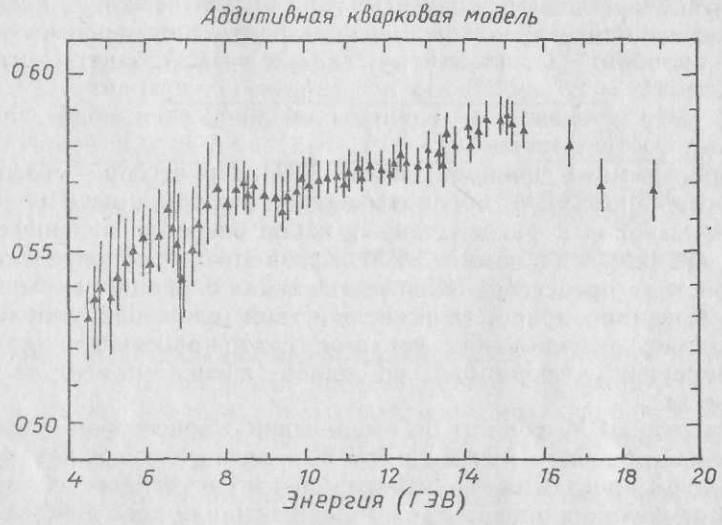


Рис. 10. Результаты математического моделирования физической системы изображены с помощью графопостроителя

Основным устройством вывода информации для непосредственного восприятия ее человеком является дисплей — телевизионный экран, на котором изображаются буквы, цифры и другие символы, имеющиеся на клавиатуре. Если этим возможности дисплея ограничиваются, то он называется алфави́тно-цифровым (так как основными символами клавиатуры являются символы алфавита и цифры). Если на дисплее (используя соответствующие программы) можно, кроме того, получать различные геометрические изображения, то дисплей называется графическим.

Помимо получения изображений на экране дисплея, ЭВМ позволяет получать их на бумаге. Если требуется вывод алфавитно-цифровой информации, то это делается с помощью алфавитно-цифрового печатающего устройства (АЦПУ). Если требуется вывод графической информации (см. рис. 2, 6, 10), это делается с помощью графопостроителя или других устройств.

Виды памяти ЭВМ. Выполнение каждой операции процессором требует определенного времени, для чтения информации из памяти и ее записи в память также требуется определенное время. Имеются различные виды памяти ЭВМ, основанные на различных принципах. При этом память с быстрым чтением и записью информации оказывается более дорогой, а память с медленным чтением и записью можно сделать более дешевой и большего объема.

В современных ЭВМ разные виды памяти применяются одновременно. Это связано со следующим. Чтобы обеспечить бесперебойную работу процессора, нужно, чтобы время чтения требуемой информации из памяти было не больше, чем время выполнения операции. Таким образом, в состав ЭВМ должна входить быстрая память. Такую память называют также оперативной (так как процессор обращается к ней постоянно в ходе выполнения своих операций) или внутренней (так как она непосредственно связана с процессором).

С другой стороны, если о некоторой информации заранее известно, что она долго не понадобится, ее можно поместить в медленную память и лишь при необходимости переписать в оперативную память. Медленную память называют внешней.

Физические основы работы ЭВМ. Как было сказано выше, обрабатываемая ЭВМ информация представляется в виде электрических сигналов. Обработку этих сигналов осуществляют электронные устройства, состоящие из тысяч и даже сотен тысяч согласованно работающих элементов. Такие устройства называют микросхемами (приставка микро- связана именно с тем, что очень большое число элементов размещается в небольшой схеме). Микросхема может представлять собой целый процессор (например, микропроцессор из числа тех, о которых говорилось выше). Если же все необходимые элементы процессора не удается поместить в одну микросхему, то несколько микросхем помещают на одну пла́ту — пластмассовую пластинку с металлическими проводниками на ней, соединяющими микросхемы (рис. 11).

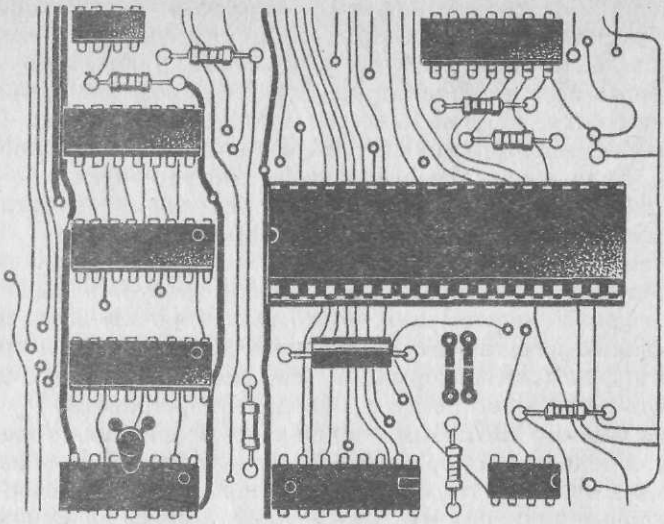


Рис. 11. Часть платы процессора ЭВМ

На такой же плате располагаются микросхемы, образующие внутреннюю память ЭВМ. Эти микросхемы напоминают микросхемы процессора и работают на тех же физических принципах (о них вы узнаете из курса физики).

Внешняя память ЭВМ устроена совсем иначе. Самым распространенным сейчас видом этой памяти является магнитная. В ней информация кодируется не электрическим сигналом, а намагниченностью частички вещества.

В качестве примера внешней памяти рассмотрим гибкие диски. Такой диск по размерам близок к гибкой грампластинке. На нем может быть записано до 0,5 Мбайт информации. Для записи и чтения информации гибкий диск помещают в устройство, называемое дисководом.

Основные характеристики современных ЭВМ. Вы знаете, что компьютеры возникли совсем недавно и развиваются очень быстро. Прогресс в этом развитии связан с достижениями информатики, усовершенствованием технологии производства и открытием новых физических принципов работы элементов компьютера, прежде всего элементов процессора. В соответствии с этим выделяют четыре поколения ЭВМ. В ЭВМ первого поколения (40—50-е гг.) использовались электронные лампы — те самые, которые сейчас употребляются в некоторых телевизорах. В ЭВМ второго поколения (50—60-е гг.) использовались транзисторы. Третье поколение ЭВМ (60—70-е гг.) уже содержало микросхемы, но небольшие, заменявшие десяток — сотню транзисторов. В теперешнем, четвертом поколении (с середины 70-х гг.) используются намного более сложные микросхемы, называемые также большими интеграль-

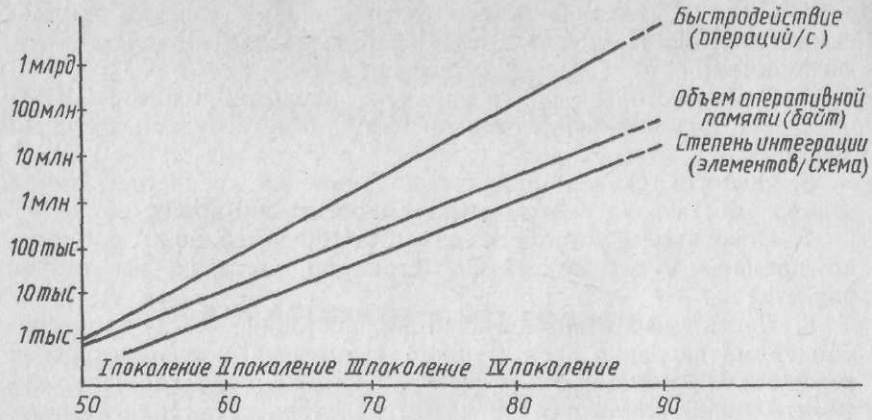


Рис. 12. Поколения ЭВМ

ными схемами. Эволюцию ЭВМ можно проиллюстрировать следующей приблизительной схемой (рис. 12).

Конечно, поколения ЭВМ различаются не только тем, из каких электронных деталей они сделаны (элементной базой), но и тем, как организована их работа (архитектурой) и какие есть для них программы (программным обеспечением).

В заключение приведем основные технические характеристики современных ЭВМ: быстродействие от сотен тысяч до сотен миллионов операций в 1 с, внутренняя память от десятков кбайт до десятков Мбайт, внешняя память от сотен кбайт до сотен Гбайт.

Вопросы

1. Какие области применения ЭВМ вам известны?
2. Приведите несколько примеров задач, для решения которых, по вашему мнению, была бы полезна ЭВМ.

Упражнения

1. Заметьте, сколько времени вам понадобится, чтобы подсчитать число букв «а» в первых 10 строках страницы, которую вы читаете. Чтобы безошибочно подсчитать количество букв «а» в тексте учебника, компьютеру понадобилось бы около 10 с. Во сколько раз это быстрее того, что вы могли бы сделать вручную без ЭВМ?

2. Предположим, что компьютер выполняет 100 000 операций в 1 с и потребляет мощность 100 Вт. Сколько операций можно сделать за 1 р.? (Цена 1 кВт/ч электроэнергии равна 4 к.)

3. На диске объемом 100 Мбайт подготовлена к выдаче на экран дисплея информация: 24 строчки по 80 символов, эта информация заполняет экран целиком. Какую часть диска она занимает?

4. Придумайте эксперимент, позволяющий узнать, сколько символов в 1 с вы читаете, пишите, произносите. Проведите этот эксперимент.

5. Текст, который вы произносите, вводится в память ЭВМ объемом 64 кбайт. Через сколько минут память будет полностью заполнена?

6. Оцените, сколько школьных сочинений среднего размера можно уместить на гибком диске емкостью 500 кбайт.

7. Печатающее устройство печатает 100 символов в 1 с. Сколько времени будет печататься страница, которую вы сейчас читаете?

8. Пусть в некотором компьютере расстояние между процессором и памятью равно 30 см, а каждое выполнение операции требует передачи информации от процессора к памяти и обратно. Покажите, что такой компьютер не может работать с быстродействием 600 млн. операций в секунду.

Раздел I

АЛГОРИТМЫ. АЛГОРИТМИЧЕСКИЙ ЯЗЫК

§ 1. АЛГОРИТМ И ЕГО СВОЙСТВА

Любой человек ежедневно встречается с множеством задач от самых простых и хорошо известных до очень сложных. Для многих задач существуют определенные правила (инструкции, предписания), объясняющие исполнителю, как решать данную задачу. Эти правила человек может изучить заранее или сформулировать сам в процессе решения задачи. Чем точнее и понятнее будут описаны правила решения задач, тем быстрее человек овладеет ими и будет эффективнее их применять.

Решение многих задач человек может передавать техническим устройствам — автоматам, электронным вычислительным машинам, ро́ботам. Применение таких технических устройств предъявляет очень строгие требования к точности описания правил и последовательности выполнения действий. Поэтому разрабатываются специальные языки для четкого и строгого описания различных правил. Это одна из задач информатики.

1. ПОНЯТИЕ АЛГОРИТМА

Под *алгоритмом* понимают понятное и точное предписание (указание) исполнителю совершить последовательность действий, направленных на достижение указанной цели или на решение поставленной задачи.

Слово алгоритм происходит от algorithmi — латинской формы написания имени великого математика IX в. аль-Хорезми, который сформулировал правила выполнения арифметических действий. Первоначально под алгоритмами и понимали только правила выполнения четырех арифметических действий над многозначными числами. В дальнейшем это понятие стали использовать вообще для обозначения последовательности действий, приводящих к решению поставленной задачи.

Рассмотрим примеры.

Пример 1.1. Вычислить значения y по формуле

$$y = (Ax + B)(Cx - D)$$

для любого значения x.

Чтобы решить эту задачу, достаточно выполнить последовательность следующих действий:

1) умножить A на x, результат обозначить R_1;
2) R_1 сложить с B, результат обозначить R_2;
3) умножить C на x, результат обозначить R_3;
4) из R_3 вычесть D, результат обозначить R_4;
5) умножить R_2 на R_4, считать результат значением y.

Это предписание является алгоритмом решения поставленной задачи. Для человека, выполняющего действия, уже необязательно знать исходную формулу для вычисления значения y. Ему нужно всего лишь строго следовать указанному предписанию, исполняя его пункт за пунктом.

В приведенном примере процесс решения задачи расчленяется на элементарные операции, арифметические действия, но это членение может быть продолжено и дальше. Например, п. 2 этого алгоритма — сложение чисел R_1 и B — можно развернуть в систему правил, описывающих процесс сложения двух чисел столбиком. Принцип расчленения сложного процесса решения задачи на элементарные действия имеет важное значение для построения алгоритмов.

Пример 1.2. Найти наибольший общий делитель двух натуральных чисел m и n.

Составим алгоритм решения этой задачи, основанный на том свойстве, что если $m>n$, то наибольший общий делитель чисел m, n такой же, как и чисел $m-n$, n.

Алгоритм будет таким:

1) если числа равны, то взять любое из них в качестве ответа, в противном случае продолжить выполнение алгоритма;
2) определить большее из чисел;
3) заменить большее число разностью большего и меньшего чисел;
4) начать алгоритм сначала.

Как видно, этот алгоритм, известный под названием алгоритма Евклида, также состоит из отдельных пунктов, предписывающих исполнителю выполнить некоторое простое действие. Его особенностью является то, что все действия, указанные в алгоритме, могут повторяться многократно.

В принципе необходимость вернуться к началу алгоритма может привести к бесконечному повторению пунктов алгоритма. В данном случае этого не произойдет, потому что величина разности большего и меньшего чисел с каждым новым вычитанием уменьшается, и поэтому после некоторого числа повторений сравниваемые числа обязательно станут равными. Алгоритм применим к любым натуральным числам и всегда приводит к решению задачи.

С помощью алгоритмов можно решать не только вычислительные, но и многие другие задачи. Приведем пример алгоритма решения графической задачи.

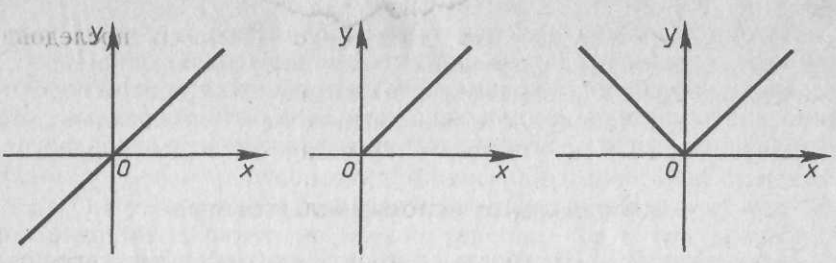


Рис. 13.

Пример 1.3. Построить график функции $y = a|x|$ при $a>0$.

Алгоритм построения имеет следующий вид (рис. 13):

1) начертить график функции $y = ax$;
2) стереть часть графика левее оси ординат;
3) симметрично отразить оставшуюся часть графика относительно оси ординат.

Пример 1.4. Игра Баше. (В игре участвуют двое.)

Рассмотрим частный случай игры. Имеется 15 предметов. Соперники ходят по очереди, за каждый ход любой из играющих может взять 1, 2 или 3 предмета. Проигрывает тот, кто вынужден взять последний предмет.

Алгоритм выигрыша для первого игрока имеет следующий вид:

1) взять два предмета;
2) второй и последующие ходы делать так, чтобы количество предметов, взятых вместе с соперником за очередной ход, в сумме составляло 4.

Данный алгоритм приводит к выигрышу для 7, 11, 15, 19, ... предметов.

Человек, пользующийся данным алгоритмом, всегда будет выигрывать в этой игре. Ему совершенно необязательно знать, почему надо поступать именно так, а не иначе. Для успешной игры от него требуется только строго следовать алгоритму.

Примеры алгоритмов показывают, что запись алгоритма распадается на отдельные указания исполнителю выполнить некоторое законченное действие. Каждое такое указание называется *командой*. Команды алгоритма выполняются одна за другой. На каждом шаге исполнения алгоритма исполнителю точно известно, какая команда должна выполняться следующей.

Поочередное выполнение команд алгоритма за конечное число шагов приводит к решению задачи, к достижению цели.

Каждый алгоритм строится в расчете на некоторого исполнителя. Для того чтобы исполнитель мог решить задачу по заданному алгоритму, необходимо, чтобы он был в состоянии выполнить каждое действие, предписываемое командами алгоритма.

Совокупность команд, которые могут быть выполнены исполнителем, называется *системой команд исполнителя*.

Алгоритм должен быть понятным исполнителю, т. е. каждая его команда должна входить в систему команд исполнителя.

Таким образом, для правильного построения алгоритма необходимо знать систему команд исполнителя и быть уверенным, что исполнение алгоритма всегда завершится за конечное число шагов.

2. ФОРМАЛЬНОЕ ИСПОЛНЕНИЕ АЛГОРИТМА

Пример 2.1. Построить алгоритм нахождения середины отрезка при помощи циркуля и линейки.

Алгоритм деления отрезка AB пополам:

1) поставить ножку циркуля в точку A;
2) установить раствор циркуля равным длине отрезка AB;
3) провести окружность;
4) поставить ножку циркуля в точку B;
5) провести окружность;
6) через точки пересечения окружностей провести прямую;
7) отметить точку пересечения этой прямой с отрезком AB.

Каждая его команда указывает исполнителю одно конкретное законченное действие, и исполнитель должен выполнить его целиком. Исполнитель не может перейти к выполнению следующей команды, не закончив полностью выполнения предыдущей. Команды алгоритма надо выполнять последовательно одну за другой, в соответствии с указанным порядком их записи. Выполнение всех команд гарантирует правильное решение задачи. Данный алгоритм будет понятен исполнителю, умеющему работать с циркулем и знающему, что такое поставить ножку циркуля, провести окружность и т. д. Для ученика I класса этот алгоритм не понятен, а для ученика IX класса понятен.

Пример 2.2. Алгоритм «отгадывания» задуманного числа.

Пусть кто-либо задумает произвольное натуральное число. Ему предлагается произвести с этим числом следующие действия и затем сообщить результат:

1) умножить задуманное число на 5;
2) прибавить 8;
3) сумму умножить на 2.

Необходимо по результату «отгадать» задуманное число.

Решение данной задачи сводится к решению уравнения $(x \cdot 5 + 8) \cdot 2 = a$, где x — неизвестное задуманное число, a — полученный результат.

«Отгадывание» x можно поручить исполнителю, совершенно незнакомому с содержанием задачи. Для этого достаточно сообщить ему следующий алгоритм:

1) вычесть из результата 16;
2) в полученной разности отбросить крайнюю правую цифру, полученное число и будет искомым.

Исполняя алгоритм, исполнитель может не вникать в смысл того, что он делает, и вместе с тем получать нужный результат.

В таком случае говорят, что исполнитель действует формально, т. е. отвлекается от содержания поставленной задачи и только строго выполняет некоторые правила, инструкции.

Это очень важная особенность алгоритмов. Наличие алгоритма формализовало процесс «отгадывания» задуманного числа, исключило рассуждения. Если обратиться к алгоритмам, которые мы рассматривали ранее, то можно увидеть, что и они позволяют исполнителю действовать формально. Таким образом, создание алгоритма дает возможность решать задачу формально, механически исполняя команды алгоритма в указанной последовательности.

Построение алгоритма для решения задачи из какой-либо области требует от человека глубоких знаний в этой области, бывает связано с тщательным анализом поставленной задачи, сложными, иногда очень громоздкими рассуждениями. На поиски алгоритма решения некоторых задач ученые затрачивают многие годы. Но когда алгоритм создан, решение задачи по готовому алгоритму уже не требует каких-либо рассуждений и сводится только к строгому выполнению команд алгоритма.

В этом случае исполнение алгоритма можно поручить не человеку, а машине. Действительно, простейшие операции, на которые при создании алгоритма расчленяется процесс решения задачи, может реализовать и машина, специально созданная для выполнения отдельных команд алгоритма и выполняющая их в последовательности, указанной в алгоритме. Это положение и лежит в основе работы автоматических устройств.

Вопросы

1. Приведите примеры известных вам алгоритмов.
2. Что понимается под командой алгоритма?
3. Что называется системой команд исполнителя? Поясните на примере.
4. Могут ли автоматические устройства быть исполнителями алгоритмов?

Упражнения

1. Сформулируйте и запишите алгоритм вычисления значения y по формуле:

а) $y = (2x + 3)(7x - 5)$; б) $y = \dfrac{2 - (x - 3)^2}{(x - 3)^2 + 4}$.

2. По приведенному алгоритму восстановите формулу для вычисления значения y:

а) 1) умножить x на x, обозначить результат R_1;
2) умножить R_1 на a, обозначить результат R_2;
3) сложить R_2 с b, обозначить результат R_3;
4) разделить R_3 на c, считать результат значением y;

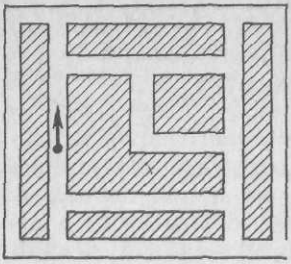


Рис. 14.

б) 1) сложить x с 1, обозначить результат A_1;
2) разделить 1 на A_1, обозначить результат A_2;
3) сложить A_2 с 1, обозначить результат A_3;
4) вычесть из A_2 единицу, обозначить результат A_4;
5) разделить A_4 на A_3, обозначить результат A_5;
6) вычесть из A_5 единицу, считать результат значением y.

3. Сформулируйте и запишите алгоритмы построения с помощью циркуля и линейки:

а) серединного перпендикуляра данного отрезка;

б) окружности, для которой данный отрезок — диаметр;

в) биссектрисы угла;

г) точки пересечения медиан данного треугольника;

д) перпендикуляра к прямой, проходящего через точку, принадлежащую прямой.

4. Двое играют в игру Баше с 20 предметами. Начинающий игрок действует по алгоритму 1.4. Может ли второй игрок у него выиграть?

5. Преобразуйте алгоритм так, чтобы его мог исполнить ученик II класса, не знающий действий возведения в квадрат и в куб:

1) возвести x в квадрат, результат обозначить a_1;
2) умножить a_1 на 2, результат обозначить a_2;
3) сложить a_2 и 15, результат обозначить a_3;
4) возвести a_3 в куб, результат обозначить a_4;
5) сложить a_4 и 25, результат обозначить a_5;
6) разделить a_3 на a_5, считать результат значением y.

6. Человек находится в лабиринте (рис. 14) и начинает двигаться в направлении, указанном стрелкой, согласно следующему предписанию: иди шаг за шагом, не отрывая руки от правой стены. Шагай, пока не выйдешь из лабиринта.

а) Достигает ли данное предписание цели вывести исполнителя из лабиринта?

б) Нарисуйте путь исполнителя этого предписания.

7. Сколько раз будет выполняться шаг 3 алгоритма Евклида для $m = 100$, $n = 18$?

§ 2. АЛГОРИТМИЧЕСКИЙ ЯЗЫК

Алгоритмический язык — это система обозначений и правил для единообразной и точной записи алгоритмов и их исполнения. Алгоритмический язык, с одной стороны, близок к обычному языку. Алгоритмы на этом языке могут записываться и читаться как обычный текст. С другой стороны, алгоритмический язык включает в

себя и математическую символику: числа, обозначения величин и функций, знаки операций и скобки и др.

Правила алгоритмического языка лежат в основе языков программирования для ЭВМ. Изучение алгоритмического языка поможет вам в будущем освоить любой язык программирования для ЭВМ.

3. ОБЩИЕ ПРАВИЛА АЛГОРИТМИЧЕСКОГО ЯЗЫКА

Как и каждый язык, алгоритмический язык имеет свой словарь. Основу этого словаря составляют слова, употребляемые для записи команд, входящих в систему команд исполнителя того или иного алгоритма. Такие команды называются *простыми* командами.

Обычно простая команда выглядит как повелительное предложение русского языка в полной или сокращенной форме, включая, если необходимо, формулы и другие символические обозначения.

Кроме того, в алгоритмическом языке используется некоторое ограниченное число слов, смысл и способ употребления которых задан раз и навсегда. Эти слова называются *служебными словами.* При записи алгоритмов они выделяются и записываются, как правило, в сокращенной форме. Использование служебных слов делает запись алгоритма более наглядной, а форму представления различных алгоритмов — единообразной.

Алгоритм, записанный на алгоритмическом языке, должен иметь название. Название выбирается так, чтобы было ясно, решение какой задачи описывает данный алгоритм. Для выделения названия алгоритма перед ним записывается служебное слово **алг** (**алг**оритм).

За названием алгоритма (обычно с новой строки) записываются его команды. Для указания начала и конца алгоритма его команды заключаются в пару служебных слов **нач** (**нач**ало) и **кон** (**кон**ец). Команды записываются последовательно. При записи одной команды можно перейти на другую строчку. Если несколько команд записываются на одной строчке, то они разделяются точкой с запятой (;).

Последовательность нескольких команд алгоритма, выполняющихся одна за другой, называется *серией.* Серия может состоять и из одной команды.

Итак, общий вид алгоритма, записанного на алгоритмическом языке, таков:

алг название алгоритма
нач
команды алгоритма (серия)
кон

Пример 3.1. Записать на алгоритмическом языке алгоритм нахождения середины отрезка (с. 20), если в систему команд исполнителя входят привычные действия с циркулем и линейкой.

алг деление отрезка AB пополам

нач

поставить ножку циркуля в точку A
установить раствор циркуля равным длине отрезка AB
провести окружность
поставить ножку циркуля в точку B
провести окружность
через точки пересечения окружностей провести прямую
отметить точку пересечения этой прямой с отрезком AB

кон

Пример 3.2. Записать на алгоритмическом языке инструкцию по использованию междугородного телефона-автомата:

Снимите трубку. Опустите монету 15 к. Услышав непрерывный сигнал, наберите код нужного вам города, а затем сразу нужный номер. Услышав ответ абонента, нажмите кнопку «Разговор».

алг использование междугородного телефона-автомата

нач

снять трубку
опустить монету 15 к.
дождаться появления непрерывного сигнала
набрать код нужного вам города
набрать номер нужного вам телефона
дождаться ответа абонента
нажать кнопку «Разговор»
говорить

кон

4. СОСТАВНЫЕ КОМАНДЫ

Некоторые типы алгоритмов: линейные, разветвляющиеся и циклические уже встречались вам в курсе алгебры.

В алгоритмическом языке линейными являются алгоритмы, состоящие из одной серии простых команд. Для записи разветвляющихся и циклических алгоритмов в алгоритмическом языке используются так называемые составные команды, аналогичные сложным предложениям русского языка.

В алгоритмическом языке употребляются две основные составные команды: команда ветвления и команда повторения (цикла). Каждая из этих двух команд отличается от простых тем, что в нее входит условие, в зависимости от которого выполняются или не выполняются команды из числа входящих в составную.

1. **Команда ветвления** записывается следующим образом:

если условие

то серия 1

иначе серия 2

все

В зависимости от условия выполняется только одна из двух серий команд, входящих в команду ветвления. Если условие соблюдено, то надо выполнить серию 1, а если нет — серию 2.

Команда ветвления используется и в сокращенной форме:

если условие

то серия

все

В этом случае, если условие соблюдено, исполнитель выполняет серию команд, следующую в записи алгоритма за служебным словом **то**, а в противном случае, пропуская серию, переходит к выполнению команды, следующей за командой ветвления (после служебного слова **все**).

Команды из каждой серии выполняются подряд, каждая по своим правилам. Команда ветвления заканчивается, как только выполнится последняя команда из серии 1 или серии 2.

В качестве условия в команде ветвления может быть использовано любое понятное исполнителю утверждение, которое может соблюдаться или не соблюдаться. Утверждение может быть выражено словами или формулой.

Читается команда ветвления так же, как и записывается.

При изображении алгоритмов в виде схем условие можно понимать как вопрос, на который возможен ответ «да» или «нет».

На рисунке 15 изображены схемы для полной и сокращенной формы записи команды ветвления.

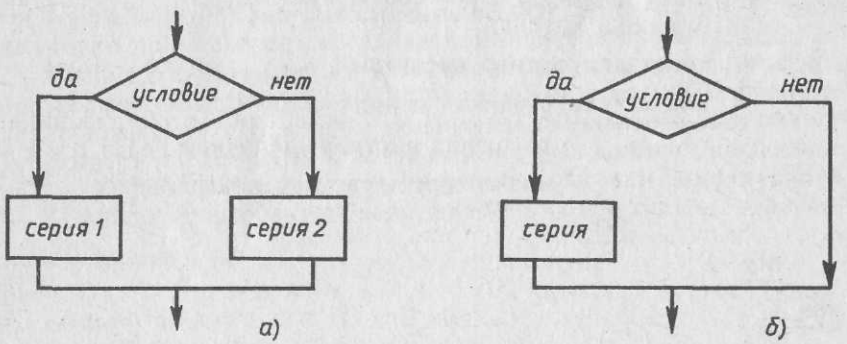


Рис. 15.

Проиллюстрируем на примерах употребление команды ветвления.

Пример 4.1. Записать алгоритм правописания приставок на «з» («с»).

алг правописание приставок на «з» («с»)
нач
    если корень слова начинается со звонкой согласной
        то на конце приставки написать «з»
        иначе на конце приставки написать «с»
    все
кон

Пример 4.2. Нужно включить электроприбор, имеющий переключатель напряжения, в сеть 220 В.

алг включение электроприбора в сеть 220 В
нач
    если переключатель прибора установлен на 127 В
        то установить переключатель прибора на 220 В
    все
    вставить вилку в розетку
кон

Пример 4.3. Определить кислотность раствора. Чтобы решить эту задачу, достаточно опустить в раствор лакмусовую бумажку и по ее цвету определить, является ли раствор кислотным, щелочным или нейтральным.

алг определение кислотности раствора
нач отлить в пробирку 1 мл раствора
    опустить в пробирку лакмусовую бумажку
    если бумажка красная
        то ответ: раствор кислотный
        иначе **если** бумажка синяя
                то ответ: раствор щелочной
                иначе ответ: раствор нейтральный
            все
    все
кон

Схема алгоритма приведена на рисунке 16.

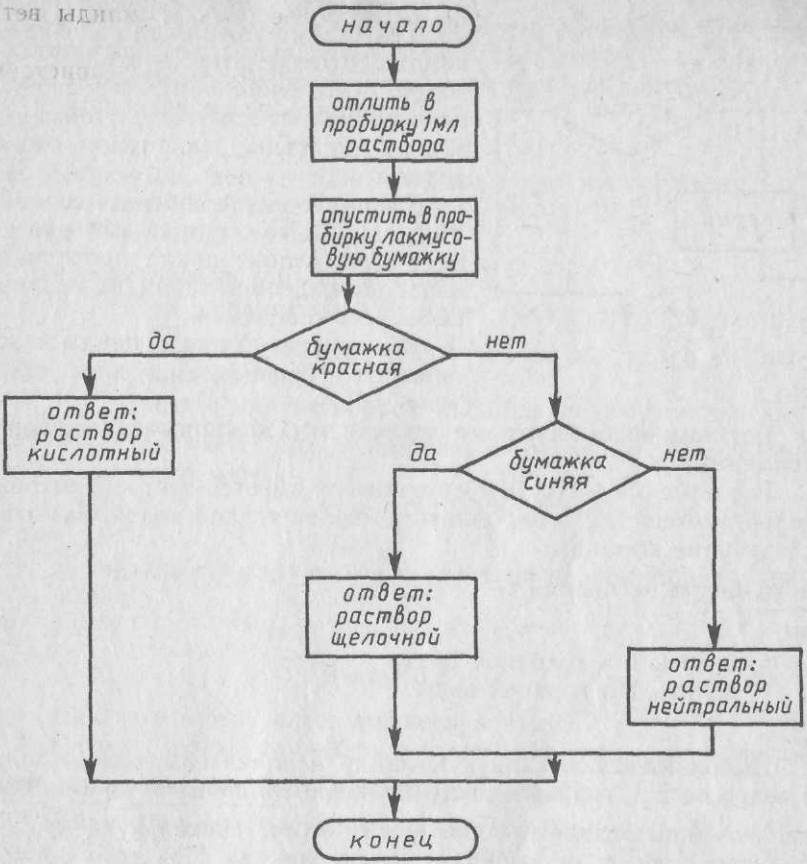


Рис. 16.

2. Команда повторения. В своей практической деятельности человек постоянно встречается с задачами, для решения которых требуется многократно повторять одни и те же действия. Именно для этого применяется составная команда повторения (цикл).

Команда повторения имеет особое значение для построения алгоритмов, исполняемых на ЭВМ, так как только ее использование позволяет с помощью сравнительно коротких алгоритмов записывать предписания о совершении очень длинной последовательности действий, для которых требуется высокая скорость ЭВМ.

Команда повторения записывается следующим образом:

пока условие
нц
серия
кц

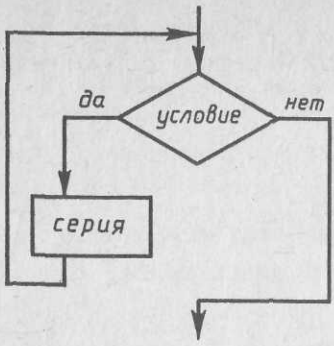


Рис. 17.

Выполнение этой команды приводит к тому, что указанная в ней серия команд выполняется несколько раз подряд. Она выполняется столько раз, сколько нужно для того, чтобы указанное условие перестало соблюдаться.

Если условие не соблюдается с самого начала, то серия не выполняется ни разу. Условие цикла проверяется перед выполнением серии, но не в процессе ее выполнения.

Выполнение команды цикла можно пояснить графически в виде схемы (рис. 17).

Поясним особенности выполнения этой команды на следующих примерах.

Пример 4.4. Пусть у исполнителя имеется пустое 7-литровое ведро, которое надо наполнить до краев теплой водой, выполняя следующие команды:

```
пока ведро не полно
нц
    долить 1 л холодной воды
    долить 1 л горячей воды
кц
```

Выполняя эту составную команду, исполнитель трижды дольет в ведро по 2 л, так как между служебными словами **нц** и **кц** стоят две команды: «долить 1 л». После этого, поскольку ведро еще не будет полным, он выполнит серию еще раз. При этом 1 л воды выльется из ведра.

Пример 4.5. Записать на алгоритмическом языке выигрышный алгоритм игры Баше, если имеется 15 предметов (см. пример 1.4).

```
алг игра Баше
нач
    взять два предмета
    пока осталось больше четырех предметов
    нц
        дать противнику сделать ход
        запомнить число К взятых противником предметов
        взять 4 — К предметов
    кц
    дать противнику сделать ход
кон
```

Пример 4.6. Используя уже известные команды ветвления и повторения, составить более сложный алгоритм — нахождения массы небольшого предмета с помощью чашечных весов.

```
алг определение массы предмета
нач
    положить на левую чашку весов исследуемый предмет
    положить на правую чашку весов гирьку 10 г
    пока чашки не уравновешены
    нц если перевесила левая чашка
            то добавить на правую чашку еще одну гирьку
               такой же массы, какую положили туда в предыду-
               щий раз
            иначе снять с правой чашки последнюю положенную
               на нее гирьку и заменить ее гирькой в 10 раз мень-
               шей массы
        все
    кц
    сосчитать суммарную массу гирек на правой чашке с гирь-
    ками
кон
```

Вопросы

1. Для чего нужен алгоритмический язык?
2. Какова роль служебных слов **алг**, **нач**, **кон**, **если**, **то**, **иначе**, **все**, **пока**, **нц**, **кц**? Поясните на примерах.
3. Какие бывают составные команды? Приведите примеры.

Упражнения

1. Запишите алгоритмы из упражнения 3, § 1 на алгоритмическом языке.

2. Измените алгоритм примера 4.4 так, чтобы вода не проливалась.

3. Придумайте и запишите выигрышный алгоритм для второго игрока игры Баше для 5 предметов.

§ 3. АЛГОРИТМЫ РАБОТЫ С ВЕЛИЧИНАМИ

Решение большинства практических задач связано с преобразованием информации.

Так, при решении квадратного уравнения вида $ax^2 + bx + c = 0$ на основании исходной информации (вида уравнения и значений a, b и c) мы получаем новую информацию — результат (имеет ли это уравнение корни, и если имеет, то получаем значения корней x_1 и x_2).

При нахождении середины отрезка AB мы на основании информации об отрезке AB получаем результат — точку, отмечающую его середину.

Под терминами «исходная информация», «преобразованная информация», «результат» понимаются вполне конкретные величины — числовые (коэффициенты a, b и c), графические (отрезок) и т. п.

5. ВЕЛИЧИНЫ

Величины делятся на переменные и постоянные. *Постоякной* называется величина, значение которой не меняется в процессе исполнения алгоритма, а остается одним и тем же, указанным в тексте алгоритма (например, 15; 2,4; 3,14 и т. д.). *Переменной* называется величина, значение которой меняется в процессе исполнения алгоритма.

При написании алгоритма для переменных величин вводятся обозначения аналогично обозначениям переменных в курсе алгебры. Такое обозначение переменной величины в алгоритмическом языке называется *именем* величины.

При исполнении алгоритма в каждый момент времени величина обычно имеет некоторое значение. Оно называется *текущим* значением. В процессе исполнения алгоритма величина может не получить конкретного значения. Такая величина называется неопределенной.

В школьном курсе математики и физики чаще всего встречаются числовые величины, значениями которых являются натуральные, целые, действительные числа. Однако в алгоритмах столь же часто встречаются и нечисловые величины: слова, таблицы, списки, тексты, графики, геометрические фигуры и т. д. В курсе математики и физики величины чаще всего обозначаются одной буквой латинского или греческого алфавита. Поскольку при применении ЭВМ и при записи алгоритмов и программ разнообразие величин очень велико, в алгоритмическом языке принято использовать в качестве имен величин произвольные буквы, буквосочетания и любые слова, поясняющие смысл и назначение величины в алгоритме.

Для того чтобы отличить имена величин от обычных слов в этом учебнике, имена величин будут выделены другим шрифтом — курсивом (например, *произведение, число шагов* и т. д.). Например, *класс 9а* в школе является переменной величиной, значение которой — множество учеников, обучающихся в этом классе в текущем учебном году.

Величины, значениями которых являются слова или текст, называются *литерными*.

Для того чтобы выделить текст, который является значением литерной величины, значения литерных величин берутся в кавычки, например „нет решения“.

Как мы видим, величины могут иметь различный тип. Они могут быть натуральными, целыми, действительными, литерными и т. д. в соответствии с тем, что может быть их назначением. Отметим, что в программировании вместо слова «действительный» в применении к числам используется слово «вещественный». Сокращенно типы переменных обозначаются словами **нат** (натуральный), **цел** (целый), **вещ** (вещественный), **лит** (литерный) и т. д.

6. ЗАГОЛОВОК АЛГОРИТМА

Запись всякого алгоритма начинается с заголовка. Вот пример заголовка алгоритма нахождения остатка от деления двух натуральных чисел:

алг остаток от деления (**нат** *делимое*, **нат** *делитель*, **нат** *остаток*)

арг *делимое, делитель*
рез *остаток*

Здесь **остаток от деления** — название алгоритма, *делимое, делитель* и *остаток* — имена величин. Перед каждым именем величины указан ее тип. Исходными данными для алгоритма нахождения остатка от деления двух натуральных чисел являются величины *делимое, делитель* типа **нат**, а в результате его выполнения величина *остаток* приобретает значение, равное остатку от деления значений величин *делимое* и *делитель*. Величины, являющиеся исходными данными для алгоритма, называются *аргументами*. Их список помещается после служебного слова **арг** (аргумент). Результаты алгоритма (в данном примере — величина *остаток*) перечисляются после служебного слова **рез** (результат).

Общий вид заголовка алгоритма таков:

алг название алгоритма (список величин с указанием типов)
арг имена аргументов
рез имена результатов

Имена аргументов и результатов алгоритма перечисляются через запятую.

Пример 6.1. Заголовок алгоритма Евклида:

алг нахождение наибольшего общего делителя (НОД) (**нат** *M*, *N*, **нат** *НОД*)
арг *M*, *N*
рез *НОД*

Пример 7.1. Рассмотрим запись алгоритма работы с величинами на примере алгоритма решения квадратного уравнения $ax^2 + bx + c = 0$.

алг решение квадратного уравнения (**вещ** a, b, c, x_1, x_2, **лит** y)
  арг a, b, c
  рез x_1, x_2, y
нач **вещ** D
  $D := b^2 - 4 \cdot a \cdot c$
  если $D < 0$
    то $y :=$ „нет решения“
    иначе $y :=$ „есть решение“
      $x_1 := \dfrac{-b + \sqrt{D}}{2a}$; $x_2 := \dfrac{-b - \sqrt{D}}{2a}$
  все
кон

Разберем подробно приведенный пример. Значение корней x_1 и x_2 надо найти при конкретных значениях a, b и c. Поэтому переменные a, b и c являются аргументами, а корни x_1 и x_2 — результатом алгоритма. Если корни уравнения не существуют, то алгоритм должен выдавать в качестве результата сообщение об отсутствии корней. Для этого введена литерная переменная y, значением которой является текст сообщения о том, имеет или нет уравнение корни.

Текст алгоритма начинается с заголовка.

Затем в тексте алгоритма следуют команды, указывающие, как решать поставленную задачу. Но перед ними после знакомого нам служебного слова **нач** появилось еще одно описание типа переменной: **вещ** D

Переменная D (дискриминант квадратного уравнения) не является ни аргументом, ни результатом алгоритма, а используется только при его выполнении для обозначения вычисляемого промежуточного значения. Такие переменные называются *промежуточными*. Их тип также должен быть указан исполнителю, и их описание приводится после служебного слова **нач**.

Далее в записи алгоритма следует команда:

$$D := b^2 - 4 \cdot a \cdot c.$$

Эта команда читается так: «Присвоить переменной D значение выражения $b^2 - 4 \cdot a \cdot c$». Команда такого вида называется *присваиванием*.

Знак присваивания (:=) делит команду присваивания на левую и правую части. В левой части может стоять любая переменная величина алгоритма, а в правой части — любое числовое или нечисловое выражение.

Знак «:=» нельзя путать со знаком математического равенства «=». Знак «:=» указывает исполнителю действие (присвоить переменной вычисленное значение), в то время как знак «=» никаких действий не предполагает.

Далее в тексте алгоритма следует уже известная нам составная команда ветвления, содержащая две серии команд. Первая серия (после служебного слова **то**) состоит из одной команды присваивания литерной переменной y:= „нет решения“.

Вторая серия (после служебного слова **иначе**) состоит из трех команд присваивания: первая присваивает значение литерной переменной y, две оставшиеся присваивают значения числовым переменным x_1, x_2.

8. ИСПОЛНЕНИЕ АЛГОРИТМА

Практическая реализация всех предусмотренных алгоритмом действий по получению результата для конкретных значений аргументов осуществляется в процессе исполнения алгоритма.

Рассмотрим, как происходит исполнение алгоритма решения квадратного уравнения (пример 7.1).

Прежде всего исполнитель составляет таблицу записи значений используемых в алгоритме переменных величин (табл. 1).

Таблица 1

Шаги алгоритма	Аргументы			Промежуточная величина	Результаты			Проверка условий
	a	b	c	D	x_1	x_2	y	

Такую таблицу мы будем называть *таблицей значений*. При исполнении алгоритма компьютером значения величин хранятся в его памяти. При исполнении алгоритма человеком таблица значений выполняет роль дополнительной памяти для исполнителя.

В данном случае исполнитель получает значения аргументов a, b и c. Пусть, например, $a = 2$, $b = 1$, $c = -6$. Эти данные исполнитель заносит в таблицу (табл. 2).

Затем он приступает непосредственно к исполнению алгоритма. Алгоритм расчленяется на отдельные команды (для удобства их последовательной регистрации в таблице имеется специальная графа «Шаги алгоритма»).

На первом шаге выполняется команда присваивания

$$D := b^2 - 4 \cdot a \cdot c,$$

т. е. исполнитель подставляет в формулу текущие значения входящих в нее переменных, производит нужные вычисления и присваивает переменной D значение 49.

Это значение вносится в таблицу (табл. 2).

Таблица 2

Шаги алгоритма	Аргументы			Промежуточная величина	Результаты			Проверка условий
	a	b	c	D	x_1	x_2	y	
	2	1	−6					
1				49				

На этом первый шаг алгоритма заканчивается, и исполнитель переходит к выполнению команды ветвления. Выполнение этой составной команды состоит из нескольких шагов.

Вторым шагом алгоритма является проверка условия $D<0$ этой команды. Текущим значением переменной D, как следует из таблицы, является число 49, поэтому условие $D<0$ в данном случае не соблюдается. Результат проверки условия для контроля заносится в соответствующую строку графы «Проверка условий» таблицы, а именно записывается условие и за ним слово «да», если условие соблюдено, и «нет» в противном случае (табл. 3).

Условие не соблюдено, нужно переходить к выполнению серии команд, следующей за служебным словом **иначе** команды ветвления, за этим словом стоит серия из трех команд присваивания.

Третий шаг алгоритма — выполнение команды присваивания $y :=$ „есть решение“. В результате выполнения этого шага исполнитель присваивает переменной y литерное значение „есть решение“ и это значение вносится в таблицу (табл. 3).

Таблица 3

Шаги алгоритма	Аргументы			Промежуточная величина	Результаты			Проверка условий
	a	b	c	D	x_1	x_2	y	
	2	1	−6					
1				49				
2								49 < 0 (нет)
3							есть решение	

Четвертый шат алгоритма — выполнение команды присваивания $x_1 := \frac{-b+\sqrt{D}}{2a}$. Исполнитель присваивает переменной x_1 значение $\frac{-b+\sqrt{D}}{2a} = \frac{-1+\sqrt{49}}{4} = 1{,}5$ (табл. 4).

Таблица 4

Шаги алгоритма	Аргументы			Промежуточная величина	Результаты			Проверка условий
	a	b	c	D	x_1	x_2	y	
	2	1	−6					
1				49				
2								49 < 0 (нет)
3							есть решение	
4					1,5			

Пятый шаг алгоритма — выполнение команды присваивания $x_2 := \frac{-b-\sqrt{D}}{2a}$. Исполнитель присваивает переменной x_2 значение $\frac{-b-\sqrt{D}}{2a} = \frac{-1-\sqrt{49}}{4} = -2$ (табл. 5).

Таблица 5

Шаги алгоритма	Аргументы			Промежуточная величина	Результаты			Проверка условий
	a	b	c	D	x_1	x_2	y	
	2	1	−6					
1				49				
2								49 < 0 (нет)
3							есть решение	
4					1,5			
5						−2		

После пятого шага закончилось выполнение составной команды ветвления и исполнение всего алгоритма. В результате пере-

менные x_1, x_2 и y получили значения: $x_2 = -2$, $x_1 = 1{,}5$, $y :=$ „есть решение“ (см. табл. 5).

Если бы аргументы алгоритма были заданы так, что значение выражения $b^2 - 4ac$ оказалось бы меньше нуля (например, $a = 2$, $b = -3$, $c = 7$), то в этом случае исполнение закончилось бы за три шага и переменная y получила бы значение „нет решения“, а переменные x_1 и x_2 никаких значений не получили бы (табл. 6).

Таблица 6

Шаги алгоритма	Аргументы			Промежуточная величина	Результаты			Проверка условий
	a	b	c	D	x_1	x_2	y	
	2	−3	7					
1				−47				
2								−47 < 0 (да)
3							нет решения	

Рассмотрим еще один пример выполнения алгоритма.

Пример 8.1. Алгоритм Евклида — нахождения наибольшего общего делителя двух натуральных чисел M и N.

```
алг нахождение наибольшего общего делителя (нат M, N, нат
    НОД)
    арг M, N
    рез НОД
нач нат x, y
    x := M; y := N
    пока x ≠ y
    нц
        если x > y
            то x := x − y
            иначе y := y − x
        все
    кц
    НОД := x
кон
```

В таблице 7 приведено исполнение алгоритма Евклида для $M = 35$, $N = 21$.

Таблица 7

Шаги алгоритма	Аргументы		Промежуточные величины		Результат	Проверка условий
	M	N	x	y	НОД	
	35	21				
1			35			
2				21		
3						$35 \neq 21$ (да)
4						$35 > 21$ (да)
5			14			
6						$14 \neq 21$ (да)
7						$14 > 21$ (нет)
8				7		
9						$14 \neq 7$ (да)
10						$14 > 7$ (да)
11			7			
12						$7 \neq 7$ (нет)
13					7	

9. ОТНОШЕНИЯ МЕЖДУ ВЕЛИЧИНАМИ В КАЧЕСТВЕ УСЛОВИЙ

Так же как в математике, в алгоритмическом языке используются следующие знаки отношения между величинами:

для числовых величин

$<$ меньше	$\geqslant$ не меньше	$=$ равно
$>$ больше	$\leqslant$ не больше	$\neq$ не равно

для литерных величин

$=$ равно	$\neq$ не равно

В алгоритмах работы с числовыми и литерными величинами именно такие отношения между величинами и используются в качестве условий.

Пример 9.1. Построить алгоритм решения следующей задачи: «Ракета запускается с точки на экваторе Земли со скоростью y (в км/с) в направлении движения Земли по орбите вокруг Солнца. Каков будет результат этого запуска ракеты в зависимости от скорости y?»

```
алг запуск ракеты (вещ y, лит A)
    арг y
    рез A
нач
    если y<7,8
        то A:= „ракета упадет на Землю“
        иначе если y<11,2
                то A:= „ракета станет спутником Земли“
                иначе если y<16,4
                        то A:= „ракета станет спутником Солнца“
                        иначе A:= „ракета покинет Солнечную систему“
                        все
                все
        все
кон
```

В этом алгоритме имеются переменные величины y и A, постоянные числовые величины 7,8; 11,2; 16,4 и постоянные литерные величины „ракета упадет на Землю“ и др. В составных командах в качестве условия используются сравнения переменной величины y и постоянных величин 7,8; 11,2; 16,4. Все условия состояли только из одного отношения между величинами. Такие условия называются *простыми*.

Пример 9.2. Построить алгоритм вычисления выражения $y=\frac{1}{x}+\frac{1}{x-1}$. При этом учесть, что при $x=0$ и при $x=1$ выражение не имеет смысла.

```
алг пример (вещ x, y, лит P)
    арг x
    рез y, P
нач
    если x=0 или x=1
        то P:= „значение y не определено“
        иначе P:= „значение y определено“;  y:=1/x+1/(x-1)
    все
кон
```

В этом алгоритме команда ветвления содержит условие, которое состоит из двух отношений, соединенных словом **или**:

$$x = 0 \text{ **или** } x = 1.$$

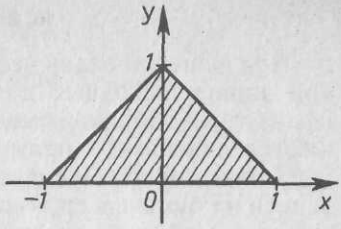


Рис. 18.

Условия такого вида называются *составными*. При записи составных условий на алгоритмическом языке пользуются служебными словами **и**, **или**, **не**. Использование служебных слов аналогично их использованию в обычном языке. Пусть a и b — условия. Условие a **и** b, соблюдается, если соблюдаются вместе и a и b. Условие a **или** b соблюдается, если соблюдается хотя бы одно из условий a, b, неважно какое. Условие **не** a соблюдается, если не соблюдается a, и наоборот.

Пример 9.3. Составить алгоритм решения следующей задачи: «Точка A задана координатами x и y. Определить, принадлежит ли точка A фигуре на плоскости (рис. 18)».

Этой фигуре будут принадлежать точки, координаты которых удовлетворяют условиям:

$$\begin{cases} y \geqslant 0, \\ |x| + |y| \leqslant 1. \end{cases}$$

Соответствующий алгоритм имеет вид:

```
алг фигура (вещ x, y, лит z)
    арг x, y
    рез z
нач
    если y ⩾ 0 и |x| + |y| ⩽ 1
        то z := „точка принадлежит фигуре“
        иначе z := „точка не принадлежит фигуре“
    все
кон
```

Используем этот алгоритм для проверки, принадлежит ли фигуре точка A с координатами $x = -0{,}4$ и $y = 0{,}95$.

Проверяем условие: $y \geqslant 0$ и $|x| + |y| \leqslant 1$. Подставим значения аргументов: $0{,}95 \geqslant 0$ и $|-0{,}4| + 0{,}95 \leqslant 1$.

Отношение $0{,}95 \geqslant 0$ является правильным, т. е. первая часть составного условия соблюдается. Однако $|-0{,}4| + 0{,}95 = 0{,}4 + 0{,}95 = 1{,}35 > 1$. Вторая часть условия и, следовательно, все условия не соблюдаются. Результату z присваивается значение „точка не принадлежит фигуре“.

10. ТАБЛИЧНЫЕ ВЕЛИЧИНЫ

При решении задач человек очень часто пользуется таблицами: при записи исходных данных, получении справочной информации и т. п. Таблицы бывают разными, но наиболее часто встречаются линейные и прямоугольные таблицы.

Значения, образующие линейную таблицу, располагаются при записи на бумаге в строчку или в столбец. Каждому значению, или элементу таблицы, соответствует его порядковый номер, и наоборот: стоит задать порядковый номер, и сразу ясно, о каком элементе таблицы идет речь.

Например, на метеостанции каждый час измеряется температура воздуха и значения измерений за сутки записываются в таблицу (табл. 8).

Таблица 8

Время измерения, ч	0	1	2	3	...	22	23
Температура, °С	17	16	15,5	14	...	18	17,5

Эта линейная таблица содержит 24 элемента, занумерованные от 0 до 23. Например, второй элемент таблицы имеет значение 15,5, а нулевой элемент — значение 17.

Запишем в таблицу средние температуры дня, измеренные за неделю (табл. 9).

Таблица 9

Дата измерения	22	23	24	25	26	27	28
Средняя температура, °С	15	15,5	17	20	18	17	17,5

Очевидно, что при хранении линейной таблицы порядковые номера хранить нет необходимости: зная начало нумерации, можно путем отсчета найти любой элемент. Кроме того, полезно знать и самый большой порядковый номер, так как это позволяет определить заранее размер таблицы. Таким образом, чтобы указать, что некоторая величина является линейной таблицей, нужно задать тип элементов таблицы, ее имя, начальный и конечный порядковые номера ее элементов.

В алгоритмах работы с табличными величинами это указание записывается следующим образом: служебное слово, указывающее тип (**цел**, **вещ**, **лит** и т. п.), затем служебное слово **таб** (**таб**лица), имя таблицы, за которым стоят в квадратных скобках начальный и конечный порядковые номера ее элементов, разделен-

ные двоеточием. Например:

вещ таб *время* [0:23]

вещ таб *средняя температура* [22:28]

Аналогичным образом происходит описание и прямоугольных таблиц.

Запишем таблицу умножения в виде прямоугольной (табл. 10).

Таблица 10

множитель \ множимое		1	2	3	...	8	9
	1	1	2	3	...	8	9
	2	2	4	6	...	16	18
	3	3	6	9	...	24	27
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	8	8	16	24	...	64	72
	9	9	18	27	...	72	81

Для прямоугольной таблицы должны быть указаны границы номеров как по вертикали, так и по горизонтали. Для данного примера это описание имеет вид:

цел таб *произведение* [1:9, 1:9]

границы множителя — границы множимого

Работа с таблицей сводится к работе с ее элементами. Для того чтобы указать, какой элемент таблицы в данный момент используется, достаточно указать его порядковый номер. Этот порядковый номер приписывается к имени таблицы в виде индекса.

Из курса математики хорошо известны обозначения a_0, a_1, b_5, a_i, x_{m+n} и т. п. Однако для того чтобы исполнителю было легче отличить переменные a_0, a_1 и т. д. от нулевого, первого и т. д. элемента таблицы a, принято порядковый номер элемента таблицы заключать в квадратные скобки и помещать вслед за именем таблицы на том же уровне строки, например:

a [i], *произведение* [2,7] и т. д.

Рассмотрим примеры использования таблиц в алгоритмах.

Пример 10.1. Построить алгоритм, который находит сумму S вещественных чисел, образующих таблицу a из 1000 элементов, занумерованных от 1 до 1000. Для отсчета количества просумми-

рованных чисел используется промежуточная целая переменная i, которая пересчитывает элементы таблицы.

```
алг сумма (вещ таб a [1:1000], вещ S)
    арг a
    рез S
нач цел i
    i:=1
    S:=0
    пока i≤1000
    нц
        S:=S+a [i]
        i:=i+1
    кц
кон
```

Использование переменной i в качестве индекса позволяет записать прибавление по очереди всех элементов таблицы в виде одной повторяемой команды присваивания $S:=S+a\,[i]$.

Пример 10.2. Построить более сложный алгоритм — алгоритм заполнения таблицы умножения (табл. 10).

```
алг таблица умножения(цел таб произведение [1:9, 1:9])
    рез произведение
нач цел i, j
    i:=1
    пока i ≤ 9
    нц
        j:=1
        пока j≤9
        нц
            произведение [i, j]:=i·j
            j:=j+1
        кц
        i:=i+1
    кц
кон
```

Рассмотрим подробнее этот алгоритм. В нем есть две команды повторения, причем одна из команд повторения входит в состав другой команды.

Сначала исполнитель присваивает i значение 1 и переходит к выполнению первой внешней команды повторения. Так как усло-

вие цикла при этом значении соблюдается, он должен выполнить все команды, входящие в состав этого цикла.

Он присваивает j значение 1 и переходит к выполнению второй внутренней команды повторения. Команда повторения считается выполненной, когда перестает соблюдаться входящее в ее состав условие, т. е. когда j станет больше 9. Это произойдет после того, как входящая в ее состав серия команд будет выполнена 9 раз. В результате окажется заполненной первая строка таблицы *произведение*.

Поясним это.

Элементы первой строки таблицы *произведение* имеют индексы:

произведение [1,1], *произведение* [1,2], ..., *произведение* [1,9].

Команда присваивания, входящая в указанную серию, имеет вид:

$$произведение\ [i,\ j] := i \cdot j,$$

при этом i равно 1, а j меняется от 1 до 9, т. е. действительно первая строка таблицы *произведение* в результате выполнения второй (внутренней) команды повторения окажется заполненной.

После этого исполнитель продолжит выполнение первой (внешней) команды повторения. Он увеличит значение i на единицу так, что i станет равным 2. Проверит условие внешней команды повторения ($i \leqslant 9$), снова присвоит j значение 1. Затем снова надо выполнять вторую (внутреннюю) команду повторения. Как мы знаем, в результате выполнения этой команды заполнится строка с номером i таблицы *произведение*.

Продолжая этот процесс, исполнитель заполнит всю таблицу *произведение*.

Вопросы

1. Какие величины встречаются в рассмотренных алгоритмах?
2. Что такое имя величины? Приведите примеры имен.
3. Что такое значение величины? Приведите примеры изменения значений величин в процессе исполнения алгоритма.
4. Приведите примеры постоянных и переменных величин.
5. Какие величины называются: а) аргументами; б) результатами алгоритма? Приведите примеры.
6. Какие типы величин используются в алгоритмическом языке? Приведите примеры.
7. Какие величины называются промежуточными? Приведите примеры.
8. Команды какого вида называются командами присваивания? Поясните на примерах их назначение.
9. Как выполняется команда присваивания?
10. Какие знаки отношений между величинами используются в алгоритмическом языке?

11. Чем различаются знаки отношений числовых и литерных величин?

12. Для чего в алгоритмическом языке служат знаки отношений между величинами?

13. Какие условия называются: а) простыми; б) составными? Приведите примеры.

14. Какие величины называются табличными? Приведите примеры.

15. Какая таблица называется: а) линейной; б) прямоугольной? Приведите примеры.

Упражнения[1]

1. Укажите тип следующих величин: 4; 4,0; «четыре»; «4,4»; «аргумент»; 0,0; «ноль»; 0.

2. Составьте и запишите алгоритм решения задачи: «В трех сосудах содержится вода. В первом сосуде содержится V_1 л воды температуры t_1, во втором — V_2 л температуры t_2, в третьем — V_3 л температуры t_3. Воду слили в один сосуд. Найдите объем и температуру t воды в этом сосуде (изменением объема воды при изменении температуры пренебречь)».

Исполните алгоритм для $V_1 = 0{,}2$ л, $V_2 = 1$ л, $V_3 = 0{,}7$ л; $t_1 = 1°C$, $t_2 = 3°C$, $t_3 = 20°C$.

3. Составьте и запишите алгоритм решения задачи.

Принадлежит ли точка $A(x, y)$ фигуре на плоскости (рис. 19)? Исполните алгоритм для следующих координат точек:

x	0,2	– 0,1	– 0,15	0,25	0,3	0,9	0	1	– 0,5
y	0,3	– 0,1	– 0,85	– 0,55	0,4	0,15	0	1	0,5

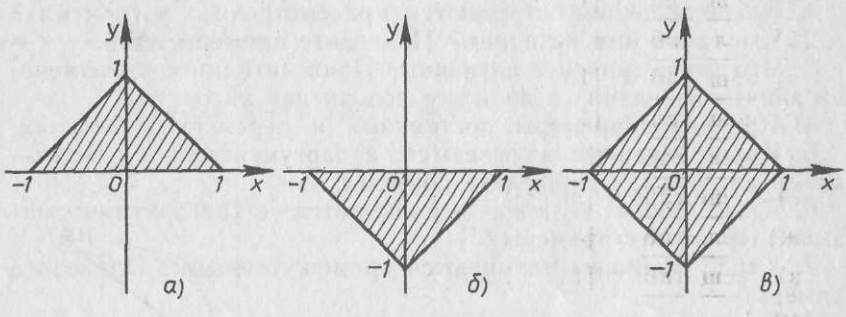


Рис. 19.

[1] Во всех упражнениях на составление алгоритмов исполнителю доступны все арифметические действия.

4. Заполните таблицу промежуточных значений для алгоритмов:

а) нахождения наибольшего общего делителя чисел 161, 253;

б) решения уравнения $2x^2 + 3x - 5 = 0$.

5. Запишите с использованием промежуточных величин алгоритмы:

а) вычисления выражения $y = \frac{a^2}{3} + \frac{a^2+4}{6} + \frac{\sqrt{a^2+4}}{4} + \frac{(\sqrt{a^2+4})^3}{4}$ при $a = 11{,}7$;

б) решения уравнения $ax^3 + bx = 0$.

6. Составьте и запишите следующие алгоритмы:

Даны площадь круга S_1 и площадь квадрата S_2. Определите, поместится ли: 1) круг в квадрате; 2) квадрат в круге. Проверьте алгоритм для нескольких значений S_1 и S_2.

7. Запишите алгоритм вычисления функции $y = f(x)$, заданной графиком (рис. 20), используя знаки отношения, приведенные на с. 37.

8. Запишите алгоритм вычисления факториала натурального числа n по формуле $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$.

9. В одной команде разрешается использовать только одну операцию. Запишите алгоритмы вычисления выражений:

а) $y = \frac{2x^2 + \sqrt{x^3+1}}{2}$; б) $y = 2\sin x^2 + 4\cos x$;

в) $y = 4\log(x^2 + 2\sqrt{\frac{1}{x} + x^3})$.

10. Объясните условие *кошка* = „кошка“. Что является именем, что — значением величины?

11. Из скольких элементов состоят таблицы, описанные следующим образом:

а) **вещ таб** x [1:10];

б) **цел таб** x [4:5];

в) **вещ таб** x [100:100];

г) **вещ таб** x [1:N];

д) **цел таб** x [−1:1];

е) **вещ таб** x [M:N];

ж) **цел таб** x [3:4, 5:10];

з) **вещ таб** x [1:N, 5:10]?

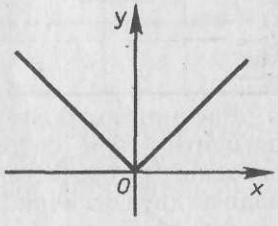


Рис. 20.

12. Как задать вектор на плоскости, таблицей?

13. Запишите правило сложения векторов (рис. 21) как действие с таблицами, их задающими.

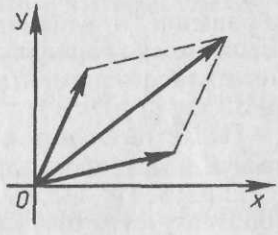


Рис. 21.

§ 4. ВСПОМОГАТЕЛЬНЫЕ АЛГОРИТМЫ

11. ПОНЯТИЕ ВСПОМОГАТЕЛЬНОГО АЛГОРИТМА

При построении новых алгоритмов могут использоваться алгоритмы, составленные раньше. Алгоритмы, целиком используемые в составе других алгоритмов, будем называть *вспомогательными* (или подчиненными) алгоритмами. Не исключено, что алгоритм, содержащий ссылку на вспомогательный, сам в определенной ситуации может оказаться в роли вспомогательного алгоритма. Использование ранее составленных алгоритмов при составлении новых находит широкое применение в практике алгоритмизации.

Все приведенные в учебнике алгоритмы, представляющие интерес для последующего использования, объединены в особый фонд — *библиотеку алгоритмов,* которая помещена в приложении.

Правила включения вспомогательных алгоритмов в основные в процессе их использования рассмотрим на примерах.

Пример 11.1. Алгоритм поиска большего из двух чисел α и β (обозначим его БИД) может быть записан следующим образом:

```
алг БИД (вещ α, β, γ)
    арг α, β
    рез γ
нач если α⩾β
         то    γ := α
         иначе γ := β
    все
кон
```

Рассмотрим более подробно использование вспомогательного алгоритма при составлении основного алгоритма. Записывая основной алгоритм, мы в какой-то момент обнаруживаем, что нам надо из двух величин x и y выбрать бóльшую и результат присвоить некоторой величине z. Чтобы воспользоваться вспомогательным алгоритмом, надо: 1) присвоить значение величины x аргументу α, значение величины y аргументу β; 2) исполнить алгоритм БИД; 3) значение результата γ присвоить переменной z. Такое обращение к вспомогательному алгоритму в алгоритмическом языке можно записать в виде одной простой команды вызова вспомогательного алгоритма, который в нашем случае будет иметь вид БИД (x, y, z).

После выполнения команды вызова выполняется следующая за ней в основном алгоритме команда.

Пример 11.2. Алгоритм поиска большего из трех чисел (обозначим его БИТ) можно составить в форме двукратного обращения к алгоритму БИД:

```
алг БИТ (вещ a, b, c, y)
    арг a, b, c
    рез y
нач вещ z
    БИД (a, b, z)
    БИД (z, c, y)
кон
```

При записи шагов исполнения основного алгоритма, содержащего в себе вызов вспомогательного алгоритма, в таблице значений записываются только величины основного алгоритма. При записи шага, состоящего в вызове вспомогательного алгоритма, записываются новые значения тех величин, которым присваиваются результаты данного исполнения вспомогательного алгоритма. Если есть необходимость проследить исполнение вспомогательного алгоритма, то для каждого обращения к нему составляется своя таблица значений.

Исполним алгоритм БИТ для аргументов 3, 9, 5. Составим таблицу значений (табл. 11).

Таблица 11

Шаги алгоритма	Аргументы			Промежуточная величина	Результат	Проверка условий
	a	*b*	*c*	*z*	*y*	
	3	9	5			
1 2				9	9	

Из этой таблицы видно, что каждое исполнение вспомогательного алгоритма — это один шаг основного алгоритма. Проследим теперь, как исполняется алгоритм БИД для аргументов 3, 9. Составим таблицу значений (табл. 12).

Таблица 12

Шаги алгоритма	Аргументы		Результат	Проверка условий
	α	β	γ	
	3	9		
1 2			9	$3 \geqslant 9$ (нет)

Аналогично можно было бы проследить исполнение алгоритма БИД для аргументов 9,5.

Пример 11.3. Составить алгоритм поиска наибольшего элемента в линейной таблице из n чисел $x\,[1:n]$, используя алгоритм БИД как вспомогательный.

Наибольшее число в линейной таблице можно отыскать путем многократного применения алгоритма БИД к очередному элементу таблицы и результату предыдущего применения алгоритма БИД.

Запись алгоритма поиска наибольшего элемента линейной таблицы (НЭЛТ) на алгоритмическом языке:

```
алг НЭЛТ (цел n, вещ таб x [1:n], вещ y)
    арг n, x
    рез y
нач цел i
    i:=2; y:=x[1]
    пока i≤n
    нц
        БИД (y, x [i], y); i:=i+1
    кц
кон
```

12. ПОСЛЕДОВАТЕЛЬНОЕ ПОСТРОЕНИЕ АЛГОРИТМА

Метод вычленения вспомогательного алгоритма может быть успешно использован в процессе поиска и разработки нового алгоритма тогда, когда вспомогательный алгоритм еще неизвестен.

Процесс последовательного построения алгоритма может выглядеть следующим образом. Алгоритм сначала формулируется в самых «крупных» командах, при этом в записи алгоритма могут использоваться команды, выходящие за рамки возможностей исполнителя. Затем на каждом последующем этапе отдельные детали алгоритма уточняются, при этом недоступные исполнителю команды записываются как вызовы вспомогательных алгоритмов. После этого так же строятся вспомогательные алгоритмы. Процесс продолжается до тех пор, пока все алгоритмы не будут состоять из команд, понятных исполнителю. Такой способ построения алгоритма называется *методом последовательного уточнения.*

Рассмотрим этот метод на примере построения алгоритма вычисления степени $y=a^x$, где x — целое число, $a\neq 0$. В курсе алгебры степень с целым показателем определяется так:

$$a^x=\begin{cases} 1, & \text{если } x=0,\\ a^x, & \text{если } x>0,\\ \frac{1}{a^{-x}}, & \text{если } x<0. \end{cases}$$

Учитывая, что $\frac{1}{a^{-x}}=\left(\frac{1}{a}\right)^{-x}$, запишем искомый алгоритм:

алг степень с целым показателем (**вещ** a, **цел** x, **вещ** y)

арг a, x
рез y
нач
если $x = 0$
то $y := 1$
иначе
если $x > 0$
то вычислить $y = a^x$, x — целое положительное число
иначе вычислить $y = \left(\frac{1}{a}\right)^{-x}$, $-x$ — целое положительное число
все
все
кон

Замечаем, что в тексте алгоритма в двух местах предписывается выполнять похожие действия, а именно вычислять степень с одним натуральным показателем, но с различными основаниями. В этих местах может быть ссылка на один и тот же вспомогательный алгоритм вычисления степени с натуральным показателем, и тогда алгоритм приобретает следующий вид:

алг степень с целым показателем (**вещ** a, **цел** x, **вещ** y)
арг a, x
рез y
нач
если $x = 0$
то $y := 1$
иначе
если $x > 0$
то степень (a, x, y)
иначе степень $\left(\frac{1}{a}, -x, y\right)$
все
все
кон

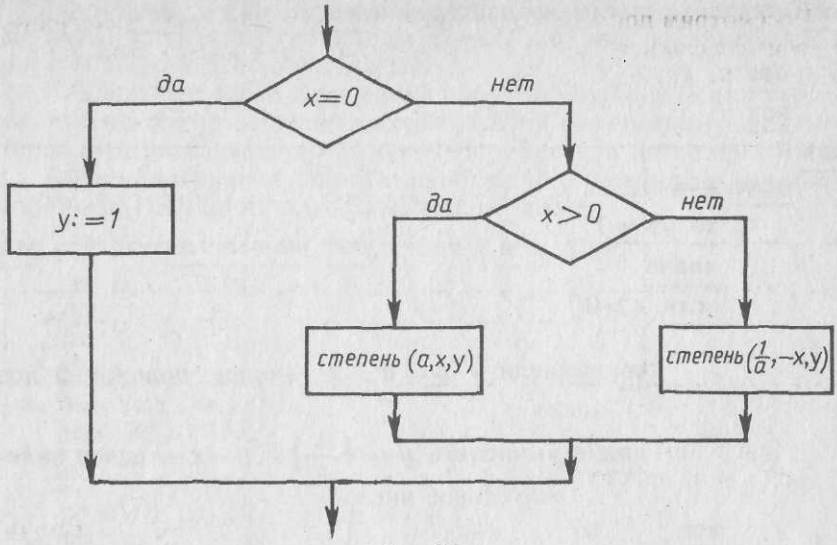


Рис. 22.

Схема этого алгоритма приведена на рисунке 22.

Теперь остается построить вспомогательный алгоритм вычисления степени с натуральным показателем $y = \alpha^n$. При вычислении α^n по формуле $\alpha^n = \underbrace{\alpha \cdot \alpha \cdot \ldots \cdot \alpha}_{n \text{ раз}}$ нужно проделать $(n - 1)$ умножение.

При составлении алгоритма удобно пользоваться формулой

$$\alpha^n = 1 \cdot \underbrace{\alpha \cdot \alpha \cdot \ldots \cdot \alpha}_{n \text{ раз}},$$

в которой число умножений равно показателю степени.

```
алг степень (вещ α, нат n, вещ y)
    арг α, n
    рез y
нач цел i
    y := 1; i := 1
    пока i ⩽ n
    нц
        y := y · α
        i := i + 1
    кц
кон
```

Рассмотрим процесс исполнения общего алгоритма вычисления степени с целым показателем $y = a^x$ для двух аргументов 5, —2 (табл. 13).

Таблица 13

Шаги алгоритма	Аргументы		Результат	Проверка условий
	a	x	y	
	5	— 2		
1				— 2 = 0 (нет)
2				— 2 > 0 (нет)
3			0,04	

Проследим теперь исполнение вспомогательного алгоритма степень для аргументов 0,2, 2 (табл. 14).

Таблица 14

Шаги алгоритма	Аргументы		Промежуточная величина	Результат	Проверка условий
	α	n	i	y	
	0,2	2			
1				1	
2			1		
3					1 ⩽ 2 (да)
4				0,2	
5			2		
6					2 ⩽ 2 (да)
7				0,04	
8			3		
9					3 ⩽ 2 (нет)

Вопросы

1. Для чего нужны вспомогательные алгоритмы?
2. Как записывается команда вызова вспомогательного алгоритма?
3. Каков порядок исполнения основного алгоритма при использовании вспомогательных алгоритмов?
4. В чем заключается метод последовательного уточнения при построении алгоритмов?
5. Может ли алгоритм, содержащий ссылку на вспомогательный, оказаться в роли вспомогательного?

Упражнения

1. Составьте алгоритм вычисления по формуле

$$y = \frac{|x+5|}{|3x^2 - x + 2| + 9},$$

используя вспомогательный алгоритм вычисления модуля числа МОД (a, m).

2. В четырехугольнике $ABCD$ $AB = x$, $BC = y$, $CD = z$, $AD = t$, $AC = d$. Составьте алгоритм вычисления площади четырехугольника, используя вспомогательный алгоритм ГЕРОН (a, b, c, S) вычисления площади треугольника по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \text{ где } p = \frac{(a+b+c)}{2}.$$

3. Постройте алгоритм решения биквадратного уравнения $ax^4 + bx^2 + c = 0$, используя как вспомогательный алгоритм решение квадратного уравнения (с. 32).

4. Постройте алгоритм нахождения наименьшего общего кратного (НОК) двух натуральных чисел, используя алгоритм нахождения наибольшего общего делителя (с. 36) как вспомогательный. Исполните алгоритм для аргументов 14 и 26, 21 и 84.

Указание. Для любых натуральных чисел a, b, c справедливо тождество $a \cdot b =$ НОД $(a, b) \cdot$ НОК (a, b).

5. Постройте алгоритм нахождения наибольшего общего делителя трех натуральных чисел, используя вспомогательный алгоритм нахождения наибольшего общего делителя двух чисел. Исполните построенный алгоритм для аргументов 24, 18, 12.

Раздел II

ПОСТРОЕНИЕ АЛГОРИТМОВ ДЛЯ РЕШЕНИЯ ЗАДАЧ

Как уже отмечалось, одна из основных целей курса информатики — научиться решать задачи с использованием ЭВМ. Решение задачи в этом случае требует предварительной работы и производится в несколько этапов. Часть этих этапов выполняется только человеком, а другая часть — человеком и ЭВМ.

Цель данного раздела — познакомиться с этапами решения задачи, предшествующими непосредственной работе на ЭВМ. Эти этапы заканчиваются построением алгоритма для решения задачи.

Следующий этап, связанный с исполнением алгоритма на ЭВМ, будет рассмотрен в X классе.

§ 5. ЭТАПЫ РЕШЕНИЯ ЗАДАЧИ С ИСПОЛЬЗОВАНИЕМ ЭВМ

Формулировка условия любой математической задачи начинается с описания исходных данных и предпосылок, которые излагаются на языке строго определенных математических понятий. Затем формулируется цель решения, т. е. указывается, что именно необходимо определить в результате решения задачи. Точную формулировку условия задачи называют также математической постановкой задачи, и решение любой задачи начинается именно с ее постановки. В результате постановки задачи выделяются исходные данные или аргументы и те величины, значение которых нужно определить, т. е. результаты. Постановка задачи является первым этапом ее решения.

При решении практических задач приходится иметь дело с реальными объектами — явлениями природы, физическими и производственными процессами, планами выпуска продукции и т. п. Для того чтобы поставить такую задачу, необходимо сначала описать объект исследования в математических терминах, т. е. построить его математическую модель, позволяющую свести исследование реального объекта к решению математической задачи. Степень соответствия модели реальному объекту проверяется практикой, экспериментом. Практика дает возможность оценить построенную модель и уточнить ее в случае необходимости.

Рассмотрим следующий пример. Пусть необходимо определить, каким образом будет двигаться тело, брошенное под углом к горизонту, и найти дальность бросания. Для решения задачи необходимо сделать некоторые допущения, при которых будет строиться математическая модель. В качестве таких допущений можно принять следующие предложения:

1. Пренебречь кривизной Земли, считая ее поверхность плоскостью.
2. Пренебречь движением Земли.
3. Считать ускорение свободного падения g постоянным.
4. Пренебречь сопротивлением воздуха.

Из курса физики известно, что при таких предположениях движение тела, брошенного под углом α к горизонту с начальной скоростью v_0, описывается системой уравнений:

$$\begin{cases} x = v_0 \cdot \cos\alpha \cdot t, \\ y = v_0 \cdot \sin\alpha \cdot t - g\dfrac{t^2}{2}. \end{cases} \qquad (1)$$

При этом предполагается, что эти уравнения описывают движение тела в неподвижной системе координат, начало которой совмещено с точкой бросания, ось x направлена вдоль поверхности Земли в сторону бросания, а ось y — вертикально вверх.

Эти уравнения являются математической моделью, описывающей движение тела. Исходя из этой модели, мы получим ответ на интересующий нас вопрос о дальности бросания.

Выразив из первого уравнения t и подставив его во второе уравнение, получим уравнение траектории:

$$y = x\,\mathrm{tg}\,\alpha - x^2\frac{g}{2v_0^2\cos^2\alpha}.$$

Эта траектория приведена на рисунке 23.

Решая квадратное уравнение $x\,\mathrm{tg}\,\alpha - x^2\dfrac{g}{2v_0^2\cos^2\alpha} = 0$, можно определить точки пересечения этой траектории с осью x: $x_1 = 0$ — точка бросания и $x_2 = \dfrac{v_0^2}{g}\sin 2\alpha$ — точка падения.

Создание математической модели исследуемого явления позволяет поставить задачу математически. В нашем примере задача формулируется так: «Пусть движение тела, брошенного под углом α к горизонту с начальной скоростью v_0, описывается системой уравнений (1), и пусть заданы значения v_0 и α. Требуется определить дальность (l) полета тела».

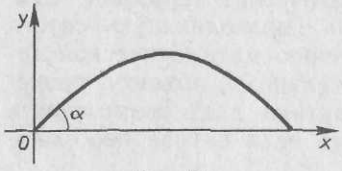


Рис. 23.

Исходными предложениями задачи являются уравнения (1), а исходными данными (аргументами) — значения v_0 и α. Результатом решения является значение l.

После постановки задачи начинается поиск метода ее решения. При использовании ЭВМ для решения задачи строится алгоритм. Таким образом, следующим этапом решения задачи является построение алгоритма. В рассмотренном примере алгоритм строится достаточно просто:

1) выразить t из первого уравнения системы (1) через x
2) подставить полученное выражение во второе уравнение системы (1)
3) положить y равным 0
4) решить полученное квадратное уравнение
5) взять в качестве ответа t ненулевой корень этого уравнения.

Такой алгоритм может быть записан на алгоритмическом языке или в виде схемы.

Теперь необходимо записать алгоритм в виде, доступном исполнению на ЭВМ. Так как ЭВМ «понимает» специальный язык, называемый языком программирования, то алгоритм должен быть записан на таком языке. Этот этап решения задачи называется записью алгоритма на языке программирования. Этот и последующие этапы решения задачи будут рассматриваться позднее, в X классе.

Следующий этап — исполнение алгоритма с помощью ЭВМ. Этот этап завершается получением результата решения.

Наконец, завершающий этап решения задачи — анализ полученных результатов. Этот анализ проводится с целью определения, насколько точно полученные результаты соответствуют реальности. Анализ результатов помогает уточнить модель, если это необходимо. Кроме того, при решении задачи могут быть получены результаты, которые противоречат смыслу задачи. Например, если математическая модель, описывающая количество изделий, выпускаемых за день некоторым предприятием, представляет собой квадратное уравнение, один из корней которого отрицателен, то такой результат противоречит смыслу задачи и должен быть отброшен.

Во многих случаях анализ результатов задачи тоже проводится на ЭВМ.

Итак, решение задачи с помощью ЭВМ разбивается на следующие этапы:

— постановка задачи, включающая построение математической модели и выделение аргументов и результатов;
— построение алгоритма;
— запись алгоритма на языке программирования;
— реализация алгоритма с помощью ЭВМ;
— анализ полученных результатов.

В следующих параграфах будут рассмотрены отдельные этапы решения задачи, причем основное внимание будет обращено на построение алгоритма для решения задач.

Упражнения

1. Постройте математическую модель движения тела, брошенного вертикально вверх.
2. Постройте математическую модель движения тела массой m по наклонной плоскости с углом α. Трением пренебречь.
3. Постройте математическую модель движения тела массой m по окружности радиуса R.

§ 6. АЛГОРИТМЫ ДЛЯ РАБОТЫ С ТАБЛИЧНЫМИ ВЕЛИЧИНАМИ

Многие задачи, которые решаются с помощью ЭВМ, связаны с обработкой больших объемов информации. Для удобства обработки информация часто сводится в линейные или прямоугольные таблицы. Тогда обработка состоит из поиска в таблице нужного элемента, записи в таблицу новых элементов, изменения порядка элементов в таблице и т. п. Преимущества ЭВМ перед другими исполнителями алгоритма и состоят в том, что ЭВМ может решать задачи по обработке таблиц, содержащих десятки и сотни тысяч элементов, достаточно быстро и тем самым освобождать человека от утомительной и непродуктивной (рутинной) работы.

Рассмотрим задачу поиска заданного элемента в линейной таблице. Будем предполагать, что элементы таблицы являются числами.

Итак, предположим, что имеется линейная таблица, имеющая имя „*ключ*“, элементы которой — вещественные числа:

вещ таб *ключ* [1:*n*].

Задача состоит в том, чтобы определить, имеется ли в этой таблице элемент, значение которого совпадает с заданным числом L. Если такой элемент существует, то необходимо в качестве ответа получить порядковый номер этого элемента. В противном случае необходимо выдать номер, равный нулю. Будем просматривать подряд все элементы таблицы, начиная с первого, и сравнивать их значения с L. Такой просмотр надо продолжать до тех пор, пока не будет найден элемент таблицы, равный L, или до тех пор, пока не будет просмотрена вся таблица (если такого элемента не существует).

Опишем алгоритм поиска в таблице. Он будет использовать величины i (число просмотренных элементов) и k (номер искомого элемента).

Вначале i будет равно 0, так как мы еще не просмотрели ни одного элемента таблицы. Затем i будет возрастать: мы будем просматривать все новые и новые элементы таблицы, пытаясь найти среди них элемент, равный L. Если такой элемент обнаружится, то k станет равным его номеру, а до тех пор k будет равно 0. Если числа L в таблице нет, то k так и останется равным нулю.

Таким образом, значение переменной k будет сигнализировать об успехе ($k>0$) или неуспехе ($k=0$) поиска.

```
алг поиск (цел n, вещ таб ключ [1:n], вещ L, цел k)
    арг ключ, n, L
    рез k
нач
    цел i
    i:=0; k:=0
    пока i≠n и k=0
    нц
        i:=i+1
        если ключ [i]=L
            то k:=i
        все
    кц
кон
```

Пояснение. Посмотрим, как будет выполняться наш алгоритм. Основная его часть — это команда повторения. Но еще до нее величинам i, k присваивается значение 0. Это соответствует смыслу этих переменных, ведь ни одного элемента еще не просмотрено и число L в таблице не найдено. Что происходит в команде повторения? Входящая в нее серия выполняется многократно и заканчивается, когда мы доходим до конца таблицы ($i=n$) или находим нужный элемент (k стало положительным).

Пусть мы уже i раз выполнили серию, причем $i\neq n$, и среди элементов (*ключ* [1], ..., *ключ* [i]) нет L (а значит, $k=0$). Тогда мы начинаем выполнение серии еще раз. В первой команде серии i увеличивается на 1, т. е. становится равным номеру элемента, который мы должны сейчас просмотреть. Если этот элемент равен L, т. е. мы нашли нужный элемент, то k становится равным номеру этого элемента. Выполнение серии закончено, и больше она выполняться не будет. Если этот элемент отличен от L, то значение $k=0$ сохраняется, серия закончена и мы снова возвращаемся к проверке условия цикла.

Поиск нужного элемента может быть ускорен, если элементы таблицы упорядочены по своей величине, например в порядке возрастания своих значений или в алфавитном порядке. Так, если фамилии в таблице расположены в алфавитном порядке, то при поиске нужной фамилии можно отыскать в таблице нужную первую букву и просматривать только те фамилии, которые начинаются с этой буквы.

Упорядочение таблицы приходится выполнять так часто, что,

по оценкам специалистов, больше четверти времени работы всех ЭВМ приходится на выполнение этого процесса, который называют также сортировкой. По этой причине для решения задачи сортировки с помощью ЭВМ было придумано много различных алгоритмов. Цель создания таких алгоритмов — обеспечить достаточно быстрое решение задачи сортировки для разного типа таблиц.

Рассмотрим один из наиболее простых алгоритмов сортировки, идея которого состоит в следующем. Будем сначала просматривать всю таблицу подряд с первого до последнего элемента и найдем элемент, имеющий наименьшее значение. Поменяем местами найденный элемент и первый элемент таблицы. В результате на первом месте окажется элемент, имеющий наименьшее значение. Теперь будем просматривать все элементы таблицы, начиная со второго, и снова найдем среди них элемент, имеющий наименьшее значение. Поменяем местами этот элемент со вторым элементом таблицы, в результате чего на втором месте будет стоять элемент, имеющий наименьшее значение среди оставшихся. Продолжим процесс. В последний раз останется просмотреть только два последних элемента таблицы, выбрать среди них элемент с наименьшим значением и поставить его на предпоследнее место в таблице. После этого таблица будет упорядочена в порядке возрастания значений составляющих ее элементов.

Описанную задачу сортировки можно разбить на две задачи: задачу поиска в таблице элемента с наименьшим значением и задачу упорядочения таблицы путем перестановки найденного элемента с очередным (первым, вторым и т. д.) элементом таблицы. Рассмотрим каждую из этих задач.

Пусть задана линейная таблица A, элементы которой нумеруются от K до N ($K < N$).

Построим алгоритм поиска наименьшего элемента линейной таблицы.

В этом алгоритме элементы таблицы будут просматриваться по порядку, от первого до последнего. Для этого будет использоваться переменная i — номер первого непросмотренного элемента таблицы. Кроме i, будут использоваться еще две переменные:

МИН — «минимум из уже просмотренных элементов».
l — «номер элемента, минимального среди уже просмотренных».

Работа алгоритма начинается с серии команд, в которой мы присваиваем переменной *МИН* значение начального элемента таблицы. Этот элемент уже просмотрен, и мы берем K в качестве значения для l; просмотрен один элемент $A\ [K]$, и он минимален среди просмотренных. Берем $K+1$ в качестве значения для i — номера первого непросмотренного элемента.

Выполнение серии, входящей в команду повторения, происходит следующим образом. Если *МИН* оказался больше, чем очередной элемент $A\ [i]$, нужно сменить *МИН*, и мы это делаем, присваи-

вая ему значение $A\ [i]$; соответственно с этим и значение l меняется, теперь оно равно i. Так как элемент с номером i уже просмотрен, серия завершается увеличением номера i на 1:

$$i := i+1.$$

Работа алгоритма завершается, когда вся таблица просмотрена и номер ее минимального элемента найден. Этот номер и является результатом.

Алгоритм имеет следующий вид:

```
алг МИНЭЛЕМЕНТ (цел K, N, вещ таб A[K:N], цел l)
    арг A, K, N
    рез l
нач цел i, вещ МИН
    МИН := A[K]; l := K; i := K + 1
    пока i ⩽ N
    нц
        если МИН > A[i]
            то МИН := A[i]; l := i
        все
        i := i + 1
    кц
кон
```

Построим теперь алгоритм упорядочения линейной таблицы, используя алгоритм МИНЭЛЕМЕНТ в качестве вспомогательного. Задача упорядочения таблицы формулируется следующим образом. Пусть задана линейная таблица C, элементы которой нумеруются от n до M $(n<M)$:

```
вещ таб C [n:M]
```

Необходимо переставить элементы этой таблицы так, чтобы они шли в порядке возрастания.

Идея алгоритма упорядочения состоит в следующем. Применив алгоритм МИНЭЛЕМЕНТ к таблице $C\ [n:M]$, мы определим номер l элемента этой таблицы, имеющего наименьшее значение. После этого поменяем значения элементов $C\ [n]$ и $C\ [l]$. Тогда на n-м месте таблицы C (напомним, что таблица C начинается с элемента с номером n) окажется самый маленький по величине элемент. На следующем шаге применяем алгоритм МИНЭЛЕМЕНТ к таблице $C\ [n+1:M]$ и снова определяем номер l минимального элемента этой таблицы. Поменяем местами элементы $C\ [n+1]$ и $C\ [l]$, тогда на $n+1$-м месте окажется самый маленький из оставшихся элементов. Далее будем применять алгоритм МИНЭЛЕМЕНТ к таблицам $C\ [n+2:M]$, $C\ [n+3:M]$, ...

...,$C\,[M-1,\ M]$ и менять местами элементы $C\,[n+2]$ и $C\,[l]$, $C\,[n+3]$ и $C\,[l]$ и, наконец, $C\,[M-1]$ и $C\,[l]$. В результате таблица C окажется упорядоченной.

На алгоритмическом языке алгоритм упорядочения будет выглядеть следующим образом:

```
алг упорядочение (цел n, M, вещ таб C[n:M])
    арг C, n, M
    рез C
нач цел i, l, вещ R
    i:= n
    пока i<M
    нц
        МИНЭЛЕМЕНТ (i, M, C, l)
        R := C [i]
        C [i] := C [l]
        C [l] := R
        i := i+1
    кц
кон
```

Заметим, что обмен местами двух элементов таблицы C осуществляется с помощью трех команд:

$$R := C\,[i]$$
$$C\,[i] := C\,[l]$$
$$C\,[l] := R$$

Это объясняется тем, что как только выполнится команда $C\,[i] := C\,[l]$, то «старое» значение $C\,[i]$ будет «забыто», поэтому предварительно нужно запомнить это значение, присвоив его вспомогательной переменной R. Тогда третья команда обеспечит присваивание этого значения переменной $C\,[l]$.

В заключение рассмотрим задачу о вычислении значений элементов в некоторой последовательности, которые сводятся в линейную таблицу, удобную для дальнейшего использования.

В качестве примера рассмотрим получение так называемых чисел Фибоначчи, названных по имени итальянского математика средневековья (Леонардо Пизанского). Эти числа образуют бесконечную последовательность, причем каждое число в ней получается по следующему правилу: первые два числа (обозначим их F_1 и F_2) равны 1, т. е. $F_1 = F_2 = 1$, а каждое следующее число, начиная с третьего, равно сумме двух предыдущих, т. е. $F_3 = F_1 + F_2$, $F_4 = F_3 + F_2$ и т. д., вообще $F_i = F_{i-1} + F_{i-2}$ для любого $i \geqslant 3$.

Построим алгоритм получения N чисел Фибоначчи, которые будем рассматривать как элементы линейной таблицы $F\,[1:N]$.

Этот алгоритм на алгоритмическом языке имеет вид:

```
алг Фибоначчи (цел N, цел таб F[1:N])
    арг N
    рез F
нач цел i
    F[1]:=1
    F[2]:=1
    i:=3
    пока i⩽N
    нц
        F[i]:=F[i-1]+F[i-2]
        i:=i+1
    кц
кон
```

Упражнения

1. Постройте алгоритм определения номера элемента с максимальным значением из линейной таблицы $B[1:n]$ в предположении, что значениями элементов таблицы B являются целые числа.

2. Используя алгоритм упражнения 1 в качестве вспомогательного, постройте алгоритм упорядочения элементов линейной таблицы по убыванию их значений.

3. Постройте алгоритм для подсчета числа положительных элементов в линейной таблице A.

4. Задана линейная упорядоченная таблица, состоящая из целых чисел. Постройте алгоритм, с помощью которого значения всех элементов таблицы, меньшие 100, заменялись бы числом 100.

5. Постройте алгоритм, определяющий, сколько раз число 10 встречается среди элементов линейной таблицы $A[1:1000]$, состоящей из целых чисел.

6. Постройте алгоритм для определения среднего арифметического n чисел, сведенных в линейную таблицу $A[1:n]$. (Напомним, что среднее арифметическое n чисел вычисляется по формуле $c=\frac{a_1+a_2+\ldots+a_n}{n}$.)

7. Постройте алгоритм для определения суммы всех положительных чисел, входящих в линейную таблицу $x[1:n]$.

8. Постройте алгоритм:

а) переписи значений элементов одной таблицы в другую;

б) заполнения линейной таблицы числом 0;

в) заполнения i-й строки прямоугольной таблицы числом 0.

§ 7. ПОСТРОЕНИЕ АЛГОРИТМОВ ДЛЯ РЕШЕНИЯ ЗАДАЧ ИЗ КУРСА МАТЕМАТИКИ

Понятие алгоритма оказывается очень важным при решении многих математических задач. При этом объекты, действия над которыми описывает алгоритм, могут быть самыми разными. Построив алгоритм решения задачи и записав его на алгоритмическом языке, можно многократно применять его, уже не вникая в условия задачи.

Пример 1. Алгоритм вычисления значения многочлена.

Пусть дан многочлен $x^4 + 2x^3 + 3x^2 - 5x - 15$. Требуется вычислить значение многочлена при $x = 2$.

Существует общепринятый способ вычисления значений многочлена: найти числа x^2, x^3 и x^4 при $x = 2$, затем вычислить $2x^3$, $3x^2$ и $5x$ и, наконец, путем сложения и вычитания получить значение многочлена. При вычислении многочлена потребовалось десять действий, так как $x^2 = x \cdot x$, $x^3 = x \cdot x^2$, $x^4 = x \cdot x^3$.

Вычисление значения многочлена может быть упрощено. Для этого перепишем заданный многочлен в виде

$$x^4+2x^3+3x^2-5x-15 = (((x+2)x+3)x-5)x-15$$

и будем выполнять операции в порядке, определяемом скобками. Тогда потребуется семь арифметических действий. Этот способ вычисления значения многочлена называется схемой Горнера.

Объясним, как применять его к произвольному многочлену.

Пусть задан многочлен

$$a_0x^3+a_1x^2+a_2x+a_3.$$

Преобразуем его к виду

$$((a_0x+a_1)x+a_2)x+a_3$$

и будем вычислять последовательно такие величины:

$a_0 \cdot x$	
	прибавляем a_1
$a_0x + a_1$	
	умножаем на x
$(a_0x + a_1)x$	
	прибавляем a_2
$(a_0x + a_1)x + a_2$	
	умножаем на x
$((a_0x + a_1)x + a_2)x$	
	прибавляем a_3
$((a_0x + a_1)x + a_2)x + a_3$	

Последняя величина и будет искомым значением многочлена. Так можно вычислить значение произвольного многочлена

$$a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n,$$

преобразовав его к виду

$$(\ldots((a_0x+a_1)x+a_2)x+\ldots+a_{n-1})x+a_n.$$

Алгоритм вычисления значения многочлена n-й степени выглядит следующим образом:

```
алг схема Горнера (цел n, вещ x, вещ таб a[0:n], вещ y)
    арг n, a, x
    рез y
нач цел i
    i := 0; y := a[0]
    пока i ≠ n
    нц
        i := i + 1
        y := y · x + a[i]
    кц
кон
```

Пример 2. Алгоритм приближенного вычисления площадей. Пусть график функции $f(x)$ имеет вид, изображенный на рисунке 24.

Пусть требуется приближенно вычислить площадь фигуры, ограниченной графиком функции $f(x)$ и прямыми $x=a$, $x=b$, $y=0$. Эта фигура называется криволинейной трапецией и отмечена на рисунке штриховкой.

Идея алгоритма вычисления площади криволинейной трапеции состоит в следующем. Разобьем отрезок $[a, b]$ на n равных отрезков точками $a=x_0<x_1<x_2<\ldots<x_n=b$ и на каждом из полученных отрезков построим прямоугольник, одной стороной которого будет отрезок $[x_i, x_{i+1}]$, а другой — отрезок, длина которого равна $f(x_i)$. Случай при $n=4$ показан на рисунке 25.

Площадь криволинейной трапеции можно приближенно считать равной сумме площадей заштрихованных прямоугольников. Можно получить другую картину (см. рис. 26), если в качестве

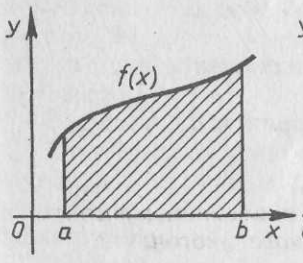


Рис. 24.

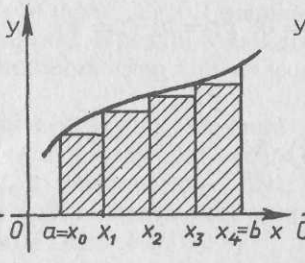


Рис. 25.

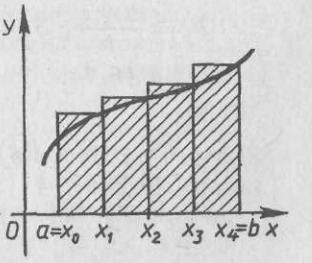


Рис. 26.

второй стороны прямоугольников выбирать отрезки, длины которых равны $f(x_{i+1})$, $i = 0, 1, 2, \ldots, n-1$.

Ясно, что если увеличить число отрезков $[x_i, x_{i+1}]$, т. е. отрезок $[a, b]$ разбить на большее число равных отрезков, то сумма их площадей все с большей точностью будет совпадать с площадью криволинейной трапеции. Значит, точность вычисления площади криволинейной трапеции определяется величиной n.

Площадь каждого прямоугольника можно вычислить так. Одна из сторон прямоугольника, построенного на отрезке $[x_i, x_{i+1}]$, равна $h = \frac{b-a}{n}$, а вторая — $f(x_i)$ либо $f(x_{i+1})$. Поэтому в первом случае площадь «левого» прямоугольника $\frac{b-a}{n} \cdot f(x_i)$, а во втором случае площадь «правого» прямоугольника $\frac{b-a}{n} \cdot f(x_{i+1})$.

Площадь криволинейной трапеции в первом случае приближенно равна сумме «левых» прямоугольников:

$$\frac{b-a}{n} \cdot f(x_0) + \frac{b-a}{n} f(x_1) + \ldots + \frac{b-a}{n} \cdot f(x_{n-1}) =$$

$$= \frac{b-a}{n}(f(x_0) + f(x_1) + \ldots + f(x_{n-1})),$$

а во втором случае — сумме «правых» прямоугольников:

$$\frac{b-a}{n}(f(x_1) + f(x_2) + \ldots + f(x_n)),$$

причем, чем больше n, тем больше точность вычисления площади криволинейной трапеции.

Обозначив величину площади S, запишем алгоритм приближенного вычисления площади криволинейной трапеции для случая «левых» прямоугольников.

```
алг площадь (вещ a, b, S, цел n)
    арг a, b, n
    рез S
нач вещ h, x, нат i

    h := (b - a)/n; S := 0; x := a; i := 1
    пока i ≤ n
    нц
        S := S + f(x) · h
        x := x + h
        i := i + 1
    кц
кон
```

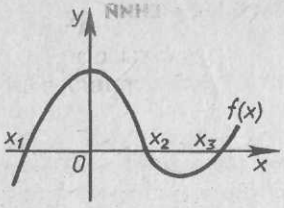


Рис. 27.

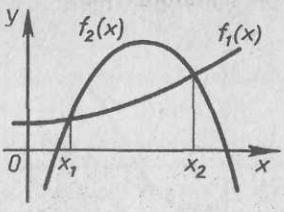


Рис. 28.

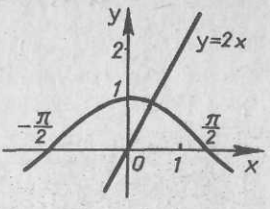


Рис. 29.

Для решения квадратного уравнения $ax^2 + bx + c = 0$ используются известные формулы вычисления корней: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Соответствующие алгоритмы были рассмотрены выше. Это алгоритмы, основанные на так называемых точных методах решения. Не всегда, однако, для решения уравнения можно применить точный метод. Например, для уравнения $2x - \cos x = 0$ уже не существует равносильных преобразований, приводящих к выражению для переменной x. На практике часто встречаются такие уравнения.

Поэтому широко используются приближенные методы решения уравнений, позволяющие тем не менее получать ответ с любой желаемой степенью точности. Особенно широкое применение эти методы получили в связи с применением вычислительных машин.

Рассмотрим этот вопрос сначала в общем виде. Пусть имеется уравнение с одной переменной $f(x) = 0$, где $f(x)$ — некоторая непрерывная функция. С геометрической точки зрения корень уравнения $f(x) = 0$ — это точка пересечения графика функции $f(x)$ с осью x. Корней может быть не один, а несколько, тогда на графике будет соответствующее число точек пересечения (рис. 27).

Очевидно, что графический метод может в отдельных случаях использоваться как метод грубого решения уравнения. Эта задача значительно облегчается, если удается преобразовать уравнение $f(x) = 0$ к равносильному виду $f_1(x) = f_2(x)$ таким образом, чтобы графики функции $f_1(x)$ и $f_2(x)$ могли быть легко построены. В этом случае корень уравнения — это абсцисса точек пересечения графиков функций $f_1(x)$ и $f_2(x)$ (рис. 28). Графическим путем можно достаточно точно определить отрезки, в каждом из которых содержится один корень уравнения. Это так называемый этап отделения корня.

Пример 3. Отделить корень уравнения $2x - \cos x = 0$.

Преобразуем уравнение к виду $2x = \cos x$ и построим графики функций $f_1(x) = 2x$ и $f_2(x) = \cos x$ (рис. 29). Из рисунка видно, что уравнение $2x - \cos x = 0$ имеет единственный корень, принадлежащий отрезку $[0; 1]$.

Предположим, что непрерывная функция $f(x)$ имеет на концах некоторого отрезка значения разных знаков. В этом случае она обя-

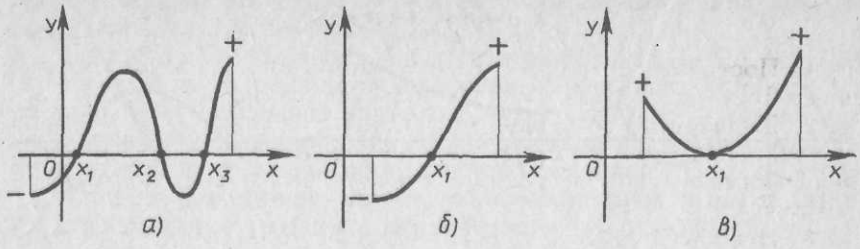


Рис. 30.

зательно имеет на этом отрезке хотя бы один корень (рис. 30, *а, б*). Как видно из рисунка 30, *а*, функция может иметь несколько корней. Если же она возрастает на этом отрезке (или убывает на нем), то корень единственный (рис. 30, *б*). Приведем один из методов отыскания корня — метод половинного деления. Этот метод применим в том случае, когда функция принимает значения разных знаков на концах некоторого отрезка. Он позволяет найти один из корней функции на этом отрезке. Если их несколько (как на рис. 30, *а*) или если значения функции на концах отрезка одного знака (рис. 30, *в*), то метод половинного деления не позволяет найти все корни функции на данном отрезке. Тем не менее он часто оказывается полезным.

Пусть функция $f(x)$ непрерывна на отрезке $[a, b]$ и принимает на его концах значения разных знаков. Составим алгоритм вычисления корня уравнения $f(x) = 0$ с заданной точностью.

Будем последовательно сужать отрезок $[a; b]$, пользуясь следующим методом. Разделим отрезок $[a; b]$ пополам точкой $c = \frac{(a+b)}{2}$ и вычислим значение $f(c)$. После этого отбросим ту часть отрезка $[a; b]$, на которой функция $f(x)$ знака не меняет (на рис. 31 такой частью является отрезок $[a; c]$). Процесс будем продолжать до тех пор, пока длина отрезка, содержащего корень, не станет меньше 2ε. Действительно, если в этом случае в качестве корня Δx взять середину отрезка $\frac{a+b}{2}$ (см. рис. 32), то допущенная при этом погрешность x не будет превышать ε (на рис. 32 точное значение корня обозначено x).

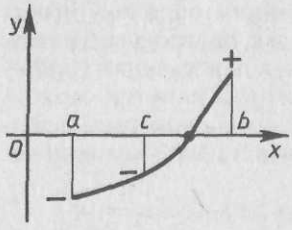


Рис. 31.

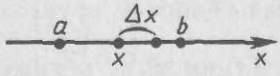


Рис. 32.

Запись алгоритма уточнения корня уравнения $f(x)$ на отрезке $[a; b]$ с заданной точностью ε на алгоритмическом языке приведена в библиотеке алгоритмов (приложение).

Пользуясь этим алгоритмом, можно искать корни уравнений с помощью калькулятора.

Упражнения

1. Постройте алгоритм «прямого» вычисления значения многочлена $2x^3 - 3x + 5$ при $x = 6$, не пользуясь схемой Горнера.

2. Вычислите значение многочлена $3x^4 + 2x^2 + x + 1$ при $x = 0{,}1$ «прямым» алгоритмом и по схеме Горнера. Сравните полученные результаты. Оцените точность вычисления каждым методом.

3. Используя схему Горнера, составьте таблицу значения многочлена:

а) $2x^4 - 1{,}4x^3 - 3{,}5x^2 + 4$ на отрезке $[1; 2]$ с шагом 0,2 (имеет ли данная функция точки экстремума на отрезке $[1; 2]$?);

б) $4x^3 - 0{,}7x^2 + x - 2{,}18$ на отрезке $[0; 4]$ с шагом 0,5;

в) $x^3 - 2{,}1x^5 + 0{,}2x^2 - 4$ на отрезке $[-2; 2]$ с шагом 0,4.

4. Решите квадратное уравнение:

а) $3x^2 - 1{,}8x - 2{,}4 = 0$; б) $6{,}2x^2 + 12{,}1x + 3{,}81 = 0$

и найдите приближенные значения корней с точностью до 0,001.

5. Стороны прямоугольника a и b удовлетворяют соотношению $\frac{a+b}{b} = \frac{b}{a}$, которое называется «золотым сечением». В этом случае отношение сторон прямоугольника $q = \frac{b}{a}$ удовлетворяет квадратному уравнению $q^2 - q - 1 = 0$ (докажите). Найдите приближенно с точностью до 0,01 положительный корень этого уравнения.

6. Через сколько секунд упадет на Землю камень, брошенный с высоты 2 м вверх с начальной скоростью 10 м/с? Напомним, что высота в этом случае меняется по закону:

$$h(t) = 2 + 10t - \frac{g}{2}t^2, \text{ где } g = 9{,}8 \text{ м/с}^2.$$

7. Вычислите приближенно площадь одной «арки» синусоиды, применяя:

а) метод «правых» прямоугольников;

б) метод «левых» прямоугольников.

8. Вычислите площадь криволинейной трапеции, ограниченной графиком функции $y = \sqrt{x + 1}$ и прямыми $y = 0$, $x = 0$, $x = 3$, методом «левых» и «правых» прямоугольников. Сравните результаты.

9. Уточните корни уравнений, пользуясь калькулятором:

а) $\sin x - 0{,}2x = 0$ (наименьший положительный корень);

б) $\cos x = x^2$;

в) $x - 10 \sin x = 0$ (наименьший положительный корень);

г) $\lg x = (x - 2)^2$.

10. Постройте алгоритм решения биквадратного уравнения, используя алгоритм решения квадратного уравнения в качестве вспомогательного.

11. Постройте алгоритм решения уравнения $\sqrt[4]{l + x} + \sqrt[4]{l - x} = m$, используя вспомогательные алгоритмы из биб-

лиотеки. При построении алгоритма используйте метод последовательного уточнения.

12. Постройте алгоритм решения системы уравнений, используя вспомогательный алгоритм из библиотеки:

$$\begin{cases} x^2 \cdot y + x \cdot y^2 = 30, \\ \frac{1}{x} + \frac{1}{y} = \frac{5}{6}. \end{cases}$$

§ 8. ПОСТРОЕНИЕ АЛГОРИТМОВ ДЛЯ РЕШЕНИЯ ЗАДАЧ ИЗ КУРСА ФИЗИКИ

В данном параграфе рассматривается несколько физических задач. В каждой из них речь идет о каком-либо физическом явлении, для которого имеется математическая модель. По исходным данным задачи, производя определяемые моделью вычисления, можно найти требуемый ответ. Вычисления, как это обычно и бывает в практических задачах, имеют довольно большой объем. Правила их выполнения формулируются в виде алгоритма на алгоритмическом языке. На основе этого алгоритма может быть разработана программа для компьютера. Вы, производя вычисления по алгоритму, можете использовать калькулятор.

Модели, которые мы сейчас будем рассматривать, можно назвать вычислительными. Они не дают явной формулы, по которой можно сразу найти требуемые значения искомых величин. Вместо этого есть алгоритм, позволяющий приближенно вычислять эти искомые значения. При этом вычислении промежутки времени, интервалы длины, участки поверхности разбиваются на мелкие доли, фрагменты. Вычисления производятся отдельно для каждого участка. Объем вычислений бывает очень большим. Вычислительные модели стали особенно важны после появления ЭВМ, которые могут реально осуществить огромный объем вычислений, требуемый для таких моделей. В примере 1 речь идет о важной вычислительной задаче, возникающей в физических исследованиях,— определении значений параметров по результатам измерений.

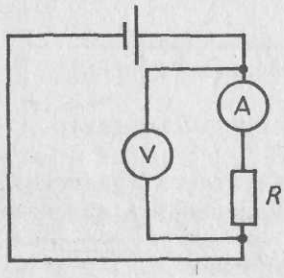


Рис. 33.

Пример 1. Пусть задана электрическая цепь (рис. 33). Измеряя напряжение U вольтметром, а силу тока I амперметром, можно найти сопротивление R. Результаты будут более точными и надежными, если повторим измерения несколько раз, а еще лучше — проведем измерения при нескольких различных значениях U (меняя источник тока в цепи).

Предположим, что при таком эксперименте для неизвестного резистора получились следующие результаты (табл. 15).

Таблица 15

U, B	1	2	3	4	5	6	7
I, A	0,010	0,018	0,031	0,042	0,050	0,061	0,072

Нужно найти сопротивление этого резистора. Нанесем эти точки на график (рис. 34).

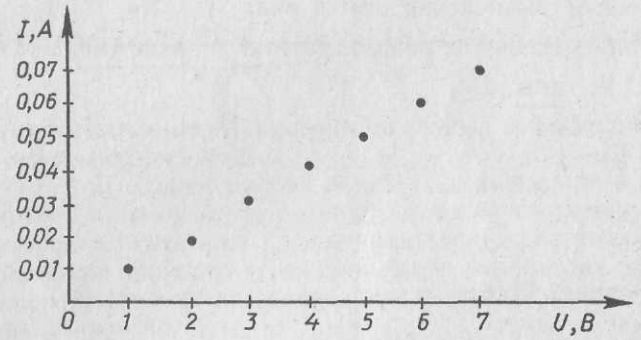


Рис. 34.

По закону Ома сила тока, проходящего через резистор, зависит от приложенного к нему напряжения:

$$I = \frac{1}{R} \cdot U.$$

Таким образом, графиком этой зависимости является прямая, проходящая через начало координат, у которой $k = \frac{1}{R}$. Но на одной прямой эти точки не лежат (рис. 34). Это связано с тем, что показания амперметра и вольтметра не совсем точные.

Постараемся провести прямую таким образом, чтобы точки графика были по возможности близки к этой прямой. По методу наименьших квадратов[1] угловой коэффициент искомой прямой вычисляется по формуле:

$$K = \frac{U_1 I_1 + U_2 I_2 + \ldots + U_7 I_7}{U_1^2 + U_2^2 + \ldots + U_7^2},$$

где U_1, I_1 — значения для первой экспериментальной точки, U_2, I_2 — для второй точки и т. д. Если все величины удалось измерить точно (все точки легли на прямую $I = \frac{1}{R} U$), то в этом

[1] Метод наименьших квадратов используется при построении математической модели, описывающей результаты эксперимента в физике, биологии, социологии и других областях. Его обоснование не входит в школьную программу и проводится в курсе высшей математики.

случае получаем $K = \frac{\frac{1}{R}(U_1^2 + \ldots + U_7^2)}{(U_1^2 + \ldots + U_7^2)} = \frac{1}{R}$, т. е. формула дает ожидаемый результат.

При таком количестве экспериментальных точек значение K можно вычислить вручную или с помощью калькулятора. Если же экспериментальных точек много или вычисления должны производиться многократно для разных наборов экспериментальных точек, разумно составить программу для ЭВМ. Алгоритм, реализующий данный метод вычисления, имеет вид:

```
алг метод наименьших квадратов (нат N, вещ таб U[1:N],
    I[1:N], вещ K)
    арг N, U, I
    рез K
нач
    цел i
    вещ числ, знам
    i := 0
    числ := 0
    знам := 0
    пока i ≠ N
    нц
        i := i + 1
        числ := числ + U[i] × I[i]
        знам := знам + U[i] × U[i]
    кц
    K := числ/знам
кон
```

Пояснение. Между выполнениями серии команд, входящих в цикл, в переменных *числ* и *знам* хранится сумма первых i членов в числителе и знаменателе.

Для нашего набора экспериментальных точек в результате вычисления по этой программе или после вычисления на калькуляторе получается следующий результат:

$$k = \frac{1}{R} = 0{,}01019 \text{ или } R = 98 \text{ Ом}.$$

(При проверке этих результатов на калькуляторе удобно выразить значение силы тока не в амперах, а в миллиамперах. При этом в формулу подсчета углового коэффициента искомой прямой нужно ввести множитель 0,001.)

Построим прямую (рис. 35), соответствующую полученному коэффициенту.

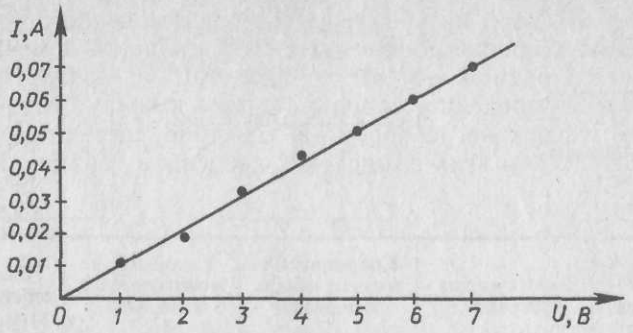


Рис. 35.

Видно, что отклонение экспериментальных точек от найденной прямой действительно мало.

Пример 2. Шарик массой $m = 0{,}1$ кг подвесили к пружине от школьного динамометра, оттянули вниз от положения равновесия на 1 см и отпустили. Он начинает двигаться вверх и через некоторое время проходит через положение равновесия. Найти это время, если жесткость пружины, определенная опытным путем, оказалась равной 40 Н/м.

Начнем с построения математической модели. Введем систему координат. Начало координат O поместим в положение равновесия, ось x направим вертикально вниз. Сила, с которой отклоненная на расстояние x от положения равновесия пружина действует на тело, равна $F = -kx$, где k — жесткость пружины, знак «минус» поставлен, так как сила, с которой пружина действует на тело, направлена противоположно отклонению тела. Ускорение a тела равно $\frac{F}{m}$, где m — масса тела. Пусть отрезок времени $\triangle t$ мал, тогда за время $\triangle t$ ускорение и скорость изменятся мало. Если ускорение равно a, то за время $\triangle t$ скорость изменяется на $\triangle v = a \cdot \triangle t$. Если скорость равна v, то за время $\triangle t$ координата тела изменяется на $\triangle x = v \cdot \triangle t$. Будем вычислять положение, ускорение и скорость тела, разбив ось времени на участки по $\triangle t = 0{,}01$ с.

Начнем с первого интервала: от $t = 0$ до $t = 0{,}01$. Координата в момент 0 равна 1 см $= 10^{-2}$ м по условию задачи. Ускорение находится по формуле $a = -\frac{k}{m}x$ и равно -4 м/с^2. В начальный момент скорость по условию равнялась 0. В середине интервала скорость изменилась на $a \cdot \frac{\triangle t}{2}$ (прошло $\frac{\triangle t}{2}$ с) и стала равной $0 - 4 \cdot \frac{10^{-2}}{2} = -2 \cdot 10^{-2}$. За первый интервал координата изменилась на $v \cdot \triangle t$ и к началу второго интервала стала равной $0{,}01 - 2 \cdot 10^{-2} \cdot 0{,}01 = 0{,}98 \cdot 10^{-2}$. В этот момент ускорение становится равным $a = \frac{-k}{m}x = -3{,}92$. Найдем скорость тела

в середине второго интервала. От середины первого интервала до середины второго прошло $\triangle t = 0{,}01$, скорость изменилась на $a \cdot \triangle t$ и стала равной $-2 \cdot 10^{-2} - 3{,}92 \cdot 10^{-2} = -5{,}92 \cdot 10^{-2}$. Зная ее, мы можем определить координату тела к концу второго интервала, его ускорение, скорость в середине третьего интервала и т. п. Результаты этих вычислений сведены в таблицу 16.

Таблица 16

Номер интервала	Время начала интервала, с	Координата в момент начала интервала, 10^{-2} м	Ускорение в момент начала интервала, м/с^2	Скорость в середине интервала, 10^{-2} м/с
1	0,00	1,00	−4,00	−2,0
2	0,01	0,98	−3,92	−5,92
3	0,02	0,92	−3,68	−9,60
4	0,03	0,82	−3,2	−12,88
5	0,04	0,69	−2,76	−15,64
6	0,05	0,53	−2,12	−17,76
7	0,06	0,35	−1,40	−19,16
8	0,07	0,16	−0,64	−19,80
9	0,08	−0,04	0,16	−19,64
10	0,09	−0,24	0,96	−18,68

Анализируя результаты вычислений, приведенных в таблице, мы видим: модуль ускорения тела по мере его приближения к положению равновесия уменьшается, а модуль скорости увеличивается вплоть до точки, которая соответствует 8-му интервалу. В этот момент координата меняет знак. Это значит, что тело проходит положение равновесия в промежутке между 0,07 и 0,08 секунды от начала движения и по инерции движется дальше, при этом его движение замедляется (модуль скорости уменьшается).

Запишем теперь все сказанное выше в виде алгоритма, вычисляющего координату x точки, в которой находится тело массой m по истечению N промежутков времени, каждый из которых равен $\triangle t$ (k — жесткость пружины, v_0 — начальная скорость).

```
алг колебание (вещ m, k, Δt, N, v0, x0, x)
    арг m, k, Δt, N, v0, x0
    рез x
нач цел i; вещ a, v
    i := 1; x := x0; a := − k · x/m; v := v0 + a · Δt/2
    пока i ≠ N + 1
    нц
        i := i + 1; x := x + v · Δt; a := − k · x/m; v := v + a · Δt
    кц
кон
```

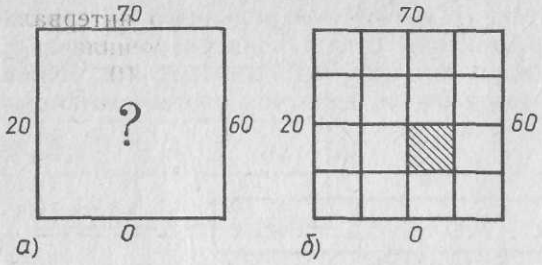


Рис. 36.

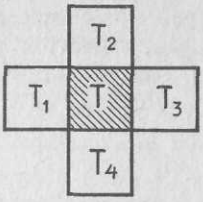


Рис. 37.

Пояснение. Между выполнением повторяющейся серии x равно координате тела в начале i-го интервала времени, a — ускорение в тот же момент, v — скорость в середине интервала.

Пример 3. На краях однородной квадратной пластины (рис. 36, *а*) поддерживается температура, указанная на рисунке. Какова будет температура во внутренних точках пластины?

Эта температура будет меняться от точки к точке: у нижнего края она будет близка к 0 °С, у верхнего — к 70 °С, у левого — к 20 °С, у правого — к 60 °С. Внутри пластины она будет принимать какие-то промежуточные значения. Чтобы найти их, нам придется сделать допущение, что теплообмен с пластиной происходит по ее краям, а в середине ее не подогревают и не охлаждают.

Такие задачи могут решаться с помощью ЭВМ. Поясним метод их решения на примере этой задачи.

Прежде всего разобьем нашу пластину на много маленьких квадратиков. Будем считать, что внутри каждого квадратика температура постоянна. Это предположение будет справедливо лишь приближенно, но при достаточно малом размере квадратика ошибка будет мала (рис. 36, *б*).

Затем мы составим систему уравнений, исходя из следующего принципа: температура каждого квадратика равна среднему арифметическому температуры четырех соседних квадратиков. Этот принцип можно обосновать следующим образом. Если квадратик температурой T граничит с квадратиком температурой T_1, то через границу идет поток тепла, пропорциональный разности температур с некоторым коэффициентом пропорциональности c. Таким образом, через границу квадратика, изображенного на рисунке 37, проходит суммарный поток тепла:

$$c(T_1 - T) + c(T_2 - T) + c(T_3 - T) + c(T_4 - T).$$

Поскольку мы считаем квадратик теплоизолированным, то эта сумма должна равняться 0. Сокращая на c и перенося T в другую часть, получаем:

$$T_1 + T_2 + T_3 + T_4 = 4T.$$

Мы рассмотрели квадратик, не примыкающий к краю. Если квадратик примыкает к краю, то у него не четыре соседа, а только

три (или два, если он в углу). В этом случае роль соседа со стороны края выполняет окружающая среда, температура которой нам известна. Разобьем нашу пластину на 9 квадратиков. Через $t[1,1]$, $t[1,2]$, ..., $t[3,3]$ обозначим температуру соответствующих квадратиков.

70

	t [1,1]	t [1,2]	t [1,3]	
20	t [2,1]	t [2,2]	t [2,3]	60
	t [3,1]	t [3,2]	t [3,3]	

0

Мы получаем такие уравнения:

$$t[1,1] = \frac{20 + 70 + t[1,2] + t[2,1]}{4};$$

$$t[1,2] = \frac{t[1,1] + 70 + t[1,3] + t[2,2]}{4};$$

$$t[1,3] = \frac{70 + 60 + t[1,2] + t[2,3]}{4};$$

$$t[2,1] = \frac{20 + t[1,1] + t[2,2] + t[3,1]}{4};$$

$$t[2,2] = \frac{t[2,1] + t[1,2] + t[2,3] + t[3,2]}{4};$$

$$t[2,3] = \frac{60 + t[2,2] + t[1,3] + t[3,3]}{4};$$

$$t[3,1] = \frac{20 + 0 + t[2,1] + t[3,2]}{4};$$

$$t[3,2] = \frac{0 + t[3,1] + t[2,2] + t[3,3]}{4};$$

$$t[3,3] = \frac{0 + 60 + t[3,2] + t[2,3]}{4}.$$

Эта система имеет 9 уравнений и 9 неизвестных. Если бы мы разбили каждую сторону квадрата не на три, а на N частей, то неизвестных и уравнений было бы N^2. На практике значения N — это сотни и даже тысячи. Для решения таких трудоемких задач применяются супер-ЭВМ, делающие миллиарды операций в секунду. Мы, однако, проиллюстрируем их решение на примере всего с девятью квадратиками. В этом примере вычисления можно провести с помощью калькулятора или даже вручную.

Один из методов решения нашей системы уравнений таков. В качестве первого приближения мы считаем, что все температуры t равны 35. Разумеется, при этом уравнения не будут выполнены: их левые части будут равны 35, а правые — нет. Запишем те и другие в таблицу на соответствующие места.

Левые

	70			
	35	35	35	
20	35	35	35	60
	35	35	35	
	0			

Правые

	70			
	40	43,75	50	
20	31,25	35	41,25	60
	22,5	26,25	32,5	
	0			

Правые части мы вычислили в соответствии с нашими уравнениями: например, 40 есть $\frac{70+20+35+35}{4}$; 43,75 есть $\frac{35+70+35+35}{4}$ и т. д. Числа в правой таблице и будут следующими приближениями к искомой температуре.

Они снова не будут решениями нашего уравнения. Если мы подставим их в уравнение, то, округляя до сотых, получим такие левые и правые части:

Левые

	70			
	40	43,75	50	
20	31,25	35	41,25	60
	22,5	26,25	32,5	
	0			

Правые

	70			
	41,25	48,75	53,75	
20	29,38	35,63	44,38	60
	19,38	22,5	31,88	
	0			

Все расчеты происходят по тому же алгоритму: число в клетке правой таблицы равно среднему арифметическому чисел в левой таблице, расположенных на соседних местах. При этом для клеток с краю используются известные нам температуры (20, 70, 60 и 0):

$41{,}25 = \frac{20+70+43{,}75+31{,}25}{4}$; $48{,}75 = \frac{40+70+50+35}{4}$ и т. д.

Числа в правой таблице и составят следующее приближение. Будем повторять это много раз: переписывая на каждом шаге числа из правой таблицы в левую и затем находя новые значения чисел в правой таблице описанным способом. Мы приведем еще несколько шагов. При этом не будем дважды повторять одну и ту же таблицу, будем выписывать только правые части.

	70			
	42,03	50,16	55,78	
20	29,07	36,25	45,32	60
	17,97	21,72	31,72	
	0			

	70			
20	42,31	51,02	56,37	60
	29,06	36,57	45,94	
	17,70	21,49	31,76	
		0		

	70			
20	42,52	51,31	56,74	60
	29,15	36,88	46,18	
	17,64	21,51	31,86	
		0		

Видно, что с каждым разом значения чисел в каждой следующей таблице становятся все более близкими к значениям соответствующих чисел в предыдущей таблице. Это значит, что разница между левой и правой частями уравнений становится все меньше и меньше. Таким образом, мы находим решение наших уравнений все точнее и точнее.

Мы можем остановиться на нашем последнем приближении, поскольку все равно надеяться на точное решение исходной физической задачи не приходится: уж слишком мало мы взяли квадратиков. При большем числе квадратиков решать систему «вручную» трудно, поэтому приходится поручать исполнение алгоритма компьютеру.

Запишем на алгоритмическом языке заголовок использованного нами алгоритма. Его аргументами будут температуры t_1, t_2, t_3,t_4, заданные на четырех сторонах квадрата, а также число N отрезков, на которые мы разбивали стороны квадрата. Результатом этого алгоритма будет таблица размером $N \times N$, описывающая распределение температуры.

алг теплопроводность (**вещ** t_1, t_2, t_3, t_4, **нат** N, **вещ таб** *температура* [1:N, 1:N])
арг t_1, t_2, t_3, t_4, N
рез *температура*

Мы не приводим текст этого алгоритма полностью, поскольку ход его выполнения был подробно описан. Задачи, аналогичные рассмотренным в данном параграфе, и методы их решения типичны для современных применений ЭВМ. Аналогичными методами решаются упоминавшиеся во введении задачи прогноза погоды и моделирования термоядерной реакции.

Упражнения

1. Выпишите формулу для нахождения углового коэффициента прямой по методу наименьших квадратов при условии, что ток измерен в миллиамперах, напряжение — в вольтах, а сопротивление — в омах (пример 1). Произведите подсчет по формуле.

2. Измерьте толщину нити. Для этого возьмите карандаш и намотайте на него 10, 20, 30, ..., 100 витков нити (укладывая ее виток

к витку). Измерьте линейкой длину участка карандаша, покрытого нитью (рис. 38). С помощью разобранного метода (пример 1) найдите толщину нити, используя результаты всех 10 измерений.

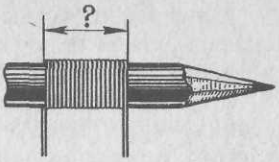


Рис. 38.

3. Найдите толщину листа бумаги, складывая его в несколько раз и измеряя толщину получившейся стопки.

4. В задаче о движении шарика на пружинке постройте график зависимости положения шарика x от времени на промежутке в 0,1 с.

5. В задаче о движении шарика на пружинке проведите расчет движения шарика, взяв в качестве $\triangle t$ не 0,01 с, а 0,02 с. Как изменится результат?

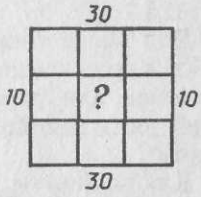


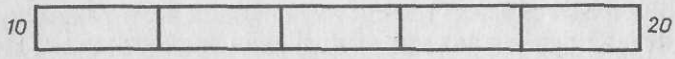
Рис. 39.

6. Найдите распределение температуры во внутренних точках квадратной однородной пластины, считая, что пластина разбита на 9 квадратиков, а распределение температуры на границе пластины дано на рисунке 39.

Выполните четыре шага вычислений.

7. Найдите распределение температуры во внутренних точках тонкого однородного стержня, считая, что стержень разбит на 5 отрезков и что на концах стержня температура равна 10 и 20° (рис. 40). Начните с температуры 15° на всех отрезках. Выполните

Рис. 40.



5 шагов вычислений. Так же как в случае пластины, исходите из предположения, что температура каждого отрезка равна среднему арифметическому температур соседних отрезков.

§ 9. АЛГОРИТМЫ РАБОТЫ С ГРАФИЧЕСКОЙ ИНФОРМАЦИЕЙ

На практике встречаются задачи, результатом решения которых являются чертежи, графики, диаграммы, рисунки и другая графическая информация.

Если задача решается с помощью компьютера, то графическая информация выводится либо на графический дисплей, на экране которого как в телевизоре эта информация высвечивается, либо на графопостроитель, который вычерчивает графическую информацию на листе бумаги. Выбрав координаты на экране дисплея или на листе бумаги, мы будем рассматривать их как часть плоскости и в дальнейшем говорить о рисовании на плоскости.

Для того чтобы вывести графическую информацию, необходимо указать, какие ее элементы (точки, отрезки, окружности и т. д.) надо вывести и в какой последовательности.

Поэтому, так же как и в случае других задач, решение графических задач требует составления алгоритмов. Эти алгоритмы состоят из специальных графических команд, каждая из которых указывает, какое элементарное графическое действие надо совершить (нарисовать точку, отрезок, окружность, многоугольник и т. д.).

Для рисования с помощью компьютера разработаны удобные наборы графических команд. Наш набор команд очень маленький и удобен для рисования простейших геометрических фигур. Он будет использовать прямоугольную систему координат на плоскости.

Для лучшего понимания правил выполнения графических команд удобно представлять себе исполнителя двигающимся по плоскости и рисующим на ней. Сначала исполнитель находится в точке плоскости с координатами (0,0) и смотрит вдоль оси y.

При выполнении графических команд может изменяться как точка, где находится исполнитель, так и направление, куда он смотрит.

1. Команды вычерчивания: вперед (a), назад (a)

Для того чтобы нарисовать отрезок, необходимо задать его начальную точку, направление и длину.

Исполнитель по команде вперед (a) вычерчивает отрезок длиной a с начальной точкой и с направлением таким же, как у исполнителя перед началом выполнения этой команды. После выполнения команды исполнитель попадает в концевую точку нарисованного отрезка, а его направление остается неизменным.

Команда назад (a) отличается от команды вперед (a) только тем, что отрезок вычерчивается в направлении, противоположном направлению исполнителя. Направление исполнителя при этом не изменяется, а сам он попадает в конечную точку нарисованного отрезка.

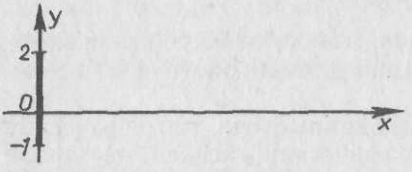


Рис. 41.

Если выполнить команды

назад (1)
вперед (3)

то получим отрезок, изображенный на рисунке 41.

2. Команды поворота: направо (*b*), налево (*b*)

Исполнитель по команде направо (*b*) поворачивается на *b* градусов вправо.

Исполнитель по команде налево (*b*) поворачивается на *b* градусов влево.

Пример 1 (рис. 42).

```
алг ломаная
нач
    направо (45)
    вперед (3)
    направо (90)
    вперед (3)
    налево (90)
    вперед (3)
кон
```

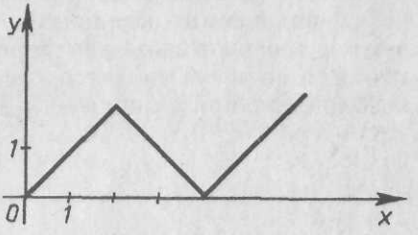


Рис. 42.

Пример 2 (рис. 43).

```
алг квадрат
нач
    вперед (4)
    направо (90)
    вперед (4)
    направо (90)
    вперед (4)
    направо (90)
    вперед (4)
кон
```

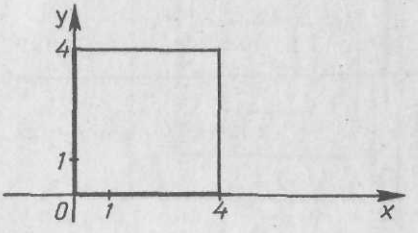


Рис. 43.

В примере 2 команды вперед (4) и направо (90) многократно повторяются. Запишем алгоритм вычерчивания квадрата со стороной длины *N*.

```
алг квадрат (вещ N)
    арг N
нач цел I
    I := 1
    пока I ⩽ 4
    нц
        вперед (N)
        направо (90)
        I := I + 1
    кц
кон
```

3. Команды для вычерчивания изображения, состоящего из отдельных элементов: рисуй, не рисуй

В некоторых случаях возникает необходимость перемещаться по плоскости без вычерчивания. Для этого вводится команда не рисуй. После этой команды исполнитель перестает рисовать отрезки, задаваемые командами вперед и назад, которые используются теперь только для передвижения. Для отмены действия команды не рисуй вводится команда рисуй. После команды рисуй исполнитель при выполнении команд вперед и назад начинает рисовать отрезки. Поясним сказанное на примере.

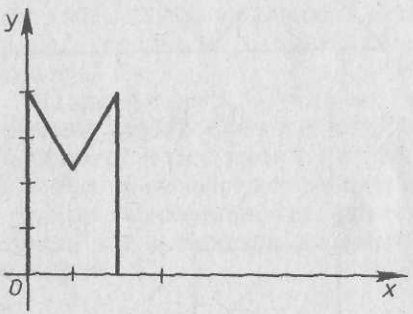


Рис. 44.

Пример 3. Написать слово МИР.

```
алг написание слова МИР
нач
    рисуй
    вперед (4); налево (30)
    назад (2); направо (60)
    вперед (2); налево (30)
    назад (4)
```

Нарисована буква М (рис. 44).

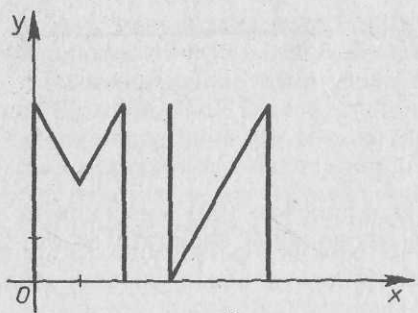


Рис. 45.

Теперь необходимо оставить промежуток между буквами М и И.

```
    не рисуй
    направо (90); вперед (1)
    налево (90); рисуй
```

Продолжаем, рисуем букву И (рис. 45).

```
    вперед (4); не рисуй
    назад (4); рисуй
    направо (30); вперед (4,8)
    налево (30); назад (4)
```

Делаем промежуток между буквами И и Р и рисуем букву Р (рис. 46).

```
    не рисуй
    направо (90); вперед (1)
    налево (90)
    рисуй; вперед (4)
    направо (90); вперед (2)
    направо (90); вперед (2)
    направо (90); вперед (2)
кон
```

Рис. 46.

Используя в качестве исполнителя графических команд компьютер, можно оформлять результаты решения задач по математике, физике, химии и т. д. в виде графиков, таблиц, диаграмм, гистограмм, моделировать на экране дисплея явления природы, играть и создавать свои компьютерные игры.

Упражнения

1. Составьте алгоритмы рисования следующих изображений:

а) слова МИР, используя только команды вперед, вправо, рисуй, не рисуй;

б) своих инициалов;

в) проекции куба.

2. Исполните алгоритм при $N = 4$, $N = 20$.

```
алг многоугольник (нат N)
    арг N
нач цел I
    I:= 1
    пока I ⩽ N
    нц
        вперед (1/N)
        направо (360/N)
        I:= I + 1
    кц
кон
```

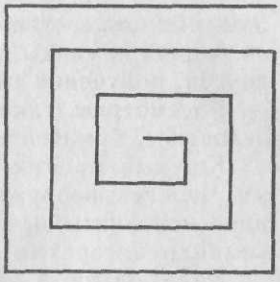


Рис. 47.

Что будет происходить с изображением при увеличении N? Чему равен периметр нарисованного многоугольника?

3. Составьте и запишите алгоритмы рисования фигур, изображенных на рисунках 47—51.

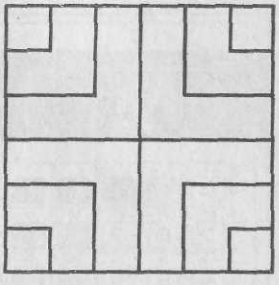


Рис. 48.

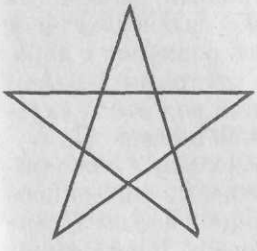


Рис. 49.

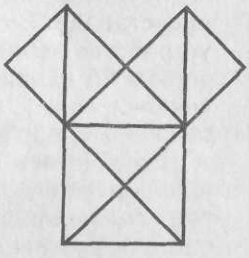


Рис. 50.

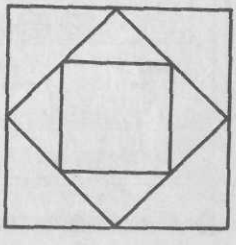


Рис. 51.

ПРИЛОЖЕНИЕ

I. РАБОТА С КАЛЬКУЛЯТОРОМ

Всякий алгоритм строится в расчете на какого-либо конкретного исполнителя. Однако исполнение каждого алгоритма включает в себя ряд действий, обязательных для любого исполнителя. Эти действия состоят в задании значений аргументов, выполнении очередной команды, осуществлении перехода от одной команды к другой, получении результата исполнения алгоритма.

Рассмотрим таких исполнителей алгоритма, как, например, человек с бумагой и карандашом и человек с калькулятором.

Каждый из этих исполнителей обладает своими особенностями. Человек, вооруженный бумагой и карандашом, может в принципе исполнить вычислительный алгоритм, однако при этом все команды алгоритма выполняются вручную.

Калькулятор позволяет в ряде случаев выполнить отдельные вычислительные команды алгоритма. В этом смысле калькулятор является вычислительным средством исполнителя-человека.

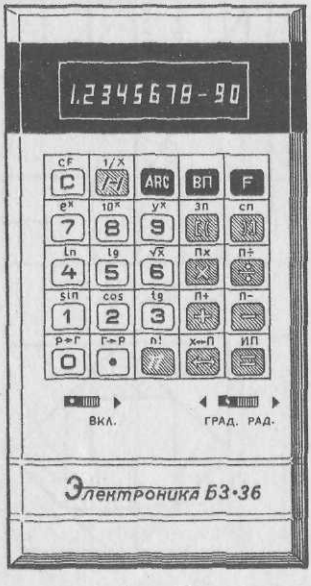


Рис. 52.

1. Выполнение вычислительных команд алгоритма

Одним из распространенных типов инженерного калькулятора, или, как его иначе называют, калькулятора для научно-технических расчетов, является калькулятор «Электроника БЗ-36». Он представляет собой вычислительное устройство небольших размеров с набором из 25 клавиш и световым экраном (индикатором). Общий вид этого калькулятора изображен на рисунке 52.

Прежде чем переходить к исполнению алгоритма, нужно научиться выполнять при помощи калькулятора часто встречающиеся вычисления. С этой целью ниже приводятся мате-

матические задачи, которые помогают освоить технику вычислений.

А. Задачи на непосредственное вычисление

1. Какое из чисел больше:

а) $\sqrt{10}+1$ или $\sqrt{17}$;

б) $\operatorname{tg} 70^\circ$ или $10\sqrt{19}$;

в) $60!$ или 30^{60};

г) π^{26} или 200π;

д) $100 \sin 40^\circ$ или 8^3;

е) $10^{\sin 29^\circ}$ или $\sqrt{10}$;

ж) $\sin(\cos 1)$ или $\cos(\sin 1)$;

з) 41^{53} или 53^{41};

и) $2^{\sin 3}$ или $3^{\sin 2}$;

к) $\sqrt{3}+2\sqrt{2}$ или $\sqrt{\sqrt{5}+\sqrt{6}}$;

л) 2^{100} или 100^{20};

м) $\operatorname{tg}\left(\frac{\pi}{2}-\frac{1}{10\,000}\right)$ или 2^{16}?

2. К какому значению стремятся следующие выражения при увеличении n:

а) $\underbrace{\sqrt{\cdots\sqrt{\sqrt{5}}}}_{n \text{ раз}}$;

б) $\frac{100\,n}{n^2+1}$;

в) $\underbrace{\sqrt{2+\sqrt{2+\ldots+\sqrt{2}}}}_{n \text{ раз}}$;

г) $\left(1+\frac{1}{2}+\frac{1}{4}+\ldots+\frac{1}{2^n}\right)$;

д) $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots+\frac{1}{n}(-1)^{n+1}$;

е) $\left(1+\frac{1}{1!}+\frac{1}{2!}+\ldots+\frac{1}{n!}\right)$;

ж) $\left(1+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{n^2}\right)$;

з) $\left(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}-\ln n\right)$;

и) $\left(1+\frac{1}{2^8}+\frac{1}{3^8}+\ldots+\frac{1}{n^8}\right)$?

Б. Задачи с буквенными обозначениями

С помощью калькулятора нельзя доказывать теоремы. Например, нельзя доказать, что $\sin^2 x+\cos^2 x=1$. Для этого пришлось бы проверить тождество для всех значений x, которых существует бесконечно много. Однако можно доказать неправильность какого-нибудь тождества. Для этого достаточно найти всего лишь один пример, который его опровергает. Например, тождество

$(x+1)^2 = x^2 + x + 1$ не выполняется, поскольку, подставив значение $x = 1$, получаем $4 = 3$.

3. Какие из следующих тождеств можно опровергнуть:

а) $\sin^2 x + \cos^2 x + 2 \sin x \cos x = \frac{1}{2}$;

б) $\operatorname{tg}\left(x + \frac{\pi}{2}\right) = 2 \operatorname{ctg} x$;

в) $\operatorname{tg}\left(x + \frac{5+\pi}{2}\right) = 4 \operatorname{tg} x$?

В. Задачи на построение графиков

4. Составьте таблицу значений функции $f(x)$ на отрезке $[a, b]$ с шагом h и в соответствии с таблицей постройте график функции $f(x)$ на отрезке $[a, b]$:

а) $f(x) = 2x^3 - 7x^2 + 12$, $[a, b] = [-5, 5]$, $h = 0{,}5$;
б) $f(x) = 4\sqrt{x} - \sin 3x$, $[a, b] = [0, 2]$, $h = 0{,}2$;
в) $f(x) = x^2 - 4\sqrt{x} + 2$, $[a, b] = [0, 4]$, $h = 0{,}2$;
г) $f(x) = x^3 - 3x^2 - 2x$, $[a, b] = [0, 4]$, $h = 0{,}5$.

Г. Аналитические задачи

Эти задачи, кроме непосредственного вычисления на калькуляторе, требуют анализа результатов. Каждая из задач представляет собой небольшое исследование. Выводы, которые можно из него получить, могут оказаться полезными в дальнейшем.

5. Над числами $a_1 = \frac{8}{7}$, $a_2 = 1{,}2$, $a_3 = \sqrt{5} - 1$, $a_4 = \frac{\sqrt{10}}{3}$, $a_5 = 1$ произведите следующие действия:

а) расположите в порядке возрастания;

б) вычислите среднее арифметическое этих чисел ($a_{ср}$) с точностью до 0,01;

в) для каждого из данных чисел найдите его отклонение от среднего арифметического $\triangle_i = |a_i - a_{ср}|$;

г) вычислите $\sum_{i=1}^{5} \triangle_i$;

д) вычислите среднее геометрическое $\sqrt[5]{\Pi a_i}$ и сравните его с $\sum_{i=1}^{5} \triangle_i$ и с $a_{ср}$.

6. Последовательность задана формулой ее n-го члена: $a_n = \frac{1}{n^2 + 4}$.

а) Вычислите первые восемь членов последовательности.

б) Найдите a_{40} и a_{200}.

в) Чему равно a_{967} с точностью до 10^{-3}?

г) Укажите какой-нибудь член последовательности, отличающийся от 0 менее чем на 10^{-5}.

7. Последовательность задана формулой ее n-го члена:

$$x_n = \frac{3n+4}{n+1}.$$

а) Найдите первые восемь членов последовательности.

б) Вычислите x_{100} и x_{1000}.

в) Для первых восьми членов последовательности вычислите $|x_n - 3|$.

8. Для последовательности (u_n) известно:

$$u_1 = 1, \quad u_{n+1} = \frac{u_n + \frac{3}{u_n}}{2} \ (n \geqslant 2).$$

а) Вычислите первые восемь членов этой последовательности.

б) Найдите $u_{n+1} - u_n$ для $n = 1, 2, \ldots, 8$.

в) Вычислите $|u_n - 1{,}7|$ для $n = 1, 2, \ldots, 8$.

г) Верно ли, что предел последовательности равен 1,7 при увеличивающихся значениях n?

9. Последовательность задана формулой ее n-го члена:

$$b_n = q^n, \text{ где } q = 2;\ 1;\ 2;\ 0{,}4;\ -0{,}7;\ \frac{\sqrt{2}}{3}.$$

а) Вычислите первые шесть, двадцатый и сотый члены последовательности для каждого значения q.

б) Найдите значения q, для которых $q^n \to 0$.

в) Для каждого из указанных значений q подберите номер, начиная с которого $q^n \approx 0$, с точностью до 10^{-3}.

10. Известны формулы n-х членов последовательностей (a_n) и (b_n): $a_n = \frac{1}{n^3}$, $b_n = \frac{1}{2^n}$.

а) Выбрав произвольно 10 значений n, вычислите a_n, b_n, $a_n + b_n$, $a_n - b_n$, $b_n - a_n$, $\frac{a_n}{b_n}$, $\frac{b_n}{a_n}$.

б) Верно ли, что $a_n \to 0$, $b_n \to 0$, $(a_n - b_n) \to 0$, $\frac{a_n}{b_n} \to 0$?

11. Для последовательности (F_n) известно, что $F_1 = F_2 = 1$, $F_n = F_{n-1} + F_{n-2}$ $(n \geqslant 3)$ (последовательность Фибоначчи).

а) Выпишите 10 первых членов последовательности.

б) Проверьте для F_2, F_3, F_{10} формулу

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}.$$

в) Составьте новую последовательность $q_n = \frac{F_{n+1}}{F_n}$.

г) Стремится ли q_n к какому-нибудь числу при увеличении n?

2. Вычислительные задачи из курсов физики и химии

Инженерный калькулятор может быть использован при решении различных задач, которые встречаются в курсах физики и химии.

Рассмотрим, например, следующие задачи:

З а д а ч а 1. Чему равна масса воздуха, впускаемого в цилиндр дизельного двигателя объемом 1,58 л при температуре 67 °С и давлении $8 \cdot 10^4$ Па ($R \approx 8{,}31$ Дж/(моль · К))?

Р е ш е н и е. Введем обозначение: $V = 1{,}58$ л, $t = 67°C$, $T = 340$ К, $P = 8 \cdot 10^4$ Па, $M = 0{,}029$ кг/моль.

Запишем уравнение: $PV = \frac{m}{M} RT$, откуда $m = \frac{PVM}{RT}$:

$$m = \frac{8 \cdot 10^4 \cdot 0{,}029 \cdot 1{,}58 \cdot 10^{-3}}{8{,}31 \cdot 340}.$$

Для вычисления значения этого выражения запишем все числа в следующем виде:

$$m = \frac{0{,}8 \cdot 10^5 \cdot 0{,}29 \cdot 10^{-1} \cdot 0{,}158 \cdot 10^{-2}}{0{,}831 \cdot 10 \cdot 0{,}34 \cdot 10^3} =$$

$$= \frac{0{,}8 \cdot 0{,}29 \cdot 0{,}158}{0{,}831 \cdot 0{,}34} \cdot \frac{10^5 \cdot 10^{-1} \cdot 10^{-2}}{10 \cdot 10^3} = \frac{0{,}8 \cdot 0{,}29 \cdot 0{,}158}{0{,}831 \cdot 0{,}34} \cdot 10^{-2}.$$

Вычислим на калькуляторе выражение $\frac{0{,}8 \cdot 0{,}29 \cdot 0{,}158}{0{,}831 \cdot 0{,}34}$ по программе 0,8 [×] 0,29 [×] 0,158 [÷] 0,831 [÷] 0,34.

Получим 0,1297373.

Поскольку данное выражение состоит только из произведений и частного, то согласно правилу округления его надо округлить до такого числа значащих цифр, какое имеет наименее точное исходное данное.

В данном случае 0,8 имеет одну значащую цифру. Поэтому ответ округляется до одной значащей цифры: $m \approx 1 \cdot 10^{-3}$ кг.

З а д а ч а 2. Рассчитать массу железа, образующегося при восстановлении оксида железа (III) массой 475 кг, содержащего 7% примесей, оксидом углерода (II).

Р е ш е н и е. Составим уравнение реакции:

$$\underset{\substack{m' = 160 \text{ кг} \\ (1 \text{ моль})}}{Fe_2O_3} + 3CO = \underset{\substack{m'' = 56 \text{ кг} \\ (2 \text{ моль})}}{2Fe} + 3CO_2$$

Найдем массу оксида железа (III) без примесей по программе 475 [П] [ЗП] [÷] [100] [×] 7 [П].

Получим 442 кг.

Определим массу полученного железа. Из 160 кг Fe_2O_3 получается $2 \cdot 56$ кг Fe, из 442 кг: $x = \frac{442 \cdot 112}{160}$.

Вычислим по программе: 442 [×] 112 [÷] 160 [=].

После округления получим ответ 309 кг.

Используя калькулятор, решите следующие задачи:

1. Определите среднюю квадратическую скорость движения молекул водорода при температуре 330К. Масса молекул водорода составляет $3{,}35 \cdot 10^{-27}$ кг.

2. Паровой молот массой 9,5 т падает с высоты 2,3 м на стальную болванку массой 230 кг. Сколько раз он должен упасть, чтобы температура болванки поднялась на 45°С? На нагревание болванки идет 50% количества теплоты, полученной при ударе.

3. Найдите отношение массы протона, равной $1{,}6726 \cdot 10^{-27}$ кг, к массе электрона, равной $9{,}1095 \cdot 10^{-31}$ кг.

4. Три резистора, сопротивления которых соответственно равны $R_1 = 12$ Ом, $R_2 = 17$ Ом, $R_3 = 2{,}9$ Ом, соединены параллельно. Определите общее сопротивление цепи.

5. Смешали раствор гидроксида калия массой 185 г с массовой долей (%) растворенного вещества 27% и раствор соляной кислоты массой 65,0 г с массовой долей (%) растворенного вещества 36,5%. Найдите массу образовавшегося хлорида калия.

6. Из азота массой 59 кг был синтезирован аммиак массой 47 кг. Каков выход аммиака в процентах от теоретически возможного?

7. Вычислите массовую долю (%) азота в кальциевой селитре.

8. К раствору, содержащему 17,6 хлорида меди (II), прилили раствор, содержащий 23,2 г гидроксида натрия. Вычислите массу образовавшегося гидроксида меди (II) (с точностью до 0,1 г).

9. Сколько литров воздуха необходимо для полного сгорания 27,0 л этана?

10. Запишите молекулярную формулу вещества по данным состава: С — 52%, Н — 13%, О — 35%, если относительная плотность паров этого газа по воздуху 1,59.

3. Использование калькуляторов для выполнения алгоритма

Пример 1. Алгоритм Евклида.

Запишем алгоритм в виде, удобном для исполнителя-человека, имеющего инженерный калькулятор. В этой записи наряду с командами алгоритмического языка будут использованы команды калькулятора, которые будут обозначаться в виде квадрата с указанием символа команды внутри него. Команды будут разделяться точкой с запятой, как это принято в алгоритмическом языке:

нач a; [—]; b; **пока** $a \neq b$ **нц если** $a < b$ **то** [↔]; [=] **кц**; **кон**

Проследим на конкретном примере, как исполнитель с калькулятором будет выполнять такой алгоритм. Пусть надо найти НОД(30, 18). Запишем в таблицу (табл. 17) аргументы a и b. В эту же таблицу в процессе выполнения алгоритма будем записывать измененные значения a и b. В соответствии с алгоритмом исполнитель вводит в калькулятор значение $a = 30$, нажимает клавишу [—] и вводит значение $b = 18$. После этого в РИ будет находиться число 18, а в РР — число 30. Так как $a > b$, то команда [↔] пропускается и исполнитель нажимает клавишу [=]. На индикаторе и в РИ появляется новое значение переменной $a = 12$. Исполнитель заносит это значение в таблицу (шаг № 2).

Так как $a \neq b$, то необходимо вернуться к началу цикла, что в программе отмечено стрелкой. Условие $a < b$ выполнено, поэтому исполнитель выполняет команду [↔], в результате чего происходит обмен чисел 12 и 18, записанных в регистрах РИ и РР. Этот обмен отражается в таблице (шаг № 3).

Теперь после выполнения команды [=] на индикаторе и в регистре РИ появляется новое значение $a = 6$, которое также записывается в таблицу (шаг № 4).

Заметим, что калькулятор работает в режиме констант, «помня» операцию, которую он должен выполнять (вычитание), поэтому после выполнения команды [=] число в РР (значение b) не изменяется.

Процесс будет повторяться до тех пор, пока значения a и b не станут равными (шаг № 6).

Таблица 17

Номер шага	1	2	3	4	5	6
a	30	12	18	6	12	6
b	18	18	12	12	6	6

Теперь условие окончания цикла ($a = b$) выполнено, поэтому повторение действий, указанных в цикле, прекращается и исполнитель записывает в качестве результата значение 6.

Итак, НОД (30, 18) = 6. Задача решена.

Конечно, для чисел 30 и 18 наибольший общий делитель можно легко найти и без калькулятора. Такие аргументы были выбраны лишь для того, чтобы показать, как используется калькулятор в процессе выполнения алгоритма. Для других аргументов, например 143 524 и 237 176, выполнить алгоритм Евклида без калькулятора уже не так просто.

Пример 2. Вычислить значения многочлена (по схеме Горнера).

Алгоритм для исполнителя, использующего калькулятор, будет иметь следующий вид:

нач a; [×]; x; $i := 1$; **пока** $i \leqslant n$ **нц** [=]; [F] [ЗП]; a_i; [F] [П+]; [F] [ИП]; $i := i + 1$ **кц кон**

При выполнении этого алгоритма калькулятор также работает в режиме констант. Значение x находится в РР и остается неизменным в процессе выполнения алгоритма. Значение y накапливается в регистре памяти РП. Результат получается в РИ и высвечивается на индикаторе.

Рассмотрим действия исполнителя при вычислении значения многочлена $2x^3 + 7x^2 + 4x + 1$ при $x = 2$.

В этом случае $n = 3$ и цикл в программе должен быть выполнен три раза при $i = 1, 2, 3$.

Запишем значения коэффициентов многочлена в таблицу 18. Во второй строке этой таблицы будем помещать значения y после каждого выполнения цикла.

Исполнение алгоритма начинается с того, что исполнитель набирает на клавиатуре калькулятора значение $a_0 = 2$. Это значение является начальным значением y, поэтому оно заносится в таблицу в графу под значением a_0. Затем исполнитель нажимает клавишу [×] и набирает значение $x = 2$.

При первом выполнении цикла ($i = 1$) после нажатия клавиши [=] в РИ получается $a_0 \cdot x = 4$, а значение $x = 2$ переписывается в РР, где и остается неизменным до окончания работы программы. После выполнения команды [F] [ЗП] значение $a_0 \cdot x = 4$ запоминается в РП. Далее исполнитель набирает на клавиатуре значение $a_1 = 7$ и выполняет команду [F] [П+]: значение $a_1 = 7$ прибавляется к значению, хранящемуся в РП. После выполнения команды [F] [ИП] полученное после первого выполнения цикла значение $a_0 \cdot x + a_1 = 11$ переписывается в РИ. Это значение y записывается в таблицу под значением a_1 (шаг № 1).

При втором выполнении цикла после выполнения команды [=] получается значение $(a_0 x + a_1) \cdot x = 22$, которое снова записывается в РП, «стирая» находившееся там ранее значение. После вторичного выполнения всех команд цикла получается новое значение $y = (a_0 x + a_1) \cdot x + a_2 = 26$ (шаг № 2). Наконец, после третьего выполнения команд цикла в РИ получается результат $y = ((a_0 x + a_1) \cdot x + a_2) \cdot x + a_3 = 53$, который исполнитель записывает в последнюю графу таблицы (шаг № 3).

Таблица 18

Номер шага		0	1	2	3
$x = 2$	a_i	2	7	4	1
	y	2	11	26	53

В рассмотренном примере вычисляли значение многочлена с целочисленными коэффициентами при $x = 2$ только для того, чтобы проще объяснить существо дела. В действительности калькулятор оказывается полезным, когда приходится вычислять значение многочлена с вещественными коэффициентами, например такого:

$62{,}135x^4 + 12{,}721x^3 + 5{,}965x^2 + 9{,}317x + 123{,}156$ при $x = 7{,}121$.

II. БИБЛИОТЕКА АЛГОРИТМОВ

А 1. Алгоритм вычисления модуля (МОД) действительного числа

```
алг МОД (вещ x, вещ y)
    арг x
    рез y
нач
    если x ⩾ 0
        то y := x
        иначе y := − x
    все
кон
```

А 2. Алгоритм поиска большего из двух чисел a и b (БИД)

```
алг БИД (вещ a, b, вещ y)
    арг a, b
    рез y
нач
    если a ⩾ b
        то y := a
        иначе y := b
    все
кон
```

А 3. Алгоритм решения линейного уравнения $ax = b$ (ЛУР), где a и b — произвольные вещественные числа

```
алг ЛУР (вещ a, b, вещ x, лит y)
    арг a, b
    рез x, y
нач
    если a ≠ 0
        то y := „есть решение“
           x := b/a
        иначе
            если b = 0
                то y := „x — любое число“
                иначе y := „решений нет“
            все
    все
кон
```

А 4. Алгоритм решения квадратного уравнения $ax^2 + bx + c = 0$ (КВУР), где a, b, c — произвольные вещественные числа ($a \neq 0$)

```
алг КВУР (вещ a, b, c, вещ x1, x2, лит y)
    арг a, b, c
    рез x1, x2, y
нач вещ D
    D := b^2 − 4 · a · c
    если D < 0
        то y := „нет решения“
        иначе y := „есть решения“
                x1 := (−b + √D)/(2a)
                x2 := (−b − √D)/(2a)
    все
кон
```

А 5. Алгоритм решения неравенства $ax>b$ (НЕР), где a и b — произвольные вещественные числа

```
алг НЕР (вещ a, b, c, лит y)
    арг a, b
    рез c, y
нач
    если a≠0
        то c:=b/a
           если a>0
               то y:="x>c"
               иначе y:="x<c"
           все
        иначе
              если b<0
                  то y:="x — любое число"
                  иначе y:= "решений нет"
              все
    все
кон
```

А 6. Алгоритм нахождения наибольшего общего делителя (НОД)

```
алг НОД (нат M, N, нат НОД)
    арг M, N
    рез НОД
нач нат x, y
    x:=M; y:=N
    пока x≠y
        нц
           если x>y
               то x:=x−y
               иначе y:=y−x
           все
        кц
       НОД:=x
кон
```

А 7. Алгоритм нахождения наименьшего элемента в линейной таблице чисел

```
алг МИНЭЛЕМЕНТ (цел K, N, вещ таб A [K:N], цел l)
    арг A, K, N
    рез l
нач цел i, вещ МИН
    МИН: = A [K]
    l: = K
    i: = K + 1
    пока i ⩽ N
    нц
        если МИН > A [i]
            то
            МИН: = A [i]
            l: = i
        все
        i: = i + 1
    кц
кон
```

А 8. Алгоритм упорядочения по возрастанию элементов линейной таблицы чисел

```
алг УПОРЯДОЧЕНИЕ (цел n, M, вещ таб C [n:M])
    арг C, n, M
    рез C
нач цел i, l, вещ R
    i: = n
    пока i < M
    нц
        МИНЭЛЕМЕНТ (i, M, C, l)
        R: = C [i]
        C [i]: = C [l]
        C [l]: = R
        i: = i + 1
    кц
кон
```

А 9. Алгоритм вычисления значения многочлена $P_n(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$ по схеме Горнера

```
алг СХЕМА ГОРНЕРА (цел n, вещ x, вещ таб a[0:n],
    вещ y)
    арг n, a, x
    рез y
нач цел i
    i := 0
    y := a[0]
    пока i ≠ n
    нц
        i := i + 1
        y := y · x + a[i]
    кц
кон
```

А 10. Алгоритм нахождения площади криволинейной трапеции, ограниченной прямыми $x = a$, $x = b$, $y = 0$ и кривой $y = f(x)$ при условии, что для всех x из отрезка $[a; b]$ значения $f(x) \geqslant 0$

```
алг ПЛОЩАДЬ (вещ a, b, S, нат n)
    арг a, b, n
    рез S
нач вещ h, x, нат i
        h := (b − a)/n
        S := 0
        x := a
        i := 1
    пока i ⩽ n
    нц
        S := S + h · f(x)
        x := x + h
        i := i + 1
    кц
кон
```

А 11. Алгоритм уточнения корня уравнения $f(x) = 0$ на отрезке $[a; b]$ с заданной точностью ε при условии, что функция $f(x)$ на отрезке $[a; b]$ монотонна и меняет знак

```
алг УТОЧНЕНИЕ КОРНЯ (вещ a, b, ε, вещ x)
    арг a, b, ε
    рез x
нач вещ c
    пока b − a > 2 · ε
    нц
        c := (a + b)/2
        если f(a) · f(c) ⩽ 0
            то b := c
            иначе a := c
        все
    кц
    x := (a + b)/2
кон
```

СОДЕРЖАНИЕ

Андрей Петрович Ершов
Вадим Макариевич Монахов
Сергей Александрович Бешенков
Ян Эдуардович Гольц
Александр Андреевич Кузнецов
Эдуард Иванович Кузнецов
Михаил Павлович Лапчик
Дмитрий Олегович Смекалин

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Пробное учебное пособие для средних учебных заведений. В двух частях

Часть первая

Зав. редакцией *Р. А. Хабиб*
Редакторы *Т. А. Бурмистрова, Н. И. Никитина*
Мл. редакторы *Л. Е. Козырева, Е. А. Сафронова*
Художник *Б. Л. Николаев*
Художественный редактор *Е. Н. Карасик*
Технический редактор *В. Ф. Коскина*
Корректор *Н. И. Новикова*

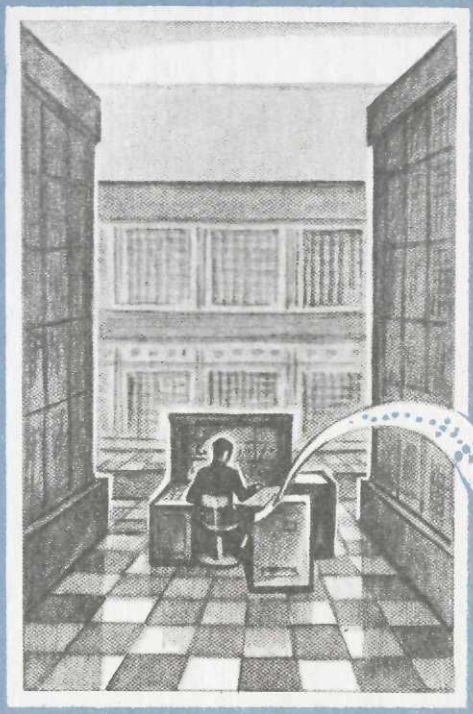
ИБ № 9984

Сдано в набор 17.04.85. Подписано к печати 25.04.85. Формат $60\times90^1/_{16}$. Бумага кн.-журн. офс. Гарнит. литерат. Печать офсетная. Усл. печ. л. 6. Усл. кр.-отт. 6,25. Уч.-изд. л. 4,91. Тираж 3 300 000 (1—2 000 000) экз. Заказ № 279. Цена 15 коп.

Ордена Трудового Красного Знамени издательство «Просвещение» Государственного комитета РСФСР по делам издательств, полиграфии и книжной торговли. 129846, Москва, 3-й проезд Марьиной рощи, 41.

Отпечатано с диапозитивов ордена Трудового Красного Знамени фабрики «Детская книга» № 1 Росглавполиграфпрома Государственного комитета РСФСР по делам издательств, полиграфии и книжной торговли на Калининском ордена Трудового Красного Знамени полиграфкомбинате детской литературы им. 50-летия СССР Росглавполиграфпрома Госкомиздата РСФСР. Калинин, проспект 50-летия Октября, 46.

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

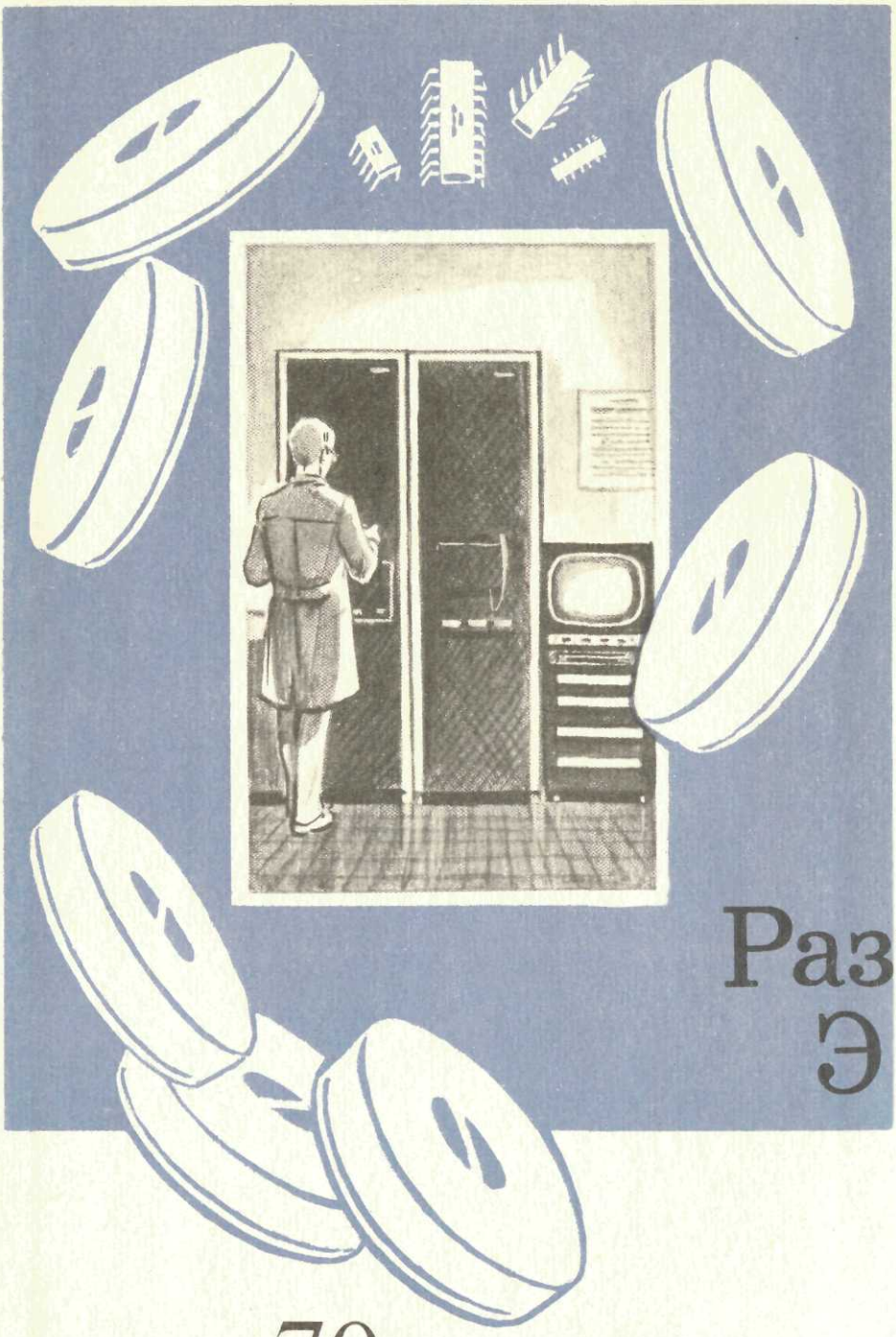


Разв
ЭВ

50-е годы

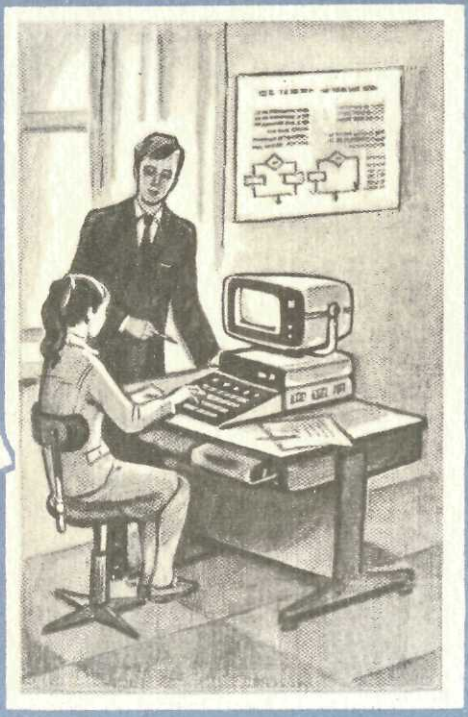
итие
ЭМ

60-Е ГОДЫ



Разв
ЭВ

70-е годы



ИТИЕ
ВМ

80-Е ГОДЫ

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ПРОБНОЕ УЧЕБНОЕ ПОСОБИЕ
для средних учебных заведений
В двух частях

•

ЧАСТЬ ВТОРАЯ

Под редакцией
А. П. Ершова и В. М. Монахова

Рекомендовано
Управлением информатики
и электронно-вычислительной техники
Министерства просвещения СССР

МОСКВА
«ПРОСВЕЩЕНИЕ» 1986

ББК 73я72
О-75

А. П. ЕРШОВ, В. М. МОНАХОВ, А. А. КУЗНЕЦОВ,
Я. Э. ГОЛЬЦ, М. П. ЛАПЧИК, А. С. ЛЕСНЕВСКИЙ,
Ю. А. ПЕРВИН, Д. О. СМЕКАЛИН

О-75 **Основы** информатики и вычислительной техники: Проб. учеб. пособие для сред. учеб. заведений. В 2 ч. Ч. 2/ А. П. Ершов, В. М. Монахов, А. А. Кузнецов и др.; Под ред. А. П. Ершова, В. М. Монахова.— М.: Просвещение, 1986.— 143 с.: ил.

Данное пробное учебное пособие предназначено для продолжения изучения курса «Основы информатики и вычислительной техники» учащимися средних специальных учебных заведений, профессионально-технических училищ и средней общеобразовательной школы.

О $\frac{4306020400-399}{103(03)-86}$ инф. письмо —86, доп. № 2 ББК 73я72+32.973я72

Раздел I

УСТРОЙСТВО ЭВМ

В первой части курса были изложены первоначальные сведения об электронных вычислительных машинах (ЭВМ). Коротко их повторим.

ЭВМ — это машина для автоматической обработки информации. Она состоит из *процессора, памяти* и *внешних устройств.*

В памяти хранится разнообразная информация, закодированная в виде последовательностей цифр 0 и 1, которые принято называть *битами* (англ. *bit* — сокращение от *binary digit* — двоичная цифра).

Работа процессора происходит под управлением *программы.* Программа находится в памяти ЭВМ и состоит из отдельных команд. Команды, как и любая информация, кодируются последовательностями цифр 0 и 1. Выполнение одной команды процессором может состоять в арифметическом действии над числами, перемещении информации в памяти, выдаче команды внешнему устройству (например, «изобразить букву А на экране») и т. д.

Внешние устройства ЭВМ обеспечивают ввод и вывод информации, взаимодействие ЭВМ с человеком и другими ЭВМ. Кроме того, внешние устройства позволяют хранить информацию на так называемых внешних носителях: гибких магнитных дисках, кассетах магнитофона и т. д. На рисунке 1 изображен гибкий магнитный диск (он лежит на учебнике) и накопитель на гибких магнитных дисках (дисковод), в который вставлен другой такой же магнитный диск.

С физической точки зрения информация, циркулирующая внутри ЭВМ, кодируется с помощью электрических сигналов. Обработку этих сигналов осуществляют электронные устройства, состоящие из тысяч и даже сотен тысяч согласованно работающих элементов. Такие устройства называют *микросхемами* (рис. 2). Если микросхема содержит целиком весь процессор ЭВМ, то она называется *микропроцессором.* Микропроцессор и другие микросхемы размещаются на плáте, которая видна на рисунке 3. На рисунке 4 изображена плата, на которой: 1 — микропроцессор, 2 — микросхемы памяти.

В этом разделе разберем подробнее устройство ЭВМ и ее частей, познакомимся с кодированием информации в машине,

Рис. 1

увидим, как выглядят машинные программы, и узнаем, как работает ЭВМ при их исполнении.

Будем рассматривать устройство вычислительных машин на примере выпускаемых нашей промышленностью персональных ЭВМ ДВК-2М (рис. 5), построенных на базе микропроцессора К1801ВМ1 (рис. 6).

§ 1. ОБЩАЯ СХЕМА УСТРОЙСТВА ЭВМ

Рассмотрим сначала, как хранится информация и как соединяются и обмениваются между собой информацией компоненты ЭВМ.

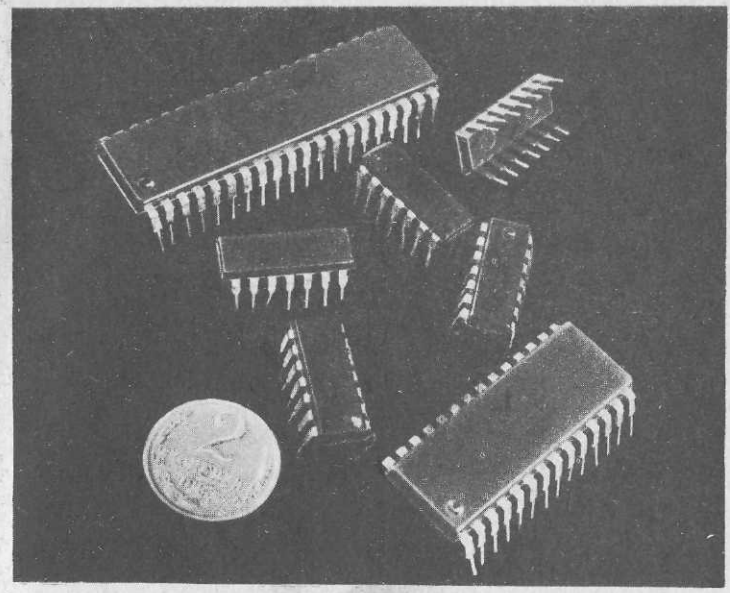


Рис. 2

Рис. 3

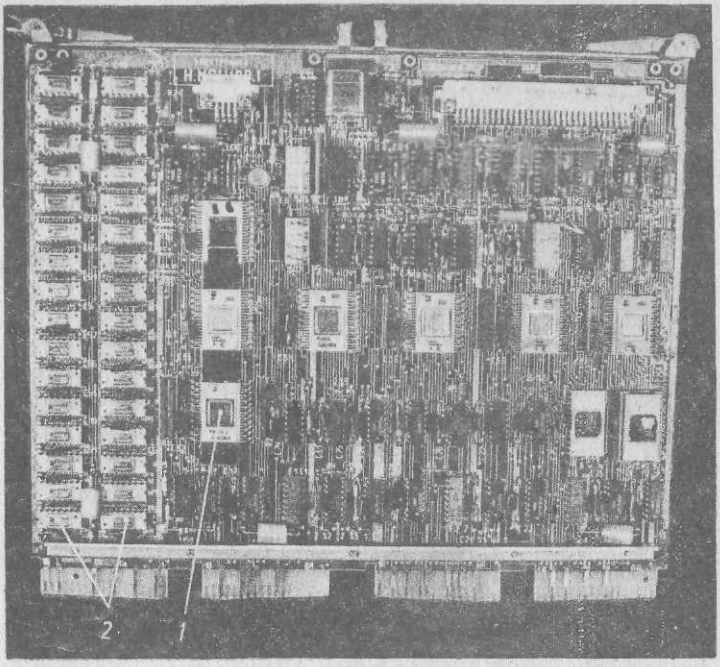


Рис. 4

Рис. 5

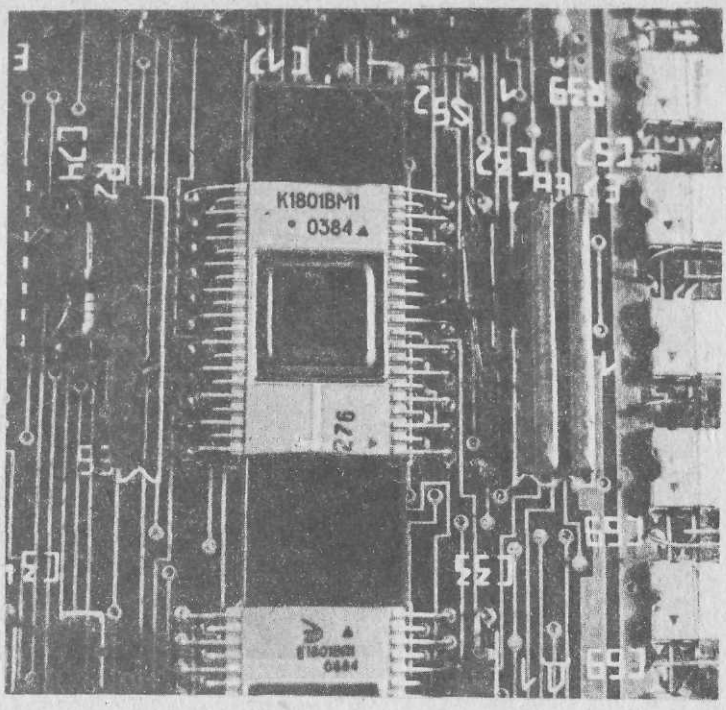


Рис. 6

1. Память ЭВМ. Вся память разбита на отдельные участки из восьми битов. 8 битов информации называются *байтом*. Более крупная единица измерения — байт — часто оказывается удобнее, чем бит. Байты имеют номера: 0, 1, 2, 3, ... Эти номера называются также *адресами*. Два соседних участка: нулевой и первый, второй и третий, четвертый и пятый и т. д.— образуют ячейки памяти ЭВМ. Как мы видим, одна ячейка памяти может хранить два байта, другими словами — 16 битов информации. Для содержимого одной ячейки используется название — *машинное слово* или просто *слово*. Каждое слово тоже имеет адрес — это адрес начального байта слова. Таким образом, адресами слов будут четные числа 0, 2, 4, 6, ... Адреса слов и байтов тоже необходимо хранить в памяти ЭВМ и передавать между различными устройствами машины. При этом, как и другая информация, адреса кодируются последовательностями нулей и единиц. Каждый адрес кодируется 16 битами.

Например, адреса 0, 1024 и 65535 кодируются так:

0000000000000000
0000010000000000
1111111111111111

2. Связь и обмен информацией между компонентами ЭВМ осуществляется с помощью *магистрали* (рис. 7). Магистраль можно представить себе как пучок проводов, к которому параллельно подсоединены все компоненты ЭВМ. Посылая по магистрали электрические сигналы, любая компонента ЭВМ может передавать информацию другим компонентам.

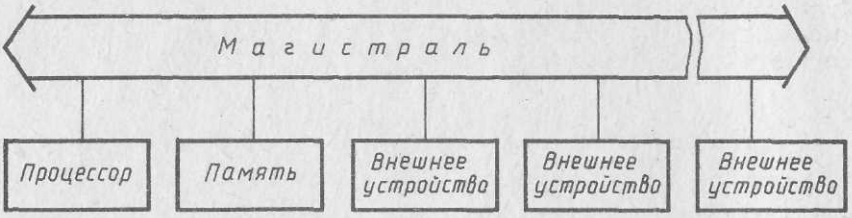


Рис. 7

Рассмотрим, как происходит обмен информацией через магистраль. Пусть процессору необходимо прочесть порцию информации из памяти. Прежде всего необходимо указать, в каком месте памяти расположена требуемая информация. Это место указывается с помощью адреса. Для передачи адреса (16 битов) используются 16 проводов магистрали. По тем же 16 проводам передается и информация. Таким образом, максимальная порция информации, передаваемая за один раз, состоит из 16 битов.

Кроме проводов для передачи адресов и информации, в магистрали есть и другие провода. По ним передаются управляющие сигналы, например указание, что надо делать с информацией, расположенной по данному адресу: прочесть или записать.

Чтение процессором слова, записанного в памяти по некоторому адресу, происходит следующим образом:

1. Процессор передает на магистраль адрес нужного слова и указание "читай слово".
2. Память считывает адрес с магистрали и передает на магистраль содержимое соответствующего слова.
3. Процессор считывает слово с магистрали.

Примерно так же взаимодействуют между собой и все остальные компоненты ЭВМ. Более подробно об этом смотрите в приложении III.

Вопросы для повторения

1. Что такое ЭВМ? Для чего она используется?
2. Перечислите внешние устройства ЭВМ. Расскажите об их назначении.
3. Что такое бит и байт? Сколько битов в одном байте?

Вопросы

1. Что такое микросхема, микропроцессор?
2. Что такое магистраль? Для чего она служит?
3. Как происходит чтение слова из памяти?

Упражнения

1. Память ЭВМ ДВК-2М позволяет хранить 56 кбайтов. Учитывая, что 1 кбайт = 1024 байта, определите, сколько слов может поместиться в памяти этой ЭВМ.

§2. ОСНОВНОЙ АЛГОРИТМ РАБОТЫ ПРОЦЕССОРА

ЭВМ можно запрограммировать для исполнения очень сложных алгоритмов и переработки огромных объемов информации. Однако при исполнении этих алгоритмов ЭВМ выполняет громадное число простых шагов; за один шаг ЭВМ выполняет только одну команду над элементарными порциями информации — битами, байтами, словами.

Поэтому прежде всего познакомимся с тем, как устроен процессор, как он выполняет элементарный шаг и как эти шаги следуют друг за другом.

1. Собственная память процессора. Процессор — это важнейшее устройство ЭВМ, обеспечивающее автоматическое исполнение хранящейся в памяти программы. Процессор имеет небольшую собственную память, из которой нас интересуют 8 слов, называемых *регистрами* и имеющих имена R0, R1, R2, R3, R4, R5, R6, R7.

Каждый регистр, как и слово обычной памяти, хранит 16 битов; кроме того, в собственной памяти процессора нас интересуют еще два бита с именами N и Z.

2. Основной алгоритм работы процессора. Как и всякая информация, команды программы размещаются в памяти ЭВМ, и каждая команда занимает одно слово. Адреса слов — четные числа, каждое слово занимает два байта. Поэтому, например, программа из трех команд может храниться по адресам 1500, 1502, 1504.

Процессор в каждый момент исполнения программы помнит, какую команду он должен выполнять следующей. Для этой цели используется регистр R7, называемый также Счетчиком Команд (СК). Если после выполнения очередной команды СК = 1500, то это значит, что следующей выполняемой командой будет команда, хранящаяся по адресу 1500.

Как же работает процессор? Чтобы выполнить команду, он должен прежде всего узнать, в чем она состоит, т. е. получить из памяти слово по адресу, хранящемуся в Счетчике Команд. Далее, процессор должен изменить содержимое Счетчика Команд, иначе на следующем шаге будет снова выполняться та же самая коман-

да. Затем процессор приступает собственно к выполнению команды. Пусть, например, это команда "переслать слово R1 в R2". В этом случае процессор помещает в регистр R2 содержимое регистра R1. В нашем примере хранящийся в Счетчике Команд адрес увеличивается на 2. В результате этого процессор переходит к выполнению команды, которая хранится в памяти следом за только что выполненной.

Таким образом, действия процессора по выполнению каждой команды состоят из четырех этапов, процессор

1) читает адрес из Счетчика Команд;
2) читает слово из памяти по этому адресу;
3) увеличивает Счетчик Команд на 2;
4) выполняет команду, записанную в прочитанном слове.

Этот циклический процесс прекращается, когда выполняемой командой окажется специальная команда *стоп*.

В этом и состоит основной алгоритм работы процессора: процессор выполняет в 4 этапа очередную команду, затем в 4 этапа выполняет следующую и т. д. до тех пор, пока не выполнится команда *стоп*.

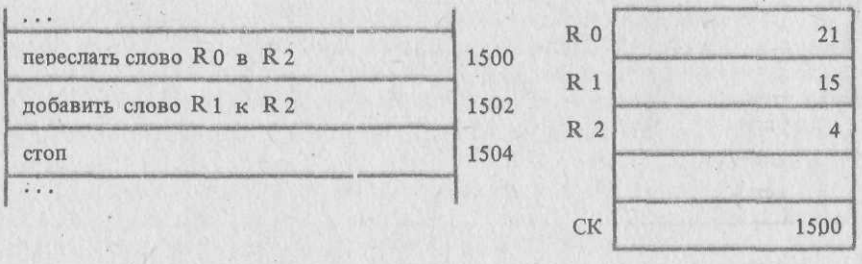
3. Пример программы. Пусть требуется целое число, находящееся в регистре R0, сложить с целым числом в регистре R1 и поместить результат в регистр R2, не меняя R0 и R1. Это можно сделать с помощью двух команд:

переслать слово R0 в R2,
добавить слово R1 к R2.

На алгоритмическом языке смысл этих команд можно выразить как R2 : = R0 и R2 : = R1 + R2.

Чтобы процессор выполнил эти две команды, их надо поместить в памяти друг за другом, например в ячейки с адресами 1500 и 1502, и записать в СК адрес первой команды, т. е. 1500. Чтобы процессор остановился после выполнения этих двух команд, в следующее слово памяти с адресом 1504 нужно поместить команду *стоп*. Предположим, что это сделано, и что в начальный момент содержимое регистров такое: R0 = 21, R1 = 15, R2 = 4 (табл. 1).

Таблица 1



Напомним, что и команды в памяти ЭВМ, и числа в регистрах процессора изображены в таблице в привычном для человека виде. В машине они кодируются последовательностями из 0 и 1.

4. Пример исполнения программы процессором. Посмотрим, как работает процессор при исполнении этой программы. В соответствии с основным алгоритмом, на первом шаге процессор

1) читает содержимое СК (1500);

2) читает слово из памяти по адресу 1500, т. е. команду "переслать слово R0 в R2";

3) увеличивает СК на 2 (до 1502);

4) выполняет прочитанную команду, т. е. пересылает содержимое регистра R0 (21) в регистр R2, после чего в регистре R2 также будет содержаться число 21.

После этого процессор выполняет второй шаг:

1) читает содержимое СК (1502);

2) читает слово из памяти по адресу 1502, т. е. команду "добавить слово R1 к R2";

3) увеличивает СК на 2 (до 1504);

4) выполняет прочитанную команду, т. е. добавляет содержимое регистра R1 (15) к содержимому регистра R2 (21), после чего в R2 будет содержаться число 36.

После этого процессор выполняет третий шаг, т. е. снова

1) читает содержимое СК (1504);

2) читает слово из памяти по адресу 1504, т. е. команду *стоп;*

3) увеличивает СК на 2 (до 1506);

4) выполняет прочитанную команду, т. е. останавливается.

Итак, в результате исполнения этой программы в регистре R2 появится число 36 — сумма содержимого регистров R0 (21) и R1 (15).

5. Таблица значений. Процесс исполнения рассматриваемой нами программы можно изобразить таблицей значений (табл. 2), в которой указаны начальные значения регистров, а также выполняемые команды и состояния регистров. После выполнения каждой команды в таблице указываются значения регистров, изменившихся при выполнении команды.

Таблица 2

Шаг	Команда	СК	R0	R1	R2
		1500	21	15	4
1	переслать слово R0 в R2	1502			21
2	добавить слово R1 к R2	1504			36
3	стоп	1506			

Из третьей колонки таблицы видно, что Счетчик Команд после выполнения каждой команды указывает адрес следующей вы-

полняемой команды программы. В процессе работы он «скользит» по программе, автоматически увеличиваясь при выполнении каждой команды, как бы «считая» команды программы. Этим и объясняется его название.

Вопросы для повторения

1. Каков общий вид алгоритма, записанного на алгоритмическом языке?
2. Какие бывают составные команды? Приведите примеры.
3. Какие типы величин используются в алгоритмическом языке? Приведите примеры.
4. Какие величины называются:

а) аргументами; б) результатами; в) промежуточными величинами? Приведите примеры.

5. Как выполняется команда присваивания?

Вопросы

1. Что надо сделать, чтобы процессор исполнил заданную программу? Где нужно разместить команды программы? Чему должно равняться содержимое СК?
2. Процессор выполнил команду *стоп*, расположенную в слове по адресу 1022. Каково после этого содержимое регистра СК?

Упражнения

1. Заполните таблицу значений для рассмотренной программы, предполагая, что в начальный момент:

а) СК = 1500, R0 = −21, R1 = −15, R2 = −4;
б) СК = 1502, R0 = 21, R1 = 15, R2 = 4;
в) СК = 1504, R0 = 22, R1 = 15, R2 = 4.

2. Напишите программу, которая складывает целые числа, находящиеся в регистрах R0, R1, R2, и помещает результат в:

а) регистр R3; б) регистр R2.

3. Заполните таблицу значений для программ упражнения 2, считая, что программа размещена начиная с адреса 1500, в начальный момент СК = 1500, R0 = 5, R1 = 7, R2 = 8.

4. Напишите программу, которая складывает удвоенное содержимое регистра R0 с утроенным содержимым регистра R1 и помещает результат в:

а) регистр R2; б) регистр R1; в) каждый из регистров R3 и R4.

5. Напишите программу, которая помещает в регистр R1 учетверенное содержимое регистра R0.

§ 3. КОМАНДА ВЕТВЛЕНИЯ И КОМАНДА ПОВТОРЕНИЯ

Для написания программ с ветвлениями и повторениями используются специальные команды, изменяющие содержимое Счетчика Команд (СК) в зависимости от значений битов N и Z процессора.

1. Назначение битов N и Z процессора. При выполнении некоторых команд процессора меняется не только содержимое регистров, но и биты N и Z в процессоре. Например, при выполнении команды "сравнить слово R1 с R2" в зависимости от знака числа (R1 — R2) устанавливаются следующие значения битов N (*negative* — отрицательный) и Z (*zero* — нуль) (табл. 3):

Таблица 3

Соотношения между R1 и R2	Знак числа R1 – R2	N	Z
R1 < R2	R1 – R2 < 0	1	0
R1 = R2	R1 – R2 = 0	0	1
R1 > R2	R1 – R2 > 0	0	0

Содержимое регистров R1 и R2 при выполнении этой команды не изменяется.

2. Команды условного и безусловного перехода. При выполнении рассмотренных команд содержимое СК увеличивалось на 2. Процессор нашей машины имеет целую группу команд, меняющих СК в зависимости от значений битов N и Z.

Примером такой команды является команда

если меньше, переход на +2 слова.

Ее выполнение зависит от значения бита N. Если оно равно 0, то, не выполняя никаких других действий, процессор увеличивает содержимое СК на 2. Если же бит N равен 1, то содержимое СК в дополнение к обычному увеличению на 2 будет увеличено еще на 4, т. е. увеличится на 6. В результате этого две команды, идущие в памяти вслед за этой командой, пропускаются. (Это объяснение в короткой форме содержится в названии команды.)

Существует и команда *безусловного перехода,* которая независимо ни от чего увеличивает или уменьшает СК на указанное число слов в дополнение к обычному увеличению на 2. Например, после выполнения команды

переход на +1 слово

содержимое СК увеличится на 4 (вместо обычного увеличения на 2) и одна команда будет пропущена.

3. Пример программы с ветвлением. Пусть требуется написать программу, которая помещает в регистр R3 большее из двух целых чисел, находящихся в регистрах R1 и R2. Запишем это на алгоритмическом языке:

```
алг большее число (цел R1, R2, R3)
    арг  R1, R2
    рез  R3
нач
    если R1 ⩾ R2
        то
            R3 := R1
        иначе
            R3 := R2
    все
кон
```

Расположим соответствующую программу в памяти ЭВМ, начиная с адреса 1500 (табл. 4).

Таблица 4

сравнить слово R1 с R2	1500
если меньше, переход на + 2 слова	1502
переслать слово R1 в R3	1504
переход на + 1 слово	1506
переслать слово R2 в R3	1508
стоп	1510

Приведем таблицу значений (табл. 5) для этой программы, считая, что в начальный момент СК = 1500, R1 = 5 и R2 = 7. Напомним, что в таблице значений указывается содержимое СК после выполнения команды.

Таблица 5

Шаг	Команда	СК	R1	R2	R3	N
		1500	5	7		
1	сравнить слово R1 с R2	1502				1
2	если меньше, переход на + 2 слова	1508				
3	переслать слово R2 в R3	1510			7	0
4	стоп	1512				

Заметьте, что при исполнении этой программы в результате шага 2 пропускаются две команды (с адресами 1504 и 1506). После шага 3 бит N станет нулевым, так как пересылается положительное число.

Если в начальный момент СК = 1500, R1 = −5, а R2 = −7, то таблица значений будет выглядеть так (табл. 6):

Таблица 6

Шаг	Команда	СК	R1	R2	R3	N
		1500	−5	−7		
1	сравнить слово R1 с R2	1502				0
2	если меньше, переход на + 2 слова	1504				
3	переслать слово R1 в R3	1506			−5	1
4	переход на + 1 слово	1510				
5	стоп	1512				

Заметьте, что при исполнении этой программы в результате шага 4 пропускается команда с адресом 1508.

4. Пример программы с повторением. Пусть требуется написать программу, которая должна поместить в регистр R2 значение выражения $K + (K - 1) + \ldots + 2 + 1$, где число K записано в регистре R1 и не должно меняться. При исполнении этой программы в регистре R2 будут храниться последовательно:

$$0,\ K,\ K + (K - 1),\ K + (K - 1) + (K - 2) \quad \text{и т. д.}$$

В регистре R0 будет очередное слагаемое: вначале K, затем $K - 1$, $K - 2$ и т. д. Когда R0 станет равным нулю, исполнение программы закончится.

Расположим соответствующую программу в памяти ЭВМ, начиная с адреса 1500 (табл. 7).

Таблица 7

очистить регистр R2	1500
переслать слово R1 в R0	1502
добавить слово R0 к R2	1504
уменьшить слово R0 на единицу	1506
если больше, переход на − 3 слова	1508
стоп	1510

Команда "очистить регистр R2" помещает в регистр R2 число 0 и записывается на алгоритмическом языке как R2 : = 0. Следующие команды означают R0 : = R1 и R2 : = R2 + R0. Команда "уменьшить слово R0 на единицу", помимо этого уменьшения, устанавливает значения битов N и Z: если новое, уменьшенное, значение R0 отрицательно, то выполняются действия N : = 1, Z : = 0. Если новое значение R0 равно 0, то N : = 0, Z : = 1. Наконец, если новое значение R0 положительно, то N: = 0, Z : = 0.

Команда "если больше, переход на —3 слова" проверяет значение битов N и Z. Если оба бита N и Z оказываются равными 0, то при ее выполнении СК уменьшается на 4 (вместо обычного увеличения на 2).

Приведем таблицу значений для этой программы, считая, что в начальный момент СК = 1500 и R1 = 3 (табл. 8). Голубым цветом показаны значения битов N и Z, проверяемые при выполнении следующего шага.

Таблица 8

Шаг	Команда	СК	R0	R1	R2	N	Z
		1500		3			
1	очистить регистр R2	1502			0	0	1
2	переслать слово R1 в R0	1504	3			0	0
3	добавить слово R0 к R2	1506			3	0	0
4	уменьшить слово R0 на единицу	1508	2			0	0
5	если больше, переход на —3 слова	1504					
6	добавить слово R0 к R2	1506			5	0	0
7	уменьшить слово R0 на единицу	1508	1			0	0
8	если больше, переход на —3 слова	1504					
9	добавить слово R0 к R2	1506			6	0	0
10	уменьшить слово R0 на единицу	1508	0			0	1
11	если больше, переход на —3 слова	1510					
12	стоп	1512					

Из этой таблицы видно, что команды с адресами 1500, 1502, 1510 выполнились один раз, команды с адресами 1504, 1506, 1508 выполнились по три раза каждая. Это вызвано тем, что после шагов 5 и 8 происходил возврат к команде с адресом 1504. Если бы в начальный момент содержимое регистра R1 равнялось числу 100, то команды с адресами 1504, 1506, 1508 выполнились бы по 100 раз каждая.

Вопросы для повторения

1. Какие знаки отношений между величинами используются в алгоритмическом языке?
2. Какие величины называются табличными? Приведите примеры.
3. Для чего нужны вспомогательные алгоритмы? Как записывается команда их вызова?

Вопросы

1. Известно, что R1 = 5, R2 = 5, R3 = −7, R4 = 0. Выпишите значения битов N и Z после выполнения команды а) "сравнить слово R1 с R2", б) "сравнить слово R2 с R3", в) "сравнить слово R3 с R4".
2. По адресу 1508 расположена команда "переход на +1 слово". Чему будет равно содержимое СК после выполнения этой команды?
3. В программе 5 команд. Может ли число шагов при исполнении этой программы оказаться равным: а) 1, б) 5, в) 100?

Упражнения

1. Напишите программу, после исполнения которой в регистре R0 будет содержаться:

а) 0;
б) −1;
в) наибольшее из чисел, лежащих в R0, R1;
г) наибольшее из чисел, лежащих в R1, R2, R3.

2. Напишите программу, после исполнения которой в регистре R0 будет содержаться сумма $2K + (2K - 2) + \ldots + 4 + 2$, если перед началом работы число K хранится в регистре R1.

3. Напишите программу, после исполнения которой в регистре R0 будет содержаться $999 + 997 + \ldots + 3 + 1$. Перед началом работы R1 = 500.

§4. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В ЭВМ

Мы уже говорили, что любая информация кодируется в ЭВМ с помощью последовательностей цифр 0 и 1. Рассмотрим теперь, как именно кодируются в ЭВМ адреса, числа, символы и команды.

1. Биты, байты и слова. Байт состоит из 8 битов, каждый из которых может принимать два значения: 0 или 1. Таким образом, байт может принимать $2^8 = 256$ различных значений, а состоящее из 16 битов машинное слово может принимать $2^{16} = 65536$ различных значений. Принято нумеровать биты справа налево, начи-

ная с нуля. Биты в слове "0001001001001111" нумеруются так (табл. 9):

Таблица 9

№ бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Бит	0	0	0	1	0	0	1	0	0	1	0	0	1	1	1	1

Значение нулевого бита обозначается i_0, значение 1-го бита обозначается i_1 и т. д.

2. Кодировка адресов. В ЭВМ каждый адрес кодируется последовательностью из 16 нулей и единиц, т. е. задается словом. В наших программах мы обозначали адреса числами от 0 до 65535. Соответствие между этими числами и последовательностями из 16 нулей и единиц задается формулой

$$\text{адрес} = i_{15} \cdot 2^{15} + i_{14} \cdot 2^{14} + \ldots + i_0 \cdot 2^0.$$

Приведем несколько примеров кодировки адресов (табл. 10).

Таблица 10

Адрес	Двоичный код
0	0000000000000000
1	0000000000000001
2	0000000000000010
3	0000000000000011
...	...
1024	0000010000000000
...	...
32767	0111111111111111
32768	1000000000000000
...	...
65534	1111111111111110
65535	1111111111111111

3. Кодировка целых чисел. Целые числа, так же как и адреса, кодируются последовательностями из 16 нулей и единиц, т. е. словами. Поэтому можно закодировать лишь 65536 различных целых чисел. Способ кодировки выбран так, что он охватывает целые числа от -32768 до $+32767$. Соответствие между целым числом и его двоичным кодом задается формулой

$$\text{число} = i_{15} \cdot 2^{15} + i_{14} \cdot 2^{14} + \ldots + i_0 \cdot 2^0 - i_{15} \cdot 2^{16}.$$

Приведем несколько примеров кодировки чисел (табл. 11).

Таблица 11

Число	Двоичный код
0	0000000000000000
1	0000000000000001
2	0000000000000010
3	0000000000000011
4	0000000000000100
5	0000000000000101
6	0000000000000110
7	0000000000000111
...	...
1024	0000010000000000
...	...
32766	0111111111111110
32767	0111111111111111
– 32768	1000000000000000
– 32767	1000000000000001
...	...
– 3	1111111111111101
– 2	1111111111111110
– 1	1111111111111111

4. Кодировка символов. Символы (цифры, русские и латинские буквы, знаки препинания и пр.) обычно кодируются последовательностями из 8 нулей и единиц, т. е. байтами. Поскольку байт имеет 256 различных значений, можно закодировать 256 различных символов. Существует несколько различных способов кодировки символов. Конкретный способ кодировки символов задается таблицей, в которой указаны коды всех символов.

5. Кодировка команд. В наших программах мы писали команды в обозначениях, удобных для человека. Однако в ЭВМ каждая команда кодируется последовательностью из 16 нулей и единиц, т. е. словом. Конкретные примеры кодировки команд приведены в приложении III.

Вопросы для повторения

1. Какие вы знаете команды работы с графической информацией? Приведите примеры алгоритмов, использующих эти команды.
2. Величины каких типов используются в алгоритмическом языке?

Вопросы

Каково число различных последовательностей из трех цифр, каждая из которых либо нуль, либо единица?

Упражнения

1. Какой адрес кодируется последовательностью 1000000000000000?
2. Какой последовательностью кодируется адрес 1024?
3. Какое целое число кодируется последовательностью 1000000000000000?

§ 5. ФИЗИЧЕСКИЕ ПРИНЦИПЫ РАБОТЫ ЭВМ

Мы уже знаем, что вся информация хранится в ЭВМ в виде большого количества нулей и единиц. В каком же виде записываются в памяти и обрабатываются процессором эти нули и единицы? Один из возможных способов таков. Будем считать, что наличие электрического сигнала, более точно — электрического напряжения (разности потенциалов), на данном участке электронной схемы кодирует единицу, а его отсутствие — нуль. (Реально — при одном из самых распространенных способов кодирования — к микросхеме присоединяется источник питания, вырабатывающий напряжение +5 В, нулю соответствует потенциал от 0 до 0,5 В, а единице — потенциал от 2,5 до 5 В по отношению к заземленным участкам схемы.)

Простейшим электронным устройством обработки информации, закодированной с помощью наличия или отсутствия напряжения, является инвертор (рис. 8). Инвертор имеет два контакта — входной и выходной. Он устроен так. Если подать на его вход напряжение, то на выходе напряжение пропадает, а если на вход не подавать напряжение, то на выходе оно будет. (Точнее нужно говорить, конечно, не о наличии или отсутствии напряжения, а о высоком или низком напряжении.)

Работа инвертора описывается такой таблицей (табл. 12): Мы знаем, что наличие напряжения кодирует единицу, а его отсутствие — нуль. Получаем такую таблицу (табл. 13):

Рис. 8

Таблица 12

Напряжение на	
входе	выходе
есть	нет
нет	есть

Таблица 13

Сигнал на	
входе	выходе
1	0
0	1

На тех же принципах, что и инвертор, основан и более сложный элемент, имеющий два входа и один выход (рис. 9). Его работа описывается таблицей 14.

Другими словами, на выходе такого элемента появляется нуль, если на обоих его входах — единицы. Во всех остальных случаях на его выходе появляется единица.

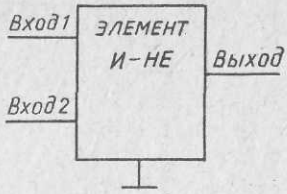


Рис. 9

Таблица 14

вход 1	вход 2	выход
0	0	1
0	1	1
1	0	1
1	1	0

Это можно записать так:

(выход=1) равносильно **не** (вход 1=1 **и** вход 2=1),

поэтому этот элемент иногда называют элементом И-НЕ.

Из тысяч и миллионов такого рода элементов строятся ЭВМ.

Расскажем теперь об устройстве и физических принципах работы клавиатуры, печатающего устройства, экрана дисплея и накопителя на гибких магнитных дисках.

Клавиатура (см. рис. 5) предназначена для ввода информации в ЭВМ с помощью нажатий на клавиши. Простейший вариант устройства клавиатуры таков: при нажатии на клавишу замыкается контакт (как в кнопке электрического звонка) и тем самым в электронную схему клавиатуры попадает сигнал о том, какая клавиша нажата.

Такая конструкция клавиатуры далеко не единственно возможная. Конструкторы ЭВМ постоянно придумывают новые способы устройства клавиатуры, чтобы повысить ее надежность и долговечность, снизить стоимость изготовления.

Печатающее устройство (рис. 10), или *принтер* (от англ. *print* — печатать), позволяет выводить результаты работы ЭВМ на бумагу. С его помощью можно печатать и рисунки (рис. 11), и тексты. Если посмотреть на рисунок 11 при большом увеличении, то мы увидим, что он составлен из отдельных точек (рис. 12). Все геометрические фигуры, которые может напечатать принтер, являются множествами точек. Представим себе клетчатую бумагу, в которой разрешается закрашивать отдельные клеточки (рис. 13). Закрашивая их, мы можем нарисовать на клет-

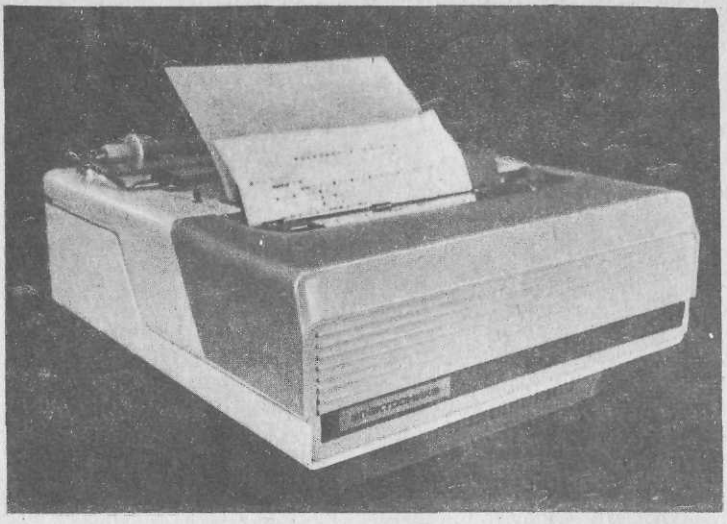


Рис. 10

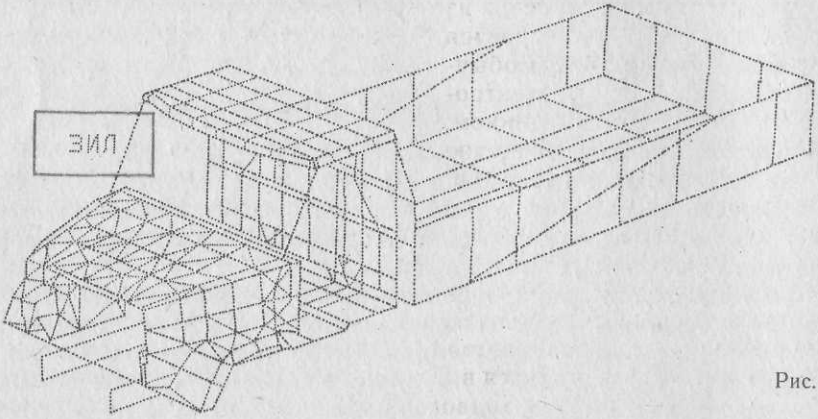


Рис. 11

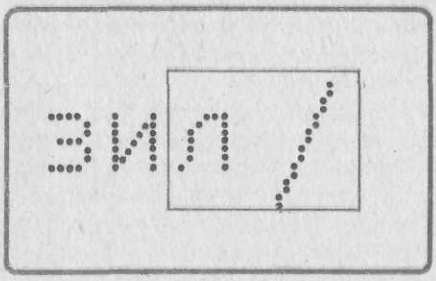


Рис. 12

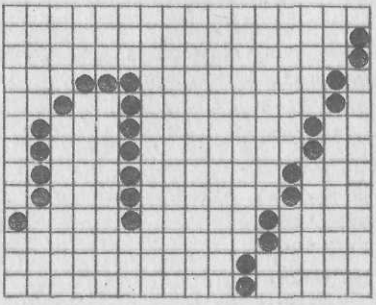


Рис. 13

чатой бумаге картинку из отдельных точек. Они будут мало заметными, особенно если рассматривать ее с большого расстояния, а расстояние между точками (сторона клетки) мало.

Такие картинки принтер печатает с помощью печатающей головки, перемещающейся относительно бумаги. Она содержит иголочки — обычно около десяти, управляемые каждая своим электромагнитом. Иголочки могут ударять по красящей ленте, оставляя черную точку в том месте, где лента коснулась бумаги.

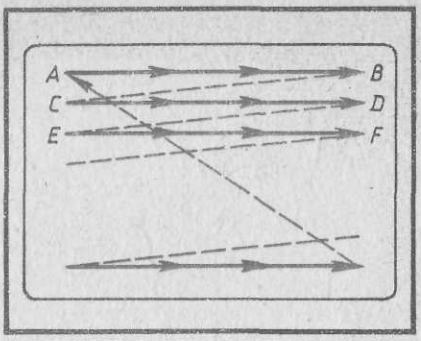


Рис. 14

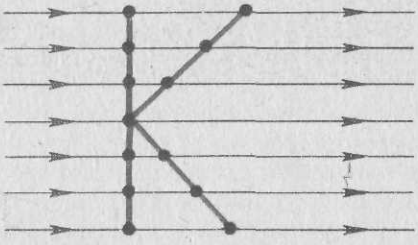


Рис. 15

Изображение на *экране дисплея*, так же как и печатаемые принтером картинки, состоит из отдельных точек. Оно строится таким же образом, как в обычном телевизоре. Пучок электронов, испускаемый источником, попадает на экран, и на экране возникает светящаяся точка. Яркость этой точки зависит от подаваемого на кинескоп управляющего потенциала и может управляться электронной схемой дисплея. (Потенциал может быть и таким, что точки вовсе не будет видно.) Отклоняющая система перемещает эту точку по экрану, создавая для этой цели магнитное поле, отклоняющее летящие электроны. При этом точка движется из A в B, затем перескакивает из B в C, движется в D, перескакивает в E, движется в F и т. д. (рис. 14). Дойдя до правого нижнего края экрана, она перескакивает в точку A, и цикл начинается сначала. Все это происходит быстро: точка обходит весь экран за сотые доли секунды. Двигаясь по экрану, точка меняет яркость, и на экране возникает картинка из светлых и темных точек. Например, чтобы на экране появилась светящаяся буква K, двигающаяся по экрану точка должна ярко вспыхивать на тех местах, где она попадает на линии, составляющие эту букву. В результате на экране появляется набор светящихся точек, которые образуют изображение буквы K (рис. 15). Поскольку весь путь проходится лучом за сотые доли секунды, глаз человека не замечает движения луча и из отдельных точек создается впечатление светящейся картинки на экране дисплея.

Накопитель на гибких магнитных дисках (см. рис. 1) позволяет сохранять информацию между сеансами работы с ЭВМ на гибком

диске. Если бы такого устройства не было, то все результаты работы машины терялись бы при выключении и каждый сеанс работы приходилось бы начинать с того, что вводить заново все программы, исходные данные и т. п.

Гибкий диск (см. рис. 1) представляет собой пластмассовую круглую пластинку, покрытую специальным составом, обладающим магнитными свойствами. (Похожим составом покрыта магнитофонная лента.) Он напоминает гибкую грампластинку, помещенную в закрытый конверт. В центре диска и конверта имеются круглые отверстия. Кроме того, в конверте есть еще одно продолговатое отверстие, через которое магнитная головка, читающая и записывающая информацию на диск, может приближаться к поверхности диска. (На рисунке 1 сквозь отверстия в конверте видна часть диска.) Когда диск вставляется в накопитель, отверстие в нем надевается на вращающуюся ось и диск начинает крутиться, совершая несколько оборотов в секунду. После этого магнитная головка, похожая на те, которые используются в магнитофонах, приближается к вращающейся поверхности диска и начинает записывать информацию или считывать ранее записанную. Запись и воспроизведение происходят так же, как в магнитофонах, только записывается и воспроизводится не музыка, а последовательности нулей и единиц, кодирующие нужную нам информацию.

Подобно тому как в проигрывателе игла может быть поставлена в любое место пластинки, магнитная головка накопителя может перемещаться по радиусу. Таким образом, чтобы подвести магнитную головку к нужному месту вращающегося гибкого диска, нужно поместить ее на заданное расстояние от центра и подождать, пока нужная часть диска, вращаясь, не подойдет к ней. Это происходит за доли секунды.

Именно благодаря такой возможности сравнительно быстрого доступа к любой части записанной на диске информации накопитель на магнитных дисках оказывается удобнее в качестве запоминающего устройства, чем магнитофон. Ведь в магнитофоне, прежде чем найти нужную запись, иногда приходится несколько минут перематывать магнитную ленту. Однако и магнитофон используется как внешнее устройство ЭВМ в тех случаях, когда не нужна большая скорость поиска и записи.

Емкость одного гибкого диска оказывается достаточной, чтобы запомнить несколько сотен тысяч байтов. Таким образом, один диск может содержать столько же информации, сколько книга из нескольких сотен страниц.

Заключение. При исполнении сложнейших программ ЭВМ работает *по шагам,* выполняя за один шаг одну из элементарных команд над элементарными порциями информации.

Автоматическая работа ЭВМ обеспечивается основным алгоритмом работы процессора и использованием специального ре-

гистра — Счетчика Команд. И программа, и обрабатываемая с ее помощью информация кодируются в двоичном коде.

Составление программ в двоичных кодах и размещение их в памяти ЭВМ очень трудоемкая работа. Мы видели, как непросто закодировать даже элементарную конструкцию **если-то-иначе-все,** а ведь мы писали не в двоичных кодах, а в привычных для человека обозначениях. Поэтому в кодах, как правило, не работают — человек составляет программу на алгоритмическом языке или на каком-то языке программирования, а в коды эту программу переводит уже сама ЭВМ под управлением специальных *системных* программ. Как именно это делается, мы рассмотрим в § 17.

В этом разделе вы познакомились с устройством *конкретной* ЭВМ и командами *конкретного* микропроцессора. Хотя все остальные ЭВМ и микропроцессоры устроены примерно так же (команды и информация кодируются двоичными кодами, программа располагается в памяти, используется Счетчик Команд и пр.), их конструкция может существенно различаться в деталях. Например, микропроцессор может иметь другой набор команд, машинное слово может состоять из четырех байтов и т. д.

Вопросы

1. Как кодируется информация в ЭВМ? Как представляются биты при хранении информации в памяти и обработке ее процессором?
2. Что такое инвертор?
3. Что такое элемент И-НЕ?
4. Как может быть устроена клавиатура ЭВМ?
5. Как образуется изображение, печатаемое на бумаге?
6. Как образуется изображение на экране дисплея?
7. Как записывается информация на магнитном диске?

Раздел II

ЗНАКОМСТВО С ПРОГРАММИРОВАНИЕМ

АЛГОРИТМИЧЕСКИЙ ЯЗЫК

В первой части курса мы изучали основные конструкции алгоритмического языка. Сейчас продолжим это изучение.

§6. КОМАНДА ВЫБОРА

Напомним, что команда ветвления записывается так:

```
если условие
|   то серия 1
|   иначе серия 2
все
```

Исполнитель выполняет эту команду следующим образом. Прежде всего он проверяет, соблюдается ли условие. В случае, когда условие соблюдается, исполнитель выполняет команды из серии 1. В противном случае выполняются команды из серии 2.

Однако часто оказывается, что возможных случаев не два, а больше. Например, вспомним алгоритм определения кислотности раствора (п. 4, ч. I).

```
алг определение кислотности раствора
нач
|   отлить в пробирку 1 мл раствора
|   опустить в пробирку лакмусовую бумажку
|   если бумажка красная
|   |   то ответ: раствор кислотный
|   |   иначе если бумажка синяя
|   |   |       то ответ: раствор щелочной
|   |   все
|   все
кон
```

Здесь, как мы видим, есть три возможности:
1) бумажка покраснела;
2) бумажка посинела;
3) бумажка не изменила цвет.

Команда *выбора* позволит записать этот же алгоритм более просто и наглядно. Сначала мы приведем пример записи команды выбора, а потом объясним, как такая команда выполняется.

```
алг определение кислотности раствора
нач
| отлить в пробирку 1 мл раствора
| опустить в пробирку лакмусовую бумажку
| выбор
|   при бумажка красная:
|       ответ: раствор кислотный
|   при бумажка синяя:
|       ответ: раствор щелочной
|   при бумажка не изменила цвет:
|       ответ: раствор нейтральный
| все
кон
```

Команда выбора может записываться в полной и сокращенной форме. В этом примере использована сокращенная форма команды:

```
выбор
|   при условие 1: серия 1
|   при условие 2: серия 2
|   ...
|   при условие N: серия N
все
```

Выполнение команды выбора происходит так. Сначала исполнитель проверяет, соблюдается ли условие 1. Если да, то он выполняет команды, входящие в серию 1, и на этом выполнение команды выбора заканчивается. Если нет, то исполнитель проверяет, соблюдается ли условие 2. Если проверяемое условие соблюдается, то он выполняет команды, входящие в серию 2, и на этом выполнение команды выбора заканчивается. Если же и условие 2 не соблюдается, то исполнитель проверяет условие 3. Если оно соблюдается, то он выполняет серию 3 и т. д. Другими словами, исполнитель последовательно проверяет все условия команды, пока не обнаружит первое из них, которое соблюдается. Найдя первое такое условие, исполнитель выполняет стоящую за ним се-

рию команд, и на этом выполнение команды выбора заканчивается. Если, например, перед началом выполнения команды выбора соблюдается и условие 2, и условие 3, то исполнитель все равно выполнит только серию 2.

В сокращенной форме команды выбора не предусмотрено никаких действий на случай, если ни одно из условий команды выбора не соблюдается. В этом случае исполнитель завершает выполнение команды выбора, не делая никаких других действий. Если же мы хотим предусмотреть какое-то действие и на этот случай, мы должны записать команду выбора в такой форме:

```
выбор
    при условие 1: серия 1
    при условие 2: серия 2
    при условие 3: серия 3
    ...
    при условие N: серия N
    иначе серия
все
```

Записав команду выбора таким образом, мы заставим исполнителя в случае, когда ни одно из условий не соблюдается, выполнить команды серии, стоящей после слова **иначе**. Используя эту конструкцию, алгоритм определения кислотности раствора можно записать таким образом:

```
алг определение кислотности раствора
нач
    отлить в пробирку 1 мл раствора
    опустить в пробирку лакмусовую бумажку
    выбор
        при бумажка красная:
            ответ: раствор кислотный
        при бумажка синяя:
            ответ: раствор щелочной
        иначе ответ: раствор нейтральный
    все
кон
```

Рассмотрим еще три примера использования команды выбора при записи алгоритмов.

Пример 1. Записать алгоритм, определяющий число корней уравнения $x^2=a$ в зависимости от a.

```
алг число корней (вещ a, лит A)
    арг a
    рез A
нач
|   выбор
|   |   при a < 0: A:="корней нет"
|   |   при a = 0: A:="один корень"
|   |   при a > 0: A:="два корня"
|   все
кон
```

Пример 2. Записать алгоритм, позволяющий получить словесное наименование школьных оценок, т. е. позволяющий получить их значение в виде литерных величин.

```
алг название оценки (цел x, лит A)
    арг x
    рез A
нач
|   выбор
|   |   при x=5: A :="отлично"
|   |   при x=4: A :="хорошо"
|   |   при x=3: A :="удовлетворительно"
|   |   при x=2: A :="неудовлетворительно"
|   |   при x=1: A :="плохо"
|   |   иначе A:="такой отметки нет"
|   все
кон
```

Пример 3. Составить алгоритм вычисления значений функции, график которой изображен на рисунке 16.

Область определения этой функции разбивается на три участка. На промежутке $]-\infty, 0]$ значения функции тождественно равны нулю. На промежутке $]0, 1]$ значения данной функции совпадают со значениями функции $y = x$. Наконец,

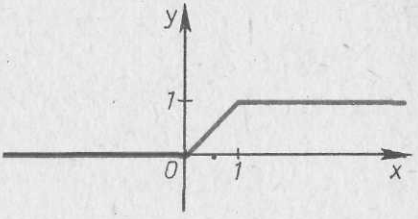


Рис. 16

на промежутке $]1, +\infty[$ значения функции тождественно равны 1. Таким образом различаются три случая:

1) $x \leqslant 0$;
2) $0 < x \leqslant 1$;
3) $x > 1$.

Выписанные условия и будут фигурировать в команде выбора; мы заменим лишь запись $0 < x \leqslant 1$ на принятую в алгоритмическом языке запись $0 < x$ **и** $x \leqslant 1$. Получается такой алгоритм:

```
алг значение функции (вещ x, вещ y)
    арг x
    рез y
нач
|   выбор
|       при x ⩽ 0: y := 0
|       при 0 < x и x ⩽ 1: y := x
|       при x > 1: y := 1
|   все
кон
```

Вспомним, что условия в команде выбора проверяются последовательно. При этом второе условие будет проверяться только в том случае, если первое условие не соблюдается и, следовательно, $x > 0$. Таким образом, часть второго условия $(0 < x)$ можно не проверять: если дело дошло до проверки второго условия, то заведомо $0 < x$. Рассуждая аналогично, замечаем, что проверку третьего условия $(x > 1)$ можно исключить. Действительно, если второе условие $(x \leqslant 1)$ не соблюдается, то третье условие $(x > 1)$ будет соблюдаться автоматически. Приходим к такому алгоритму:

```
алг значение функции (вещ x, вещ y)
    арг x
    рез y
нач
|   выбор
|       при x ⩽ 0: y := 0
|       при x ⩽ 1: y := x
|       иначе y := 1
|   все
кон
```

З а м е ч а н и е. Команду ветвления можно рассматривать как частный случай команды выбора. Ее сокращенная форма

если условие
| **то** серия
все

может быть записана как команда выбора с одним условием:

выбор
| **при** условие: серия
все

Ее полная форма

если условие
| **то** серия 1
| **иначе** серия 2
все

может быть записана в виде

выбор
| **при** условие: серия 1
| **иначе** серия 2
все

Вопросы

1. Как записывается команда выбора? Как она выполняется? Приведите примеры.
2. В каких случаях исполнитель выполняет серию команд, стоящих после слова **иначе** в команде выбора?
3. Может ли команда ветвления рассматриваться как частный случай команды выбора? Приведите примеры.

Упражнения

1. Запишите в виде команды выбора фразу из сказки: «Направо пойдешь — коня потеряешь, налево пойдешь — смертью умрешь, прямо пойдешь — друга найдешь».
2. Напишите алгоритм вычисления значений функции, график которой изображен на рисунке 17.

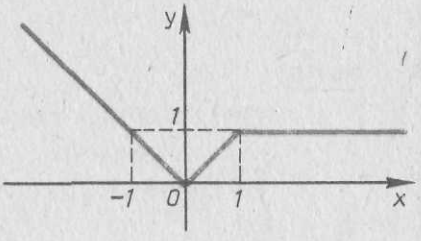


Рис. 17

3. Запишите алгоритм нахождения числа корней уравнения $x^2 = a + 1$ в зависимости от значения параметра a.

4. Запишите алгоритм вычисления абсолютной величины (модуля) вещественного числа с помощью команды выбора.

5. Нарисуйте график функции, значения которой вычисляются таким алгоритмом:

```
алг значение функции (вещ x, вещ y)
    арг x
    рез y
нач
|   выбор
|   |       при x ⩽ 0: y:=1
|   |       при 0 < x и x ⩽ 1: y:=1−x
|   |       при x > 1: y:=0
|   все
кон
```

6. В тексте приведены два варианта алгоритма «значение функции» с использованием команды выбора. Поменяем в каждом из них местами два первых условия команды выбора, записав:

а)
```
выбор
        при 0 < x и x ⩽ 1: y:=x
        при x ⩽ 0: y:=0
```

б)
```
выбор
        при x ⩽ 1: y:=x
        при x ⩽ 0: y:=0
```

Останутся ли алгоритмы правильными? Почему?

7. Запишите с помощью команды выбора:

```
если условие 1
|   то если условие 2
|   |   |   то серия 1
|   |   все
|   иначе
|       если условие 3
|       |   то серия 2
|       |   иначе серия 3
|       все
все
```

8. Напишите алгоритм нахождения числа корней уравнения $ax + b = 0$ в зависимости от значений параметров a, b. Не забудьте рассмотреть случай $a = 0$, а также $a = 0$ и $b = 0$.

9. Напишите алгоритм нахождения числа дней в месяце, если даны: номер месяца n — целое число от 1 до 12; целое число a, равное 1 для високосного года и равное 0 в противном случае.

10. Напишите алгоритм вычисления значений функции $y = |x - 1| + |x - 4|$, различающий случаи $x \leqslant 1$, $1 < x \leqslant 4$ и $x > 4$.

11. Чтобы определить, на какую цифру оканчивается квадрат целого числа, достаточно знать лишь последнюю цифру самого числа. Напишите алгоритм, который по одной из цифр 0, 1, 2, ..., 9 — последней цифре числа n — находит последнюю цифру числа n^2.

12. Напишите алгоритм, который по номеру дня недели — целому числу от 1 до 7 — выдает в качестве результата количество уроков в вашем классе в соответствующий день.

§7. КОМАНДА ПОВТОРЕНИЯ С ПАРАМЕТРОМ

В первой части курса (п. 4) мы изучали команду повторения

пока условие
нц
| серия
кц

Эта команда позволяет выполнить одну и ту же серию многократно.

Пример 1. Записать алгоритм, заполняющий нулями таблицу **вещ таб** $a[1:100]$.

```
алг заполнение нулями (вещ таб a[1:100])
    рез a
нач цел i
|   i:=1
|   пока i ⩽ 100
|   нц
|   |   a[i]:=0
|   |   i:=i + 1
|   кц
кон
```

При исполнении этого алгоритма команда

$$a[i]:=0$$

выполняется сначала для $i = 1$, затем для $i = 2, \ldots, 100$. Более наглядно этот алгоритм может быть записан с помощью *команды повторения с параметром*. При этом он будет выглядеть так:

```
алг заполнение нулями (вещ таб a[1:100])
    рез a
нач цел i
|   для i от 1 до 100
|   нц
|   | a[i]:=0
|   кц
кон
```

В данном случае параметром является целочисленная переменная величина i, которая в ходе выполнения команды принимает все значения от 1 до 100. Для каждого из этих значений переменной выполняется команда $a[i]:=0$. В результате таблица a заполняется 100 нулями.

Общий вид команды повторения с параметром таков:

```
для x от Xmin до Xmax
нц
| серия
кц
```

Здесь x — целочисленная переменная величина, а X_{min} и X_{max} — выражения, принимающие целочисленные значения. Выполнение команды происходит так:

1) вычисляются значения выражений X_{min} и X_{max};

2) переменной x последовательно присваиваются значения X_{min}, $X_{min} + 1$, $X_{min} + 2$, ..., X_{max}, и для каждого из этих значений выполняется серия команд, заключенная между **нц** и **кц**, в частности при $X_{min} = X_{max}$ эта серия выполнится только один раз. Если же с самого начала $X_{min} > X_{max}$, то серия не выполнится ни разу.

Приведем еще несколько примеров, где применение команды повторения с параметром (будем называть ее циклом «**для**») вместо команды повторения (цикла «**пока**») позволяет упростить запись алгоритмов.

Пример 2. Записать алгоритм суммирования элементов линейной таблицы (п. 10) с использованием команды повторения с параметром.

В п. 10 (см. ч. I) этот алгоритм был записан в виде:

```
алг сумма (вещ таб a[1:1000], вещ S)
    арг a
    рез S
нач цел i
|   i:=1; S:=0
|   пока i ⩽ 1000
|   нц
|   |  S:=S + a[i]; i:=i + 1
|   кц
кон
```

Теперь это можно записать его проще:

```
алг сумма (вещ таб a[1:1000], вещ S)
    арг a
    рез S
нач цел i
|   S:=0
|   для i от 1 до 1000
|   нц
|   |  S:=S + a[i]
|   кц
кон
```

Пример 3. Заполнить таблицу *произведение* [1:9, 1:9], сделав *произведение* [*i*, *j*] равным произведению чисел *i* и *j*.

В п. 10 для этого использовался такой алгоритм:

```
алг таблица умножения (цел таб произведение [1:9, 1:9])
    рез произведение
нач цел i, j
|   i:=1
|   пока i ⩽ 9
|   нц
|   |  j:=1
|   |  пока j ⩽ 9
|   |  нц
|   |  |  произведение [i, j]:=i*j; j:=j + 1
|   |  кц
|   |  i:=i + 1
|   кц
кон
```

Мы используем обозначение $i*j$ для произведения чисел i и j, чтобы постепенно привыкать к обозначениям, принятым в языках программирования.

Сейчас, применяя команду повторения с параметром, можно записать алгоритм «таблица умножения» более наглядно:

```
алг таблица умножения (цел таб произведение [1:9, 1:9])
    рез произведение
нач
|   цел i, j
|   для i от 1 до 9
|   нц
|   |   для j от 1 до 9
|   |   нц
|   |   |   произведение [i, j]:=i*j
|   |   кц
|   кц
кон
```

В этом примере одна команда повторения используется внутри другой команды повторения. Мы видим, что команда повторения с параметром позволяет записывать алгоритмы более коротко и наглядно.

Пример 4. Записать алгоритм, переписывающий данные из одной таблицы в другую, т. е. делающий вторую таблицу копией первой.

```
алг копирование таблицы (цел таб a[1:100], цел таб b[1:100])
    арг a
    рез b
нач
    цел i
    для i от 1 до 100
    нц
    |   b[i]:=a[i]
    кц
кон
```

Если нам нужно, чтобы данные в таблице b располагались в обратном порядке (по сравнению с их порядком в таблице a), то можно составить алгоритм «обращение» следующим образом:

```
алг обращение (цел таб a[1:100], цел таб b[1:100])
    арг a
    рез b
нач цел i
|   для i от 1 до 100
|   нц
|   |  b[i]:=a[101 − i]
|   кц
кон
```

При исполнении этого алгоритма команда

$$b[i]:=a[101-i]$$

выполняется сначала при $i=1$:

$$b[1]:=a[101-1],$$

затем при $i=2$:

$$b[2]:=a[101-2],$$

...

наконец, при $i=100$:

$$b[100]:=a[101-100].$$

В результате этого значения $b[1]$, $b[2]$, ..., $b[100]$ становятся равными соответственно $a[100]$, $a[99]$, ..., $a[1]$, что и требовалось.

Пример 5. Записать алгоритм, который подсчитывает число отрицательных, нулевых и положительных элементов в таблице.

Этот алгоритм иллюстрирует совместное применение команд повторения с параметром и выбора.

```
алг подсчет (цел таб a[1:100], цел отр, нул, пол)
    арг a
    рез отр, нул, пол
нач цел i
|   отр:=0; нул:=0; пол:=0
|   для i от 1 до 100
|   нц
|   |   выбор
|   |   |       при a[i] < 0: отр:=отр + 1
|   |   |       при a[i] = 0: нул:=нул + 1
|   |   |       при a[i] > 0: пол:=пол + 1
|   |   все
|   кц
кон
```

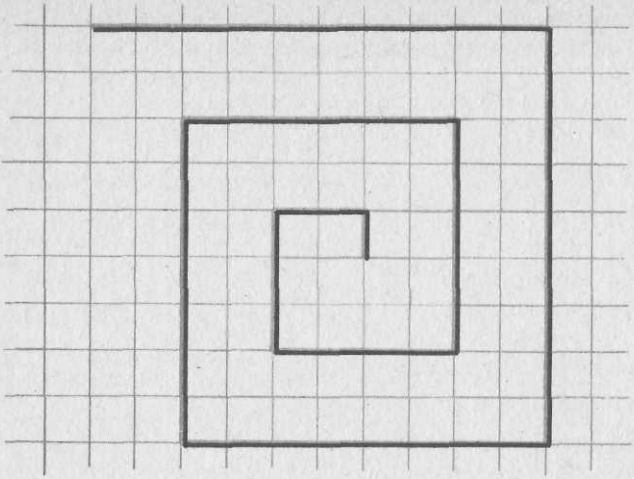


Рис. 18

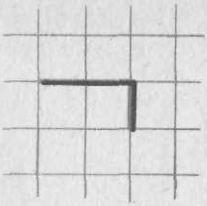


Рис. 19

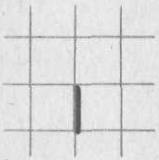


Рис. 20

Во всех предыдущих примерах команда повторения с параметром использовалась для работы с таблицами. Но ее удобно использовать и в других случаях. Например, с ее помощью можно рисовать картинки.

Пример 6. Составить алгоритм рисования спирали.

```
алг спираль 1 (цел n)
    арг n
нач цел i
|   для i от 1 до n
|   нц
|   |  вперед (i); налево (90)
|   кц
кон
```

При исполнении этого алгоритма с $n = 10$ появится такая же картинка, как на рисунке 18.

Интересно посмотреть, как будет выполняться этот алгоритм при малых значениях n. При $n = 2$ будет нарисована ломаная из двух звеньев (рис. 19), при $n = 1$ нарисован отрезок (рис. 20).

Что же получится, если взять $n = 0$? При этом будет выполняться команда:

```
для i от 1 до 0
нц
|   вперед (i); налево (90)
кц
```

Оказывается, что не будет нарисовано ничего. Дело в том, что в алгоритмическом языке принято такое соглашение: в команде вида

```
для x от X_min до X_max
нц
| серия
кц
```

при $X_{min} > X_{max}$ серия команд не выполняется ни разу. Поэтому алгоритм рисования спирали при $n = 0$ или при $n < 0$ ничего не нарисует.

Пример 7. Записать алгоритм рисования правильного n-угольника со стороной 1.

```
алг n-угольник (цел n)
    арг n
нач
|   цел i
|   для i от 1 до n
|   нц
|   |   вперед (1)
|   |   налево (360/n)
|   кц
кон
```

При исполнении этого алгоритма исполнитель n раз повторяет такие действия:

нарисовать отрезок длины 1,

повернуться налево на угол в $\frac{360}{n}$ градусов.

Обратите внимание, что в этом алгоритме серия команд

```
вперед (1)
налево (360/n),
```

входящая в команду повторения с параметром i, не использует значение этого параметра. Он используется только как счетчик числа повторений.

В разобранных примерах при выполнении команды повторения с параметром значение параметра всякий раз увеличивается на 1 (говорят: шаг изменения параметра равен 1). Алгоритмический язык позволяет задавать в команде повторения произвольный целочисленный положительный шаг изменения параметра. В общем виде команда повторения с параметром выглядит так:

```
для x от X_min до X_max шаг x_шаг
нц
| серия
кц
```

Выполнение этой команды происходит так же, как и команды на с. 39; разница состоит в том, что после каждого повторения значение параметра x увеличивается не на 1, а на значение целочисленного выражения $x_{шаг}$.

Пример 8. Записать алгоритм вычисления суммы всех двузначных нечетных чисел.

```
алг сумма нечетных (цел S)
    рез S
нач
|   цел i
|   S:=0
|   для i от 11 до 99 шаг 2
|   нц
|   | S:=S+i
|   кц
кон
```

Пример 9. Составить алгоритм заполнения таблицы, содержащей число дней для 1901, ..., 1999 годов. Напомним, что каждый четвертый из них, начиная с 1904 года, високосный и содержит на один день больше, чем обычно, а все остальные содержат 365 дней.

```
алг заполнение таблицы (цел таб число дней [1901:1999])
    рез число дней
нач
|   цел i
|   для i от 1901 до 1999
|   нц
|   | число дней [i]:=365
|   кц
|   для i от 1904 до 1999 шаг 4
|   нц
|   | число дней [i]:=число дней [i]+1
|   кц
кон
```

Первая команда повторения заполняет таблицу без учета високосных лет. Она выполняется для

$$i = 1901, 1902, ..., 1999.$$

Вторая вносит поправку для високосных лет и выполняется для

$$i = 1904, 1908, ..., 1996.$$

Значение параметра i, равное $1996 + 4 = 2000$, оказывается бо́льшим указанной в команде границы 1999, и для него серия команд между **нц** и **кц** (в данном случае — единственная команда) не выполняется.

Команда повторения с параметром, не содержащая явного указания шага, эквивалентна команде повторения с шагом 1.

Обратите внимание, что во всех приведенных выше примерах команды серии, входящей в команду повторения с параметром, не меняют значений параметра, границ или шага.

Вопросы

1. Какова форма записи команды повторения с параметром?
2. Приведите примеры применения команды повторения с параметром.
3. Может ли команда повторения с параметром содержать внутри себя команду выбора или другую команду повторения с параметром? Приведите примеры.
4. Приведите пример применения команды повторения с параметром и шагом, не равным 1.

Упражнения

1. Используя команду повторения с параметром, составьте алгоритм, который прибавляет по 1 ко всем элементам таблицы a. Его заголовок должен иметь вид:

алг прибавление 1 (**цел таб** a [1 : 100])
арг a
рез a

2. Составьте алгоритм заполнения таблицы умножения чисел от 1 до 99 двумя способами: с использованием и без использования команды повторения с параметром. Заголовок должен иметь вид:

алг таблица умножения (**цел таб** *произведение* [1:99, 1:99])
рез *произведение*

Какой из двух способов вам кажется более удобным?

3. Какие изменения нужно сделать в алгоритмах «копирование таблицы» и «обращение» (с. 36, 37), чтобы фигурирующие в них таблицы из 100 элементов заменить таблицами из 200 элементов?

4. Рассмотрим следующий алгоритм:

```
алг обработка 1 (цел таб a[1:10])
    арг a
    рез a
нач
|   цел i
|   для i от 1 до 10
|   нц
|   |   a[i] := a[11 − i]
|   кц
кон
```

Каковы будут результаты исполнения этого алгоритма для величины a, значения которой приведены в таблице 15?

Таблица 15

i	1	2	3	4	5	6	7	8	9	10
$a[i]$	11	12	13	14	15	16	17	18	19	20

5. Выполните предыдущее упражнение для алгоритмов

а)
```
алг обработка 2 (цел таб a[1:10])
    арг a
    рез a
нач
|   цел i, A
|   для i от 1 до 10
|   нц
|   |   A := a[i]
|   |   a[i] := a[11 − i]
|   |   a[11 − i] := A
|   кц
кон
```

б) **алг** обработка 3 (**цел таб** *a*[1:10])

арг *a*

рез *a*

нач

цел *i*, *A*

для *i* **от** 1 **до** 5

нц

$A := a[i]$

$a[i] := a[11-i]$

$a[11-i] := A$

кц

кон

6. Что нарисует исполнитель алгоритма «*n*-угольник» (пример 7), если

а) $n = 3$; б) $n = 2$; в) $n = 1$; г) $n = 0$; д) $n = -1$?

7. Что нарисует исполнитель алгоритма «спираль» при $n = 10$, если заменить в нем команду "налево (90)" командой "направо (90)"?

8. Исходными данными являются целочисленная таблица *температура* [1 : 31], в которой записана температура за каждый день января, и величина *S*, равная средней январской температуре за последнее столетие. Составьте алгоритм, подсчитывающий, сколько в январе было дней с температурой, большей, меньшей и равной средней. Его заголовок должен иметь вид:

алг подсчет (**цел таб** *температура* [1 : 31], **цел** *S*, **цел** *бол, мен, рав*)

арг *температура, S*

рез *бол, мен, рав*

9. Составьте алгоритм заполнения таблицы числа дней от 1801 до 1899 года. (Високосные годы — каждый четвертый, начиная с 1804 года.)

10. Составьте алгоритм заполнения таблицы числа дней от 1801 до 2000 года. Учтите, что 1900 год не високосный, а 2000 год високосный.

11. Составьте алгоритм вычисления числа воскресений в 1991 году. (Первое воскресенье 1991 года — 6 января.)

12. Составьте алгоритм заполнения таблицы числа дней для всех лет с 1-го до 2000-го года. Указание: високосными являются годы, номера которых делятся на 4, за исключением тех, номера которых делятся на 100, но не делятся на 400.

13. Составьте алгоритм нахождения: а) минимального; б) максимального элемента таблицы *a*[1 : 100].

14. Составьте алгоритм для рисования пятиконечной звезды.

15. Составьте алгоритм вычисления суммы квадратов всех натуральных чисел от 1 до 50: $S = 1^2 + 2^2 + 3^2 + ... + 50^2$.

16. Составьте алгоритм вычисления первых 30 чисел Фибоначчи. (Эти числа определяются так: $a_1 = 1$, $a_2 = 1$, а каждое следующее число равно сумме двух предыдущих: $a_{n+1} = a_n + a_{n-1}$. Вот первые несколько чисел Фибоначчи: 1, 1, 2, 3, 5, 8, 13, 21, ... (§ 6, ч. I).) Ответ получите в виде таблицы a [1:30].

17. Что нарисует исполнитель алгоритма «n-угольник» (пример 7), если в нем угол $360/n$ градусов заменить вдвое большим углом? Рассмотрите случаи $n = 3, 4, 5, 6, 7, 8$.

18. Составьте алгоритм циклической перестановки элементов таблицы a [1 : 1000], при которой $a[i]$ перемещается в $a[i + 1]$, а a [1000] перемещается в a [1].

19. Составьте алгоритм циклической перестановки элементов таблицы a [1 : 1000] на заданное число k шагов, так что элемент $a[i]$ перемещается в $a[i + k]$, а последние k элементов таблицы, которым «не хватило места», перемещаются в «освободившиеся» первые k элементов таблицы.

20. Дана таблица a [1 : 500]. Напишите алгоритм, записывающий в качестве i-го элемента этой таблицы сумму чисел от a [1] до $a[i]$ включительно.

§ 8. ВСПОМОГАТЕЛЬНЫЕ АЛГОРИТМЫ ВЫЧИСЛЕНИЯ ЗНАЧЕНИЙ ФУНКЦИЙ

В первой части курса (§ 4) мы познакомились с возможностью использовать при построении одного алгоритма другой алгоритм, который назывался вспомогательным. Часто вспомогательные алгоритмы используются для вычисления значений некоторых функций. В этих случаях алгоритмический язык допускает более короткий и наглядный (чем уже знакомый нам) способ написания программ. Поясним сказанное на примерах.

Пример 1. В библиотеке алгоритмов приведен алгоритм вычисления модуля вещественного числа (ч. I, с. 90).

```
алг МОД (вещ x, вещ y)
    арг x
    рез y
нач
|   если x ⩾ 0
|       то y := x
|       иначе y := −x
|   все
кон
```

Этот алгоритм можно использовать, например, при вычислении значений функции $y = |2x + 1| - |3x + 7|$.

```
алг вычисление 1 (вещ x, вещ y)
    арг x
    рез y
нач
|   вещ a, b
|   МОД (2*x + 1, a)
|   МОД (3*x + 7, b)
|   y := a - b
кон
```

При исполнении этого алгоритма мы дважды обращаемся к вспомогательному алгоритму МОД. После его первого применения a становится равным $|2x + 1|$, после второго b становится равным $|3x + 7|$. После этого команда

$$y := a - b$$

дает нужное значение переменной y.

Этот же алгоритм можно записать иначе:

```
алг вычисление 2 (вещ x, вещ y)
    арг x
    рез y
нач
|   y := abs (2*x + 1) - abs (3*x + 7)
кон
```

Чтобы исполнитель мог исполнить этот алгоритм, ему нужно сообщить алгоритм вычисления модуля, т. е. абсолютной величины abs (m) заданного числа m. Это делается в такой форме:

```
алг вещ abs (вещ x)
нач
|   если x ⩾ 0
|       то знач := x
|       иначе знач := -x
|   все
кон
```

Первая строка этой записи является заголовком. Записывая заголовок вспомогательного алгоритма вычисления функции, мы

указываем тип значений функции (в данном случае **вещ**), название функции (в данном случае abs), затем в скобках список ее аргументов с указанием типа (в данном случае функция имеет единственный вещественный аргумент x). Больше ничего указывать в заголовке не требуется — и так ясно, что все перечисленные в скобках величины являются аргументами, а результатом является значение функции, обозначаемое сокращенно служебным словом **знач**. Эта величина используется как переменная, тип которой совпадает с типом значений функции. Указывать тип этой величины не следует.

За заголовком следует алгоритм вычисления значения $\mathrm{abs}(x)$ при заданном x. Он гласит, что при неотрицательном x значение вычисляемой функции равно x, а в противном случае (при отрицательном x) значение равно $-x$.

Пример 2. Рассмотрим алгоритм нахождения большего из двух чисел (ч. I, с. 90).

```
алг БИД (вещ a, b, вещ y)
    арг a, b
    рез y
нач
|   если a ⩾ b
|       то y := a
|       иначе y := b
|   все
кон
```

Используя этот алгоритм как вспомогательный, мы записывали алгоритм нахождения большего из трех чисел так:

```
алг БИТ (вещ a, b, c, вещ y)
    арг a, b, c
    рез y
нач вещ z
|   БИД (a, b, z)
|   БИД (z, c, y)
кон
```

В этом алгоритме используется промежуточная величина z, значение которой равно большему из чисел a и b. Алгоритм БИТ

основан на таком свойстве: если через $\max 2(a, b)$ обозначить большее из чисел a, b, а через $\max 3(a, b, c)$ — большее из трех чисел a, b, и c, то

$$\max 3(a, b, c) = \max 2(\max 2(a, b), c).$$

При новом способе записи вспомогательных алгоритмов алгоритм БИД приобретает такой вид:

```
алг вещ max 2(вещ a, b)
нач
| если a ⩾ b
| | то знач := a
| | иначе знач := b
| все
кон
```

а использующий его алгоритм нахождения БИТ — такой:

```
алг вещ max 3(вещ a, b, c)
нач
| знач := max 2(max 2(a, b), c)
кон
```

Приведем еще несколько примеров вспомогательных алгоритмов вычисления функций.

В следующем примере аргумент функции и результат разных типов.

Пример 3. Записать алгоритм вычисления значений функции $y = \text{sign}(x)$, определенной следующим образом:

$$\text{sign}(x) = \begin{cases} -1, & \text{если } x < 0, \\ 0, & \text{если } x = 0, \\ 1, & \text{если } x > 0. \end{cases}$$

График этой функции изображен на рисунке 21.

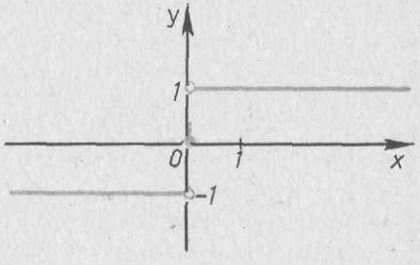


Рис. 21

Заголовок алгоритма вычисления значений функции $y = \text{sign}(x)$ запишется так:

```
алг цел sign (вещ x)
```

В нем указано, что аргументом функции $\text{sign}(x)$ должно быть вещественное число, а значением будет число целое. Алгоритм вычисления использует

команду выбора и выглядит так:

```
алг цел sign (вещ x)
нач
| выбор
|       при x<0: знач := -1
|       при x=0: знач := 0
|       при x>0: знач := 1
| все
кон
```

Включив алгоритм вычисления значений какой-то функции в библиотеку вспомогательных алгоритмов, мы получаем возможность использовать эту функцию в других алгоритмах.

Для единообразия будем записывать значения всех функций с использованием скобок и писать, например, $\sin(x)$ вместо обычного $\sin x$.

При таком способе записи со скобками известная формула для синуса суммы запишется следующим образом:

$$\sin(x + y) = \sin(x)*\cos(y) + \cos(x)*\sin(y),$$

а теорема косинусов, связывающая стороны a, b, c треугольника с углом C, противолежащим стороне c, будет иметь вид:

$$c**2 = a**2 + b**2 - 2*a*b*\cos(C).$$

Здесь операция возведения в степень обозначена, как это принято во многих языках программирования, двумя звездочками, т. е. вместо a^2 пишем $a**2$, вместо 10^x пишем $10**x$ и т. д.

Аргументом функции может быть не только число (или несколько чисел), но и таблица. В следующем примере аргументом является линейная таблица $a[1:100]$ из 100 элементов.

Пример 4. Записать алгоритм вычисления среднего арифметического всех элементов таблицы $a[1:100]$.

```
алг вещ среднее (вещ таб a[1:100])
нач
| вещ S, цел i
| S := 0
| для i от 1 до 100
| нц
|   S := S + a[i]
| кц
| знач := S/100
кон
```

Пример 5. Записать алгоритм вычисления значений функции факториал (n) натурального аргумента n, определяемой как произведение всех натуральных чисел от 1 до n:

$$\text{факториал } (n) = 1*2*...*n.$$

В математике эта функция обозначается $n!$ Алгоритм вычисления значений функции факториал (n) таков:

```
алг цел факториал (нат n)
нач
| цел i
| знач := 1
| для i от 1 до n
| нц
| | знач := знач * i
| кц
кон
```

Найдем теперь сумму факториалов всех целых чисел от 1 до 10, используя алгоритм вычисления факториала как вспомогательный.

```
алг сумма факториалов (цел S)
    рез S
нач цел i
| S := 0
| для i от 1 до 10
| нц
| | S := S + факториал (i)
| кц
кон
```

Существуют так называемые *рекурсивные* алгоритмы вычисления значений функций. Приведем два примера таких алгоритмов.

Иногда значение функции для некоторого аргумента можно выразить через значение этой же функции от другого аргумента, например $\sin(x + \pi) = -\sin(x)$ для всех вещественных x.

В некоторых случаях такие соотношения можно применить для вычисления функций.

Пример 6. Определение функции факториал можно записать в виде следующего соотношения:

$$\text{факториал } (n) = \begin{cases} 1, & \text{если } n = 1, \\ \text{факториал } (n-1)*n, & \text{если } n > 1. \end{cases}$$

Покажем, как, используя это соотношение, можно вычислить, например, факториал (4).

Подставляя $n = 4$, находим, что

факториал (4) = факториал (3) $*$ 4.

Теперь используем наше соотношение при $n = 3$ и подставим полученное выражение для факториал (3). Получается

факториал (4) = факториал (2) $*3*4$.

Наконец, применим соотношение при $n = 2$ и получаем

факториал(4) = факториал (1)$*2*3*4$.

Сделав еще одну подстановку, получаем, наконец, что

факториал (4) = $1*2*3*4 = 24$.

Пример 7. Вычислим значение функции квадрат $(m) = m{**}2$ для натурального m, используя лишь операции сложения и вычитания. Вспомним формулу квадрата суммы и запишем ее для $(m + 1){**}2$:

$$(m + 1){**}2 = m{**}2 + 2{*}m + 1,$$

т. е.

$$\text{квадрат}\ (m + 1) = \text{квадрат}\ (m) + 2{*}m + 1.$$

Перепишем теперь эту формулу, взяв вместо m выражение $m - 1$:

$$\text{квадрат}\ (m) = \text{квадрат}\ (m - 1) + 2{*}(m - 1) + 1 = \text{квадрат}\ (m - 1) + m + m - 1$$

(записываем $2{*}m$ как $m + m$, избегая умножений).

Добавив к этому равенству правило для вычисления значения квадрат (1), получим следующее соотношение:

$$\text{квадрат}\ (m) = \begin{cases} 1, & \text{если } m = 1, \\ \text{квадрат}\ (m - 1) + m + m - 1, & \text{если } m > 1. \end{cases}$$

Это соотношение, как и предыдущее (для факториалов), связывает значения искомой функции от некоторого натурального аргумента со значением этой же функции от предыдущего по величине аргумента, а также задает прямое правило определения значения функции для минимального значения аргумента. Такие соотношения называют *рекуррентными*.

Правила алгоритмического языка позволяют составить алгоритмы вычисления значений функций на основе рекуррентных соотношений. Соответствующие алгоритмы имеют следующий вид:

```
алг нат факториал (нат n)
нач
| если n = 1
| | то знач := 1
| | иначе знач := факториал (n - 1)*n
| все
кон
```

```
алг нат квадрат (нат m)
нач
|   если m = 1
|   |   то знач := 1
|   |   иначе знач := квадрат (m − 1) + m + m − 1
|   все
кон
```

В этих алгоритмах значения функций выражаются через значения тех же функций от других аргументов. Такие алгоритмы называют рекурсивными.

Способ исполнения рекурсивных алгоритмов остается тем же самым, что и у обычных алгоритмов. Нужно лишь при каждом новом обращении к вычислению значений функции составлять новый экземпляр таблицы значений.

Вопросы

1. Как используются вспомогательные алгоритмы вычисления значений функций? Приведите примеры.
2. Как составляется заголовок алгоритма вычисления значений функции?
3. Для чего используется величина **знач** в алгоритме вычисления значений функции? Каков тип этой величины?

Упражнения

1. Напишите алгоритм вычисления значения выражения

$$y = |x + 1| + |x - 1|,$$

используя в качестве вспомогательного алгоритм вычисления значений функции abs(x).

2. Напишите алгоритм вычисления значений функции $y =$ $=$ max4(a, b, c, d), значение которой равно большему из четырех ее аргументов.

3. Напишите алгоритм

```
алг вещ больший (вещ таб a[1 : 100])
```

вычисляющий значения функции больший (a), значением которой является наибольший из элементов таблицы a.

4. Напишите алгоритм

```
алг вещ сумма (вещ таб a[1 : 100])
```

вычисляющий сумму всех элементов таблицы a.

5. Используя алгоритм вычисления функции факториал (n) в качестве вспомогательного, напишите алгоритм вычисления суммы факториалов всех четных чисел от 2 до 100.

6. Напишите алгоритм нахождения приближенного значения функции $y = \sin x$ при малых значениях x по формуле

$$y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}.$$

7. Напишите алгоритм, находящий: а) последнюю цифру десятичной записи натурального числа x; б) предпоследнюю цифру десятичной записи натурального числа x. При этом можно использовать алгоритм «остаток», имеющийся в библиотеке алгоритмов.

8. Напишите алгоритм, находящий сумму цифр десятичной записи трехзначного натурального числа x.

9. Напишите алгоритм, находящий сумму трехзначного числа N с трехзначным числом $\overline{N}$, полученным из первого перестановкой его цифр в обратном порядке. (Например, если $N = 123$, то $\overline{N} = 321$, а искомая сумма равна 444.)

10. Напишите алгоритм подсчета количества «счастливых» автобусных билетов. (Билет называется счастливым, если сумма первых трех цифр номера билета равна сумме последних трех цифр.)

§9. АЛГОРИТМЫ РАБОТЫ С ЛИТЕРНЫМИ ВЕЛИЧИНАМИ

Мы уже рассматривали величины, значениями которых являются тексты: "нет решения", "ракета станет спутником Солнца", "удовлетворительно". Такие величины назывались *литерными*. Говоря о текстах, мы имеем в виду произвольные последовательности символов (необязательно имеющие смысл). Например, значением литерной величины может быть текст

"абракадабра"

или текст

"щклмнтуеь12395+".

До сих пор у нас не было никаких операций, которые выполнялись бы над литерными величинами. Сейчас мы введем несколько таких операций.

Операция *соединения*, или склеивания, обозначается знаком + и соединяет два текста в один. Например,

"ин" + "форматика" = "информатика",

а

"12" + "345" = "12345".

Обратите внимание, что результат соединения текстов зависит от их порядка:

$$"345" + "12" = "34512".$$

Длиной текста называется количество букв в нем. Длина текста A обозначается **длин**(A).

Например,

длин ("ин") = 2,

длин ("форматика") = 9,

длин ("информатика") = 11.

Существует текст длины 0, не содержащий ни одной буквы. Он обозначается двумя стоящими рядом кавычками "" и называется пустым текстом.

Очевидно,

$$\textbf{длин}\ (A + B) = \textbf{длин}\ (A) + \textbf{длин}\ (B).$$

Будем считать, что буквы в тексте пронумерованы слева направо, начиная с единицы (табл. 16).

Таблица 16

1	2	3	4	5	6	7	8	9	10	11
и	н	ф	о	р	м	а	т	и	к	а

Еще одна операция над текстами, которую мы будем рассматривать, называется *вырезкой*. С ее помощью можно «вырезать» из текста фрагмент, указав номера первой и последней буквы. Например, если A = "информатика", то

$A\ [3:7]$ = "форма", а

$A\ [8:10]$ = "тик".

Фрагмент может содержать всего одну букву:

$A\ [7:7]$ = "а".

Комбинируя операции вырезки и склеивания, можно получать из одних текстов другие: если A = "информатика", то, например,

$$A\ [8:8] + A\ [4:5] + A\ [8:8] = "торт".$$

Введенные операции используются в алгоритмах работы с литерными величинами.

Приведем примеры таких алгоритмов.

Пример 1. Написать алгоритм подсчета числа букв "а" в тексте.

```
алг цел число букв а (лит x)
нач
  цел i
  знач := 0
  для i от 1 до длин (x)
  нц
    если x[i:i] = "а"
      то знач := знач + 1
    все
  кц
кон
```

Пример 2. Написать алгоритм обращения слова.

Поясним смысл этого алгоритма. Если, например, его аргументом является слово "кот", то результат — слово "ток", если аргумент — "акитамрофни", то результат — "информатика" и т. д.

Этот алгоритм может быть записан таким образом:

```
алг лит обращение (лит x)
нач
  цел i
  знач := ""
  для i от 1 до длин (x)
  нц
    знач := x [i:i] + знач
  кц
кон
```

При исполнении этого алгоритма значением величины **знач** становится вначале пустой текст, не содержащий ни одной буквы и обозначаемый "", а затем к его началу приписываются слева поочередно все буквы слова x.

Значение литерной величины (как и величины любого другого типа) может быть изменено командой присваивания. При этом старое значение этой величины «теряется». Мы будем рассматривать также команды частичного изменения значения литерной величины, позволяющие заменить часть слова.

Например, при A = "квант" команда

$$A\ [4:5] := \text{"рк"}$$

заменит часть слова "квант" с 4-й по 5-ю букву на "рк". После выполнения этой команды значением литерной величины A станет

слово "кварк". При этом важно, чтобы новая часть слова имела ту же длину, что и старая. Например, команда

$$A\ [7:9] := \text{"абвгдейка"}$$

не имеет смысла, так как часть слова с 7-й по 9-ю букву содержит ровно 3 буквы.

Приведем пример алгоритма, заменяющего в данном слове все буквы "а" на буквы "б".

```
алг замена а на б (лит x)
    арг x
    рез x
нач
|   цел i
|   для i от 1 до длин (x)
|   нц
|   |   если x [i:i] = "а"
|   |   |   то x [i : i] := "б"
|   |   все
|   кц
кон
```

Перебирая все буквы слова от начала к концу, этот алгоритм проверяет, не является ли i-я буква слова x буквой "а". Если это так, то команда

$$x\ [i : i] := \text{"б"}$$

заменяет эту букву буквой "б". Например, применение этого алгоритма к аргументу "абба" дает результат "бббб".

Мы познакомились с основными конструкциями алгоритмического языка в русском варианте (Русского Алгоритмического Языка — РАЯ). В разных республиках нашей страны и в других странах используемые в нем слова могут изменяться, но содержание основных конструкций будет одним и тем же. Конструкции алгоритмического языка служат основой языков программирования, применяемых при решении различных возникающих из практики задач на больших и малых ЭВМ. И хотя различные языки программирования имеют индивидуальные особенности, связанные с областью их применения, историей развития и т. д., основные конструкции этих языков являются «переводом» конструкций нашего алгоритмического языка.

Эти конструкции образуют как бы «общее ядро» различных языков программирования, в том числе и тех, с которыми мы познакомимся в последующих параграфах.

Вопросы

1. Как применяется операция соединения к литерным величинам? Приведите примеры.

2. Как применяется операция вырезки к литерным величинам? Приведите примеры.

3. Как применяется команда частичного изменения значения литерной величины? Приведите примеры.

Упражнения

1. Пусть A = "компьютеризация". Найдите значения:

а) $A\,[4:4] + A\,[2:2] + A\,[7:9] + A\,[15:15]$;
б) $A\,[4:4] + A\,[9:11]$.

2. Составьте алгоритм подсчета суммарного числа букв "а" и букв "б" в данной литерной величине.

3. Составьте алгоритм замены в какой-нибудь литерной величине всех букв "а" на буквы "б" и наоборот (при такой замене, например, из слова "баба" должно получиться слово "абаб").

4. Составьте алгоритм, удваивающий каждую букву в заданном тексте (при этом, например, из слова "бейсик" должно получиться слово "ббееййссиикк").

5. Составьте алгоритм, выясняющий, является ли данное слово «перевертышем» (так называются слова, читающиеся одинаково слева направо и справа налево, например: ПОТОП, ҚАЗАҚ).

6. Составьте алгоритм, вычеркивающий из данного слова все буквы "а" (так, чтобы, например, из слова "заноза" получалось слово "зноз").

7. Составьте алгоритм, который каждую встреченную в слове букву "б" заменял бы сочетанием букв "ку" (так, чтобы, например, из слова "абракадабра" получалось слово "акуракадакура").

8. С помощью операций вырезки и склеивания составьте из частей слова "интеграл" слова "гантели", "рентген", "тигр", "агент".

9. Напишите алгоритм, проверяющий, является ли частью данного слова слово "сок". Ответ должен быть "да" или "нет". Например, для слов "сокол" и "кусок" ответ "да", а для слова "кокос" — "нет".

10. Напишите алгоритм, посчитывающий, сколько раз в данном слове встречается сочетание "со" (например, в слове "согласование" оно встречается 2 раза).

11. Напишите алгоритм, подсчитывающий, сколько раз в данном слове x встречается (в качестве его части) данное слово y. Если слово y длиннее, чем x, то результат должен быть равен нулю. Примеры: если x = "аттестат", y = "т", результатом должно быть число 4; если x = "абракадабра", y = "бра" — результат равен 2; если x = "забой", y = "еж" — результат равен 0; если x = "аааааа", y = "ааа", то результат равен 3.

12. В примере 2 поменяйте местами «слагаемые» в команде присваивания, т. е. команду

$$\underline{\textbf{знач}} := x[i:i] + \underline{\textbf{знач}}$$

замените командой

$$\underline{\textbf{знач}} := \underline{\textbf{знач}} + x[i:i].$$

а) Пусть аргументом этого алгоритма является слово "школа". Что будет результатом?

б) Что будет результатом применения этого алгоритма к произвольному слову?

ЯЗЫК ПРОГРАММИРОВАНИЯ РАПИРА

В следующих параграфах мы перейдем от изучения алгоритмического языка к знакомству с *языками программирования*, т. е. с правилами записи программ, исполняемых на ЭВМ. И хотя алгоритмы, записанные на алгоритмическом языке, при некотором их уточнении, тоже могут исполняться на ЭВМ, все же главным назначением алгоритмического языка является его употребление человеком (независимо от какой бы то ни было машины): для составления, записи и изучения алгоритмов решения задач из самых разнообразных областей человеческой деятельности. При решении задачи на ЭВМ запись алгоритма на алгоритмическом языке является исходной информацией для реализации этого алгоритма на ЭВМ с помощью ее конкретного (рабочего) языка программирования.

Язык программирования Рапира разработан в начале 80-х годов в СССР для использования в школьном курсе информатики. Школьные компьютеры снабжены программами, позволяющими им исполнять алгоритмы, записанные на этом языке. Вместе с тем этот язык имеет широкие возможности, позволяющие составлять сложные производственные программы. Здесь описаны только некоторые простейшие элементы языка Рапира. Его конструкции близки к соответствующим конструкциям алгоритмического языка. Поэтому в большинстве случаев мы будем вводить их по аналогии с известными элементами алгоритмического языка.

§ 10. ЗАПИСЬ АЛГОРИТМОВ В ВИДЕ ПРОЦЕДУР НА РАПИРЕ

Чтобы познакомиться с языком Рапира, рассмотрим какой-нибудь известный нам из первой части учебника алгоритм (например, алгоритм решения квадратного уравнения) и сравним его записи на алгоритмическом языке и на Рапире. На алгоритмическом языке он запишется так:

алг решение квадратного уравнения (**вещ** a, b, c, x_1, x_2, **лит** y)

арг a, b, c

рез x_1, x_2, y

нач **вещ** D

$D := b^2 - 4 \cdot a \cdot c$

если $D < 0$

то $y :=$ "нет решения"

иначе $y :=$ "есть решения"

$x_1 := \frac{-b + \sqrt{D}}{2a}$; $x_2 := \frac{-b - \sqrt{D}}{2a}$

все

кон

Теперь запишем этот алгоритм на языке Рапира:

```
ПРОЦ РЕШЕНИЕ_КВАДРАТНОГО_УРАВНЕНИЯ
( => A, => B, => C, X1 => , X2 => , Y => )
ИМЕНА: D
НАЧ
    D := B**2 - 4*A*C
    ЕСЛИ D < 0
        ТО Y := "НЕТ РЕШЕНИЯ"
        ИНАЧЕ
        Y := "ЕСТЬ РЕШЕНИЯ"
        X1 := (-B + SQRT(D))/(2*A)
        X2 := (-B - SQRT(D))/(2*A)
    ВСЕ
КОН
```

Мы видим, что записи этих алгоритмов похожи, но есть и некоторые отличия. Перечислим основные из них.

1. В программах на алгоритмическом языке употреблялись любые буквы (латинские и греческие), математические знаки и т. п. Поскольку на клавиатуре компьютеров есть не все из этих букв, то алфавит языка Рапира ограничен: можно использовать только русские и латинские буквы. Кроме того, переменные с индексами записываются в одну строку: x_1 пишется как X1 и т. п.

2. По тем же причинам возведение в степень обозначается двумя звездочками, умножение — одной, а квадратный корень из числа x записывается как SQRT(X) (сокращение английских слов *square root* — квадратный корень), деление обозначается /.

3. Вместо служебного слова **алг** используется слово ПРОЦ (сокращение от «процедура»). Процедурой называется и сама запись алгоритма на Рапире.

4. В названии алгоритма отдельные слова соединены черточками (символами подчеркивания). Имена в Рапире (названия алгоритмов, переменных и т. п.) должны представлять собой последовательность букв, цифр и символов подчеркивания (_) и не должны содержать пробелов. Такое соглашение помогает компью-

теру не спутать имя "РЕШЕНИЕ_КВАДРАТНОГО_УРАВНЕНИЯ" с написанными подряд тремя именами "РЕШЕНИЕ", "КВАДРАТНОГО" и "УРАВНЕНИЯ".

5. Нет отдельных списков аргументов и результатов (**арг** a, b, c или **рез** x_1, x_2, y). Вместо этого в перечислении величин в заголовке процедуры (их называют *параметрами* процедуры) ставится стрелка =>> (состоящая из двух символов = и >) перед аргументами и после результатов. (Если какая-то величина Z является и аргументом, и результатом алгоритма, то мы стрелку ставим и до, и после: =>> Z =>> .) Указание аргументов и результатов алгоритма с помощью стрелок удобно, так как избавляет от необходимости повторно писать имена параметров. Тем самым уменьшается вероятность ошибки.

6. В заголовке процедуры на Рапире не указаны типы параметров A, B, C, X1, X2 и Y. Это одна из особенностей Рапиры. Дело в том, что в Рапире одна и та же величина может в разные моменты времени принимать значения различных типов. Другими словами, типы имеют не сами величины, а их значения.

Хотя в заголовке процедуры и не указано, что A, B, C должны принимать числовые значения, попытка применить процедуру, например, к литерным значениям величин A, B, C приведет к ошибке: при вычислении значения выражения B**2 — 4*A*C выяснится, что операции возведения в квадрат и умножения неприменимы к литерным значениям.

7. По той же причине запись **вещ** D заменена записью ИМЕНА: D, не конкретизирующей тип промежуточной величины D.

8. Команды повторения и выбора пишутся на Рапире так же, как и на алгоритмическом языке, только с использованием заглавных букв при записи служебных слов.

Вопросы

1. Приведите примеры различий между алгоритмическим языком и языком Рапира. Чем объясняются эти различия?

2. Как указываются аргументы и результаты алгоритма в записи на алгоритмическом языке и на Рапире?

Упражнения

1. Запишите выражение

$$b^3 + \sqrt{c} - \sqrt{a}$$

на языке Рапира.

2. Заголовок алгоритма, записанного на Рапире:

ПРОЦ ПРИМЕР (=>A, B=>, =>C=>).

Укажите аргументы и результаты этого алгоритма.

3. Арифметическое выражение имеет вид:

$$\frac{1024 : 2^5 + 128 : 4^2}{(16 - 2 \cdot 4) - 2^2}.$$

Запишите на Рапире команду, которая вычисляет значение этого выражения и присваивает его переменной с именем ВЕЛИЧИНА.

4. Запишите на Рапире команду, которая вычисляет величину h по формуле:

$$h = h_0 + \frac{9{,}81 \cdot t^2}{2}.$$

5. Составьте описание процедуры РАССТОЯНИЕ, которая имеет в качестве аргументов координаты двух точек: (X1, Y1), (X2, Y2) и вычисляет расстояние между этими точками.

6. Напишите процедуру с заголовком

ПРОЦ ПРОВЕРКА (=> P, => Q, => R, Z =>), где P, Q, R — коэффициенты квадратного уравнения, а Z — литерная переменная, принимающая одно из трех значений:

"КОРНИ ОДНОГО ЗНАКА",
"КОРНИ ПРОТИВОПОЛОЖНЫХ ЗНАКОВ",
"ВЕЩЕСТВЕННЫХ КОРНЕЙ НЕТ".

Предусмотрите вызов процедуры РЕШЕНИЕ_КВАДРАТНОГО_УРАВНЕНИЯ внутри описания процедуры ПРОВЕРКА.

Процедура ПРОВЕРКА должна вычислять корни и, если они существуют, выяснять совпадение и несовпадение их знаков.

§ 11. ЗАПИСЬ АЛГОРИТМОВ ВЫЧИСЛЕНИЯ ЗНАЧЕНИЙ ФУНКЦИЙ НА РАПИРЕ

Запишем алгоритм нахождения наибольшего общего делителя в виде алгоритма вычисления значений функции НОД (x, y). На алгоритмическом языке этот алгоритм будет выглядеть так:

```
алг нат НОД (нат m, нат n)
нач нат x, y
|   x := m; y := n
|   пока x ≠ y
|   нц
|   |   если x > y
|   |   |   то x := x − y
|   |   |   иначе y := y − x
|   |   все
|   кц
|   знач := x
кон
```

Напомним, что этот алгоритм называется алгоритмом Евклида и основан на том, что

НОД $(x, y) =$ НОД $(x - y, y)$ при $x > y$,
НОД $(x, y) =$ НОД $(x, y - x)$ при $x < y$,
НОД $(x, x) = x$.

На Рапире этот алгоритм будет выглядеть так:

```
ФУНК НОД (M, N)
ИМЕНА: X, Y
НАЧ
    X:=M; Y:=N
    ПОКА X/=Y
    НЦ
        ЕСЛИ X > Y
            ТО X:=X - Y
            ИНАЧЕ Y:=Y - X
        ВСЕ
    КЦ
    ЗНАЧ :=X
КОН
```

Отметим некоторые особенности записи.

1. В этом примере вместо служебного слова **алг** используется не слово ПРОЦ (процедура), а слово ФУНК (функция). Это объясняется следующим. В алгоритмическом языке алгоритмы вычисления значений функций отличаются тем, что после слова **алг** в них указывается тип значений функции (в данном случае **нат**). В Рапире тип значений функции заранее не указывается. Вместо этого алгоритмы вычисления функций начинаются со слова ФУНК, что выделяет их среди остальных.

2. Как и в алгоритмическом языке, аргументы M и N функции НОД не нуждаются в специальном указании, что они являются аргументами, и стрелки перед ними не ставятся.

3. Вместо знака $\neq$ используется комбинация знаков / =. Аналогично вместо знаков $\leqslant$ и $\geqslant$ используются комбинации знаков < = и > =.

Приведем еще один пример записи алгоритма вычисления функции.

```
ФУНК ПЛОЩАДЬ (A, B, C)
ИМЕНА: ПОЛУПЕРИМЕТР
НАЧ
    (*ВЫЧИСЛЯЕТ ПЛОЩАДЬ ТРЕУГОЛЬНИКА СО СТОРОНАМИ A, B, C
    ПО ФОРМУЛЕ ГЕРОНА*)
    ПОЛУПЕРИМЕТР:= (A + B + C)/2
    ЗНАЧ : = SQRT (ПОЛУПЕРИМЕТР*(ПОЛУПЕРИМЕТР — A)
        *(ПОЛУПЕРИМЕТР — B) * (ПОЛУПЕРИМЕТР — C) )
КОН
```

В этой записи текст, заключенный в скобках между звездочками, предназначен для человека, читающего программу. Исполни-

тель же программы (компьютер) пропускает части текста программы, заключенные между «(*» и «*)».

Вопросы

1. Как записываются алгоритмы вычисления значений функций на языке Рапира? Приведите пример.
2. Почему возникает необходимость использовать в языке Рапира два слова — ФУНК и ПРОЦ — вместо одного слова **алг** в алгоритмическом языке?

Упражнения

1. Запишите на языке Рапира алгоритм вычисления функции MAX2(M, N), значение которой равно большему из чисел M и N. Заголовок должен быть таким:

ФУНК MAX2(M, N)

2. Используя алгоритм предыдущего упражнения как вспомогательный, запишите алгоритм вычисления значений функции MAX3(M, N, S), значение которой равно большему из трех чисел M, N и S.

3. Опишите алгоритм вычисления функции, значение которой равно числу букв "А" в заданном тексте СЛОВО.

4. Расширьте возможности алгоритма предыдущего задания так, чтобы функция могла вычислять:

а) число вхождений любой буквы, заданной в качестве аргумента алгоритма, в тексте СЛОВО;

б) процентный состав задаваемой буквы в тексте СЛОВО.

5. Опишите функцию ДЕНЬ_НЕДЕЛИ, которая по аргументу ДЕНЬ — литерной величине, принимающей значение

"ПОНЕДЕЛЬНИК", "ВТОРНИК", ..., "ВОСКРЕСЕНЬЕ",

вычисляет номер дня недели — целое число 1, 2, ..., 7 соответственно.

Воспользуйтесь командой ВЫБОР.

§ 12. КОРТЕЖИ НА РАПИРЕ

Табличные величины называются в Рапире *кортежами*. Элементы кортежа нумеруются всегда с единицы. Правила Рапиры позволяют записывать кортежи в виде списка элементов, разделенных запятыми и заключенных в угловые скобки ⟨ и ⟩. Например, после выполнения команды

A := ⟨5, 4, 3, 2, 1⟩

значением переменной величины A станет такой кортеж (таблица) из пяти элементов:

$$A[1] = 5,$$
$$A[2] = 4,$$
$$A[3] = 3,$$
$$A[4] = 2,$$
$$A[5] = 1.$$

Как и в алгоритмическом языке, через A[N] в Рапире обозначается N-й элемент кортежа (таблицы) A.

Длина кортежа X обозначается ДЛИН(X) (так же как и длина текста). Используя введенные обозначения, составим алгоритм суммирования всех элементов кортежа:

```
ФУНК СУММА(X) (*СУММА ВСЕХ ЭЛЕМЕНТОВ КОРТЕЖА*)
ИМЕНА: К
НАЧ
    ЗНАЧ :=0
    ДЛЯ К ОТ 1 ДО ДЛИН(X)
    НЦ
        ЗНАЧ := ЗНАЧ + X [K]
    КЦ
КОН
```

Элементами кортежа могут быть не только числа. Например, выполнив команду

ИМЯ_МЕСЯЦА:= ⟨"ЯНВАРЬ", "ФЕВРАЛЬ", "МАРТ", "АПРЕЛЬ", "МАЙ","ИЮНЬ","ИЮЛЬ","АВГУСТ","СЕНТЯБРЬ","ОКТЯБРЬ","НОЯБРЬ", "ДЕКАБРЬ"⟩, мы получим кортеж ИМЯ_МЕСЯЦА, для которого ДЛИН(ИМЯ_МЕСЯЦА) = 12, причем ИМЯ_МЕСЯЦА[К] будет названием К-го месяца года, например:

ИМЯ_МЕСЯЦА[9] = "СЕНТЯБРЬ"

Элементами кортежа могут быть и другие кортежи. Например, после выполнения команды

$$A := \langle \langle 1, 2\rangle, \langle 4, 4\rangle \rangle$$

значением A[1] является кортеж ⟨1, 2⟩, а значением A[2] — кортеж ⟨4, 4⟩. Тем самым A[1][1] означает первый элемент кортежа A[1] (т. е. кортежа ⟨1, 2⟩) и, следовательно, A[1][1] = 1. Аналогичным образом

$$A\ [1]\ [2] = 2, \quad A\ [2]\ [1] = 4, \quad A\ [2]\ [2] = 4.$$

Такие «кортежи кортежей» заменяют в Рапире прямоугольные таблицы, имеющиеся в алгоритмическом языке. Приведенный пример кортежа ⟨⟨1, 2⟩, ⟨4, 4⟩⟩ соответствует таблице

1	2
4	4

Таблица a [1 : m, 1 : n] изображается в Рапире в виде кортежа из m элементов, каждый из которых является кортежом из n элементов. Подчеркивая сходство «кортежа кортежей» с прямоугольными таблицами, в Рапире разрешают наряду с A[I] [J] писать A[I, J].

Применение кортежей проиллюстрируем на примере. Пусть у нас имеется список класса с указанием оценок по истории, по математике и по информатике за IX класс. Запишем сведения о каждом ученике в виде кортежа:

⟨ФАМИЛИЯ, ОЦЕНКА_ПО_ИСТОРИИ, ОЦЕНКА_ПО_МАТЕМАТИКЕ, ОЦЕНКА_ПО_ИНФОРМАТИКЕ⟩.

Например, кортеж

⟨"СОРОКИН", 4, 5, 4⟩

содержит информацию о том, что ученик по фамилии Сорокин имеет четверку по истории, пятерку по математике и четверку по информатике. А весь список запишется в виде кортежа, элементы которого — кортежи, содержащие сведения о каждом из учеников.

Например, список (табл. 17)

Таблица 17

Фамилия	Оценки по		
	истории	математике	информатике
Алексеева	5	5	4
Вербицкий	3	4	3
Иванов	5	3	3
Кириллов	4	4	4
Макаров	5	5	5
Полякова	5	5	5
Сорокин	4	5	4

будет представлен в виде кортежа:

⟨ ⟨"Алексеева", 5, 5, 4⟩,
⟨"Вербицкий", 3, 4, 3⟩,
⟨"Иванов", 5, 3, 3⟩,
⟨"Кириллов", 4, 4, 4⟩,
⟨"Макаров", 5, 5, 5⟩,
⟨"Полякова", 5, 5, 5⟩,
⟨"Сорокин", 4, 5, 4⟩ ⟩.

По такому списку можно выполнять разнообразные подсчеты. Например, можно подсчитать число учеников, имеющих оценки «5» по всем трем предметам.

```
ПРОЦ КРУГЛЫЕ_ОТЛИЧНИКИ( => СПИСОК, ЧИСЛО => )
ИМЕНА : НОМЕР
НАЧ
    ЧИСЛО := 0
    ДЛЯ НОМЕР ОТ 1 ДО ДЛИН(СПИСОК)
    НЦ
        ЕСЛИ СПИСОК[НОМЕР] [2] = 5 И
              СПИСОК[НОМЕР] [3] = 5 И
              СПИСОК[НОМЕР] [4] = 5
              ТО ЧИСЛО := ЧИСЛО + 1
        ВСЕ
    КЦ
КОН
```

Применяя эту процедуру к нашему списку, получим в ответе 2 (в этом списке двое круглых отличников — Макаров и Полякова).

Записанная выше в нашей процедуре команда ветвления имеет составное условие, использующее союз И. Если мы заменим И на ИЛИ, получим процедуру подсчета числа учеников, имеющих пятерку хотя бы по одному из трех предметов.

```
ПРОЦ ОТЛИЧНИКИ( => СПИСОК, ЧИСЛО => )
ИМЕНА : НОМЕР
НАЧ
    ЧИСЛО := 0
    ДЛЯ НОМЕР ОТ 1 ДО ДЛИН(СПИСОК)
    НЦ
        ЕСЛИ СПИСОК[НОМЕР] [2] = 5 ИЛИ
              СПИСОК[НОМЕР] [3] = 5 ИЛИ
              СПИСОК[НОМЕР] [4] = 5
              ТО ЧИСЛО := ЧИСЛО + 1
        ВСЕ
    КЦ
КОН
```

Выполнив эту процедуру в применении к нашему списку, получим в ответе 5 (все ученики, кроме Вербицкого и Кириллова, имеют хотя бы одну пятерку).

Среднюю оценку по информатике можно подсчитать так:

```
ПРОЦ СРЕДНЯЯ (=> СПИСОК, ОЦЕНКА = > )
ИМЕНА : НОМЕР, СУММА
НАЧ
    СУММА := 0
    ДЛЯ НОМЕР ОТ 1 ДО ДЛИН(СПИСОК)
    НЦ
        СУММА := СУММА + СПИСОК[НОМЕР] [4]
    КЦ
    ОЦЕНКА := СУММА/ДЛИН(СПИСОК)
КОН
```

Результатом исполнения этого алгоритма будет средняя оценка «4».

Над кортежами, как и над текстами, определены операции вырезки и соединения (склеивания). Так, вырезая из кортежа $\langle 4, 2, 1, 3 \rangle$ второй и третий элементы, получаем кортеж $\langle 2, 1 \rangle$:

$$\langle 4, 2, 1, 3 \rangle \ [2:3] = \langle 2, 1 \rangle,$$

а соединяя два кортежа ⟨4, 9, 7⟩ и ⟨5, 8, 6, 3⟩, получаем кортеж ⟨4, 9, 7, 5, 8, 6, 3⟩:

$$\langle 4, 9, 7\rangle + \langle 5, 8, 6, 3\rangle = \langle 4, 9, 7, 5, 8, 6, 3\rangle.$$

Вопросы

1. Что соответствует в Рапире имеющимся в алгоритмическом языке табличным величинам?
2. Могут ли быть элементами кортежа: а) литерные величины; б) кортежи?
3. Какие операции над кортежами вам известны?

Упражнения

1. Составьте процедуру вычисления произведения всех элементов кортежа. Его заголовок должен иметь вид:

```
ФУНК ПРОИЗВЕДЕНИЕ(Х)
(*ПРОИЗВЕДЕНИЕ ВСЕХ ЭЛЕМЕНТОВ КОРТЕЖА Х*)
```

2. Чему равно А [3], если

А = ⟨"КОТ", "СЛОН", "ТИГР", "КРОЛИК"⟩?

3. Составьте процедуру, определяющую по списку класса (в форме кортежа) число учеников, у которых отметка по математике ниже отметки по информатике.

4. Элементами кортежа ЧИСЛА являются числовые величины. Найдите наименьший из элементов кортежа.

5. В условии предыдущей задачи найдите номер наименьшего элемента.

6. Кортеж КЛАСС состоит из кортежей пар. Первый элемент пары — литерная величина, значением которой является фамилия школьника, а второй — литерная величина, принимающая значение "А" для мальчиков и значение "Б" для девочек.

Опишите процедуру, которая сосчитает процентный состав девочек и мальчиков в классе.

7. Шахматную доску можно представить в виде кортежа кортежей. Кортеж ДОСКА состоит из 8 кортежей — горизонталей. Каждый кортеж — горизонталь — в свою очередь состоит из 8 элементов — полей.

Напишите процедуру ЛАДЬЯ, аргументами которой являются два целых числа M и N, между 1 и 8 — номер горизонтали и номер вертикали поля. Эта процедура должна «пометить» все поля, побиваемые ладьей, стоящей на поле (M, N). «Пометить» поле — это значит присвоить единицу соответствующему элементу — полю. В исходном состоянии все элементы — поля содержат нулевые значения.

§ 13 КОМАНДЫ ВВОДА И ВЫВОДА НА РАПИРЕ

До сих пор мы ничего не говорили о том, каким образом исходные данные для работы алгоритмов сообщаются программе и каким образом программа информирует нас о результатах своей работы. Сейчас мы расскажем, как это делается при использовании языка Рапира.

Для ввода исходных данных в компьютер пользуются клавиатурой, а для вывода результатов работы программы — экраном дисплея. В языке Рапира имеются следующие команды ввода данных с клавиатуры. По команде

ВВОД : X

работа программы приостанавливается, и мы имеем возможность ввести в компьютер желаемое значение. Пусть, например, мы нажали клавиши [А], [Б], [В], а затем специальную клавишу, сигнализирующую о завершении ввода. После этого исполнение программы продолжается, причем переменная величина X принимает литерное значение "АБВ". Если вместо клавиш [А], [Б], [В] мы нажмем клавиши [5] и [7] (и после этого нажмем специальную клавишу, сообщающую компьютеру об окончании ввода), то величина X примет литерное значение "57". Если же мы хотим, чтобы вводимые данные рассматривались как числа, то применяем команду

ВВОД ДАННЫХ : X

При выполнении этой команды машина воспринимает нажимаемые цифровые клавиши как цифры вводимого числа. Например, чтобы ввести в машину приближенное значение числа π по команде

ВВОД ДАННЫХ : ПИ

нужно нажать последовательно клавиши

[3], [.], [1], [4], [1], [6]

и клавишу, сообщающую об окончании ввода, после чего ПИ примет значение 3,1416. (Обратите внимание, что в языке Рапира, как и в других языках программирования, для отделения целой части числа от дробной применяется не запятая, а точка!)

Приведем пример использования команд ввода на примере программы отыскания наибольшего общего делителя:

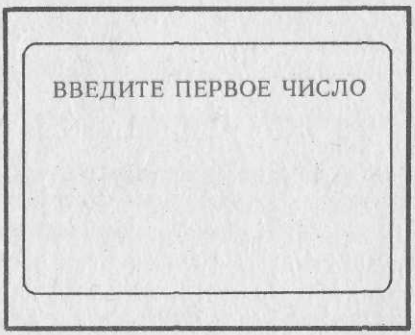
```
ПРОЦ НАИБОЛЬШИЙ_ОБЩИЙ_ДЕЛИТЕЛЬ
ИМЕНА: A, B
НАЧ
    ВЫВОД: "ВВЕДИТЕ ПЕРВОЕ ЧИСЛО"
    ВВОД ДАННЫХ: A
    ВЫВОД: "ВВЕДИТЕ ВТОРОЕ ЧИСЛО"
    ВВОД ДАННЫХ: B
    ВЫВОД: "НОД РАВЕН"
    ВЫВОД: НОД (A, B)
КОН
```

Наряду с командами ввода, эта программа использует команды вывода. Команда

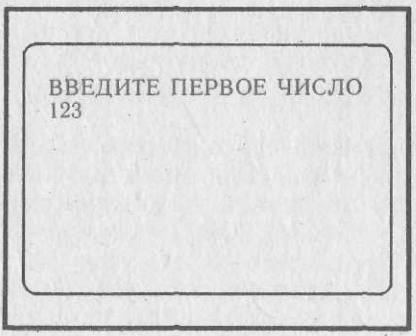
ВЫВОД: X

выводит на экран значение X. (При этом нет необходимости в двух отдельных командах вывода для чисел и текста, так как тип значений величины X машине известен.) Теперь проследим за работой программы.

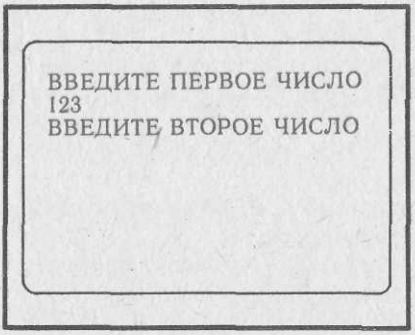
Исполнение программы начинается с выполнения первой команды. Это команда вывода. На экране появляется текст



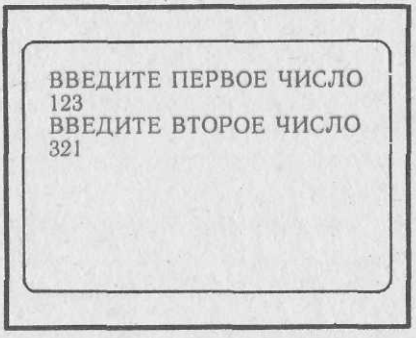
Затем выполняется команда ВВОД ДАННЫХ: A и работа программы приостанавливается до тех пор, пока мы не введем число с клавиатуры. Нажмем клавиши [1], [2], [3], соответствующие цифры появятся на экране:



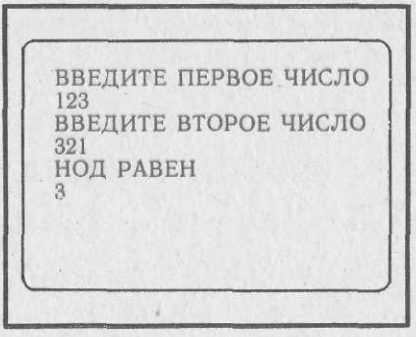
Но исполнение программы будет продолжено только после того, как мы нажмем клавишу окончания ввода. Как только мы это сделаем, исполнение программы продолжится и на экране появится следующий текст:



Нажимаем клавиши [3], [2], [1]. На экране появляется текст:



Теперь при нажатии на клавишу окончания ввода работа программы продолжается и на экране появляются надписи:



На этом исполнение программы заканчивается.

Обратите внимание, что приведенная программа использует вспомогательный алгоритм вычисления функции НОД(А,В). Этот алгоритм должен быть заранее сообщен компьютеру.

Вопросы

1. Для чего применяются команды ввода и вывода?
2. Как выполняются команды ввода и вывода?

Упражнения

1. Составьте программу, которая вводила бы два числа и выводила бы их произведение.

2. Составьте программу, которая вводила бы две литерные величины и выводила бы их соединение.

3. Программа должна ввести два числа — координаты левого нижнего угла прямоугольника, а затем еще два числа — координаты правого верхнего угла этого прямоугольника. После этого программа запрашивает у ученика значение периметра этого прямоугольника и в зависимости от введенного значения либо хвалит ученика, выводя текст "МОЛОДЕЦ", либо сообщает о неверном результате, выводя текст "НЕВЕРНО". Составьте такую программу.

4. На соревнованиях по прыжкам в высоту установлена квалификационная норма — 180 см. По программе надо ввести фамилию спортсмена и взятую им высоту.

Программа проверяет результат и выдает фамилию прыгуна и одно из двух сообщений

"НОРМА ВЫПОЛНЕНА"

или

"ВЫСОТА НЕ ВЗЯТА".

Составьте такую программу.

5. Перепишите процедуру

РЕШЕНИЕ_КВАДРАТНОГО_УРАВНЕНИЯ

так, чтобы коэффициенты уравнения можно было не задавать в виде аргументов, а вводить во время исполнения программы.

ЯЗЫК ПРОГРАММИРОВАНИЯ БЕЙСИК

В этом разделе мы познакомимся с языком программирования Бейсик.

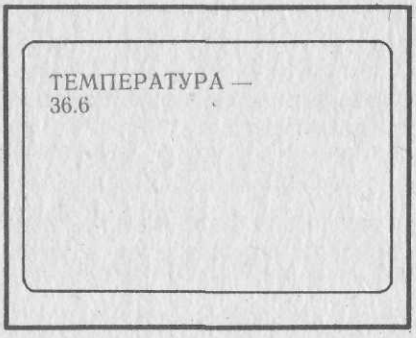
Язык программирования Бейсик был создан в США более двадцати лет назад. В настоящее время Бейсик — один из самых распространенных в мире языков программирования на персональных компьютерах.

§ 14. ОБЩИЕ СВЕДЕНИЯ О ЯЗЫКЕ БЕЙСИК

1. Общий вид программы на Бейсике. Программа на языке Бейсик состоит из последовательности *строк*. Каждая строка начинается целым числом — *номером* строки; за номером строки следует команда языка Бейсик. (Правила Бейсика допускают несколько команд в одной строке. В этом случае команды отделяются друг от друга двоеточием.) Приведем пример Бейсик-программы:

```
5  CLS
10 t = 36.6
20 text$ = "ТЕМПЕРАТУРА—"
30 PRINT text$
40 PRINT t
```

Из этого примера видно, что в языке Бейсик в основном используются буквы латинского алфавита и английские названия команд. Приведенная программа содержит пять строк по одной команде в каждой. Строка с номером 5 содержит команду очистки экрана дисплея CLS (*clear screen* — очистить экран). В строке 10 содержится уже знакомая нам команда присваивания. В отличие от алгоритмического языка в Бейсике вместо знака «:=» в команде присваивания применяется знак равенства «=». Строка 20 содержит команду присваивания для литерной переменной. Строки 30 и 40 содержат две команды печати PRINT. Первая выводит на экран значение указанной в ней литерной переменной text$, а вторая — значение вещественной переменной t. В результате этой программы на экране будут напечатаны две строки:



Замечание. Строки в Бейсике необязательно нумеровать подряд. Важно лишь, чтобы номера шли в возрастающем порядке. Однако обычно строки нумеруются с интервалом в 10:

10, 20, 30 ... .

2. Константы и переменные в языке Бейсик. Как и в алгоритмическом языке, в Бейсике величины делятся на переменные и постоянные (константы); каждая величина может быть целого, вещественного или литерного типа. Постоянные величины в Бейсике записываются так же, как и в алгоритмическом языке. Единственное отличие — в способе записи десятичных чисел. В Бейсике для отделения целой части от дробной пользуются не запятой, а точкой (например, 12.5, 0.5, 3.1415926 и т. д.). Это правило принято практически во всех языках программирования. Тип константы ясен из ее записи. Например, 2, 3, —1 — целые числовые константы, 2.0, 36.6, 3.14 — вещественные, а "сколько ему лет?", "введите число", "есть решения", "абвгдежзикля" — литерные. В отличие от алгоритмического языка в Бейсике тип переменной определяется по ее имени. Если имя оканчивается знаком % — переменная целого типа; если знаком $ — литерного типа, во всех остальных случаях — вещественного. Например, среди величин

agel, A, B999, name1$, y$, count%, y%

три первые — вещественные, две следующие — литерные, две последние — целые.

Для описания табличных величин (их называют *массивами*) используется служебное слово DIM (от английского слова *dimension* — размер). В Бейсике нумерация элементов таблиц всегда начинается с 0, поэтому при описании таблицы указывается только число элементов таблицы. Строка

10 DIM A(5)

содержит описание вещественной линейной таблицы A, в которой 5 элементов:

A(0), A (1), A (2), A (3), A (4).

На алгоритмическом языке описание соответствующей табличной величины выглядело бы так:

вещ таб *A* [0,4].

Номер элемента табличной величины, в отличие от алгоритмического языка, в Бейсике записывается в круглых скобках. Например, строка

20 A(4) = 36.6

содержит команду присваивания 4-му элементу таблицы A значения 36.6.

3. Выражения в языке Бейсик. Использование функций в выражениях. Как и в алгоритмическом языке, в Бейсике из переменных и констант с помощью знаков операций и круглых скобок можно строить выражения. Для обозначения арифметических операций используются следующие символы (табл. 18):

Таблица 18

Символ	Название операции
+	сложение
–	вычитание
*	умножение
/	деление
^	возведение в степень

Например,

A + B; A — B; (A + B)*2.1; x/(3.0*y);
z ^ 2; z ^ N%; (A(1) + B(1))/C(3).

Для литерных переменных в Бейсике разрешено использовать только одну операцию — соединение текстов. Эта операция обозначается знаком +. Например,

"коло" + "бок" = "колобок".

В выражениях языка Бейсик, как и в алгоритмическом языке, можно использовать названия функций, например:

2*SIN(X)*COS(X), X*ABS(A — B) и т. д.

Бейсик — язык программирования, и алгоритмы, записанные с его помощью, предназначены для ввода в ЭВМ. Поэтому для записи алгоритмов на языке Бейсик можно использовать только те символы, которые есть на клавиатуре ЭВМ. Это почти все символы, которыми мы привыкли пользоваться в математике (кроме греческих букв); операция извлечения квадратного корня ($\sqrt{a}$) обозначается SQR(a); знака «не равно» ($\neq$) на клавиатуре тоже нет, и выражение $a \neq b$ записывается: a <> b, т. е. с помощью двух знаков: «меньше» (<) и «больше» (>).

Еще некоторые отличия в записи вы заметите сами; все они связаны с заменой отсутствующих на клавиатуре символов имеющимися, требованием записи символов в одну строку и английским «происхождением» этого языка.

§ 15. КОМАНДЫ ЯЗЫКА БЕЙСИК

1. Команда ветвления. Так же как и в алгоритмическом языке, в Бейсике есть команда ветвления. Ее общий вид следующий:

IF условие THEN команда ELSE команда

В этой команде участвуют английские слова IF, THEN, ELSE, которым соответствуют русские слова **если**, **то**, **иначе**. Выполняется эта команда аналогично команде ветвления в алгоритмическом языке.

Приведем пример:

```
10 IF a> b THEN a=5 ELSE a=10
20 c=a
```

После исполнения этой программы вещественная переменная a будет равна 5, если условие соблюдается (т. е. a> b), во всех иных случаях переменной a будет присвоено значение 10.

Условия в Бейсике записываются так же, как в алгоритмическом языке; отличия видны из таблицы 19.

Таблица 19

Алгоритмический язык	$A=B$	$A<B$	$A>B$	$A\geqslant B$	$A\leqslant B$	$A\neq B$
Бейсик	A=B	A<B	A>B	A>=B	A<=B	A<>B

Так же как и в алгоритмическом языке, в Бейсике можно образовывать составные условия, только форма записи будет немного другая: вместо слов **и**, **или**, **не** нужно писать английские слова AND, OR, NOT соответственно; исходные условия можно брать в круглые скобки. Например, условие

$$(x>5)\ \text{AND}\ (x<=7)$$

означает, что $5<x\leqslant 7$.

2. Команда повторения в Бейсике бывает двух типов. Первый из них аналогичен команде повторения **пока** — **нц** — **кц** в алгоритмическом языке. Вот пример использования этой команды:

```
10 DIM A(10)
15 I%=0
20 WHILE I%<10
30 A(I%)=0
35 I%=I%+1
40 WEND
```

Приведенная программа заполняет нулями линейную таблицу из десяти элементов. Английское слово WHILE означает **пока**. Работает эта команда так: проверяется условие, стоящее после WHILE. Если оно соблюдается, то выполняются все команды, стоящие после слова WHILE, но до слова WEND, играющего ту же роль, что и **кц** в алгоритмическом языке. После этого снова проверяется условие. Как только выяснится, что условие не соблюдается, выполнение команды WHILE закончится и начнется выполнение команды, следующей за WEND.

Второй тип команды повторения аналогичен команде повторения с параметром и выглядит следующим образом:

```
10 DIM A(10)
15 B%=9
20 FOR I%=0 TO B%
30 A(I%)=0
40 NEXT I%
```

Эта программа делает то же самое, что и предыдущая. Приведенная в программе команда

```
20 FOR I%=0 TO B%
```

работает так же, как и команда повторения с параметром в алгоритмическом языке: переменная I% меняется последовательно с шагом 1 от 0 до B% и для каждого значения I% выполняются команды, стоящие между FOR и NEXT. (Перевод английских слов FOR, TO и NEXT такой: для, до, следующий.) После слова NEXT можно указывать имя параметра.

Как и в алгоритмическом языке, значение параметра может изменяться с шагом, отличным от 1. Для этого используется служебное слово STEP (шаг). Например, запись

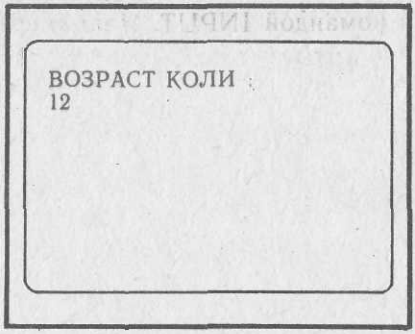
```
20 FOR I%=0 TO B% STEP 2
```

означает, что шаг изменения переменной I% равен 2.

3. Команда обращения к подпрограмме — GOSUB. Это слово происходит от двух английских слов *go* — идти, *subroutine* — подпрограмма. Команда позволяет перейти к исполнению программы с той строки, номер которой указан в команде. Как только первый раз после выполнения команды GOSUB встретится команда RETURN (вернуться), исполнение программы продолжится со строки, следующей за командой GOSUB. Рассмотрим следующий пример:

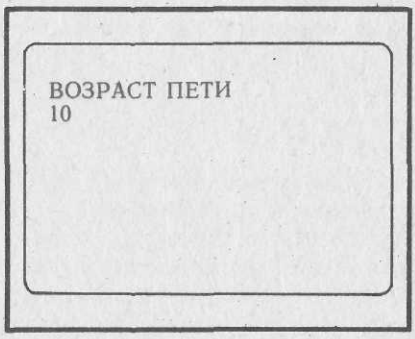
```
10  IF y$="да" THEN GOSUB 100 ELSE GOSUB 130
20  PRINT name$
25  PRINT age
30  END
...
100 name$="ВОЗРАСТ КОЛИ"
110 age=12
120 RETURN
130 name$="ВОЗРАСТ ПЕТИ"
140 age=10
150 RETURN
```

Если при исполнении этой программы значение литерной переменной y$ окажется равным "да", то будет выполнена команда GOSUB 100. Произойдет переход к строке 100, переменная name$ получит значение "ВОЗРАСТ КОЛИ", а переменная age — значение 12. После этого команда RETURN (в строке 120) вернет нас на строку 20 и на экране появится



На этом исполнение закончится (в строке 30 стоит команда END).

Если же у$ ≠ "да", то на экране появится



Команда END (конец) — команда окончания исполнения программы.

Команда REM позволяет вносить в текст Бейсик-программы комментарии: любой текст от слова REM (от английского слова *remark* — пояснение) до конца строки игнорируется во время исполнения программы.

Например, в предыдущую программу можно добавить строку

2 REM программа "ПЕТЯ И КОЛЯ"

4. Команды ввода-вывода. Мы познакомились с командами языка Бейсик, похожими на те, которые уже встречались при изучении алгоритмического языка.

Теперь перейдем к командам вывода информации на экран и приема данных с клавиатуры. Именно команды ввода-вывода позволяют писать программы, «ведущие диалог с человеком».

Команда PRINT используется для вывода на экран цифровой и буквенной информации. Она уже встречалась в наших примерах (см. с. 71).

Команда INPUT позволяет приостановить исполнение программы и ввести с клавиатуры значения переменных, имена ко-

торых следуют за командой INPUT. Например, команда

```
10 INPUT A, B, N%, TEXT$
```

позволяет ввести два вещественных числа, одно целое и одну литерную величину, сделав их значениями переменных величин A, B, N% и TEXT$.

Следующая программа

```
10 INPUT A, B, C
20 PRINT A+B+C
30 END
```

напечатает на экране сумму трех введенных с клавиатуры чисел.

Вопросы

1. Какой тип имеют величины a, b, c%, d%?
2. Приведите примеры переменных: а) целого; б) вещественного; в) литерного типа.
3. Какой тип имеют константы 2; —3; —2.0; "константа"; "целое"?
4. Приведите примеры констант: а) целого; б) вещественного; в) литерного типа.

Упражнения

1. Напишите на Бейсике программу, в которой используется по крайней мере один раз каждая из указанных команд:

а) присваивания;
б) ветвления;
в) повторения WHILE — WEND;
г) повторения FOR — NEXT;
д) ввода данных INPUT.

Результат работы программы должен быть выведен на дисплей с помощью команды PRINT. Поясните, что делает ваша программа.

2. Что появится на экране после исполнения такой программы?

```
10 A=5
20 B=7
30 IF A> B THEN C=A ELSE C=B
40 PRINT A, B, C
```

3. Напишите программу, которая вводит с клавиатуры два числа и выводит на экран их произведение.

4. Напишите программу, которая вводит с клавиатуры два числа и выводит на экран большее из них.

5. Напишите программу, которая вводит с клавиатуры два числа и выводит на экран разность между большим и меньшим.

Раздел III

РОЛЬ ЭВМ В СОВРЕМЕННОМ ОБЩЕСТВЕ. ПЕРСПЕКТИВЫ РАЗВИТИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

§ 16. КРАТКАЯ ИСТОРИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Возникновение электронных вычислительных машин. Еще в XVII в. в связи с бурным развитием мореплавания и астрономии появилась потребность в быстром и точном составлении различных математических таблиц, в том числе таблиц синусов, логарифмов, квадратных корней и др. Возникла мысль создать устройство, облегчающее проведение арифметических операций над большим количеством чисел.

В 1642 г. французский ученый Б. Паскаль сконструировал первый механический вычислитель, позволяющий складывать и вычитать числа. В 1673 г. немецкий ученый Г. Лейбниц разработал счетное устройство, в котором использовал механизм, известный под названием «колеса Лейбница». Его счетная машина выполняла не только сложение и вычитание, но и умножение и деление. Механические счетные машины — арифмометры — с видоизмененными «колесами Лейбница» использовались до середины нашего столетия, пока не были вытеснены электрическими цифровыми вычислителями, а впоследствии современными электронными калькуляторами.

В середине 30-х годов XIX столетия английский математик Ч. Беббидж предложил структуру автоматического вычислителя, названного им «аналитической машиной», состоящего из двух отдельных устройств: устройства хранения, где находятся команды и данные, введенные в машину, и перерабатывающего устройства (процессора), которое выполняет операции, пользуясь находящимися в памяти командами и данными. Самому автору реализовать идею «аналитической машины» не удалось.

В начале 40-х годов XX в. болгарский инженер Д. Атанасов и американский ученый Д. Моучли предложили использовать радиолампы для выполнения вычислений, предполагая тем самым полностью исключить механические узлы компьютера.

Первая вычислительная машина, полностью реализующая эту идею, была создана в США в 1946 г. Она получила название ЭНИАК — по первым буквам полного английского названия, которое по-русски выглядит как «электронный числовой интегратор и вычислитель». Именно с этого времени начинается эра электронных вычислительных машин.

Прошло, однако, еще несколько лет, пока не сложились основные принципы устройства, или, как говорят, *архитектура* современных ЭВМ. Эти принципы были обоснованы выдающимся математиком Дж. фон Нейманом в 1946 г. Первая ЭВМ, основанная на этих принципах,— ЭДСАК была создана в Англии в 1949 г. ученым и конструктором М. Уилксом.

Первая советская электронная вычислительная машина (получившая впоследствии название МЭСМ — малая электронная счетная машина) была создана в 1949 г. в Киеве, а еще через три года, в 1952 г., в Москве вошла в строй машина БЭСМ (быстродействующая электронная счетная машина). Обе машины были созданы под руководством выдающегося советского ученого Сергея Алексеевича Лебедева (1902—1974), основоположника советской электронной вычислительной техники.

Поколения ЭВМ. Компьютер состоит из очень большого числа электронных компонент, имеющих самые простые функции. Различных простейших функций совсем немного.

Эти функции элементов компьютера за истекшие 35—40 лет почти не изменялись, в то время как их физическое устройство менялось очень сильно. Каждый этап развития компьютеров определялся совокупностью элементов, из которых строились компьютеры,— *элементной базой*.

С изменением элементной базы ЭВМ значительно изменялись характеристики, внешний вид и возможности компьютеров.

Ни одно техническое устройство, изобретенное человеком до сих пор, не развивалось так стремительно, как электронные вычислительные машины. Каждые 10—12 лет происходил резкий скачок в конструкции и способах производства ЭВМ. Появляющиеся в результате такого скачка новые модели ЭВМ быстро вытесняли старые, при этом область применения ЭВМ постоянно расширялась.

Именно поэтому уместно говорить о *поколениях* ЭВМ, сменявших друг друга в ходе общего развития вычислительной техники.

Итак, в основе смены поколений вычислительной техники лежит смена элементной базы ЭВМ. Появление новой элементной базы, в свою очередь, обычно связано с прогрессом в области физики и химии, который приводит к открытию новых принципов работы компонент ЭВМ, новых свойств материалов и новых способов производства.

Смена элементной базы требует полной перестройки заводов, производящих вычислительную технику, разработки новых технологических линий, станков и другого оборудования. Поэтому такая смена оправдана лишь тогда, когда переход на новую элементную базу существенно улучшает характеристики вычислительной техники.

Естественно, что смена поколений заключалась не только в обновлении элементной базы. С каждым новым поколением в прак-

тику применения ЭВМ входили новые способы решения задач и новые компоненты программного обеспечения.

В настоящее время основу вычислительной техники составляют ЭВМ третьего и четвертого поколений и ведутся экспериментальные разработки машин пятого поколения.

В ЭВМ первого поколения все элементы электронных схем изготовлялись в виде отдельных деталей. Важнейшими среди них были вакуумные *электронные лампы,* которые сейчас еще можно увидеть в телевизорах и старых радиоприемниках. Несколько таких ламп устанавливалось на металлической панели — шасси, которое вставлялось в корпус ЭВМ. На этом же шасси укреплялись и другие элементы схемы.

Сама ЭВМ имела вид большого количества металлических шкафов (стоек), сплошь заполненных шасси с электронными лампами. Машины первого поколения занимали громадные залы, весили сотни тонн и расходовали сотни киловатт электроэнергии.

Появление ЭВМ второго поколения стало возможным благодаря изобретению *транзисторов*. Резкое уменьшение размеров транзисторов по сравнению с радиолампами позволило делать блоки ЭВМ в виде так называемых печатных плат. Такая плата представляет собой пластмассовую пластинку, на которую с одной ее стороны вставляются и припаиваются транзисторы и другие элементы, а с другой стороны прямо на поверхности располагаются проводники в виде металлических полосок, соединяющих элементы схемы.

Использование транзисторов и печатных плат позволило частично автоматизировать производство ЭВМ и значительно сократить их размеры и потребление энергии.

Основу ЭВМ третьего поколения составляют так называемые *интегральные схемы*. Элементы компьютеров предыдущих поколений производились в виде отдельных деталей и лишь потом соединялись друг с другом, образуя нужную схему.

Исследования в области физики и химии показали, что схемы можно формировать на небольшом участке пластинки из чистого кристаллического кремния, нанося на этот участок в нужной комбинации тончайшие пленки разных веществ. Формирование элементов можно осуществлять сразу на многих участках пластинки.

Такая схема, имеющая вид многослойной пленки веществ, нанесенных на кристаллы кремния, получила название интегральной схемы. Уже в первых интегральных схемах на одном кристалле размещалось до сотни элементов.

Изобретение интегральных схем открыло перспективы дальнейшего развития элементной базы ЭВМ, которые до сих пор далеко не исчерпаны. Сразу резко повысился уровень надежности электронных схем при столь же резком падении их стоимости благодаря уменьшению размеров и, главное, автоматизации их проектирования и производства. Ступени прогресса в разработке интегральных схем стали измеряться количеством элементов, которые можно раз-

местить на одном кристалле кремния стандартных размеров. В ЭВМ третьего поколения применялись интегральные схемы, содержащие более тысячи элементов на одном кристалле.

ЭВМ четвертого поколения используют *большие интегральные схемы* (БИС), в которых количество переключательных элементов на кристалле кремния равно десяткам тысяч. Замечательным достижением XX в. стало создание процессора ЭВМ, который целиком размещается на одном кристалле кремния. Такие однокристальные процессоры получили название *микропроцессоров*. В результате на одной плате оказалось возможным разместить электронные схемы всех устройств ЭВМ, а саму ЭВМ, которая еще двадцать лет назад занимала большой зал, сделать по габаритам и по стоимости доступной для индивидуального применения на рабочем месте пользователя. Так появились персональные ЭВМ, а также карманные и настольные микрокалькуляторы. Итак, мы видим, что развитие технологии производства ЭВМ позволяет сделать компьютер с заданными вычислительными ресурсами все более и более дешевым и доступным.

В настоящее время появляется возможность изготавливать «супер-ЭВМ», рекордные по скорости работы, объему памяти и т. п. Такие ЭВМ изготавливаются на самой современной элементной базе, выпускаются в небольшом количестве и очень дороги, но они незаменимы во многих областях науки и техники.

Одним из основных направлений экономического и социального развития СССР на период до 2000 года является всемерное ускорение научно-технического прогресса, повсеместное применение его результатов в производстве и управлении, сфере обслуживания и в быту. Катализатором научно-технического прогресса является использование вычислительной техники. Именно поэтому перед нашей страной в нынешней пятилетке стоит задача ускоренного развития выпуска средств автоматизации инженерного труда, малых ЭВМ высокой производительности и персональных ЭВМ.

Мы уже говорили, что за несколько десятков лет, прошедших со времени изобретения компьютеров, они стали в тысячи раз производительнее и в тысячи раз дешевле. Возможность такого быстрого развития связана с тем, что компьютер, в отличие от других машин, является устройством для обработки информации.

Поясним сказанное. Можем ли мы надеяться, что через несколько лет автомобиль станет в десять раз меньше по размеру? Нет, конечно: ведь тогда человек в него просто не поместится. Размеры и другие параметры автомобиля тесно связаны со свойствами окружающего мира, в котором он должен работать. Размеры же компьютера никак не связаны с его функциями по переработке информации и определяются лишь технологией производства, которая все время совершенствуется.

Это привело к тому, что уже сейчас затраты на изготовление компьютера невелики по сравнению с затратами на разработку программ. Это парадоксально. Ведь разработать программы

нужно всего лишь раз, а потом их можно почти без затрат копировать в любом количестве экземпляров. В то же время каждый экземпляр компьютера нужно сделать заново. Тем не менее даже после деления на число выпускаемых компьютеров данного типа затраты на разработку программ остаются весьма значительными.

Такое развитие привело к изменениям во взглядах на соотношение аппаратуры компьютера и программ. Если раньше считалось, что задача программистов — наиболее полно использовать все возможности аппаратуры, то теперь стало ясно, что аппаратура компьютера должна максимально облегчать главную задачу — разработку программ. Если раньше программы предназначались для управления машинами, то теперь машины предназначаются для исполнения программ. Это приводит к повышению роли науки программирования в индустрии информатики.

§ 17. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

Как мы уже знаем, ЭВМ не может работать без программы. Только поместив программу в память ЭВМ, можно заставить ее сделать что-нибудь полезное. Сменив программу в памяти ЭВМ, можно переключить ЭВМ с одной работы на другую. Именно благодаря этой особенности одни и те же ЭВМ могут применяться в разных областях человеческой деятельности. Сфера применения конкретной ЭВМ определяется набором созданных для нее программ. Этот набор называется *программным обеспечением*. Программное обеспечение современных ЭВМ включает десятки и сотни тысяч программ: от широко распространенных компьютерных игр и программ обработки текстов до программ специального назначения (например, программ расчета ядерных реакторов).

Рассмотрим различные применения ЭВМ и тем самым различные виды программного обеспечения.

Обработка текстов. Одно из наиболее распространенных применений компьютеров, особенно персональных,— обработка текстов. Всем нам время от времени приходится писать какие-то тексты — письма друзьям, школьные сочинения, заметки в стенгазету и т. д. Многие профессии связаны с обработкой и подготовкой большого количества текстов. Это приходится делать инженеру и служащему, писателю и программисту. Нет особого смысла использовать сейчас компьютер для писания писем своим близким — проще написать их от руки. Однако в будущем, когда компьютеры появятся в каждой семье и будут соединены друг с другом в сеть (как сейчас телефоны), может оказаться удобным быстро передать по такой сети подготовленный на компьютере текст и не прибегать к услугам почты.

Уже сегодня компьютер оказывается неоценимым помощником при оперативной профессиональной обработке текстов, например

в редакциях газет. Удобен компьютер и при тщательной подготовке текстов большого объема, например книг. Сейчас книга обычно готовится так. Автор отдает рукопись книги машинистке для перепечатки. После этого он дает читать перепечатанную рукопись рецензенту, редактору и другим специалистам. Их замечания обычно требуют изменений в тексте. Эти изменения нужно аккуратно сделать во всех экземплярах. Когда, наконец, все поправки сделаны, книга отдается в типографию, наборщик садится за клавиатуру наборной машины и буква за буквой набирает текст (неизбежно допуская при этом некоторое количество ошибок).

Применение ЭВМ позволяет в корне упростить и ускорить описанную работу. Для этого необходимо заменить пишущую машинку компьютером и вместо печатания на машинке вводить текст в компьютер в помощью специальной программы — редактора текстов. С введенным в ЭВМ текстом легко работать: его можно напечатать в любом количестве экземпляров (для рецензента, редактора и т. п.), в него можно вносить изменения и улучшения. Наконец, гибкий диск с записанным на нем окончательным текстом книги можно поместить в установленный в типографии компьютер, который соединен с фотонаборной машиной, и сделать набор автоматически, с помощью ЭВМ.

Такая технология в последнее время находит все большее распространение. При ее применении отпадает необходимость в корректорах, сверяющих набранный текст книги с рукописью автора, облегчается исправление замеченных ошибок, уменьшается количество опечаток и т. д.

Как же выглядит работа с редактором текстов? Работая над рукописью с помощью такой программы, мы видим на экране дисплея текст, или, точнее, ту его часть, которая умещается на экране.

Нажимая специальные клавиши, можно осмотреть любой интересующий нас фрагмент текста. Можно впечатать в любое место текста любую букву (вместо той, которая была в этом месте раньше) и таким образом исправить опечатку, заменить неудачно выбранное слово другим, вставить в любое место текста новые слова, строчки и т. д.

Программы текстовой обработки составляют сейчас наиболее часто используемые программы для микрокомпьютеров.

Языки программирования. Работа ЭВМ состоит в исполнении программы, состоящей из размещенных в памяти ЭВМ машинных команд. Однако создать вручную текст на машинном языке очень сложно. Гораздо проще записать алгоритм на алгоритмическом языке и «научить» ЭВМ исполнять программы, написанные на алгоритмическом языке или на языке, близком к нему. Таких языков сейчас существует много, и называются они *языками программирования.*

Один из языков программирования — Фортран (от англ. *Formula Translator* — переводчик формул) был создан в конце

50-х годов и получил широкое распространение. Он был предназначен для программирования научно-технических расчетов. Другой язык, оказавший большое влияние на развитие информатики,— Алгол-60 (от англ. *Algorithmic* — алгоритм и *Language* — язык, его название отражает дату создания — 1960 г.) послужил основой для разработки многих языков, например языка Паскаль (назван в честь ученого Б. Паскаля), который является одним из наиболее распространенных языков программирования для микрокомпьютеров.

Число относительно широко используемых языков программирования в настоящее время исчисляется десятками, а общее их число — тысячами. Существуют узкоспециализированные языки, скажем для программирования определенного вида технологических процессов, и более универсальные. Развитие и распространение языков программирования, как и вообще всего программного обеспечения, происходит медленнее, чем развитие элементной базы ЭВМ. Для того чтобы новый язык программирования получил широкое распространение, нужно, чтобы его выучило достаточное количество программистов, чтобы на этом языке было написано достаточно большое количество программ и т. п.

Существуют два типа программ-переводчиков с языка программирования на машинный язык. Они называются *компиляторами* и *интерпретаторами*. Программа-компилятор читает текст на языке программирования от начала и до конца, создавая эквивалентную программу на машинном языке. Интерпретатор читает исходную программу по частям, сразу выполняя соответствующие действия. Разницу между использованием компиляторов и интерпретаторов можно пояснить таким сравнением. Если у нас имеется книга на малознакомом нам языке, можно поступить двояко. Можно перевести ее целиком и в дальнейшем пользоваться только переводом. Этот способ соответствует действиям компилятора. А можно пользоваться книгой в оригинале, переводя только нужную ее часть по мере надобности. При этом нам придется переводить кое-какие места много раз, если они нам многократно понадобятся (ведь результаты перевода не запоминаются). Зато другие разделы, которые не понадобятся ни разу, не придется переводить вовсе. Этот вариант соответствует применению интерпретатора.

И компиляторы, и интерпретаторы имеют и преимущества, и недостатки. Из упоминавшихся языков программирования Фортран, Алгол-60, Паскаль обычно компилируются, а Бейсик и Рапира — интерпретируются (хотя это и необязательно: существуют и интерпретаторы для Паскаля, и компиляторы для Бейсика).

Автоматизация труда программиста. Использование ЭВМ позволяет повысить производительность труда в любой области человеческой деятельности. Не является исключением и труд программиста.

На заре программирования программы составлялись в машин-

ных кодах, записывались на бумаге и вручную заносились в память ЭВМ. Позже появились специальные устройства, позволяющие вводить в память ЭВМ информацию, закодированную с помощью отверстий на специальных тонких картонных карточках — *перфокартах*. Это облегчило работу по вводу программы в память ЭВМ. Появление программ-компиляторов и программ-интерпретаторов избавило программистов от необходимости работать в машинных кодах и коренным образом повысило производительность их труда. Программы, однако, по-прежнему писались на бумаге, а затем пробивались на перфокартах. Следующим шагом в повышении производительности труда программистов стало использование текстовых редакторов для подготовки текстов программ сразу на ЭВМ.

Если при этом используется компилятор, то общая схема работы оказывается такой: (1) человек вводит в память ЭВМ текст программы на языке программирования с помощью текстового редактора; (2) подготовленный текст обрабатывается компилятором, который проверяет, не нарушены ли правила записи программ на данном языке, и, если все правильно, создает программу в машинных кодах; (3) эта программа в машинных кодах исполняется на ЭВМ. Если на этапе (2) или (3) в программе обнаруживается ошибка, то все приходится повторять заново. Для сложных программ такой цикл приходится повторять многократно.

Следующий этап повышения производительности труда программиста — использование вместо редактора текстов так называемого *редактора программ,* учитывающего правила записи программ на используемом языке программирования. Например, редакторы программ на языке Бейсик, как правило, могут автоматически нумеровать строчки программы и вставлять в текст программы служебные слова при нажатии на одну кнопку клавиатуры. Это убыстряет ввод текста программы и избавляет от ошибок в записи служебных слов.

Редактор программ можно совместить с компилятором или интерпретатором. В этом случае ЭВМ сможет проверять правильность программы по мере ее ввода человеком. Такой редактор программ может, кроме того, при нажатии на специальные клавиши формировать не только служебные слова, но и целые конструкции языка, делая невозможными ошибки в записи этих конструкций.

Приведем примеры работы редактора программ для алгоритмического языка.

При создании новой программы при нажатии на специальную кнопку клавиатуры на экране дисплея появляется конструкция «алгоритм», т. е. должным образом расположенные служебные слова АЛГ, АРГ, РЕЗ, НАЧ, КОН (рис. 22). Курсор (маленький прямоугольник) устанавливается после слова АЛГ, как бы приглашая человека ввести имя алгоритма. Если в процессе составления алгоритма человек нарушит правила алгоритмического языка, то редактор программ поможет человеку немедленно обнаружить и

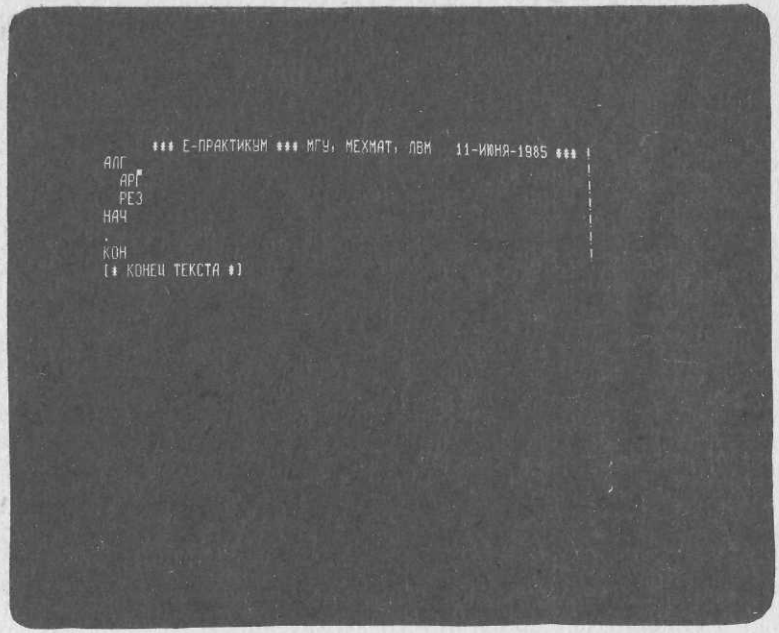

*** Е-ПРАКТИКУМ *** МГУ, МЕХМАТ, ЛВМ 11-ИЮНЯ-1985 ***
АЛГ
АРГ
РЕЗ
НАЧ
КОН
[* КОНЕЦ ТЕКСТА *]

Рис. 22

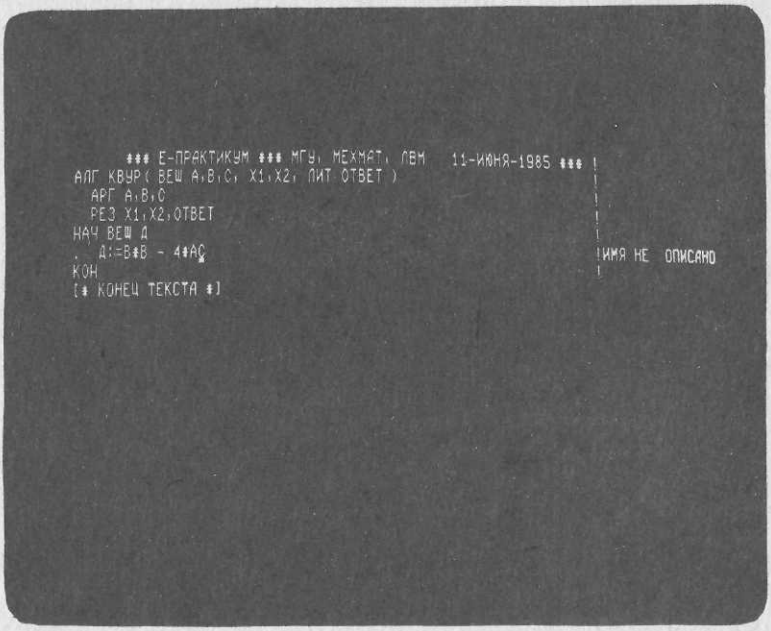

*** Е-ПРАКТИКУМ *** МГУ, МЕХМАТ, ЛВМ 11-ИЮНЯ-1985 ***
АЛГ КВУР(ВЕЩ A,B,C, X1,X2, ЛИТ ОТВЕТ)
АРГ A,B,C
РЕЗ X1,X2,ОТВЕТ
НАЧ ВЕЩ Д
Д:=B*B - 4*AC
КОН
[* КОНЕЦ ТЕКСТА *]
ИМЯ НЕ ОПИСАНО

Рис. 23

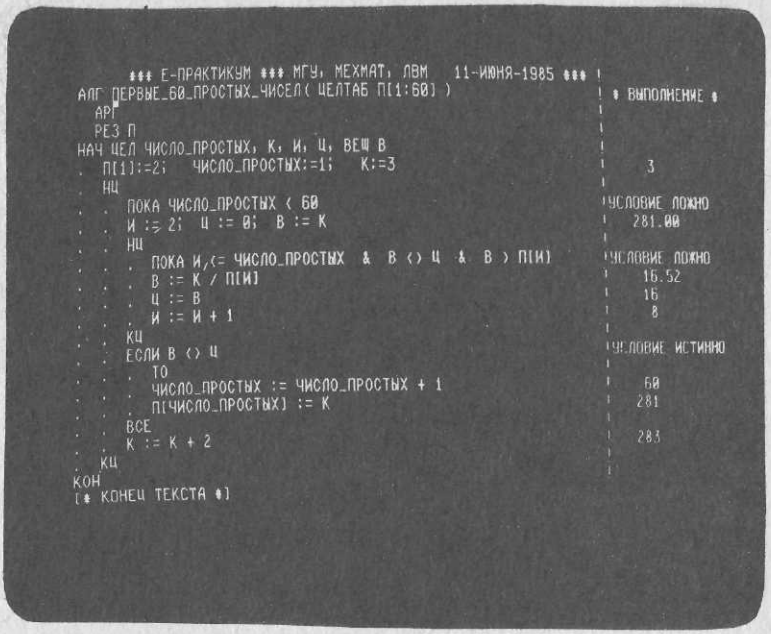


Рис. 24

исправить ошибку. Пусть, например, при составлении алгоритма КВУР человек забыл написать знак умножения между А и С в формуле для дискриминанта. Тогда редактор программ воспримет АС как имя переменной величины, а так как правила алгоритмического языка требуют, чтобы тип любой величины был описан, то редактор программ сообщит на полях справа, что величина с именем АС не описана, и установит курсор под этой ошибкой (рис. 23). После того как человек вставит между А и С знак умножения, сообщение об ошибке исчезнет.

Редактор программ может и исполнять их, показывая в ходе их исполнения на полях программы результаты проверок условий и выполнения операций присваивания. На рисунке 24 изображен результат исполнения программы вычисления первых 60 простых чисел: видно, что 60-е простое число (результат присваивания "П[ЧИСЛО_ПРОСТЫХ]:=К") равно 281.

Существуют и другие способы автоматизации труда программиста. Самый радикальный — вообще избавить программиста от необходимости явно описывать в программе последовательность действий. Достаточно описать (на специальном языке), *что* должна сделать программа, а не *как* это она должна делать. Такие языки называются *непроцедурными языками программирования.*

Операционная система. Мы уже знаем, что перед исполнением программы, написанной на языке машинных команд, необходимо

разместить ее в памяти ЭВМ и поместить в Счетчик Команд процессора адрес первой команды программы. На одной и той же ЭВМ обычно исполняют много разных программ. Эти программы хранят во внешней памяти ЭВМ — чаще всего на диске. Следовательно, необходима специальная *программа-загрузчик*, которая найдет нужную нам программу на диске, скопирует («загрузит») ее в память ЭВМ и установит начальное значение Счетчика Команд. Если наша программа использует вспомогательные алгоритмы (подпрограммы), то их также надо найти и поместить в память ЭВМ.

Большинство программ использует внешние устройства ЭВМ: дисплей, клавиатуру, диски и др. При управлении ими в разных программах постоянно приходится повторять одни и те же последовательности действий. Эти часто встречающиеся действия можно записать в виде специальных вспомогательных подпрограмм для работы с внешними устройствами.

Во внешней памяти ЭВМ (на диске) могут храниться не только машинные программы, но и другая информация. Порция информации, хранящаяся во внешней памяти, называется *файлом*. Для организации записи, хранения, поиска и считывания файлов используется комплекс специальных программ, который называется *файловой системой*.

Поскольку быстродействие ЭВМ велико, то она может одновременно делать несколько дел. Например, она может одновременно печатать текст на печатающем устройстве, выяснять, сколько свободного места осталось на диске, и обрабатывать символы, набираемые человеком на клавиатуре. Каждое такое «дело» называется *процессом*. Для организации параллельного (одновременного) выполнения нескольких процессов также нужны специальные управляющие программы.

Итак, для любой работы на ЭВМ нужен целый комплекс специальных вспомогательных программ (как перечисленных выше, так и многих других). Этот комплекс программ называется *операционной системой* и входит в так называемое *базовое программное обеспечение* ЭВМ.

Прикладные программы для научно-технических расчетов. Покажем работу таких программ на примере программы-решателя задач на треугольники. Из курса математики известны следующие соотношения между элементами треугольника (рис. 25):

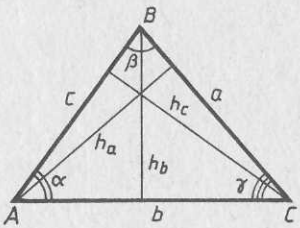


Рис. 25

$$\alpha + \beta + \gamma = \pi \qquad (1)$$

$$\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c} \qquad (2)$$

$$a^2 = b^2 + c^2 - 2bc \cdot \cos\alpha \qquad (3)$$

$$S = \frac{1}{2} a h_a \qquad (4)$$

$$S = \frac{1}{2} bh_b \qquad (5)$$

$$S = \frac{1}{2} bc \sin \alpha \qquad (6)$$

$$S = \frac{1}{2} ab \sin \gamma \qquad (7)$$

Используя эти соотношения, программа-решатель задач позволяет найти одни элементы треугольника, если известны другие. Пусть, например, известны площадь S, высоты h_a и h_b треугольника. Требуется найти стороны a и b, угол γ.

Получив такого рода условие, программа пытается найти цепочку равенств из числа соотношений (1) — (7), ведущую от известных величин к искомым.

Предполагается, что из каждого равенства можно выражать одну из входящих в него величин через остальные. Например, из равенства (6) вытекает, что

$$\alpha = \arcsin \left(\frac{2S}{bc}\right) .$$

При поиске цепочки равенств, ведущих от аргументов задачи к ее результатам, возникает задача типа поиска пути в лабиринте, игры в домино или отгадывания головоломки.

В нашем примере требуемая цепочка может быть такой:

из (4): a по S и h_a,
из (5): b по S и h_b,
из (7): γ по S, a и b.

В ряде случаев получение из уравнения явных формул, выражающих одну величину через другие, может быть выполнено программой автоматически. В более трудных случаях соответствующие явные формулы и реализующие их алгоритмы находятся разработчиками заблаговременно.

При решении нашей задачи ЭВМ согласно найденной цепочке уравнений автоматически выстраивает и затем исполняет следующую программу решения поставленной задачи:

$$a := 2*S/h_a$$
$$b := 2*S/h_b$$
$$\gamma := \arcsin (2*S/(a*b))$$

Информационные системы. Задачи, решаемые такими системами, поясним на примере библиотеки. Каждая библиотека имеет катало́г. Этот каталог состоит из большого числа ящиков, заполненных карточками. Каждая карточка содержит информацию об

одной из книг: кто ее автор, как она называется, в каком году издана и т. д. Наконец, на карточке указан «адрес» книги — в каком шкафу и на какой полке эту книгу следует искать. Обращаясь к библиотекарю, читатель должен назвать ему этот адрес — иначе библиотекарь не сможет найти нужную книгу среди множества книг библиотеки.

Если бы карточки стояли в беспорядке, то найти карточку интересующей нас книги было бы практически невозможно. Поэтому их располагают в алфавитном порядке по фамилиям авторов — вначале идут книги, авторы которых имеют фамилии, начинающиеся на А, затем — на Б, после них — на В и так далее. Это позволяет найти нужную книгу (точнее, ее карточку) довольно быстро — конечно, в том случае, если мы знаем фамилию ее автора. А что делать, если фамилия автора нам неизвестна, а известно только название книги? Приходится заводить еще одну картотеку, где карточки расположены в алфавитном порядке названий. А если у книги два автора? Вероятно, нужно включить эту карточку в алфавитный каталог дважды — по фамилии каждого из авторов. Уже из сказанного видно, что работа по составлению и поддержанию в рабочем состоянии библиотечного каталога связана с огромным количеством проблем, причем о многих из них мы даже и не упоминаем. (Что делать, если карточка из картотеки потеряна? Что делать, если книга отдана на некоторое время в переплет и на это время читателям недоступна? Что делать, если из-за переезда библиотеки расположение книг в шкафах изменилось?)

В решении этих проблем существенную помощь может оказать компьютер, а также специальная программа, которая организует хранение в памяти компьютера сведений о всех книгах библиотеки и позволяет читателям получать необходимые им сведения. Например, такая программа должна уметь выдавать список всех книг данного автора или всех книг с данным названием (если их много). Хорошо бы даже, чтобы она могла, например, выдать список всех книг, в названии которых есть слово «зубофрезерование» или слова «мелиорация болот». Кроме выполнения такого рода запросов, программа должна позволять работникам библиотеки вносить изменения в информацию, хранящуюся в памяти компьютера (например, регистрировать приобретаемые книги).

Такая программа может создать много удобств читателям и работникам библиотеки. Например, читателям уже не понадобится переписывать сведения об интересующей их книге на отдельный листок и нести его библиотекарю. Достаточно будет, выбрав нужное название книги на экране дисплея, нажать специальную клавишу — и заказ на эту книгу будет принят. Но есть и много опасностей, которые могут сделать эксплуатацию такой компьютерной программы бесполезной и даже вредной. Что, если в какой-то момент из-за неполадок в аппаратуре, или ошибки в программе, или просто из-за плохого качества компьютера список книг библио-

теки будет утрачен? Что, если дисплеев для читателей окажется слишком мало и за простейшей справкой придется стоять в очереди? Что, если программа окажется неудобной и читатели будут ежесекундно звать сотрудников библиотеки, требуя научить пользоваться программой? Потенциальные опасности тут столь же велики, как и потенциальные выгоды. Таким образом, внедрение информационных систем требует всестороннего предварительного анализа и решения возникающих проблем.

До сих пор мы предполагали, что сами книги и способ их хранения остаются прежними, а компьютеры применяются лишь в качестве информационно-справочной службы. В будущем может оказаться выгодным хранить книги не напечатанными на бумаге, а записанными в памяти машины. Тогда пользование библиотекой еще упростится. Текст интересующей нас книги будет выдаваться на экран дисплея без всякого участия библиотекаря. Необязательно при этом пользоваться дисплеем, установленным в самой библиотеке. Вместо этого можно соединить стоящий дома компьютер с библиотечным и пользоваться библиотекой, не выходя из дома. Если при этом все библиотеки соединены в единую сеть и могут обмениваться информацией, то мы, не выходя из своего дома, получаем доступ ко всем фондам библиотек этой сети.

Но и здесь одновременно с большими удобствами возникают и опасности, которые трудно сейчас даже предвидеть. Например, если книги перестанут выпускаться в обычном печатном виде, а будут храниться в памяти библиотечных компьютеров, то возрастает опасность уничтожения какой-либо книги.

Приведем еще несколько примеров использования информационно-поисковых систем.

1. Большое значение для развития физики имеют опыты на ускорителях элементарных частиц. Многократно ускоряясь в кольцевых вакуумных камерах огромного диаметра, частицы, сталкиваясь с мишенью или друг с другом, создают новые частицы, которые разлетаются от места столкновения. Специальные средства наблюдения позволяют зафиксировать траектории разлетающихся частиц в виде картинки. Использование ЭВМ при проведении физического эксперимента на ускорителях позволило записывать эти картинки с большой скоростью прямо в память ЭВМ и затем использовать эту же ЭВМ для автоматического анализа изображений и предварительного отбора тех из них, которые могут быть интересны экспериментатору. В современных экспериментах для обнаружения физического явления, ради которого проводится исследование, иногда приходится фиксировать и анализировать сотни тысяч изображений, накапливая их в памяти машины. Без ЭВМ и соответствующих информационно-поисковых систем такие эксперименты были бы невозможны.

2. Волжский автомобильный завод в г. Тольятти ежегодно выпускает около миллиона легковых автомашин «Жигули», «Лада» и «Нива». Сборочный конвейер в главном корпусе этого завода

работает под управлением ЭВМ. Ее задача — следить за своевременным поступлением деталей на конвейер со складов и из цехов вспомогательных производств.

Для выполнения своей работы управляющая ЭВМ программа должна хранить сведения о тысячах деталей, образующих автомобиль, о наличии деталей на складах, об их движении по транспортным линиям, о поступлении на рабочие места конвейера и т. д.

3. В настоящее время во многих населенных пунктах нашей страны существует автоматическая междугородная телефонная связь. При обеспечении этой связи серьезной проблемой является организация учета длительности разговора и последующей оплаты за разговор. На междугородней АТС средних размеров приходится одновременно поддерживать несколько сот междугородных телефонных разговоров, а общее число абонентов может насчитывать больше ста тысяч.

Для каждого абонента нужно знать его номер телефона, фамилию, имя и отчество и почтовый адрес. Для каждого телефонного разговора нужно зафиксировать его продолжительность, запомнить номера телефонов говорящих, найти нужный тариф, подсчитать стоимость разговора, а затем автоматически подготовить бланк счета за телефонный разговор и напечатать его в виде служебного письма, которое потом будет послано по почте абоненту. Все это стало возможным только благодаря применению ЭВМ.

Численное моделирование физических процессов. В первой части учебника говорилось об использовании компьютеров для моделирования различных процессов. Моделироваться может состояние земной атмосферы, поведение плазмы в термоядерном реакторе, распространение радиоволн, биологическая эволюция, экономическое развитие.

Остановимся на одном из примеров моделирования. Пусть необходимо спроектировать новый пассажирский самолет. Самолет должен обладать заданной грузоподъемностью и быть экономичным — расходовать наименьшее количество топлива. Для обеспечения экономичности нужно, в частности, выбрать самую подходящую форму отдельных деталей самолета и самолета в целом — такую форму, для которой, скажем, сопротивление воздуха при заданной скорости полета было бы наименьшим. Как это сделать? С самого начала самолетостроения и до 50-х годов нашего века использовался один способ: уменьшенные модели нужных деталей помещались в устройство, называемое аэродинамической трубой, через которую с большой скоростью продувался воздух. Такой эксперимент позволял получать достоверные данные о сопротивлении воздуха в реальном полете. К сожалению, необходимым условием получения таких данных является большой размер аэродинамической трубы. Дело в том, что при малых размерах трубы (а значит, и модели самолета) существенно изменяется физическая картина исследуемого явления.

Из сказанного выше видно, что эксперименты с аэродинамической трубой и конструирование самолета очень дороги. Физический процесс обтекания самолета потоком воздуха можно смоделировать совсем иначе, не изготовляя никаких физических моделей и не пользуясь аэродинамической трубой. Дело в том, что еще задолго до первого полета самолета была построена математическая модель полета. Эта модель основана на системе уравнений. Решив ее, мы можем найти, в частности, интересующую нас силу сопротивления воздуха.

Зачем же тогда строились аэродинамические трубы? Неужели конструкторы, которые их использовали, ничего не знали о существовании столь нужной им системы уравнений? Конечно, знали. Дело в том, что сколько-нибудь точно математически описать обтекание воздухом самолета можно, только разбив его поверхность и прилегающий к ней слой воздуха на очень мелкие части и для каждой из этих частей построив свои уравнения. Решение этих уравнений отчасти напоминает процесс нахождения температуры в отдельных точках пластинки, рассмотренный в первой части учебника, только задача гораздо сложнее.

Итак, математическая модель существовала, но инженеры не могли ее применять. Ситуация изменилась с появлением компьютеров. Уже первые из них назывались «быстродействующими» и действительно работали существенно быстрее человека. Все же пришлось немного подождать, пока компьютеры стали использоваться для решения сначала самых простых задач аэродинамики, а потом все более и более сложных.

В 60-е годы ЭВМ использовались для моделирования обтекания воздухом простых деталей. Сейчас наиболее мощные ЭВМ позволяют моделировать обтекание самолета целиком (хотя и для малых скоростей). Потребуется еще несколько лет, чтобы моделирование стало возможным для самолета в целом и сверхзвуковых скоростей. Однако уже сейчас многие из используемых самолетов были бы созданы со значительно большими затратами времени и материальных ресурсов, если бы не помощь ЭВМ.

Станки с числовым программным управлением. Все, конечно, видели в своей жизни винты, но задумывались ли о том, как их изготавливают? Один из возможных способов таков. Круглую заготовку, на которой нужно нарезать резьбу, зажимают в патрон токарного станка и вращают. В это время резец движется параллельно оси заготовки, прорезая в ней винтовую канавку. При этом необходимо, чтобы за то время, пока заготовка повернется на один оборот, резец сдвигался на строго определенное расстояние (называемое шагом резьбы). Если мотор, вращающий заготовку, начнет крутиться чуть быстрее, то и резец должен начать двигаться быстрее.

Как же этого достичь? Давно известный способ состоит в том, чтобы двигать заготовку и резец с помощью одного и того же мотора. При этом механизм, передвигающий резец, с помощью зубча-

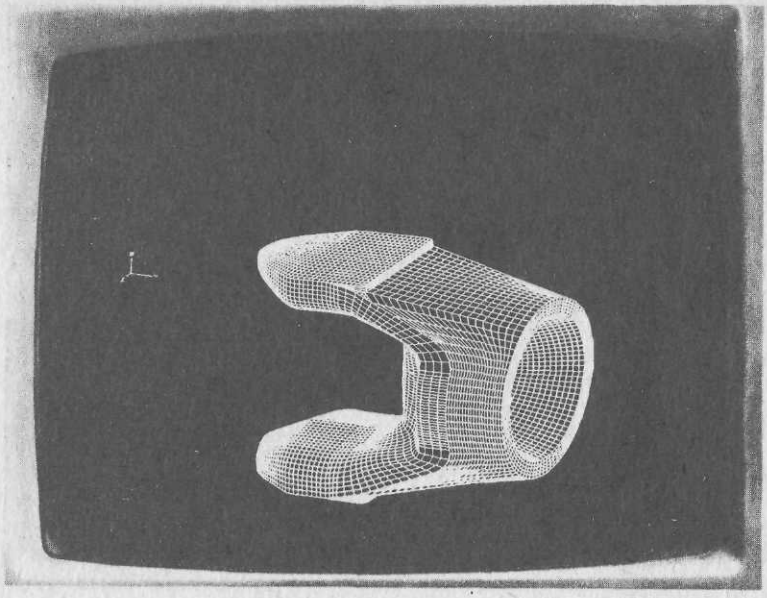


Рис. 26

тых колес соединяется с механизмом, вращающим заготовку, и работают они согласованно.

Но шаг резьбы может потребоваться разный. Из-за этого необходимо иметь в станке систему зубчатых колес, которые могли бы зацепляться по-разному, давая разные соотношения между скоростью вращения заготовки и скоростью движения резца. Настройка такой системы на данный шаг резьбы — непростое дело. (Да и не всякий шаг можно получить таким способом.)

Как видим, даже при изготовлении такой простой вещи, как винт, встречается много трудностей. А современные изделия неизмеримо сложнее, и для их изготовления еще больше разных частей станка должно двигаться согласованно. Добиться такой согласованности чрезвычайно трудно. Если, например, нужно изготовить криволинейную поверхность заданной формы, то часто вначале вручную изготовляют модель, имеющую такую поверхность, а только затем с ее помощью на копировальном станке делают нужную деталь.

Современная технология позволяет упростить этот процесс. Рассчитав нужную деталь с помощью компьютера (рис. 26), можно с помощью того же компьютера рассчитать, как должен двигаться резец и остальные части станка, чтобы такую деталь изготовить. Далее нужно дать компьютеру возможность управлять этими движениями. Для этого используются специальные двигатели, перемещающие части станка по командам ЭВМ. Оснащенные таким образом станки называются станками с ЧПУ (числовым

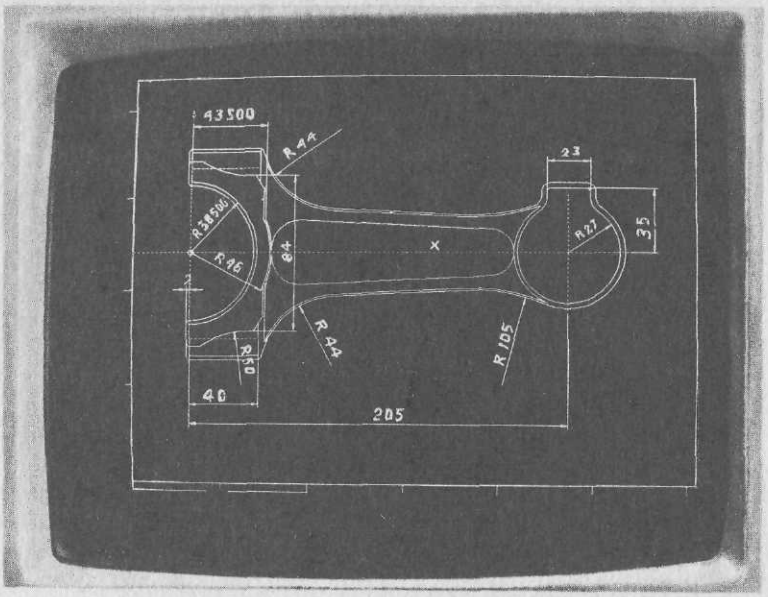


Рис. 27

программным управлением). Применение таких станков совместно с компьютерами позволяет изготавливать сложные детали, причем переход от изготовления одного типа деталей к другому осуществляется легко и быстро путем смены программы в управляющей ЭВМ.

Системы машинной графики. Прежде чем станет возможным изготовление детали на станке с ЧПУ, необходимо создать в компьютере модель детали. Эта модель играет ту же роль, что и чертеж при изготовлении детали традиционным способом. При создании таких моделей применяются специальные устройства. Опишем некоторые из них.

Световое перо имеет вид авторучки, соединенной проводом с ЭВМ. На кончике светового пера находится электронное устройство, при поднесении которого к выбранной точке экрана в ЭВМ поступает информация о координатах точки на экране, к которой поднесено световое перо. Двигая перо, можно указывать на экране нужные точки, рисовать линии и т. д.

Шаровой рычаг имеет вид шарового шарнира, от которого отходит стержень. Поворот стержня поворачивает шаровой шарнир, и информация об углах поворота (вперед, назад, влево, вправо) передается в ЭВМ.

«Мышь» имеет вид коробочки, лежащей на поверхности стола и соединенной проводом с ЭВМ. Коробочка свободно катается по столу. При ее движении информация о положении «мыши» поступает в ЭВМ.

С помощью «мыши» можно также рисовать линии, например перемещая ее по поверхности стола.

Опишем в общих чертах работу при подготовке модели детали с помощью ЭВМ. Перемещая, например, «мышь» и вводя с клавиатуры числа, человек может создать чертеж интересующей его детали (рис. 27) примерно так же, как это делается с помощью циркуля и линейки.

Однако можно не только построить чертеж. Возможности ЭВМ позволяют создать объемную модель детали, рассматривать ее с разных точек зрения (рис. 28 и 29) и даже строить поверхность детали автоматически по нескольким основным линиям, которые указывает человек (рис. 30, 31).

Когда модель уже создана, можно с помощью ЭВМ заняться ее анализом. Например, можно задать нагрузки, которые будут действовать на шатун (см. рис. 27), произвести прочностной расчет и узнать, выдержит ли шатун эти нагрузки, в каких местах его надо усилить, а где, наоборот, можно сэкономить часть металла.

После этого ЭВМ может, как мы уже говорили, изготовить деталь «в натуре», управляя станками с ЧПУ. Таким образом, при использовании ЭВМ чертежи становятся ненужными. Такую технологию принято называть *безбумажной*.

Чего не могут ЭВМ. Развитие информатики и вычислительной техники за последние 40 лет превзошло самые смелые ожидания и опрокинуло самые оптимистические прогнозы. Создатели первых ЭВМ не поверили бы, если бы услышали, что через несколько десятков лет компьютеры станут в тысячи раз быстрее и в тысячи раз дешевле. В то же время в области применения компьютеров иногда наблюдается противоположная картина. Некоторые задачи, о которых еще на заре «компьютерной эры» говорилось как о доступных машинам, до сих пор остаются недоступными компьютерам, и прогресс в их решении гораздо меньше ожидавшегося. Представление о таких задачах помогает понять, каковы возможности компьютеров, каковы границы этих возможностей и где использование компьютеров наиболее целесообразно.

Посмотрев на фотографию, где собака играет со щенками, легко можно сказать, где на снимке собака, где щенки и сколько видно щенков. Прослушав песенку один или несколько раз, можно записать ее слова. Каждый человек может разобрать свой почерк и даже почерк своих товарищей. Все эти задачи кажутся совсем простыми, и трудно поверить, что для компьютера (во всяком случае, сегодняшнего) они непосильны. И дело тут не в том, что компьютер «не видит» фотографии или не «слышит» песенки, как раз эта часть задачи проста: изображение и звук нетрудно автоматически закодировать в доступной для ЭВМ форме. Дело в том, что совершенно непонятно, каков должен быть алгоритм обработки закодированных таким образом изображений и звуков. Все мы легко отличаем на фотографии собаку от кошки, но при попытке составить алгоритм, позволяющий отличить закодированную соответствующим

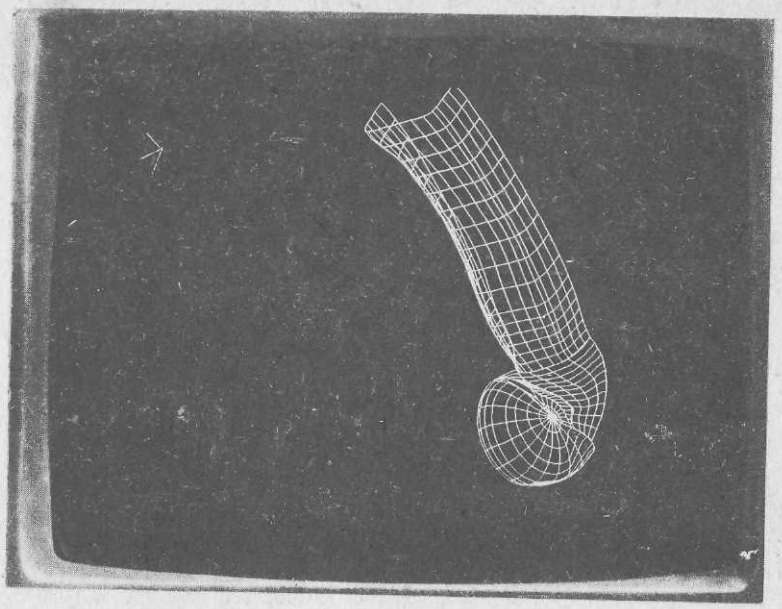


Рис. 28

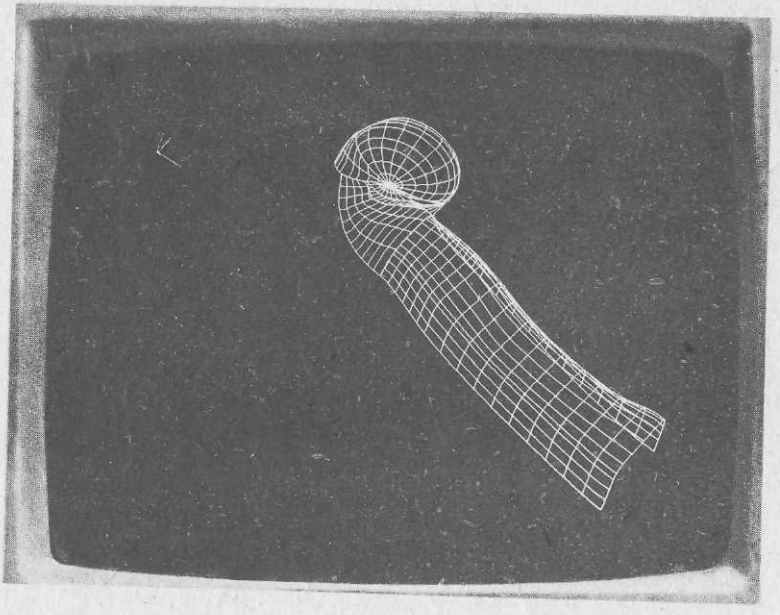


Рис. 29

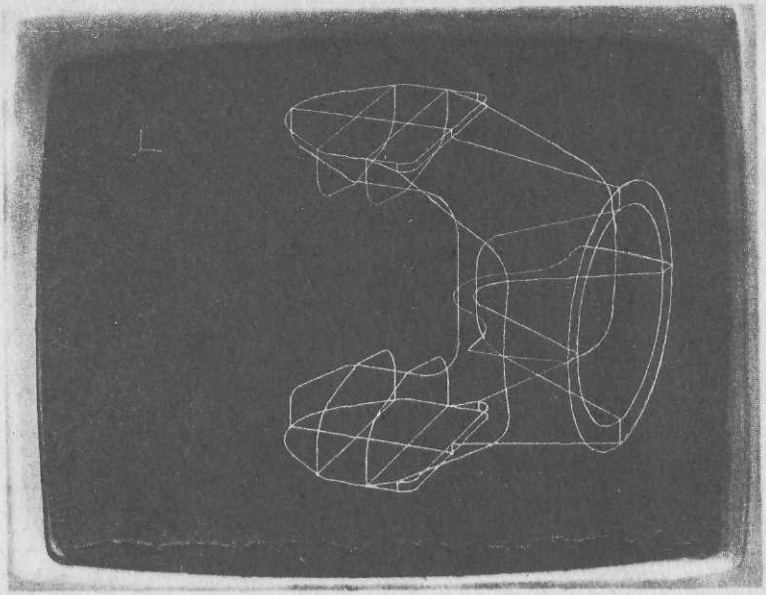


Рис. 30

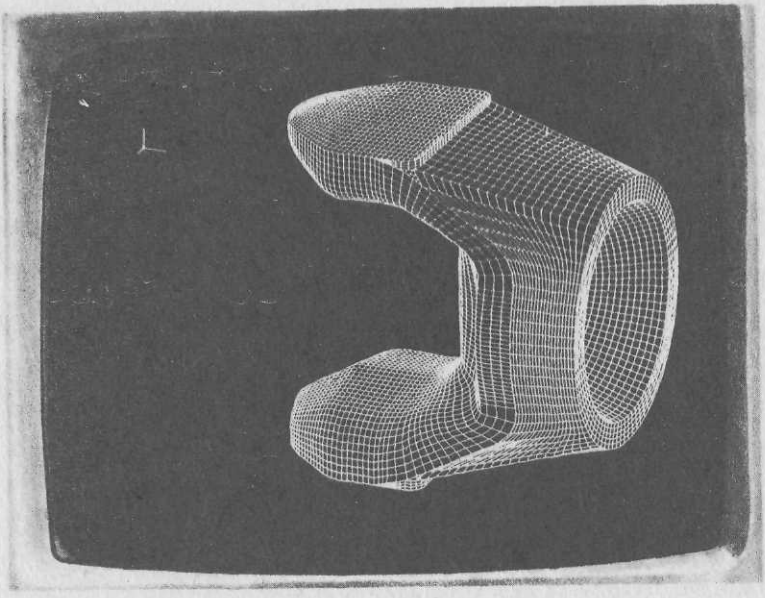


Рис. 31

образом фотографию собаки от фотографии кошки, мы встречаемся с непреодолимыми (во всяком случае, на сегодняшнем уровне развития науки) трудностями.

Таким образом, задачи, легкие для человека, могут оказаться трудными для компьютера, и, напротив, трудные для человека задачи (например, поиск информации в огромном массиве или вычисления большого объема) могут быть простыми для компьютера. Это и хорошо — если бы трудные для нас задачи были бы трудны и для компьютера, то чем бы он нам смог помочь? Подытоживая все сказанное, можно сказать:

> применение компьютера может быть эффективно, если он используется для выполнения рутинных работ большого объема.

Поучительна история попыток написать программу для игры в шахматы. Говоря совсем грубо, можно разделить деятельность шахматистов за доской на две части — расчет вариантов и оценку позиций. Первая из них носит рутинный характер, и компьютер справляется с ней гораздо быстрее человека. Благодаря этому быстро работающие специализированные шахматные компьютеры в состоянии играть на уровне мастера спорта. Однако неумение запрограммировать интуицию шахматиста, которая сразу же подсказывает ему, что некоторая позиция безнадежна или что такой-то ход оптимален, приводит к тому, что более высокий уровень игры пока компьютерам недоступен.

Итак, Вы познакомились с основами информатики и вычислительной техники. Знание этих основ поможет Вам, независимо от того, в какой отрасли народного хозяйства Вы будете трудиться, решать возникающие задачи, грамотно используя ЭВМ.

УПРАЖНЕНИЯ ДЛЯ ПОВТОРЕНИЯ

1. Найдите число ненулевых элементов в таблице **цел таб** $A[1:100]$.

2. Найдите количество элементов в таблице **вещ таб** $A[1:1000]$, абсолютная величина которых больше 7.

3. Составьте алгоритм, дающий ответ «да» или «нет» в зависимости от того, встречается или нет число 7 в таблице **цел таб** $A[1:100]$.

4. Дана целочисленная таблица $A[1:100]$. Найдите разность наибольшего и наименьшего чисел в этой таблице.

5. Даны две целочисленные таблицы $A[1:100]$, $B[1:100]$. Подсчитайте количество тех i, для которых: а) $A[i] < B[i]$; б) $A[i] = B[i]$; в) $A[i] > B[i]$.

6. Дана целочисленная таблица $A[1:100]$. Подсчитайте количество таких i, что $A[i]$ не меньше всех предыдущих элементов таблицы $(A[1], A[2], \ldots, A[i-1])$.

7. Дана таблица **вещ таб** $A[1:100]$. Найдите количество элементов этой таблицы, больших среднего арифметического всех ее элементов.

8. Дана целочисленная таблица $A[1:100]$. Заполните вещественную таблицу $B[1:100]$, i-й элемент которой равен среднему арифметическому первых i элементов таблицы A:

$$B[i] = (A[1] + \ldots + A[i])/i.$$

9. Дана целочисленная таблица $A[1:100]$. Подсчитайте, сколько раз встречается в этой таблице максимальное по величине число.

10. Дана целочисленная прямоугольная таблица $A[1:100, 1:50]$. Измените все элементы этой таблицы на противоположные по знаку.

11. Дана целочисленная прямоугольная таблица $A[1:100, 1:50]$. Найдите наибольшее из чисел, встречающихся в этой таблице.

12*[1]. Дана целочисленная таблица $A[1:100]$. Проверьте, есть ли в ней элементы, равные нулю. Если есть, найдите номер первого из них, т. е. наименьшее i, при котором $A[i] = 0$.

13*. Дана целочисленная таблица $A[1:100]$. Проверьте, есть ли в ней отрицательные элементы. Если есть, найдите наибольшее i, при котором $A[i] < 0$.

14*. Проверьте, является ли прямоугольная таблица **цел таб** $A[1:100, 1:100]$ «магическим квадратом» (это значит, что суммы чисел во всех ее вертикалях, всех горизонталях и двух диагоналях одинаковы).

15*. Дана целочисленная таблица $A[1:1000]$. Подсчитайте наибольшее число одинаковых идущих в ней подряд элементов.

16*. Подсчитайте количество различных чисел, встречающихся в таблице **цел таб** $A[1:100]$. Повторяющиеся числа учитывайте один раз.

17*. Дана таблица **цел таб** $A[1:1000]$. Постройте таблицу **цел таб** $B[1:1000]$, которая содержит те же числа, что и таблица A, но в которой все отрицательные элементы предшествуют всем неотрицательным.

18*. Даны целочисленные таблицы $A[1:100]$, $B[1:100]$, причем

$$A[1] \leqslant A[2] \leqslant \ldots \leqslant A[100],$$
$$B[1] \leqslant B[2] \leqslant \ldots \leqslant B[100].$$

Постройте таблицу **цел таб** $C[1:200]$, содержащую все элементы таблиц A и B, в которой

$$C[1] \leqslant C[2] \leqslant \ldots \leqslant C[200].$$

19*. Дана прямоугольная целочисленная таблица $A[1:100, 1:50]$. Найдите количество тех чисел i от 1 до 100, для которых $A[i, j] = 0$ при некотором j от 1 до 50.

20*. Дана прямоугольная целочисленная таблица $A[1:100, 1:50]$. Найдите наименьшее целое число K, обладающее таким свойством: хотя бы в одной строке таблицы все элементы не превосходят K.

21*. Дана прямоугольная целочисленная таблица $A[1:100, 1:50]$. Найдите наибольшее целое число K, обладающее таким свойством: в любой строке таблицы есть элемент, больший или равный K.

22*. Постройте алгоритм разложения натуральных чисел на простые множители. В какой форме будут представлены результаты работы этого алгоритма?

23*. Натуральное число называют *совершенным*, если оно равно сумме всех своих делителей, не считая его самого (например,

[1] Знаком * отмечены трудные задачи.

$6 = 1 + 2 + 3$ — совершенное число). Напишите алгоритм, проверяющий, является ли заданное число совершенным.

24*. Напечатайте в порядке возрастания первые 1000 чисел, которые не имеют простых делителей, кроме 2, 3 и 5. (Начало списка: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ...)

25*. Придумайте способ нахождения самой легкой и самой тяжелой из 100 монет различной массы, если можно сделать не более 150 взвешиваний на чашечных весах без гирь. (На каждую чашку весов помещается по одной монете; весы показывают, какая из них тяжелее.)

26*. Имеется 1000 монет, из которых одна фальшивая (она легче других). Докажите, что нельзя придумать способ, гарантирующий нахождение фальшивой монеты за 6 взвешиваний на чашечных весах без гирь. Придумайте способ, гарантирующий нахождение фальшивой монеты за 7 взвешиваний.

27*. Среди $2N + 1$ различных по массе монет нужно найти среднюю (т. е. такую, которая тяжелее N монет и легче N других монет). Придумайте способ, позволяющий сделать это не более чем за $100N$ взвешиваний на чашечных весах без гирь.

28*. Последовательность a_1, a_2, a_3, ... определяется так:

$$a_1 = 1,$$
$$a_{2n} = a_n,$$
$$a_{2n+1} = a_n + a_{n+1}.$$

Напишите алгоритм, вычисляющий a_n по известному n.

29*. Дана целочисленная таблица $A[1:1000]$. Найдите наименьшее число K элементов, которые нужно выкинуть из последовательности $A[1]$, $A[2]$, ..., $A[1000]$, чтобы осталась возрастающая последовательность.

30*. Даны три целочисленные таблицы $A[1:1000]$, $B[1:1000]$, $C[1:1000]$. Известно, что существуют целые числа, встречающиеся во всех трех таблицах. Найдите одно из таких чисел.

31*. Даны две литерные величины A и B. Проверьте, можно ли из букв, входящих в A, составить B. (Буквы можно переставлять, но каждую букву можно использовать не более одного раза.)

32*. Составьте алгоритм, находящий по целым величинам A, B и C такие целые числа X и Y, что $AX + BY = C$ (если такие X и Y есть).

33*. Перестановкой K чисел называется последовательность $A[1]$, $A[2]$, ..., $A[K]$, в которой встречаются по одному разу все числа от 1 до K. (Например, перестановками трех чисел являются: (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1) и других перестановок трех чисел нет.) Постройте алгоритм, печатающий все перестановки 100 чисел. Используйте в качестве вспомогательного алгоритм «печать (**цел таб** $A[1:100]$)», печатающий элементы таблицы A.

34*. Дана таблица **нат таб** $A[1:100]$. Найдите наименьшее натуральное число X, не обладающее следующим свойством: «можно так вычеркнуть некоторые числа таблицы A, чтобы сумма оставшихся равнялась X».

35*. Составьте алгоритм подсчета числа способов, которыми можно уплатить K рублей купюрами в 1, 3, 5, 10, 25, 50 и 100 рублей.

36*. В стране имеется 50 городов, соединенных авиационным сообщением. Стоимость билета из i-го города в j-й записана в прямоугольной таблице *цена*$[1:50, 1:50]$. Составьте алгоритм, находящий стоимость перелета из i-го города в j-й по самому дешевому маршруту (возможно, с пересадками).

37*. Существует способ обойти шахматным конем шахматную доску, побывав на каждом поле по одному разу. Составьте алгоритм отыскания такого способа.

38*. Существуют способы расстановки 8 ферзей на шахматной доске так, чтобы они не били друг друга. Составьте алгоритм, отыскивающий один из таких способов.

39*. Некоторые натуральные числа могут быть представлены в виде суммы кубов целых неотрицательных чисел: например, $9 = 2^3 + 1^3$, $27 = 3^3 + 0^3$. Составьте алгоритм, отыскивающий наименьшее натуральное число, имеющее два разных таких представления. (Представления $9 = 2^3 + 1^3 = 1^3 + 2^3$ считаются одинаковыми.)

40*. Составьте алгоритм, вычисляющий по заданным вещественным числам a, b, c, d величины $ac + bd$ и $ad - bc$. Этот алгоритм может использовать промежуточные величины, операции сложения, вычитания и умножения, причем умножение должно выполняться не более трех раз.

ПРИЛОЖЕНИЕ

I. ОСНОВНЫЕ КОНСТРУКЦИИ АЛГОРИТМИЧЕСКОГО ЯЗЫКА

1. Общий вид алгоритма.

алг название (список аргументов и результатов с указанием типов)
арг перечень аргументов
рез перечень результатов

нач
| список промежуточных величин
| серия
кон

алг КВУР (**вещ** a, b, c, **вещ** x_1, x_2, **лит** y)
арг a, b, c
рез x_1, x_2, y
нач
| **вещ** D
| $D := b^2 - 4ac$
| **если** $D < 0$
| | **то** $y :=$ "нет решения"
| | **иначе**
| | $y :=$ "есть решения"
| | $x_1 := \dfrac{-b + \sqrt{D}}{2a}$
| | $x_2 := \dfrac{-b - \sqrt{D}}{2a}$
| **все**
кон

В заголовке алгоритма решения квадратного уравнения указано, что этот алгоритм называется КВУР, аргументами его являются вещественные величины a, b, c (коэффициенты квадратного уравнения), а результатами — литерная величина y (есть или нет решения) и вещественные величины x_1, x_2 (корни). Далее, после слова **нач**, указано, что в алгоритме используется вещественная промежуточная величина D. После этого следует серия команд, которые должен выполнить исполнитель алгоритма.

2. Общий вид алгоритма вычисления значений функций.

```
алг тип название (список аргументов с указанием типов)
нач
| список промежуточных величин
| серия
кон
```

```
алг нат НОД (нат a, b)
нач
| нат x, y
| x := a; y := b
| пока x ≠ y
| нц
| | если x > y
| | | то x := x − y
| | | иначе y := y − x
| | все
| кц
| знач := x
кон
```

В заголовке алгоритма вычисления наибольшего общего делителя указано, что функция НОД (a, b) принимает натуральные значения и что аргументы ее также натуральные числа. Далее, после слова **нач**, указано, что используются две промежуточные величины x и y. Затем идут команды, которые должен выполнить исполнитель алгоритма. Последняя из них:

знач $:= x$

указывает, что значением функции является значение величины x.

3. Команда ветвления.

1) Полная форма:

```
если условие
|   то серия 1
|   иначе серия 2
все
```

```
если a ⩾ b
|   то max := a
|   иначе max := b
все
```

Команда ветвления выполняется так. Сначала исполнитель проверяет, соблюдается ли условие. Если оно соблюдается, то выполняется серия 1 и на этом выполнение команды ветвления заканчивается. Если же условие не соблюдается, то выполняется серия 2. В приведенном примере команды ветвления при $a \geqslant b$ значением величины max становится число a, а в противном случае (при $a < b$) значением величины max становится число b.

2) Сокращенная форма:

```
если условие
|   то серия
все
```

```
если x < 0
|   то x := −x
все
```

Команда ветвления в сокращенной форме выполняется так. Сначала исполнитель проверяет, соблюдается ли условие. Если оно соблюдается, то выполняется указанная серия команд. Если же нет, то выполнение команды ветвления на этом заканчивается.

В приведенном примере при $x < 0$ выполняется команда $x := -x$ (в данном случае серия состоит из единственной команды), величина x меняет знак и становится положительной. При $x \geqslant 0$ величина x остается без изменений.

4. Команда выбора.

1) Полная форма:

```
выбор
|   при условие 1 : серия 1
|   при условие 2 : серия 1
|       . . .
|   при условие n : серия n
|   иначе серия
все
```

```
выбор
|     при a > 100:
|         b := "очень большое"
|     при a > 10:
|         b := "большое"
|     иначе b := "небольшое"
все
```

Команда выбора выполняется так. Одно за другим проверяются условия команды. Как только исполнитель находит условие, которое соблюдается, он выполняет серию идущих за ним команд (и на этом выполнение команды выбора заканчивается). Если ни одно из условий не соблюдается, то выполняется серия команд, идущая после слова **иначе.**

В нашем примере при $a > 100$ соблюдается первое условие и значением величины b становится текст "очень большое". При $10 < a \leqslant 100$ первое условие не соблюдается, но соблюдается второе и значением величины b становится текст "большое". Наконец, при $a \leqslant 10$ не соблюдается ни одно из условий и значением величины b становится текст "небольшое".

2) Сокращенная форма:

```
выбор
|     при условие  1: серия 1
|     при условие  2: серия 2
|
|     при условие  n: серия n
все
```

```
выбор
|     при a[i] < 0: отр := отр + 1
|     при a[i] = 0: нул := нул + 1
|     при a[i] > 0: пол := пол + 1
все
```

В сокращенной форме команды выбора отсутствует слово **иначе** и серия идущих за ним команд. В этом случае, если ни одно из условий команды выбора не соблюдается, выполнение команды выбора на этом заканчивается.

5. Команда повторения.

пока условие	**пока** $x \geqslant 10$
нц	**нц**
серия	$x := x - 10$
кц	**кц**

При выполнении команды повторения входящая в нее серия команд повторяется столько раз, сколько нужно, чтобы условие перестало соблюдаться. (Если условие не соблюдается с самого начала, то серия не выполняется ни разу.) В нашем примере значение величины x будет уменьшаться каждый раз на 10, пока оно не станет меньше 10. Если начальное значение переменной x — целое положительное число, то после выполнения команды повторения ее новым значением станет остаток от деления исходного числа на 10.

6. Команда повторения с параметром.

для x **от** X_{min} **до** X_{max} **шаг** $x_{шаг}$	**для** i **от** -2 **до** 11 **шаг** 3
нц	**нц**
серия	$y := y + i$
кц	**кц**

В команде повторения с параметром x — целочисленная переменная, называемая *параметром,* а X_{min}, X_{max} и $x_{шаг}$ — целочисленные выражения, причем $x_{шаг}$ должно быть больше 0. Выполняется команда так: входящая в нее серия выполняется для последовательности значений $i = X_{min}$, $X_{min} + x_{шаг}$, $X_{min} + 2x_{шаг}$, которая продолжается до тех пор, пока не будет пройдено значение X_{max}. В нашем примере команда

$$y := y + i$$

будет выполнена для $i = -2$, затем для $i = 1$, затем для $i = 4$, $i = 7$ и, наконец, для $i = 10$. Это значение i будет последним, поскольку следующее значение $i = 13$ уже больше граничного значения 11.

Наиболее употребительна команда повторения с параметром, в которой значение $x_{шаг}$ равно 1. В этом случае слово «**шаг** 1» можно опустить и писать, например,

для i **от** 1 **до** 10		**для** i **от** 1 **до** 10 **шаг** 1
нц	вместо	**нц**
$S := S + i$		$S := S + i$
кц		**кц**

В этом примере команда $S := S + i$ будет выполнена последовательно при $i = 1$, $i = 2$, ..., $i = 10$.

7. Команда присваивания.

имя : = выражение

В команде присваивания *имя* — имя некоторой величины. В результате выполнения команды значение этой величины становится равным указанному в команде выражению, которое должно иметь соответствующий тип. При записи выражений могут быть использованы операции и функции, приведенные в таблице 20, а также функции, алгоритмы вычисления значений которых включены в библиотеку вспомогательных алгоритмов.

Таблица 20

Название операции или функции	Форма записи
сложение	$x+y$
вычитание	$x-y$
умножение	$x*y$
деление	x/y
возведение в степень	$x**y$
взятие элемента линейной таблицы	$x[i]$
взятие элемента прямоугольной таблицы	$x[i,j]$
извлечение корня	sqrt (x)
синус	sin (x)
косинус	cos (x)
тангенс	tg (x)
арксинус	arcsin(x)
арккосинус	arccos (x)
арктангенс	arctg (x)
десятичный логарифм	lg (x)
соединение строк	$s+t$
вырезка	$s[i:j]$

В левой части команды присваивания может стоять выражение:

$x[i]$, если x — линейная таблица,
$y[i, j]$, если y — прямоугольная таблица,
$s[i:j]$, если s — литерная величина.

Например, при выполнении команды

$x[3] := 7$

элемент $x[3]$ линейной таблицы x становится равным 7. При выполнении команды

$$y[5, 4] := y[5, 4] + 1$$

значение элемента $y[5, 4]$ прямоугольной таблицы y увеличивается на 1.

При выполнении команды

$$s[5:7] := \text{"абв"}$$

значение литерной переменной s меняется: пятой буквой становится буква "а", шестой — буква "б" и седьмой — буква "в".

8. Команда вызова вспомогательного алгоритма.

имя (список аргументов и результатов)

Имеющийся в библиотеке алгоритм можно использовать, записав команду вызова этого алгоритма. Для этого указывается его имя, аргументы, к которым его нужно применять, и величины, которые будут его результатами.

Например, заголовок алгоритма решения квадратного уравнения $ax^2 + bx + c = 0$ был такой:

алг КВУР (**вещ** a, b, c, **вещ** x_1, x_2, **лит** y)

арг a, b, c

рез x_1, x_2, y

В результате вызова

КВУР (1, 5, $2*m + 1$, p, q, *ответ*)

алгоритм будет применен к уравнению с коэффициентами $a = 1$, $b = 5$, $c = 2*m + 1$, значение переменной *ответ* будет "есть решения" или "нет решения" в зависимости от наличия или отсутствия решений у уравнения $x^2 + 5x + (2m + 1) = 0$, и, если решения есть, они станут значениями переменных p и q.

Если вспомогательный алгоритм вычисляет значение функции, то вызов этого вспомогательного алгоритма происходит, когда при вычислении значения выражения встречается эта функция. Например, при выполнении команды

$$y := \text{abs}(a + 1) + \text{abs}(\text{abs}(a) - b)$$

трижды вызывается вспомогательный алгоритм abs, а при выполнении команды

КВУР (1,5, abs($2*m + 1$), p, q, *ответ*)

перед вызовом вспомогательного алгоритма КВУР происходит вызов вспомогательного алгоритма abs.

9. Запись условий в алгоритмическом языке.

Простое условие в алгоритмическом языке имеет вид:

выражение знак выражение

Например, $a + b > c + d$

Составное условие состоит из простых, соединенных служебными словами **и**, **или**, **не**.

Примеры составных условий:

$$x \leqslant -1 \text{ **или** } x \geqslant 1$$
$$x > -1 \text{ **и** } x \leqslant 2$$
$$\text{**не** } (x < 2 \text{ **и** } x \neq 0)$$

Выражения в условиях составляются по тем же правилам, что и в команде присваивания. В качестве знака можно использовать $=$ и $\neq$, а также (для выражений с целыми и вещественными значениями) знаки $<$, $\leqslant$, $>$, $\geqslant$. Условия встречаются в командах ветвления, выбора и повторения.

II. БИБЛИОТЕКА АЛГОРИТМОВ

1. Алгоритм вычисления абсолютной величины действительного числа.

```
алг вещ abs (вещ x)
нач
| если x ⩾ 0
| | то знач:= x
| | иначе знач:= −x
| все
кон
```

2. Алгоритм нахождения максимальной из двух величин.

```
алг вещ max 2 (вещ a, b)
нач
| если a ⩾ b
| | то знач:= a
| | иначе знач:= b
| все
кон
```

3. Алгоритм решения квадратного уравнения $ax^2 + bx + c = 0$, где a, b, c — произвольные вещественные числа ($a \neq 0$).

```
алг КВУР (вещ a, b, c, вещ x1, x2, лит y)
    арг a, b, c
    рез x1, x2, y
нач
    вещ D
    D := b^2 - 4ac
    если D < 0
        то y := "нет решения"
        иначе
            y := "есть решения"
            x1 := (-b + sqrt(D)) / 2a
            x2 := (-b - sqrt(D)) / 2a
    все
кон
```

4. Алгоритм Евклида вычисления наибольшего общего делителя двух натуральных чисел.

```
алг нат НОД (нат m, n)
нач
    нат x, y
    x := m
    y := n
    пока x ≠ y
    нц
        если x > y
            то x := x - y
            иначе y := y - x
        все
    кц
    знач := x
кон
```

5. Алгоритм вычисления факториала натурального числа.

```
алг цел факториал (нат n)
нач цел i
|   знач := 1
|   для i от 1 до n
|   нц
|   |   знач := знач * i
|   кц
кон
```

6. Алгоритм нахождения остатка при делении целого положительного числа *(делимое)* на целое положительное число *(делитель)*.

```
алг цел остаток (цел делимое, делитель)
нач
|   знач := делимое
|   пока знач ⩾ делитель
|   нц
|   |   знач := знач — делитель
|   кц
кон
```

Пояснение. В этом алгоритме из величины *делимое* вычитают *делитель* до тех пор, пока это можно (т. е. остается неотрицательное число). После этого **знач** равно остатку от деления делимого на делитель.

7. Алгоритмы деления с остатком целых положительных чисел.

```
алг деление с остатком (цел делимое, делитель, цел частное,
    остаток)
     арг делимое, делитель
     рез частное, остаток
нач
|   частное := 0; остаток := делимое
|   пока остаток ⩾ делитель
|   нц
|   |   остаток := остаток — делитель; частное := частное + 1
|   кц
кон
```

Пояснение. Между выполнениями серии, входящей в команду повторения, соблюдается условие

$$делимое = делитель * частное + остаток.$$

После выполнения команды повторения это условие продолжает соблюдаться и, кроме того, условие $остаток \geqslant делитель$ не соблюдается, т. е. $остаток < делитель$, что и требовалось.

8. Алгоритм подсчета количества отрицательных, нулевых и положительных элементов в таблице.

```
алг подсчет (цел таб a[1 : 100], цел отр, нул, пол)
    арг a
    рез отр, нул, пол
нач цел i
|   отр: = 0; нул: = 0; пол: = 0
|   для i от 1 до 100
|   нц
|   |   выбор
|   |   |       при a[i] < 0: отр: = отр + 1
|   |   |       при a[i] = 0: нул: = нул + 1
|   |   |       при a[i] > 0: пол: = пол + 1
|   |   все
|   кц
кон
```

9. Алгоритм нахождения наименьшего делителя целого положительного числа x (не считая единицы).

```
алг цел наименьший делитель (цел x)
нач
|   цел i
|   i : = 2
|   пока остаток (x, i) ≠ 0
|   нц
|   |   i: = i + 1
|   кц
|   знач: = i
кон
```

Пояснение. В этом алгоритме значение i увеличивается до тех пор, пока не найдено такое значение i, при котором остаток $(x, i) = 0$, т. е. x делится на i. Этот алгоритм использует алгоритм 6 как вспомогательный.

Замечание. Приведенные алгоритмы (нахождение остатка, наименьшего делителя) далеко не наилучшие в отношении количества действий, которые должен проделать исполнитель.

10. Алгоритм замены букв а на буквы б

```
алг замена а на б (лит X)
    арг X
    рез X
нач
|   цел i
|   для i от 1 до длин (X)
|    нц
|    |  если X[i : i] = "а"
|    |     то X[i : i]: = "б"
|    |  все
|    кц
кон
```

III. ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ ОБ УСТРОЙСТВЕ ЭВМ

1. Работа с информацией в памяти ЭВМ. В первом разделе мы уже знакомились с устройством и работой ЭВМ ДВК-2М и аналогичных. До сих пор, однако, во всех наших примерах аргументы, результаты и промежуточные величины программы располагались в регистрах процессора. В памяти ЭВМ располагалась только сама программа. Познакомимся теперь с командами, работающими с памятью ЭВМ. По сути дела, это уже знакомые нам команды (*"переслать слово"*, *"добавить слово"* и др.). То, что информация находится в памяти ЭВМ, указывается очень просто: если в команде имя регистра взято в скобки, то это означает, что в регистре содержится не сама информация, а ее адрес в памяти. Таким образом, команда

переслать слово R0 в (R1)

пересылает информацию из регистра R0 в память по адресу, записанному в регистре R1. Команда

переслать слово (R0) в (R1)

пересылает информацию из памяти в память, а команда

добавить слово (R0) к R1

увеличивает содержимое регистра R1 на число, хранящееся по адресу, указанному в R0.

Рассмотрим пример программы, читающей информацию из памяти ЭВМ. Пусть первые три элемента линейной таблицы целых чисел расположены в ячейках памяти с адресами 2000, 2002 и 2004 и надо поместить в регистр R1 их сумму. Будем считать, что в начальный момент СК = 1000, R0 = 2000, R2 = 2 и расположим соответствующую программу в памяти ЭВМ, начиная с адреса 1000 (табл. 21):

Таблица 21

очистить регистр R1	1000
добавить слово (R0) к R1	1002
добавить слово R2 к R0	1004
добавить слово (R0) к R1	1006
добавить слово R2 к R0	1008
добавить слово (R0) к R1	1010
стоп	1012

Приведем таблицу значений (табл. 22) для этой программы, считая, что по адресам 2000, 2002, 2004 находятся числа 5, 6, 7.

Таблица 22

Шаг	Команда	СК	R0	R1	R2
		1000	2000		2
1	очистить регистр R1	1002		0	
2	добавить слово (R0) к R1	1004		5	
3	добавить слово R2 к R0	1006	2002		
4	добавить слово (R0) к R1	1008		11	
5	добавить слово R2 к R0	1010	2004		
6	добавить слово (R0) к R1	1012		18	
7	стоп	1014			

Команды для работы с табличными величинами. Заметьте, что в предыдущей программе после добавления к R1 слова по адресу, записанному в R0 (шаги 2 и 4), регистр R0 сразу увеличивается на 2 (шаги 3 и 5). Таким образом, при исполнении этой программы

регистр R0 «скользит» по адресам памяти, переходя от одного элемента целочисленной таблицы к другому. Такое скольжение часто используется при работе с табличными величинами, и для его организации существуют специальные команды: если в любой известной нам команде имя регистра не только заключить в скобки, но и написать после скобок знак «+», то процессор не только возьмет содержимое ячейки по адресу, записанному в регистре, но и автоматически увеличит содержимое этого регистра на 2, т. е. перейдет к следующему слову. Например, по команде

добавить слово (R0) + к R1

процессор не только добавит к R1 содержимое ячейки по адресу, записанному в R0, но и увеличит R0 на 2, т. е. перейдет к следующему элементу таблицы.

Таким образом приведенную выше программу можно записать короче, не используя регистр R2 (табл. 23):

Таблица 23

очистить регистр R1	1000
добавить слово (R0)+ к R1	1002
добавить слово (R0)+ к R1	1004
добавить слово (R0) к R1	1006
стоп	1008

Приведем теперь программу, которая не только читает, но и записывает информацию в память ЭВМ. Пусть требуется скопировать линейную таблицу чисел, находящуюся в 200 ячейках памяти, начиная с адреса 2000, в другую таблицу из 200 слов, начинающуюся с адреса 3000. Будем считать, что в начальный момент СК = 1000, R0 = 2000, R1 = 3000, R2 = 200, и расположим программу в памяти ЭВМ, начиная с адреса 1000 (табл. 24):

Таблица 24

переслать слово (R0) + в (R1) +	1000
уменьшить слово R2 на единицу	1002
если больше, переход на – 3 слова	1004
стоп	1006

В процессе работы этой программы R0 содержит адрес очередного еще не скопированного элемента таблицы (слова), R1 — адрес, куда надо переслать это слово, а R2 — число еще не скопи-

рованных слов (элементов таблицы). Перед началом работы программы R2 = 200, т. е. ни один элемент таблицы еще не скопирован. По окончании работы программы R2 = 0, т. е. все элементы таблицы скопированы.

Установка начальных значений регистров. До сих пор мы всегда предполагали, что перед началом работы программы в регистры уже записаны нужные адреса и числа. Например, в предыдущей программе предполагалось, что R0 = 2000, R1 = 3000, R2 = 200. Поместить нужную информацию в регистры может и сама программа (табл. 25):

Таблица 25

переслать слово (СК) + в R0	1000
2000	1002
переслать слово (СК) + в R1	1004
3000	1006
переслать слово (СК) + в R2	1008
200	1010
переслать слово (R0) + в (R1) +	1012
уменьшить слово R2 на единицу	1014
если больше, переход на – 3 слова	1016
стоп	1018

В этой программе по адресам 1002, 1006 и 1010 записаны не команды, а информация, которая в процессе работы программы будет помещена в регистры.

Первая команда этой программы

переслать слово (СК) + в R0

отличается от рассмотренной выше команды

переслать слово (R0) + в R1

только именами регистров. При выполнении этой первой команды процессор в соответствии с основным алгоритмом:

1) читает адрес из СК (1000);

2) читает слово из памяти по этому адресу (команду "переслать слово (СК) + в R0");

3) увеличивает СК на 2 (до 1002);

4) выполняет прочитанную команду. (Поскольку в команде написано (СК) +, а в этот момент СК = 1002, то процессор пере-

сылает в R0 содержимое слова памяти по адресу 1002, сам регистр СК увеличивает на 2 (до 1004).

В соответствии с основным алгоритмом на следующем шаге процессор выполнит команду, адрес которой (1004) записан в СК, т. е. перешлет 3000 в регистр R1. Затем он перешлет 200 в R2, после чего оставшаяся часть программы будет исполняться в точности так, как было описано выше.

2. Подпрограммы. В первой части курса мы знакомились с вспомогательными алгоритмами и командами их вызова. Теперь покажем, как записываются и вызываются вспомогательные программы — *подпрограммы*.

Подпрограмма очень похожа на программу — она также состоит из команд и также должна быть размещена в памяти ЭВМ. Единственное отличие заключается в том, что в конце подпрограммы вместо команды *стоп* должна стоять специальная команда

возврат из подпрограммы.

Предположим, что мы составили подпрограмму, поместили ее в память, начиная с адреса 1400 (этот адрес — адрес первой команды подпрограммы — называется *адресом подпрограммы*). Тогда вызвать эту подпрограмму можно с помощью команды

вызов подпрограммы,

вслед за которой в памяти ЭВМ расположен адрес подпрограммы (табл. 26):

Таблица 26

вызов подпрограммы	1000
1400	1002

(Будем считать, что команда вызова подпрограммы расположена в ячейке с адресом 1000.)

Как и любая другая, команда "вызов подпрограммы" выполняется за один шаг работы процессора и в соответствии с основным алгоритмом. Мы уже знаем, что при работе процессора по основному алгоритму регистр R7, т. е. СК, играет особую роль. При работе с подпрограммой особую роль будет играть и еще один регистр — R6. При выполнении команды "вызов подпрограммы" процессор

1) читает адрес из СК (1000);
2) читает команду из ячейки памяти по этому адресу (вызов подпрограммы);
3) увеличивает СК на 2 (до 1002);
4) выполняет полученную команду, а именно:
   а) получает адрес подпрограммы, т. е. берет содержимое СК (1002), читает слово по этому адресу (1400) и увеличивает СК на 2 (до 1004);
   б) уменьшает R6 на 2;

в) записывает СК (1004) в ячейку памяти по адресу, указанному в R6;

г) записывает в СК адрес подпрограммы (1400).

Таким образом, после выполнения этой команды *адрес возврата* (т. е. 1004— адрес первой следующей за "вызов подпрограммы" команды) будет записан в некоторую ячейку памяти. Адрес самого этого слова будет содержаться в регистре R6, а в СК будет находиться адрес первой команды подпрограммы.

В соответствии с основным алгоритмом на следующем шаге процессор выполнит команду, адрес которой (1400) записан в СК, т. е. первую команду подпрограммы. Затем он выполнит вторую команду подпрограммы и т. д. до тех пор, пока не встретит команду

возврат из подпрограммы.

При выполнении пункта 4 для команды "возврат из подпрограммы" процессор

а) помещает в СК содержимое ячейки памяти с адресом, записанным в R6;

б) увеличивает регистр R6 на 2.

В результате R6 примет то же значение, что и до вызова подпрограммы, а в СК окажется адрес возврата — 1004— адрес первой команды, следующей за "вызов подпрограммы". В соответствии с основным алгоритмом именно эта команда и будет выполнена на очередном шаге процессора.

Хорошо было бы проиллюстрировать вызов подпрограммы на каком-нибудь интересном примере. Однако интересные примеры длинные, поэтому мы сейчас приведем пример, который только для иллюстрации и годится: задача, которая в нем решается, могла бы быть решена проще, без всяких подпрограмм.

Рассмотрим пример программы, которая увеличивает регистр R1 на 2 с помощью подпрограммы, увеличивающей R1 на 1 (табл. 27).

Таблица 27

вызов подпрограммы	1000
1400	1002
вызов подпрограммы	1004
1400	1006
стоп	1008
...	
увеличить слово R1 на единицу	1400
возврат из подпрограммы	1402

Таблица значений для этой программы при начальных значениях СК = 1000, R6 = 1000, R1 = −1 выглядит так (табл. 28):

Таблица 28

Шаг	Команда	СК	R6	R1	Ячейка с адресом 998
		1000	1000	−1	
1	вызов подпрограммы	1400	998		1004
2	увеличить слово R1 на единицу	1402		0	
3	возврат из подпрограммы	1004	1000		
4	вызов подпрограммы	1400	998		1008
5	увеличить слово R1 на единицу	1402		1	
6	возврат из подпрограммы	1008	1000		
7	стоп	1010			

Как и требовалось, после исполнения программы регистр R1 увеличился на 2, т. е. стал равен 1.

Естественно, что подпрограмма также может вызывать другие подпрограммы и т. д. Проиллюстрируем это на простейшем примере: программа А вызывает подпрограмму В, которая, в свою очередь, вызывает подпрограмму С (табл. 29).

Таблица 29

	вызов подпрограммы	1000
А	1400	1002
	стоп	1004
	вызов подпрограммы	1400
В	1600	1402
	возврат из подпрограммы	1404
С	возврат из подпрограммы	1600

Таблица 30 значений для этой программы при начальных значениях СК = 1000, R6 = 1000 помещена на с. 122.

Стрелки на таблице показывают последовательность действий процессора при выполнении третьего шага.

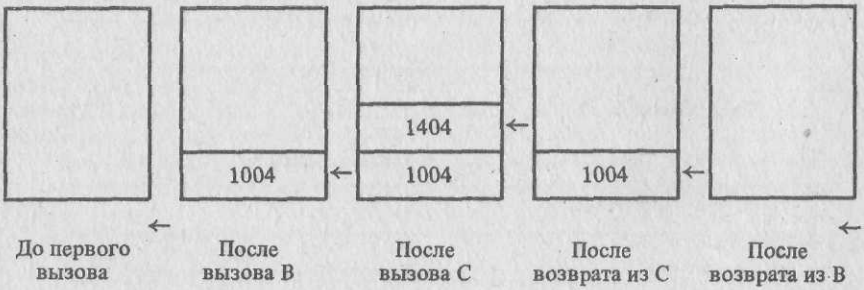
Замечание. Рассказывая про команды "вызов подпрограммы" и "возврат из подпрограммы", мы несколько погрешили против истины: на самом деле эти команды имеют аргументы и выполняются более сложным образом. Так, как описано выше, эти

Таблица 30

Шаг	Команда	СК	R6	Слово с адресом 998	Слово с адресом 996
		1000	1000		
1	вызов подпрограммы	1400	998	1004	
2	вызов подпрограммы	1600	→996		→1404
3	возврат из подпрограммы	1404	998←		
4	возврат из подпрограммы	1004	1000		
5	стоп	1006			

команды работают только при некоторых фиксированных значениях аргументов.

На примере последней программы проследим еще раз, как записываются в память и как извлекаются из нее адреса возвратов при вызовах подпрограмм. Будем изображать только область памяти с адресами, меньшими 1000, а адрес, содержащийся в регистре R6, покажем стрелкой, указывающей на соответствующее слово:



Таким образом, при каждом вызове подпрограммы (например, подпрограммы C) на уже существующие слова с адресами возвратов сверху кладется еще один «кирпичик» — слово с очередным адресом возврата. При следующем вызове сверху добавляется еще один «кирпичик» и т. д.

По команде возврата из подпрограммы верхний «кирпичик» снимается, при следующем возврате снимается еще один и т. д.

«Кирпичики» подчиняются правилу «положил позже — взял раньше». Любую структуру, элементы которой подчиняются этому правилу, принято называть *стеком*. В качестве примеров стеков можно привести детскую пирамидку (первым снимается колечко, надетое последним) или железнодорожный тупик (вагоны вывозят из тупика в порядке, обратном порядку их помещения туда).

Так как регистр R6 в каждый момент времени содержит адрес самого верхнего элемента стека (или, как принято говорить, указывает на *вершину стека*), то его называют *указателем стека*. В целом алгоритм записи и извлечения из памяти адресов возвратов при вызовах подпрограмм принято описывать фразой: *при вызовах подпрограмм адреса возвратов сохраняются в стеке*.

3. Кодировка команд. Каждая команда в ЭВМ кодируется последовательностью из 16 нулей и единиц (16 бит), т. е. словом. Для разных команд разные группы битов кодируют разную информацию. Например, для команды "очистить ячейку (R1)" эти 16 битов можно разбить на три группы: 10 битов — *код операции* (КОП), 3 бита — вид использования регистра и 3 бита — номер регистра.

КОП содержит информацию о том, что должна делать команда (очистить ячейку, уменьшить слово на единицу, переслать слово и пр.). Номер фигурирующего в команде регистра (от R0 до R7) кодируется 3 битами и может быть равен 000, 001, 010, 011, 100, 101, 110, 111 соответственно. Вид использования регистра указывает, надо ли очистить сам регистр (вид 000) или ячейку памяти по адресу, записанному в регистре (вид 001), или же, кроме очистки ячейки памяти по адресу, записанному в регистре, надо еще сам регистр увеличить на 2 (вид 010). Существуют и другие виды использования регистров (011, 100, 101, 110, 111), которые мы не рассматривали.

Примеры конкретных кодировок разных команд приведены в таблицах 31 и 32.

Таблица 31

Команда	КОП	Вид	Регистр
стоп	0000000000000000		
очистить регистр R1	0000101000	000	001
очистить ячейку (R1)	0000101000	001	001
очистить ячейку (R1)+	0000101000	010	001
очистить регистр R2	0000101000	000	010
уменьшить слово R1 на единицу	0000101011	000	001
уменьшить слово (R1) на единицу	0000101011	001	001
уменьшить слово (R1)+ на единицу	0000101011	010	001
уменьшить слово R2 на единицу	0000101011	000	010

Таблица 32

Команда	КОП	Вид	Регистр	Вид	Регистр
переслать слово R1 в R2	0001	000	001	000	010
переслать слово (R1) в R2	0001	001	001	000	010
переслать слово (R1)+ в (R2)+	0001	010	001	010	010
переслать слово (R4) в R7	0001	001	100	000	111

Таким образом, команда "переслать слово (R4) в R7", например, кодируется последовательностью 0001001100000111.

В командах перехода КОП занимает половину слова (8 битов), а оставшиеся 8 битов используются для указания, на сколько слов надо сместиться (на +2, на −3 и т. п.). Это число слов (его принято называть *смещением*) кодируется примерно так же, как и целые числа. Однако, поскольку используется только 8 битов, диапазон значений смещения меньше: от −128 до +127 (табл. 33).

Таблица 33

Команда	КОП	Смещение
переход на + 1 слово	00000001	00000001
переход на + 127 слов	00000001	01111111
переход на – 128 слов	00000001	10000000
переход на – 3 слова	00000001	11111101
переход на – 1 слово	00000001	11111111
если меньше, переход на + 2 слова	00000101	00000010

Таким образом, команда "если меньше, переход на +2 слова" имеет код 0000010100000010.

4. Работа с внешними устройствами. Разберемся теперь, как процессор управляет внешними устройствами, т. е. каким образом можно, например, что бы то ни было изобразить на экране дисплея или напечатать на печатающем устройстве (см. рис. 10).

Прежде всего отметим, что нет ни специальных команд для управления внешними устройствами, ни специальных форм записи аргументов при таком управлении. Все управление внешними устройствами осуществляется обычными командами, такими, как "переслать слово".

Посмотрим, что же происходит при выполнении этой команды и как с ее помощью можно управлять внешними устройствами.

Мы уже говорили, что при выполнении, например, команды "переслать слово R1 в (R0)" процессор записывает содержимое регистра R1 в память по адресу, указанному в регистре R0. Это верно лишь в том случае, если в регистре R0 содержится адрес памяти. Дело в том, что все множество адресов слов от 0 до 65534 поделено между различными устройствами ДВК-2М и аналогичных ЭВМ. К памяти относятся лишь адреса от 0 до 57342. Бóльшие адреса относятся к различным внешним устройствам и процессору. Например, у устройства печати есть два слова с адресами 65356 и 65358, а у экрана дисплея — два слова с адресами 65396 и 65398.

Поэтому правильнее было бы описывать действие команды "переслать слово R1 в (R0)" так:

процессор берет содержимое регистра и посылает эту информацию через магистраль по адресу, записанному в R0.

Можно представлять себе магистраль как своего рода почтовую службу, по которой процессор посылает письмо. Дальнейшие действия зависят не от процессора, а от адресата — от того, кому адресовано «письмо». Если это память, то полученная информация (содержимое регистра R1) просто запоминается в соответствующей ячейке памяти, как мы это и описывали раньше. Если же, к примеру, R0 = 65358, т. е. адресатом является устройство печати, то полученная информация воспринимается как некоторая команда и печатающее устройство выполняет соответствующие действия (печатает символ на бумаге, прогоняет бумагу до границы страницы и т. п.).

Аналогичным образом происходит и чтение информации по некоторому адресу. Например, по команде

переслать слово (R0) в R1

процессор посылает по адресу, указанному в R0, «письмо» с просьбой прислать информацию. Дальнейшие действия опять зависят от адресата. Память просто посылает обратно содержимое соответствующей ячейки. Устройство печати (при R0 = 65356) посылает обратно информацию о своем состоянии. Проанализировав эту информацию, находящуюся теперь в регистре R1, можно, например, узнать, включено ли устройство в сеть, закончено или нет выполнение предыдущей команды и пр.

Заметим, что «письма» по магистрали не обязаны проходить через процессор. Точно так же обмениваться информацией между собой и с памятью могут любые внешние устройства. Скорость передачи информации по магистрали велика и составляет миллионы байтов в секунду.

5. Магистрально-модульный принцип построения ЭВМ. Наличие магистрали позволяет собирать ЭВМ из отдельных функционально и конструктивно законченных блоков, называемых модулями. Модуль может содержать несколько компонент ЭВМ, например процессор и память (см. рис. 4). Наоборот, одна компонента ЭВМ может быть изготовлена в виде нескольких модулей. Подсоединяя к магистрали разные наборы модулей, можно получать различные ЭВМ. Такой *магистрально-модульный принцип* построения ЭВМ получил сейчас широкое распространение, так как обладает несколькими важными достоинствами:

1. Процессор может эффективно управлять внешними устройствами с помощью тех же команд, которыми он работает с памятью, т. е. для работы с внешними устройствами не нужны специальные команды процессора.

2. Можно конструировать и подключать к магистрали новые внешние устройства. При этом не требуется никаких изменений в уже существующих устройствах, процессоре, памяти.

3. Из готовых модулей можно легко составлять ЭВМ разной мощности и назначения. Состав ЭВМ можно легко изменять в процессе ее эксплуатации.

Вопросы

1. В чем различие команд "очистить регистр R0", "очистить ячейку (R0)" и "очистить ячейку (R0) + "?
2. Что такое: а) подпрограмма, б) команда вызова подпрограммы, в) команда возврата из подпрограммы, г) адрес подпрограммы, д) адрес возврата?
3. Может ли одна подпрограмма вызывать другую? Может ли одна и та же подпрограмма вызываться разными командами вызова из разных мест памяти?
4. Как используется регистр R6 при вызовах подпрограмм?
5. Что такое: а) стек, б) вершина стека, в) указатель стека?
6. Какое колечко детской пирамидки является вершиной стека?
7. Какой двоичный код имеет команда "уменьшить слово (R4) + на единицу"?
8. Какой команде соответствует двоичный код 0001000001001010?
9. Как процессор управляет внешними устройствами?
10. Что такое магистраль? Для чего она служит?

Упражнения

1. Напишите программу, после исполнения которой в регистре R0 будет содержаться:

а) число 2000;
б) число, адрес которого лежит в R1;
в) содержимое R1, увеличенное на 200;
г) максимум из чисел, адреса которых лежат в R1, R2;
д) максимум из чисел, адреса которых лежат в R0, R1.

2. В памяти, начиная с адреса 2000, расположена целочисленная таблица из 100 элементов. Напишите программу, после исполнения которой:

а) все элементы таблицы увеличатся на 1;
б) все элементы таблицы станут равны 0;
в) в таблице будет записана арифметическая прогрессия с первым членом 1 и разностью —1;
г) все элементы таблицы изменят знак на противоположный.

3. В памяти, начиная с адреса 2000, расположена целочисленная таблица из 100 элементов. Напишите программу, после исполнения которой в регистре R0 будет содержаться:

а) сумма элементов таблицы;
б) сумма модулей элементов таблицы;
в) сумма положительных элементов таблицы;
г) минимальный элемент таблицы;
д) адрес первого минимального элемента таблицы;
е) число положительных элементов таблицы;
ж) число элементов таблицы, больших 100.

IV. АЛГОРИТМЫ ПОИСКА ИНФОРМАЦИИ

Мы уже говорили о том, что одно из важнейших применений компьютеров — учет и обработка экономической информации. Рассмотрим пример. Разумеется, он является лишь упрощенной моделью реальной обработки информации, которая часто требует больших и сложных программ.

Итак, мы с вами на складе. На нем хранится 1000 различных измерительных приборов, которые выдаются по мере необходимости, а затем возвращаются обратно. Заведующий складом решил использовать компьютер для учета хранящихся на складе приборов.

Для этого каждый прибор должен иметь свой номер: от 1 до 1000, а компьютер должен уметь отвечать на вопрос, имеется ли на складе прибор с заданным номером. Сведения о наличии всех приборов будут храниться в таблице *наличие* [1:1000], при этом *наличие* [*i*] = 1, если *i*-й прибор есть на складе, и *наличие* [*i*] = 0, если *i*-го прибора нет на складе.

Теперь можно написать такой алгоритм:

```
алг лит есть на складе (цел номер, цел таб наличие [1 : 1000])
нач
| выбор
|   | при наличие [i] = 1 : знач := "да"
|   | при наличие [i] = 0 : знач := "нет"
| все
кон
```

Нужно еще уметь регистрировать выдачу прибора со склада и его возвращение. Это делается так:

```
алг выдача (цел номер, цел таб наличие [1 : 1000])
    арг номер, наличие
    рез наличие
нач
| наличие [номер] := 0
кон

алг возврат (цел номер, цел таб наличие [1 : 1000])
    арг номер, наличие
    рез наличие
нач
| наличие [номер] := 1
кон
```

Эти алгоритмы должны применяться, когда со склада берут или на склад возвращают очередной прибор. Перед применением алгоритма «выдача» полезно убедиться, что соответствующий прибор числится имеющимся на складе, а перед применением алгоритма «возврат» — что он числится выданным.

Имея таблицу *наличие,* можно выполнять и другие учетные операции. Например, можно быстро узнать, сколько приборов есть на складе в данный момент. Поскольку в таблице *наличие* стоят только нули и единицы и отсутствие прибора обозначается нулем, то количество приборов на складе будет равно просто сумме элементов этой таблицы.

```
алг цел количество (цел таб наличие [1 : 1000])
нач
| цел i
| знач := 0
| для i от 1 до 1000
| нц
|   знач := знач + наличие [i]
| кц
кон
```

Описанный выше способ хранения информации с помощью сквозной нумерации приборов и таблицы *наличие* по разным причинам не очень удобен. Например, при покупке организацией любой вещи: стула, телевизора, станка, прибора и т. д.— эта вещь получает в организации специальный номер для учета — инвентарный номер. Все эти номера различны, и удобнее не нумеровать приборы на складе специально, а воспользоваться уже имеющимися инвентарными номерами.

Можно было бы хранить информацию в таблице *наличие* $[1:N]$, где N — достаточно большое число (настолько большое, что любой инвентарный номер не больше N). К сожалению, для хранения такой таблицы в памяти ЭВМ потребуется много места. Бóльшая часть этого места будет пропадать зря — ведь инвентарные номера приборов составляют очень небольшую часть всех инвентарных номеров, используемых в данной организации. Хотелось бы хранить информацию более экономно.

Один из возможных способов хранения информации таков. Пусть, например, на складе имеются приборы с номерами 2543, 7207 и 10628. Тогда можно хранить информацию об этом в таблице *номера:*

номера [1] = 2543
номера [2] = 7207
номера [3] = 10628

При этом объем, занимаемый таблицей в памяти, будет зависеть не от величины номеров, а от числа приборов на складе.

Плохо только, что размер таблицы *номера* должен меняться с изменением числа приборов, а правила алгоритмического языка требуют заранее задать размер таблицы и не позволяют изменить этот размер во время работы программы. Тут можно поступить следующим образом. Введем величину *количество*, значение которой будет равно текущему числу приборов на складе. Тогда числа

номера [1], ..., *номера* [*количество*]

будут инвентарными номерами приборов, хранящихся на складе. При этом оставшиеся, «запасные», места в таблице могут быть заполнены чем угодно. Они понадобятся, если впоследствии количество приборов на складе увеличится.

Таким образом, размер таблицы *номера* нужно выбрать исходя из наибольшего возможного числа приборов на складе.

Поясним сказанное на примере. Пусть нам известно, что на складе будет храниться не более 1000 приборов одновременно с инвентарными номерами от 1 до 40 000. В этом случае вместо таблицы *наличие* [1:40 000] нам будет достаточно иметь таблицу *номера* [1:1000] и величину *количество*, которая принимает значения от 0 до 1000.

Пусть на складе имеются приборы с номерами 2543, 7207 и 10628. Чтобы зарегистрировать это, достаточно выполнить команды

количество : = 3
номера [1] : = 2543
номера [2] : = 7207
номера [3] : = 10628

Можно было бы вместо этих команд выполнить команды

количество : = 3
номера [1] : = 7207
номера [2] : = 2543
номера [3] : = 10628

Получившиеся значения величин *номера* и *количество* соответствуют той же самой ситуации на складе, поскольку порядок расположения в таблице *номера* не существен.

Чтобы привыкнуть к такому способу записи информации о номерах имеющихся на складе приборов, попробуем ответить на следующий вопрос. Пусть *количество* = 5, а таблица *номера* такова:

1	2	3	4	5	6	7	8	9	...	1000
57	2	444	7	91	179	19	199	19	...	1591

Какие приборы имеются на складе? Прежде всего мы знаем, что их там 5 штук. Выбрав 5 первых элементов таблицы *номера*, находим, что на складе имеются приборы с номерами

57, 2, 444, 7 и 91.

Если при той же таблице *номера* значение переменной *количество* будет равно 2, то это будет означать, что на складе имеются приборы с номерами 57 и 2. А если *количество* = 0, то это значит, что на складе не осталось ни одного прибора.

Не всякие значения величин *количество* и *номера* возможны. Например, *количество* не может быть равно — 7 или 1001. Подобным же образом мы не допускаем, например, ситуации *количество* = 3 и *номера:*

1	2	3	4	5	6	7	8	9	...	1000
37	57	37	251	52	38	339	43	49	...	8

потому, что номер 37 встречается в последовательности

номера [1], ..., *номера* [*количество*]

более одного раза (в данном случае дважды).

Другими словами, допустимы только такие значения переменных *количество* и *номера,* при которых $0 \leqslant$ *количество* $\leqslant 1000$ и все числа

номера [1], ..., *номера* [*количество*]

различны. Это соответствует той ситуации, когда на складе имеются приборы с номерами

номера [1], ..., *номера* [*количество*].

Мы договорились о новом способе хранения информации о наличии приборов на складе с помощью двух величин *номера* и *количество.* Как же теперь узнать, есть ли на складе прибор с заданным номером? Для этого прежде всего нужно построить новый алгоритм «есть на складе» с заголовком

алг лит есть на складе (**цел** *номер,* **цел** *количество,*
цел таб *номера* [1:1000]),

который давал бы ответ "да", если номер встречается среди

номера [1], ..., *номера* [*количество*],

и ответ "нет" в противном случае. Запишем этот новый алгоритм:

```
алг лит есть на складе (цел номер, цел количество,
    цел таб номера [1:1000])
нач цел i
|   знач := "нет"
|   для i от 1 до количество
|   нц
|   |   если номер = номера [i]
|   |   то знач := "да"
|   |   все
|   кц
кон
```

Работает этот алгоритм так. Вначале значением алгоритма становится величина "нет". Затем мы сравниваем значение величины *номер* со всеми значениями *номера* [*i*] при всех *i* от $i = 1$ до $i =$ *количество*. Если условие *номер* = *номера* [*i*] соблюдается при некотором *i*, то значением алгоритма становится величина "да". Если же условие *номер* = *номер* [*i*] не соблюдается ни при одном *i*, то значением алгоритма остается величина "нет".

Теперь нам нужно научиться регистрировать выдачу приборов со склада и возвращение их на склад. Начнем со второго. Если мы хотим зарегистрировать тот факт, что на склад принесли прибор с данным номером, то переменную *количество* нужно увеличить на 1, записав в очередную клеточку таблицы *номера* номер помещенного на склад прибора. Приходим к такому алгоритму:

```
алг возврат (цел номер, цел количество,
    цел таб номера [1:1000])
    арг номер, количество, номера
    рез количество, номера
нач
|   количество := количество + 1
|   номера [ количество ] := номер
кон
```

Алгоритм «возврат» можно применять только в том случае, когда *количество* < 1000 (т. е. склад не полон), а есть на складе (*номер, количество, номера*) = "нет" (т. е. элемент с указанным номером отсутствует).

Например, если значение переменной *количество* равнялось 3, а таблица *номера* была заполнена так:

1	2	3	...
333	72	29	...

то после возвращения на склад прибора номер 57 значением переменной *количество* станет число 4, а таблица *номера* станет такой:

1	2	3	4	...
333	72	29	57	...

Если мы после этого захотим зарегистрировать взятие со склада только что положенного туда прибора с номером 57, то достаточно снова уменьшить на 1 величину *количество*. (При этом в таблице *номера* нет необходимости вычеркивать число 57, так как элементы *номера* [*i*] при *i* > *количество* не играют для нас никакой роли.) Хуже, если понадобится зарегистрировать взятие со склада прибора с номером 72. Возможное решение тут таково — переставив его с элементом 57, получим таблицу:

1	2	3	4	...
333	57	29	72	...

После этого нужно уменьшить значение переменной *количество* с 4 до 3. Этот способ действий мы и будем применять в общем случае: исключаемый из таблицы элемент мы вначале поменяем местами с последним, а затем уменьшим величину *количество* на 1. Таким образом, исключение заданного номера из таблицы происходит в три этапа:

1) находим его в таблице;
2) переставляем его местами с последним элементом таблицы;
3) уменьшаем *количество* на 1.

Записывая это более подробно, приходим к такому алгоритму:

```
алг выдача (цел номер, количество, цел таб номера [1:1000])
    арг номер, количество, номера
    рез количество, номера
нач цел i, A
|   i := 1
|   пока номер ≠ номера [i]
|   нц
|   |   i := i+1
|   кц
|   A := номера [i]
|   номера [i] := номера [количество]
|   номера [количество] := A
|   количество := количество — 1
кон
```

Замечание. Этап 2 можно упростить. Нам нет необходимости переставлять местами элементы

номера [*i*] и *номера* [*количество*],

поскольку после уменьшения переменной *количество* на 1 значение одного из них будет несущественно. Достаточно выполнить команду

номера [*i*] := *номера* [*количество*].

Приходим к такому варианту алгоритма:

```
алг выдача (цел номер, количество, цел таб номера [1:1000])
    арг номер, количество, номера
    рез количество, номера
нач цел i
|   i := 1
|   пока номер ≠ номера [i]
|   нц
|   |  i := i + 1
|   кц
|   номера [i] := номера [количество]
|   количество := количество — 1
кон
```

В обоих вариантах алгоритма предполагается, что исключаемый из таблицы номер в ней имеется.

Посмотрим теперь, сколько действий придется выполнить исполнителю наших алгоритмов. Чтобы выяснить, есть ли прибор с заданным номером на складе (алгоритм «есть на складе»), исполнитель сравнивает его *номер* поочередно со всеми номерами

номер [1], ..., *номер* [*количество*].

Поэтому число выполняемых действий пропорционально числу приборов на складе. Примерно таково же будет число необходимых действий в алгоритме «выдача». Правда, при его исполнении исполнителю может повезти и исключаемый номер может найтись быстро (если он стоит в начале таблицы). Но в худшем случае если этот номер стоит в конце таблицы, то число действий исполнителя также будет пропорционально числу приборов на складе. Алгоритм «возврат» — самый простой из трех; при его исполнении число действий не зависит от числа приборов на складе.

Представим себе теперь, что проверка наличия прибора с заданным номером — наиболее часто выполняемая операция и затраты на ее выполнение хотелось бы сократить. Нельзя ли тут что-нибудь придумать?

Вспомним, как мы ищем нужное нам слово в словаре. Если бы мы читали словарь подряд, сравнивая нужное слово со всеми сло-

вами, в нем встречающимися, то поиск каждого слова занимал бы несколько часов. К счастью, есть гораздо более быстрый способ. Ведь слова в словаре упорядочены по алфавиту, и, посмотрев на любое слово из словаря, мы знаем, до или после него находится нужное нам слово. Таким образом, открыв словарь посередине и сравнив наше слово с тем, которое написано посередине, мы сразу же узнаем, в какой половине словаря расположено интересующее нас слово. Открыв эту половину посередине, мы узнаем, в какой из половин этой половины расположено нужное слово, и т. д. Каждый раз область, в которой может находиться нужное слово, сужается примерно вдвое, и довольно быстро нужное слово находится (или мы убеждаемся, что в словаре его нет).

Такая процедура поиска, в которой «круг подозреваемых» все время сужается примерно вдвое, называется *двоичным поиском*. Прежде чем говорить о применении двоичного поиска в нашей задаче, мы приведем еще несколько примеров его применения.

Все мы знаем детскую игру, в которой один из участников задумывает какой-то объект, а второй должен угадать, что задумал первый. При этом он может задавать первому вопросы, на которые можно отвечать «да» и «нет». Первый отвечает на эти вопросы, и по ответам второй должен узнать, что задумал первый. Рассмотрим вариант игры, в которой первый может задумать любое число от 1 до 100. Как действовать второму?

Самый простой способ состоит в том, чтобы последовательно задавать вопросы: «Ты задумал 1?», «Ты задумал 2?» — до тех пор, пока первый не ответит: «Да». При этом понадобится задать столько вопросов, каково задуманное число: если задумано число 1, то хватит одного вопроса, но если задумано число 100, то понадобится 100 вопросов.

Двоичный поиск позволяет обойтись гораздо меньшим числом вопросов. Разобьем числа от 1 до 100 на 2 половины: от 1 до 50 и от 51 до 100. Задав один вопрос: «Задуманное число больше 50?», мы узнаем, в какой половине находится задуманное число. Пусть, например, нам ответили «нет», и, следовательно, задуманное число находится среди чисел 1, 2, ..., 50. Снова разобьем эти числа на две части: 1, 2, ..., 25 и 26, ..., 50. Задав вопрос: «Задуманное число больше 25?», мы узнаем, в какой из этих частей находится задуманное число. Пусть, например, нам сказали «да», т. е. число находится от 26 до 50. Снова разобьем эту область на 2 части (теперь уже только *примерно* равные, так как количество «подозреваемых» чисел — 25 — нечетно). Это будут области от 26 до 38 и от 39 до 50 (в них 13 и 12 чисел соответственно). Следующий вопрос будет таков: «Задуманное число больше 38?» И т. д.

Сколько же понадобится вопросов? Составим таблицу, показывающую количество «подозреваемых» чисел после каждого из вопросов. (Если возможно несколько вариантов, например 12 или 13, мы берем худший случай.)

Количество заданных вопросов	1	2	3	4	5	6	7
Количество „подозреваемых" чисел	50	25	13	7	4	2	1

Из таблицы видно, что после первого вопроса остается максимум 50 «подозреваемых», после второго — максимум 25, после третьего — максимум 13 и т. д. После седьмого вопроса осталось одно-единственное число, которое и будет искомым.

Легко сообразить, что при таком способе отгадывания одного числа из 2^n чисел требуется n вопросов. Этот способ является наилучшим в том смысле, что нельзя придумать способ, который бы гарантировал отгадывание числа с меньшим количеством вопросов. В самом деле, каждый вопрос разбивает круг «подозреваемых» на две части. Если части окажутся равными, то на следующем этапе количество «подозреваемых» уменьшится вдвое. Если же части неравные, то в худшем случае под подозрением останется бóльшая часть и число «подозреваемых» сократится меньше чем вдвое. Так что наш способ — самый лучший.

Это рассуждение объясняет, почему, например, при отгадывании задуманной известной исторической личности вопрос: «Является ли задуманное лицо генетиком?» — неудачен: среди возможных кандидатов в задуманные фигуры генетики составляют лишь небольшую часть. Скорее всего ответ будет отрицательным, а круг «подозреваемых» сузится лишь очень немного. Гораздо лучше вопрос: «Является ли задуманное лицо ученым?», при котором круг возможных задуманных лиц делится на более близкие по размеру части.

Еще одним примером двоичного поиска является алгоритм отыскания корня функции на отрезке (см. первую часть учебного пособия). Там отрезок, на котором функция меняет знак, делился пополам и выбиралась та из половин этого отрезка, на которой знак тоже менялся. В результате нескольких таких шагов мы находили корень с заданной точностью.

В нашем случае для применения алгоритма двоичного поиска к таблице *номера* необходимо, чтобы номера приборов располагались в таблице в порядке возрастания, т. е. чтобы соблюдались условия:

$$номера\ [1] < номера\ [2] < \ldots < номера\ [количество].$$

Если это так, то с помощью двоичного поиска можно найти интересующий нас номер (или убедиться в его отсутствии) быстрее.

Будем сужать круг «подозреваемых» мест в таблице, считая «подозреваемыми» все числа таблицы от

номера [*левая граница*] включительно до
номера [*правая граница*] невключительно.

Здесь *левая граница* и *правая граница* — две переменные величины, отмечающие границы участка таблицы *номера*, находящегося «под подозрением». Вначале «под подозрением» находятся все числа таблицы, и потому *левая граница* должна быть равна 1, а *правая граница* равна *количество* + 1. (Напомним, что мы «подозреваем» числа от левой границы, включая ее, до правой границы, исключая ее.) Затем мы сужаем «подозреваемый» участок. Выберем число *середина*, находящееся в нем:

левая граница $\leqslant$ *середина* $<$ *правая граница*.

Если *номера* [*середина*] $\leqslant$ *номер*, то элемент *номер* может находиться только в правой половине «подозреваемого» участка (от *середина* включительно до *правая граница* невключительно). Если же *номера* [*середина*] $>$ *номер*, то элемент *номер* может находиться только в левой половине (от *левая граница* включительно до *середина* невключительно). Приходим к такой схеме алгоритма:

```
левая граница := 1
правая граница := количество + 1
пока «подозреваемых» чисел больше одного
нц
| разбить «подозреваемый» участок числом середина на две
| примерно равные половины
| выбор
| | при номера [середина] ⩽ номер:
| |    левая граница := середина
| | при номера [середина] > номер:
| |    правая граница := середина
| все
кц
```

Число «подозреваемых» чисел равно (*правая граница* — *левая граница*). Когда «подозреваемое» число останется одно, т. е. *правая граница* = *левая граница* + 1, достаточно будет сравнить его с искомым. Если же их не останется вовсе, т. е. *левая граница* = = *правая граница* (так будет, если *количество* = 0), то нужного числа в таблице нет.

Осталось уточнить в алгоритме два места, записанных нами пока неформально. «Подозреваемых» чисел больше одного» означает *правая граница* $>$ *левая граница* + 1. Выбрать число *середина* между левой и правой границами можно так: надо взять их полусумму и (на случай, если она окажется не целой) взять ближайшее к ней слева целое число (рис. 32). При этом нам по-

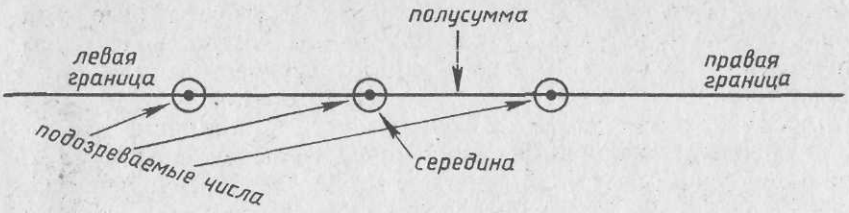


Рис. 32

надобится функция «ближайшее слева целое», которую будем обозначать целая часть (x).

Итак, получим окончательно такой алгоритм:

```
алг лит есть на складе (цел номер, цел количество,
                        цел таб номера [1:1000] )
нач цел левая граница, правая граница, середина
    левая граница := 1
    правая граница := количество + 1
    пока правая граница > левая граница + 1
    нц
        середина := целая часть ( (левая граница + правая
        граница) /2)
        выбор
            при номера [середина] ≤ номер:
                левая граница := середина
            при номера [середина] > номер:
                правая граница := середина
        все
    кц
        выбор
            при левая граница = правая граница:
                знач := "нет"
            при левая граница + 1 = правая граница и
                номера [левая граница] = номер:
                знач := "да"
            при левая граница + 1 = правая граница и
                номера [левая граница] ≠ номер:
                знач := "нет"
        все
кон
```

Решив хранить таблицу номеров в порядке возрастания, мы должны изменить и алгоритмы «выдача» и «возврат»: их применение не должно нарушать упорядоченности элементов таблицы.

V. ПРОГРАММИРОВАНИЕ — ВТОРАЯ ГРАМОТНОСТЬ

Еще несколько лет назад сравнение программирования с грамотностью могло бы показаться неправомерным. Сейчас же, когда предмет «Основы информатики и вычислительной техники» изучают в каждой школе, такое сравнение поможет нам разобраться в том, какое место знание этого предмета занимает в общекультурном багаже современного человека и какую роль оно будет играть в будущем. Заметим, что мы используем слово «программирование» в широком смысле, понимая под ним общее умение использовать ЭВМ.

Программирование легче сравнивать с грамотностью, если вспомнить, что грамотность — это историческая категория, имеющая свое возникновение и развитие. СССР — страна практически сплошной грамотности: 15 лет назад грамотные у нас составляли 99,7% от общего числа населения в возрасте 9 лет и старше. 100 лет назад этот процент был чуть выше 20%.

Мы знаем, что и в основе грамотности, и в основе программирования лежит техническое изобретение — печатный станок и ЭВМ соответственно. Если развитие и распространение книгопечатания привели к всеобщей грамотности, то развитие и распространение ЭВМ приведут к всеобщему знанию основ программирования.

Несмотря на то что возникновение книгопечатания и вычислительной техники разделено периодом в пять столетий, их сопоставление обнаруживает много сходства. И в том и в другом мы найдем «смену поколений», основанную на изменениях производственной базы и технологических процессов.

Приведем лишь некоторые данные, которые характеризуют темп, размах и взаимообусловленность развития книгопечатания и грамотности.

Появление первых изданий изобретателя печатного станка Иоганна Гутенберга датируется 1445 г. (латинская грамматика Элия Доната и др.), не прошло и 50 лет, а в мире уже работало свыше тысячи типографий, выпустивших около 10 млн. экземпляров книг. Таким образом, почти мгновенно был превышен наличный фонд рукописных изданий. В 1962 г. во всем мире было напечатано 10 млрд. книг.

На рисунке 33 показана взаимозависимость между грамотностью и книгопечатанием в нашей стране за последнее столетие.

Если мы поверим в справедливость наших аналогий, то это даст нам некоторое представление о размахе и объеме работы, которую надо предпринять для подготовки встречи людей с миром ЭВМ.

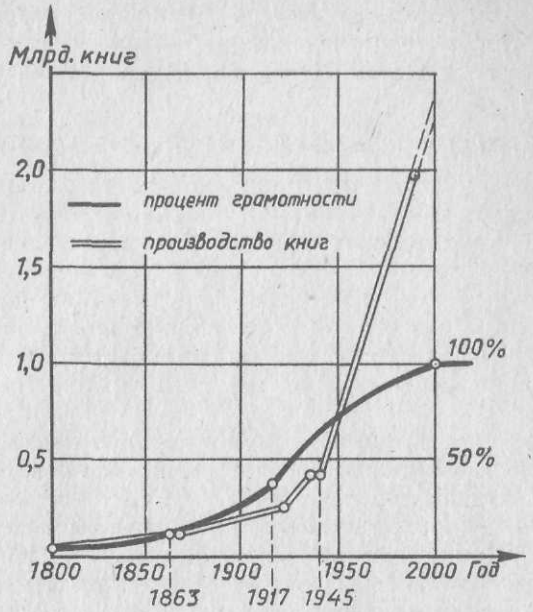


Рис. 33

Средства массовой информации, научно-популярные издания и рекламные проспекты уже создали привычное ви́дение вычислительной машины. Ее атрибуты — экран и клавиатура дисплея, бобины магнитных лент, кружево перфоленты, длинные бумажные полотна выдачи быстродействующих печатающих устройств, мигающие огоньки инженерного пульта, угловатые шкафы, забитые электронными деталями. Подобное представление о месте ЭВМ в жизни человека не только поверхностно, но и в чем-то ошибочно. ЭВМ будущего — это прежде всего крошечный срез кристалла кремния, оправленный в миниатюрную рамку с паутиной тончайших проводов и встроенный в большинство промышленных изделий: от наручных часов и бытовых приборов до станков и космических кораблей.

Речь идет, конечно, о микропроцессорах, которые хотя и появились всего чуть больше десяти лет назад, но уже производятся десятками миллионов штук в год. Первое, наиболее заметное сейчас их применение — это выпуск самых разнообразных карманных калькуляторов. Но это лишь небольшая часть из всего многообразия применений микропроцессоров. Появление и развитие микропроцессоров — по мнению специалистов — самое революционное техническое новшество XX века: микроЭВМ, построенная на базе микропроцессора, работает со скоростью сотни тысяч операций в секунду, хранит в своей оперативной памяти десятки тысяч чисел (слов), способна обмениваться информацией с внешней памятью емкостью в сотни тысяч знаков, помещается в объеме

Рис. 34

спичечного коробка, требует для своего изготовления несколько человеко-часов и может производиться практически в неограниченных количествах.

Микропроцессор, встроенный в промышленное изделие, будь то бытовой прибор или средство производства, придает ему совершенно новые качества и сильно влияет на характер взаимодействия человека с этим изделием.

Не менее сильно включение микропроцессора в схему изделия влияет на способ его проектирования, во время которого должны быть замечены, осмыслены и реализованы новые возможности изделия.

Широкое применение микропроцессоров приводит к тому, что тысячи профессий меняют свое лицо. Миллионы людей — операторов производства, наладчиков, машинисток, банковских служащих, продавцов-контролеров, библиотекарей, монтажников, секретарей, сборщиков на конвейере — начинают работать на полностью переоборудованных рабочих местах.

Мы уже сейчас говорим о миллионах людей, вовлеченных в этот процесс, а в ближайшем будущем это коснется практически каждого человека, занятого в общественном производстве.

На пути этого лавинного развития возникает, однако, одно принципиальное препятствие. В настоящее время способность человека к передаче знания машине сильно отстает от возможности создать эту машину. Если затраты общественного труда на изготовление микропроцессора и микроЭВМ исчисляются в человеко-часах, то затраты на создание программного обеспечения до сих пор выражаются в человеко-месяцах и человеко-годах. На рисунке 34 приведена кривая, показывающая динамику отношения стоимости программ к стоимости оборудования при проектировании систем обработки информации. Специалисты по программированию напряженно работают над тем, чтобы сделать труд программиста гораздо более производительным. Однако даже если им удастся создать необходимые условия для десяти-

кратного увеличения производительности труда при традиционном процессе создания программ, то и тогда, как показывают элементарные расчеты, примерно через двадцать лет придется посадить за программирование чуть ли не все взрослое население земного шара для того, чтобы обеспечить программами все производимые микропроцессоры.

Фантасты, которые силой своего воображения уже побывали в XXI в., утверждают, что в будущем человека, не знающего точных наук, с полным правом можно будет сравнить с неграмотным средневековым бароном, который с гордостью говорил, что счетом и письмом у него занимаются секретари.

То же должно случиться и с программированием. Каждый специалист, в какой бы области науки, производства и т. д. он ни работал, должен будет уметь эффективно использовать ЭВМ, уметь программировать, что и называется второй грамотностью.

Итак, мы переходим от мира машин к миру программ.

Герой Мольера, месье Журден, был в высшей степени удивлен, когда узнал, что всю жизнь говорил прозой, не подозревая этого. Благодаря появлению ЭВМ и вызванному этим возникновению вычислительной науки человечество оказывается в положении месье Журдена, с удивлением обнаруживая, что оно живет в мире программ.

Да, мы живем в мире программ и сами постоянно программируем, не сознавая этого.

Несомненно, что одно из самых выдающихся достижений биологической науки XX в.— это открытие того, что развитие организма есть исполнение генетической программы, записанной в его генном наборе. Заметим, что использование здесь программистских терминов «исполнение» и «программа» не является метафорой, а выражает суть внутриклеточных процессов роста и развития, по отношению к которым молекулярные структуры и химические процессы — своего рода элементная база и способ реализации команд.

Можно сказать, что очень многие физиологические процессы, происходящие в нашем организме,— это огромная, тщательно отлаженная и сложно устроенная библиотека программ. Анализ структуры программ этой библиотеки и их информационных связей позволяет делать выводы о состоянии организма в настоящее время и прогнозировать его поведение в будущем.

Практически весь производственный процесс на любом предприятии — это работа по программам. Причем эффективность производственного процесса зависит от степени отлаженности этих программ.

Даже если процесс включает элемент случайности, таковы, например, охота или вождение автомашины, то случайность и непредсказуемость сказываются лишь на выстраивании цепочки ситуаций, но не на реакции на эти ситуации, которая почти всегда выполняется по заранее разученной программе.

Наконец, обучение, т. е. приобретение знаний, это тоже работа по определенной программе. Действительно, ребенок научается что-то делать только после того, как он понимает, как это делается. Только после выработки такого понимания повторительная тренировка достигает успеха. Заметим, что это касается не только программ, представляющих собой цепочки логических реакций на заранее известные воздействия, но и программ, исполняемых человеком в процессе сложных движений в спорте, музыке, играх и т. д.

Повседневная жизнь человека — это во многом тоже деятельность по программам. Каждый человек, придерживающийся режима, с гордостью чувствует себя программистом. Вспомните свои утренние часы, начиная от звонка будильника и кончая выходом из дома. Эти часы заполнены до предела привычными утренними процедурами: уборка постели, умывание и т. д. Поразмышляйте над процедурой уборки квартиры, и вы увидите, что она не проста. Разработка эффективной программы уборки сделает честь любому профессиональному программисту.

Таким образом, мир программ это далеко не только начинка памяти ЭВМ. Это прежде всего огромный запас операционного знания, накопленный человечеством и теперь лишь актуализируемый вычислительными машинами, роботами, автоматическими устройствами. Еще больший запас программ хранится в генофонде всего живого: его актуализация в значительной степени составляет предмет биологии и ее новых разделов, включая генную инженерию.

Итак, мы стоим на пороге практически беспредельного развития и распространения электронно-вычислительной техники в обществе. Машина становится «партнером» практически во всех сферах деятельности человека, освобождая его от рутинных работ и усиливая его интеллект.

Существенным ускорителем этого процесса развития человеческого интеллекта должны быть законы обработки информации, способы перехода от знания к действию, способность строить программы, рассуждать о них и предвидеть результаты их исполнения. Сумма знаний по этим вопросам должна стать фундаментальной компонентой общего образования — второй грамотностью — каждого члена общества.

В Политическом докладе Центрального Комитета КПСС XXVII съезду Генеральный секретарь ЦК КПСС М. С. Горбачев отметил:

«Интересы дела требуют более глубоко поставить изучение научных основ современного производства, ведущих направлений его интенсификации. И что особенно неотложно, обеспечить компьютерную грамотность учащихся» (Правда.— 1986.— 26 февр.).

СОДЕРЖАНИЕ

Андрей Петрович Ершов
Вадим Макариевич Монахов
Александр Андреевич Кузнецов и др.

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Пробное учебное пособие
для средних учебных заведений

В двух частях

Часть вторая

Зав. редакцией *Р. А. Хабиб*
Спецредактор *В. П. Кацева*
Редакторы *Т. А. Бурмистрова, Н. И. Никитина*
Младшие редакторы *Л. Е. Козырева, Е. А. Сафронова*
Художественный редактор *Е. Н. Карасик*
Художники обл. *Б. Л. Николаев*, форз. *Б. Л. Рытман*, рис. *Е. П. Титков*
Технические редакторы *Н. А. Киселева, В. Ф. Коскина*
Корректоры *В. И. Громова, Н. И. Новикова*

ИБ № 10485

Сдано в набор 23.01.86. Подписано к печати 19.02.86. Формат 60×90$^1/_{16}$. Бумага офсетная № 2. Гарнитура литературная. Печать офсетная. Усл. печ. л. 9+форзац 0,25. Усл. кр.-отт. 18,69. Уч.-изд. л. 6,94+форзац 0,4. Тираж 4 000 000 (1 700 001—4 000 000) экз. Заказ № 844. Цена 45 коп.

Ордена Трудового Красного Знамени издательство «Просвещение» Государственного комитета РСФСР по делам издательств, полиграфии и книжной торговли. 129846, Москва, 3-й проезд Марьиной рощи, 41.

Отпечатано с диапозитивов ордена Трудового Красного Знамени фабрики «Детская книга» № 1 Росглавполиграфпрома Государственного комитета РСФСР по делам издательств, полиграфии и книжной торговли на Калининском ордена Трудового Красного Знамени полиграфкомбинате детской литературы им. 50-летия СССР Росглавполиграфпрома Государственного комитета РСФСР по делам издательств, полиграфии и книжной торговли. 170040, Калинин, проспект 50-летия Октября, 46.