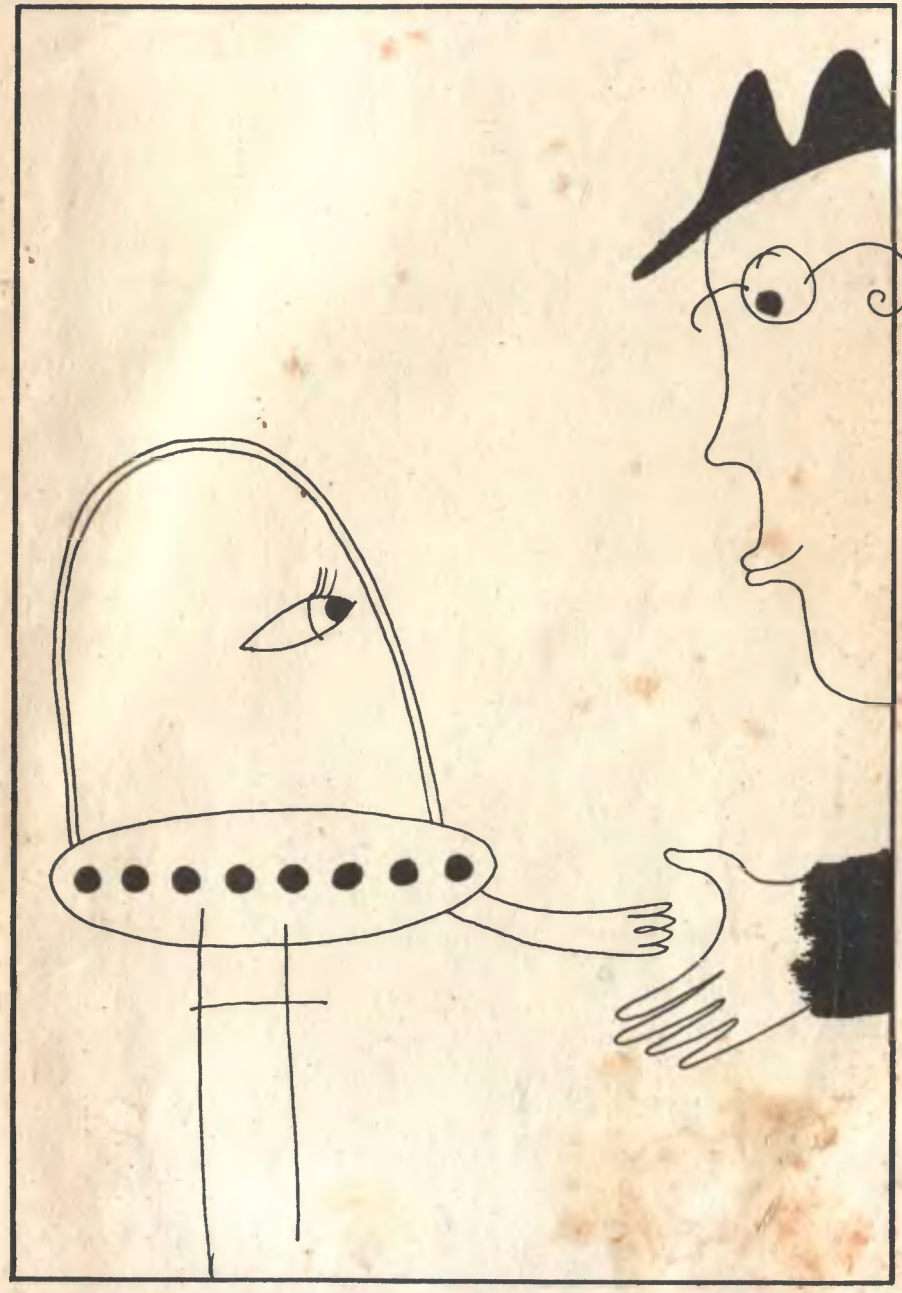
ЗНАКОМЬТЕСЬ: ПЕРСО-



НАЛЬНАЯ ЭВМ КОРВЕТ

ЗНАКОМЬТЕСЬ: ПЕРСОНАЛЬНАЯ ЭВМ КОРВЕТ

С предисловием академика Е.П.ВЕЛИХОВА

МОСКВА "НАУКА"
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1989

ББК 22.3с+22.18
3-71
УДК 53.08+519.85(023)

А в т о р ы:
С.А.АХМАНОВ (мл.), И.Г.ПЕРСИАНЦЕВ, А.Т.РАХИМОВ,
Н.Н.РОЙ, А.В.СКУРИХИН

Р е ц е н з е н т ы:
доктор физико-математических наук *А.А.Веденов*
кандидат физико-математических наук *А.Г.Кушниренко*

3-71 Знакомьтесь: Персональная ЭВМ Корвет /С.А.Ахманов (мл.), И.Г.Персианцев, А.Т.Рахимов и др.; Предисл. акад. Е.П.Велихова. - М.: Наука. Гл. ред. физ.-мат. лит., 1989. - 240 с.: ил.
ISBN 5-02-014207-7

За последние десять лет персональные ЭВМ произвели буквально революцию в организации многих сторон человеческой деятельности. Они все шире входят в нашу жизнь, становясь рабочим инструментом научного работника, исследователя, инженера, школьника. Корвет один из первых отечественных компьютеров, которыми предполагается оснастить средние школы. Авторы - разработчики этой персональной ЭВМ знакомят читателей с ее устройством, с приемами работы на ней, с возможностями ее использования.

Для широкого круга непрофессионалов, начинающих знакомство с ЭВМ на "персональном" уровне.

З 1604010000 - 125 / 053(02)-89 90-89

ББК 22.3с+22.18

ISBN 5-02-014207-7

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Развитие вычислительной техники и особенно ее массовое распространение сегодня во многом определяют темпы развития общества. Накопление, обработка и обмен информацией; создание экспертных систем, позволяющих на базе накопленных знаний прогнозировать оптимальный способ действий в самых разных областях человеческой деятельности; ускорение выполнения рутинных операций, ежедневно и многократно выполняемых представителями самых разных профессий, - все это и еще многое другое невозможно без широкого распространения вычислительной техники, без создания в обществе широкой информационно-вычислительной инфраструктуры.

Одним из главных этапов в создании такой инфраструктуры является появление персональных компьютеров. Чтобы стать персональными не по названию, а по существу, подобные компьютеры должны быть дешевыми, должны многое уметь и должны быть ориентированы на самого обычного пользователя, дружелюбие по отношению к которому только и способно заинтересовать людей вне зависимости от рода их занятий.

В нашей стране разработан и начат серийный выпуск ряда персональных компьютеров. Удачным по совокупности своих характеристик мне представляется компьютер, разработанный в НИИ ядерной физики Московского государственного университета и выпускаемый сегодня под названием "Корвет" на предприятиях Министерства радиопромышленности. Модификации этого же компьютера, выполненные с полным сохранением всех архитектурных решений и отличающиеся только конструктивным оформлением и названием, выпускаются сегодня и другими министерствами.

Корвет является одним из первых в нашей стране массовых персональных компьютеров. Предполагается широкое оснащение этими машинами средних школ. Поэтому появление книги, знакомящей широкий круг потенциальных пользователей Корвета с его устройством и основами его программирования, своевременно и полезно.

Авторы предлагаемой Вашему вниманию книги являются разработчиками машины. Им, как мне кажется, удалось в живой и пока в нашей специальной литературе нетрадиционной форме донести до читателя и общие принципы работы с Корветом, и тонкости использования его широких возможностей.

Академик *Е.П.ВЕЛИХОВ*

I. ПЕРВОЕ ЗНАКОМСТВО С ПЕРСОНАЛЬНОЙ ЭВМ КОРВЕТ

1. ЧТО ТАКОЕ ПЕРСОНАЛЬНАЯ ЭВМ

Введение в роман.
Мы впервые встретимся с персональной ЭВМ
и узнаем, чем она отличается от больших машин

У большинства людей, не имевших дела с вычислительной техникой и программированием, слово "ЭВМ" ассоциируется с огромным, залитым ярким светом залом. Нагнетаемый мощными кондиционерами охлажденный воздух создает внутри зала избыточное давление, чтобы ни одна пылинка не проникла в царство вычислительной машины. Люди, облаченные в белые без единого пятнышка одежды, склоняются над раскрытыми шкафами и тумбами, заполненными множеством плат с огромным числом разнообразных деталей. Тишина лишь изредка нарушается скупыми деловыми переговорами, в которых преобладают незнакомые нам слова: дисплей, дисковод, процессор и др. Все происходящее похоже на некий ритуальный обряд, совершенно непостижимый для непосвященного. Пользователь компьютера, уже побеседовавший с коллегами, имеющими некоторый опыт общения с ЭВМ, знает, что обслуживающие вычислительную машину люди делятся на электронщиков и системщиков. Чем они в действительности занимаются - он точно не знает, но фразы "машина на профилактике" или "системное время" вселяют в него благоговейный ужас. Кажется, что простой смертный просто не в состоянии хотя бы приблизиться к изумительным устройствам, заполняющим машинный зал.

Давайте зайдем в вычислительный центр. По условиям игры, мы - начинающие пользователи, мечтающие поработать на ЭВМ. К сожалению, первое, что мы увидим, попав в вычислительный центр, - это обилие запретительных плакатов. Во многие помещения вход подавляющему большинству простых смертных запрещен. Мы можем только через стеклянную дверь смотреть с благоговением внутрь машинного зала.

Но вот открывается дверь в другое помещение, называемое дисплейным классом, и новоявленный пользователь оказывается в тесной комнатушке, сплошь уставленной телевизорами, около каждого из которых лежит доска с кнопочками (клавиатура). Комната наполнена такими же, как он, пользователями, среди которых выделяются своими уверенными и нарочито громкими голосами пользователи-асы. При-

близительно такую картину можно было наблюдать еще лет десять назад во всех вычислительных центрах.

Но жизнь не стоит на месте: человек научился вместо радиоламп делать транзисторы, их в свою очередь упаковывать в кусок кремнезема и получать микросхемы, потом большие интегральные схемы (БИС) и наконец сверхбольшие интегральные схемы (СБИС). Слова "большие" и "сверхбольшие" не должны вводить в заблуждение. Речь идет не о размерах микросхем, которые приблизительно одинаковы, а о количестве транзисторов, входящих в микросхему. Когда говорят "большая интегральная микросхема" или "микросхема большей степени интеграции", это значит, что в ней много транзисторов. Оказалось, что блок, состоящий из сотен микросхем малой степени интеграции, можно упаковать в одну микросхему (или, как это часто называют, "корпус") немного большего, чем обычно, размера и значительно большей степени интеграции.

В результате появилась реальная возможность ту электронную начинку, которая раньше расфасовывалась по многим шкафам, соединенным друг с другом бесчисленными проводами, поместить в один корпус сравнительно небольшого размера. Самым ярким примером воплощения в жизнь такого пути развития вычислительной техники стала первая персональная ЭВМ (ПЭВМ) или персональный компьютер, разработанный в 1979 г. Стивом Джобсом и Стивом Возняком.

История создания первого персонального компьютера поучительна. Его авторы работали в фирме Atari, выпускавшей игровые автоматы. В свободное время они собирали в гараже устройство, которое потом и стало первой персональной ЭВМ. Результат их деятельности превзошел все ожидания, и было сделано еще несколько экземпляров для друзей и знакомых. Вслед за этим компьютер был запущен в серию и стал называться Apple II. Любопытно, что подобный процесс рождения новых ПЭВМ сохранился в известной степени и в наше время. Группа энтузиастов возится днями и ночами с какими-то железками. Это их хобби. Потом, когда детище готово, его показывают публике. Если оно всем нравится, его пытаются размножить и продавать всем желающим.

Слово "персональная" наводит на размышления. Раз персональная, значит вы оказываетесь с ней один на один. На ней не будет висеть табличка "профилактика" или "системное время", но зато все заботы о включении ЭВМ и поддержании ее работоспособности ложатся на ваши плечи.

Что же такое персональная ЭВМ? Внешне она не отличается от обыкновенного телевизора с клавиатурой, стоящего в любом дисплейном классе. Однако внимательный наблюдатель разглядит рядом необычные устройства типа кассетного магнитофона или ящика, называемого дисководом. Последний часто бывает встроен в саму ПЭВМ. Таким образом, ПЭВМ - это маленькая ЭВМ, целиком умещающаяся на письменном столе и весом не больше цветного телевизора. Уместно вспомнить, что когда некоторым нашим коллегам-физикам показывали

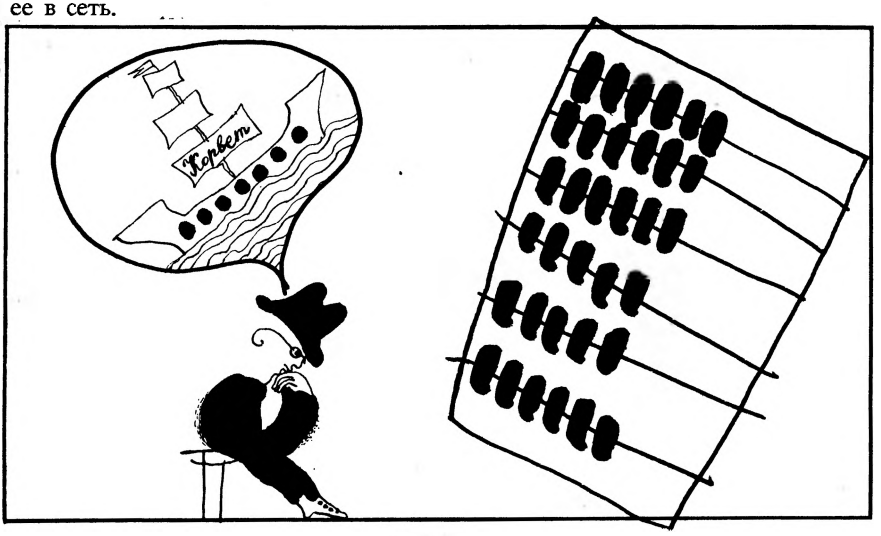
первые ПЭВМ, они норовили заглянуть в рядом стоящие шкафы или соседние помещения, чтобы найти собственно вычислительную машину. Они отказывались верить, что ПЭВМ умещается в корпусе дисплея.

Может создаться впечатление, что ПЭВМ - просто игрушка, не способная конкурировать с большими машинами серий ЕС или БЭСМ. Но это неверное впечатление. Сегодня есть ПЭВМ, которые по таким параметрам, как быстродействие и объем оперативной памяти, не уступают этим машинам.

Работа с ПЭВМ имеет ряд особенностей. Во-первых, пользователю приходится самостоятельно выполнять функции оператора. Дело в том, что ПЭВМ работает в диалоговом режиме. Что это такое? Персональная ЭВМ постоянно ведет разговор со своим хозяином. Она выводит на экран дисплея сообщения и ожидает команд с клавиатуры либо иных действий типа смены дисков, бумаги в цифропечатающем устройстве и т. д.

Во-вторых, пользователю необходимо самостоятельно работать с различными узлами ПЭВМ. Что это значит? Новая ПЭВМ попадает в руки пользователя в виде красивых, аккуратно заклеенных картонных коробок. Сгорая от любопытства, он достает из них новенькие части машины: дисплей, клавиатуру, дисководы, процессорный блок, цифропечатающее устройство и т. д. Вот, наконец, все детали извлечены и аккуратно расставлены на столе. Тут выясняется, что, кроме извлеченных устройств, в коробках лежат какие-то провода. Поскольку специалист, разбирающийся в этих проводах, к ПЭВМ не прилагается, пользователю необходимо разобраться в этой проблеме самому, т. е. правильно соединить все блоки и включить, наконец, машину.

В-третьих, пользователю приходится самому загружать систему. На человеческом языке это означает оживление ЭВМ после включения ее в сеть.



Итак, мы приходим к выводу, что переход пользователя от большой машины к персональной чреват для него рядом сложностей. Он должен выполнять обязанности и оператора ЭВМ, и системного программиста и в некоторой степени электронщика. Какие же преимущества имеет ПЭВМ перед большой машиной (если таковые вообще имеются)?

Прежде всего необходимо отметить, что реальное время счета не очень громоздких задач у ПЭВМ значительно меньше, чем у больших ЭВМ (если учесть время, затрачиваемое при работе на больших ЭВМ на поход в вычислительный центр и обратно, ожидание в очереди в дисплейном классе, ожидание в очереди задач, исполняемых машиной, ожидание выдачи результатов). Далее, ПЭВМ почти никогда не "зависает" (этот термин означает, что компьютер в данное время делает неизвестно что и на действия оператора, и тем более пользователя, не обращает никакого внимания). Неустойчивость работы больших ЭВМ приводит к столь большим потерям времени, что все большее число пользователей предпочитает ПЭВМ.

Другим преимуществом ПЭВМ является визуализация процесса счета. Богатые графические возможности ПЭВМ и диалоговый режим работы позволяют эффективно управлять решением самых различных задач, оперативно вмешиваясь в работу компьютера.

Наконец, ПЭВМ позволяют решать задачи, которые бессмысленно ставить на больших машинах. Это касается всевозможных автоматизированных картотек, записных книжек, блокнотов и т. п. Без диалогового режима работы на "персональном" уровне пользоваться такими программами невозможно. Трудно себе представить, что, пожелав прочитать телефон приятеля в записной книжке, мы согласимся ждать пару часов, пока нам вынесут распечатку. Отметим, что ПЭВМ, будучи подключенной к телефонной сети через специальное устройство, называемое модемом, может сама набирать необходимый номер и даже отвечать по телефону человеческим голосом.

За последние десять лет ПЭВМ произвели буквально революцию в организации многих сторон человеческой деятельности. Они взяли на себя решение множества бытовых задач - от организации семейного бюджета до всевозможных игр. Устанавливаемые на рабочем месте каждого научного работника и инженера, они позволяют оперативно проводить численные расчеты и конструировать новые устройства. Соединяемые в сеть, они способны управлять сложными экспериментальными установками и технологическими процессами. ПЭВМ избавляют нас от огромных каталогов и картотек в библиотеках.

Однако использование ПЭВМ теряет всякий смысл, если она не становится массовой. Такие машины должны обладать программной совместимостью, т. е. возможностью переноса программ с машин одного типа на машины другого типа.

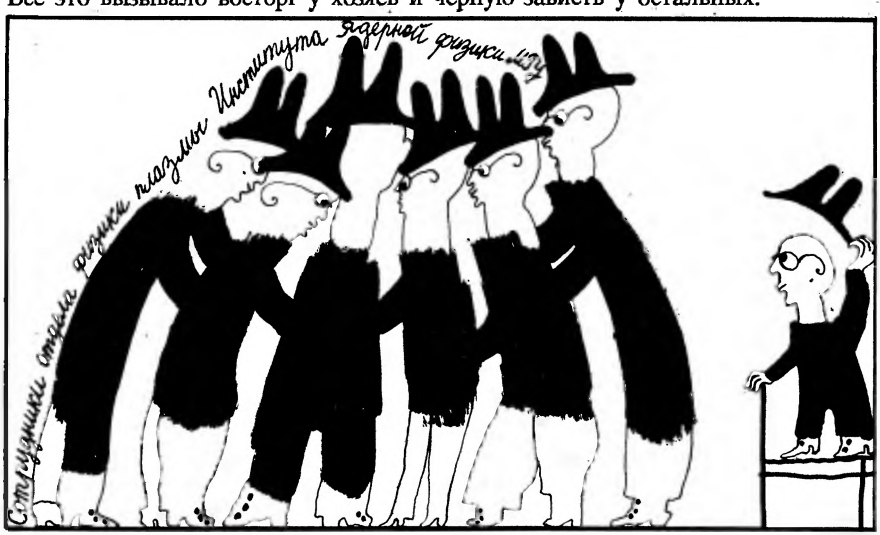
В настоящей книге мы познакомим начинающих пользователей с одним из первых массовых отечественных компьютеров - с персональной ЭВМ Корвет, с ее устройством и с приемами работы на ней. Итак, что может ПЭВМ Корвет?

2. ВОЗМОЖНОСТИ ПЭВМ КОРВЕТ

Мы знакомимся с ПЭВМ Корвет и начинаем понимать, какая это замечательная вещь

Прежде чем переходить к описанию ПЭВМ Корвет, коснемся истории создания этой машины. Разрабатывали Корвет сотрудники Отдела физики плазмы Института ядерной физики Московского государственного университета им. М. В. Ломоносова, недавние выпускники физического факультета, не имевшие до того времени большого опыта работы с вычислительными машинами. В один прекрасный день в их лаборатории появилось устройство, входящее в комплект физической аппаратуры и называемое TRS-80 (фирма Tandy, США). Этим устройством была одна из первых в мире ПЭВМ. Она сразу же прижилась в лаборатории, и через некоторое время число людей, желающих на ней поработать, стало настолько большим, что машина перестала быть персональной.

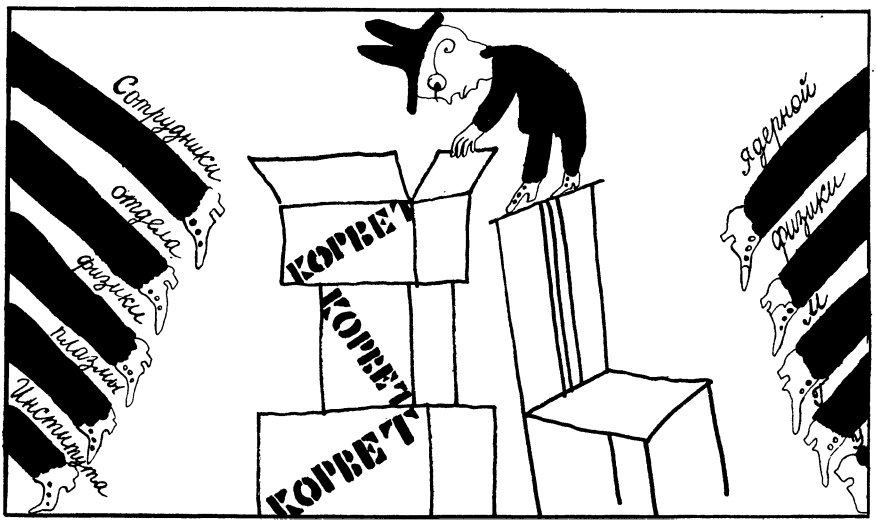
Чем же TRS-80 всем так понравилась? Прежде всего тем, что ее пользователи забыли, что такое карандаш и бумага. Все тексты статей, отчетов, характеристик вводились непосредственно в машину. Отпала необходимость многократно перепечатывать одно и то же с незначительными исправлениями. Редактирование велось уже на экране дисплея. Наличие достаточно мощных по тем временам языков программирования Бейсик и Фортран позволило достаточно быстро начать решать на этой машине вполне серьезные научные задачи. TRS-80 не обладала графическим дисплеем, однако она умела рисовать на экране картинки маленькими квадратиками. Тогда пользователи впервые увидели, что результат может быть выведен на экран в виде очень наглядной картинки вместо многометрового листинга (кипы бумаги, выползающей из цифропечатающего устройства при распечатке текстов программ и результатов счета). Все это вызывало восторг у хозяев и черную зависть у остальных.



Возможности приобрести такую или подобную машину всем желающим не представлялось. Выход напрашивался один: сделать свою собственную машину. Вначале не верилось, что из этой затеи что-нибудь получится, поскольку никто из потенциальных разработчиков до этого ЭВМ не делал. Однако не прошло и года, как на свет появился прообраз современной ПЭВМ Корвет. Оказалось, что физики могут разбираться в электронной начинке ЭВМ и в организации операционной системы так же, как и полубоги в белых халатах в вычислительных центрах. Примером этого может служить смелый эксперимент одного из создателей ПЭВМ Корвет, проделанный им еще в юности. Познавая тонкости работы машины БЭСМ-6, он ухитрился обойти многоступенчатую систему защиты и стереть системную ленту (т. е. магнитную ленту, на которой записана операционная система). Результатом эксперимента явилась остановка работы машины, пока из другого города не привезли новую ленту. Успехи программиста вызвали широкий резонанс среди общественности, и его отовсюду выгнали. К счастью, покровители проекта ПЭВМ Корвет поняли, что этот поступок свидетельствует о том, что перед ними высококлассный специалист с оригинальным взглядом на свои занятия, и он был включен в группу разработчиков.

Отметим, кстати, очень важный вывод, следующий из этого случая: все ценные программы, и в особенности операционную систему, необходимо копировать и работать в дальнейшем с копиями.

Но вернемся к ПЭВМ Корвет. Как только заработал первый образец, всем захотелось большего. Во главу угла ставилась задача сделать ПЭВМ Корвет компактной, надежной и ни в чем не уступающей аналогичным зарубежным образцам. В результате долгих споров было решено, что новая машина должна быть собрана на одной плате (это означает, что все входящие в машину радиодетали припаяны к одной и той же дощечке). Эту плату решили разместить в той же коробке, что и клавиатуру.



Разработчикам хотелось сделать процесс работы на машине максимально удобным для пользователя. Поэтому было принято решение разделить системы отображения текстовой и графической информации. В результате Корвет может обслуживать одновременно три телевизора. На одном из них отображаются тексты. В качестве дисплея может использоваться любой черно-белый телевизор. Два других предназначены для отображения различных картинок (на них также может выводиться и текст).

Здесь необходимо сделать небольшое отступление. Какие приборы могут использоваться в качестве дисплеев? К Корвету можно подключать как профессиональные дисплеи, так и бытовые телевизоры. В последнем случае врачи не рекомендуют работать с машиной больше 15 минут в неделю. Чтобы разрешить это противоречие, в Корвете предусмотрен ряд технических решений. Прежде всего это касается жесткой синхронизации работы всех устройств, входящих в его состав. В результате устранено "дрожание" изображения. Вторым важным моментом является размер матрицы для изображения символов. У Корвета она содержит 8*16 точек. В результате отображаемый текст получается в виде, не утомляющем зрение при продолжительной работе с машиной.

Если ЭВМ обладает цветным графическим дисплеем, то одной из ее характеристик является разрешение дисплея, т. е. число точек и число цветов, одновременно отображаемых на экране. Корвет имеет разрешение 512*256 точек и 16 цветов. Для сравнения приведем разрешение стандартного графического дисплея фирмы IBM (типа CGA): 300*200 точек и 4 цвета. Разрешение в конечном итоге влияет на качество цветной картинки. Корвет может отобразить на своем экране черно-белую фотографию человека, причем качество оказывается настолько высоким, что изображение почти неотличимо от оригинала.

В состав любой персональной ЭВМ входит устройство, позволяющее хранить после выключения машины программы и другую информацию. Та материальная субстанция, на которой записана эта информация, называется носителем. На заре развития вычислительной техники носитель изготавливался из бумаги. Это были перфолента и перфокарты. Информация записывалась на них путем пробивания дырочек в носителе (перфорации). Неудобство такого носителя очевидно и определяется непрочностью материала. Люди старой школы хорошо помнят рулоны перфоленты и толстенные колоды перфокарт, которые в течение нескольких секунд изрубались в лапшу неисправным считывателем. В свое время действовало неписанное правило: колоду перфокарт числом более ста считать невозможно.

К счастью, был изобретен магнитофон. Новый носитель в виде магнитной ленты позволил избавиться от неудобств использования перфоленты и перфокарт. Обычный магнитофон для большой машины - это шкаф высотой в человеческий рост и массой в добрую сотню килограммов. Естественно, таким устройством нельзя комплектовать

ПЭВМ. В ее состав входит самый обыкновенный бытовой кассетный магнитофон и запись ведется на самую обыкновенную компакт-кассету.

Чем же характеризуется способность ЭВМ записывать информацию на кассету? Естественно, в первую очередь объемом записанной на целую кассету информации.

Остановимся на единицах количества информации. На вопрос, что такое информация, ответить довольно трудно, однако измерять ее научились. Единицей количества информации является бит. Что это такое?

Любое цифровое устройство (в частности, ЭВМ) работает в двоичной системе счисления. Это значит, что любой элемент ЭВМ понимает только два состояния: "включено" и "выключено", или "1" и "0", "да" и "нет". Бит - это как раз и есть элементарная ячейка машины. Рассмотрим, например, дощечку с лунками, в которых могут размещаться шарики, по одному в лунке. В каждой лунке или есть шарик, или его нет. Припишем лунке число 1, если там есть шарик, и число 0, если его там нет. Каждая лунка и является битом. Если вы хотите сказать, есть ли шарик в лунке с номером 10 или нет, вы говорите, что десятый бит равен 1 или 0.

Восемь бит составляют другую важную единицу - байт. Почему восемь? Так как машина работает с двумя состояниями, то все числа удобно сортировать по степеням двойки. Так, $8 = 2^3$. Когда байтов становится очень много, информацию начинают измерять в килобайтах (Кбайт). Требование, чтобы 1 Кбайт составлял два в целой степени байтов, приводит к тому, что 1 Кбайт = 1024 байт, а не 1000 байт, как следовало бы из простой расшифровки приставки "кило" (например, 1 кг = 1000 г). Следующей единицей информации является мегабайт (1 Мбайт = 1024 Кбайт), и т. д.

Сколько же байтов можно записать на одну кассету? Это легко вычислить, если знать, сколько битов в секунду может передавать ЭВМ. Единица измерения скорости передачи информации называется бодом (1 бод = 1 бит/с). Корвет может передавать информацию со скоростью 2400 бод. Поскольку обычная кассета играет не меньше часа, то на нее можно записать

3600 с * 2400 бод = 8640000 бит = 1080000 байт информации

Мы видим, что кассета является довольно емким носителем информации. Чтобы наглядно представить ее емкость, подсчитаем, сколько страниц текста можно записать на одну кассету. Обычно один знак текста занимает 1 байт. Если принять, что на строке умещается 60 знаков и на странице 50 строк, то одна страница занимает 3000 байт. Таким образом, на кассету уместится

1080000/3000 = 360 страниц текста

При этом одна страница будет считываться за десять секунд.

Запись информации на компакт-кассету обладает существенным недостатком. Информация на нее записывается последовательно бит

за битом и таким же образом считывается. Поэтому мы можем считывать ее, только начиная каждый раз с начала. Следовательно, та информация, которая находится вдали от начала кассеты, попадает в машину с задержкой, связанной с перемоткой ленты.

Носителем, свободным от такого недостатка, является магнитный диск. Диск представляет собой пластинку, поверхность которой, так же как на магнитной ленте, покрыта магниточувствительным материалом. Для записи информации и считывания с диска требуется специальное устройство - дисковод. Магнитная головка дисковода размещается над вращающимся диском и может двигаться по радиусу диска от центра к периферии и обратно. Таким образом, она может быть очень быстро подведена к любой точке на диске. Благодаря этому считывание и запись информации на диск происходит во много раз быстрее, чем на ленту.

ПЭВМ должны уметь выводить информацию в виде, доступном для чтения человеком. Поэтому они, как правило, комплектуются цифропечатающим устройством (ЦПУ), или, как его еще называют, принтером (от англ. printer). Современные принтеры способны не только печатать буквы, цифры и другие знаки, но и рисовать цветные картинки. Большое впечатление производит распечатка твердой копии цветной картинки с экрана дисплея на бумаге (от англ. hard copy).

Итак, мы видим, что Корвет состоит из четырех основных блоков: клавиатуры, в которой размещена собственно ЭВМ, минимум одного телевизора, дисковода или кассетного магнитофона и принтера.

Рассмотрим теперь собственно ПЭВМ, которая спрятана в клавиатуре. ПЭВМ должна обязательно содержать три блока. Это блок, который думает (микропроцессор), блок, который помнит (оперативное запоминающее устройство ОЗУ и постоянное запоминающее устройство ПЗУ), и периферийные устройства, осуществляющие связь с внешним миром.

Что же характеризует ПЭВМ как вычислительное устройство? Безусловно, читатель неоднократно слышал, как в разговорах об ЭВМ мелькают слова "быстродействие", "число операций в секунду". Быстродействие ЭВМ - действительно важнейшая ее характеристика. Речь при этом идет, как правило, о простейших операциях сложения, сравнения или передачи информации. Само по себе быстродействие мало о чем говорит. Действительно, не ясно, что за объекты складываются, сравниваются или передаются.

В этом смысле поясняющим понятием является "разрядность" ЭВМ. Этот термин означает, какое количество информации может одновременно обработать микропроцессор ЭВМ. Бывают 8-, 16-, 24- и 32-разрядные машины. Поэтому быстродействие ЭВМ - это собирательное понятие, состоящее из числа операций в секунду и разрядности. ПЭВМ Корвет является 8-разрядной машиной и производит более 600 тысяч простейших операций в секунду. Для 8-разрядных машин это значение близко к рекордному. Отметим, что более сложные операции, скажем вычисление синуса или логарифма какого-нибудь числа, состоят

из тысяч простейших операций. Так что упомянутые 600 тысяч операций не должны порождать иллюзию, что любая программа будет выполнена мгновенно.

Следующим важным параметром, характеризующим ПЭВМ, является объем ее памяти. Память в ПЭВМ бывает двух сортов. Первый тип памяти - оперативная (ОЗУ), пользуясь которой можно в процессе работы записывать (запоминать) и считывать (вспоминать) любую информацию. Содержимое оперативной памяти пропадает при выключении машины.

Второй тип памяти - постоянная (ПЗУ). Из ПЗУ в процессе работы машины информацию можно только считывать. Для записи в ПЗУ требуются специальные устройства. При выключении ЭВМ информация из ПЗУ не пропадает. Обычно в ПЗУ содержатся программы, которые должны автоматически оживлять ЭВМ при ее включении. Как правило, в число таких программ входят тестовые программы, которые при включении машины проверяют работоспособность различных ее узлов, а также Бейсик.

Объем памяти ПЭВМ измеряется в килобайтах. Объем ОЗУ в ПЭВМ Корвет может составлять 256 Кбайт, объем ПЗУ - до 96 Кбайт. Много это или мало? Для сравнения укажем, что в TRS-80 объем ОЗУ составлял 48 Кбайт, а ПЗУ - 16 Кбайт.

Поговорим теперь о периферийных устройствах. Читатель, должно быть, несколько устал от всяких битов, байтов, ОЗУ, ПЗУ, и у него появилось сомнение, что он когда-нибудь разберется во всей этой премудрости. Не все так страшно. Работать с ПЭВМ можно и не имея представления об этих не совсем привычных вещах. Конечно, знание этих терминов совершенно необходимо для поддержания разговора среди умудренных опытом пользователей, которые, узнав, что вы слышите слово "байт" первый раз в жизни, смерят вас презрительным взглядом и изгонят из своей компании.

Что же делать с машиной человеку, который не знает, где у нее микропроцессор и что такое ПЗУ? Для того чтобы общаться с внешним миром, у ПЭВМ есть периферийные устройства. Эти устройства осуществляют связь прежде всего с пользователем. Их задача - уметь распознавать наши действия и в доступном для ПЭВМ виде вводить в нее полученную информацию или в доступном для нас виде выводить ее наружу. Спектр периферийных устройств достаточно широк и определяется запросами пользователя. Если вводимая пользователем в ПЭВМ информация имеет вид текстов (программ, статей, отчетов и т. п.), то ему достаточно единственного устройства ввода информации - клавиатуры.

Посмотрите на клавиатуру внимательно (рис. 1). Клавиши на ней делятся на четыре группы. Первая группа - это клавиши, на которых нанесены буквы, цифры и другие знаки. При нажатии на такую клавишу на экране дисплея появится отображение того знака, который на ней изображен. Называется эта группа основным полем, располагается в центре клавиатуры и однозначно соответствует клавиатуре обычной пишущей машинки.

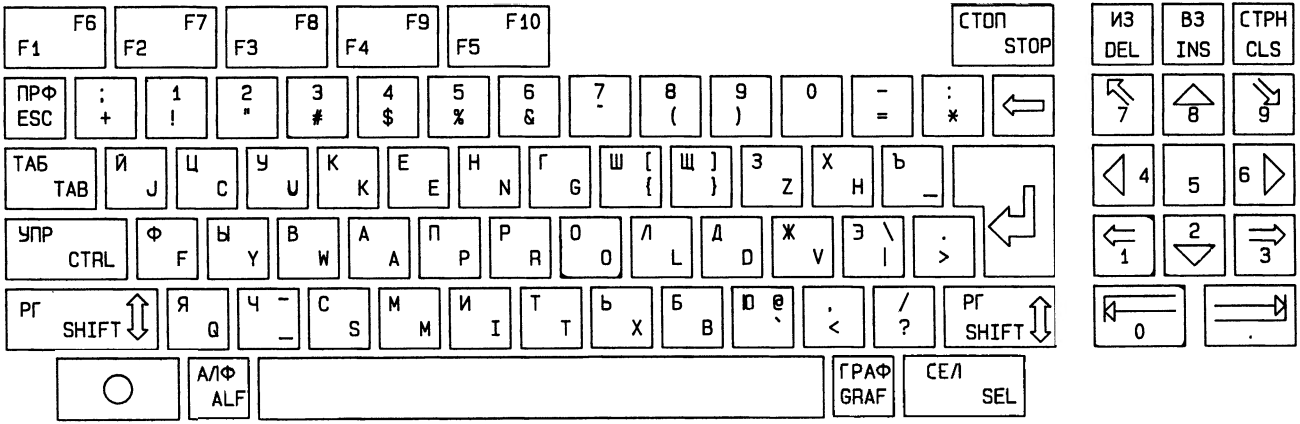


Рис.1. Клавиатура Корвета

Другая группа - это так называемые контрольные клавиши (переключение верхнего и нижнего регистров, переключение русского и латинского шрифтов и т. д.).

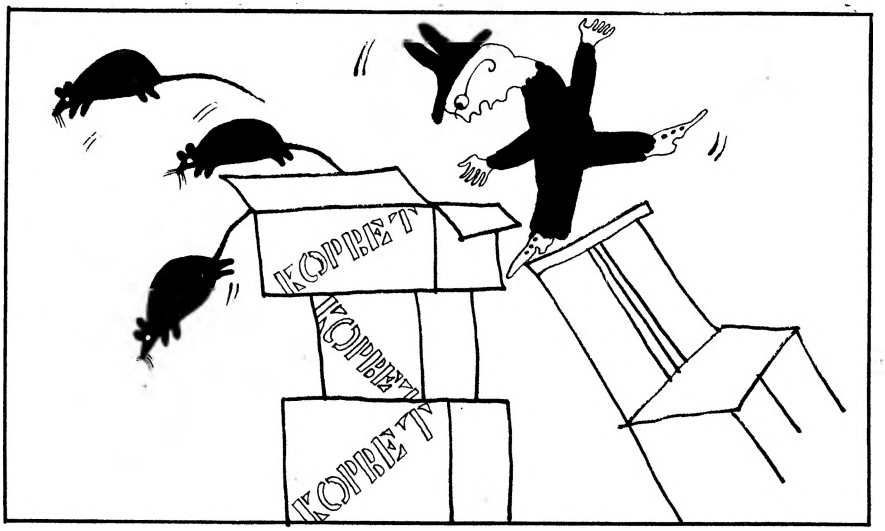
Очень полезная группа клавиш располагается в правой части клавиатуры, несколько в стороне от основного поля. Эти клавиши позволяют управлять курсором - светящейся меткой на экране дисплея, показывающей, где появится следующий знак, выводимый на экран. Где конкретно расположены все эти клавиши и как ими пользоваться - мы узнаем позднее.

Последняя группа - это функциональные клавиши. Они не имеют каких-либо жестко закрепленных за ними знаков. Их функцией является активизация различных подпрограмм в какой-нибудь серьезной программе типа редактора текста или языка Бейсик. С конкретными примерами таких подпрограмм мы познакомимся в разделе 8. Обычно на таких клавишах наносится буква F и следом - номер клавиши (например, F1).

Работа с клавиатурой обладает существенным недостатком: если пользователь плохо умеет печатать на пишущей машинке, то он работает с клавиатурой очень медленно.

Неужели такой красивый и совершенный аппарат, как Корвет, может только высвечивать на экране буквы и цифры, которые мы набираем на клавиатуре? Конечно, нет. Корвет может работать с массой устройств для ввода информации (это часто называется "поддерживать" устройства). Самыми привлекательными из них являются "мышь" и джойстик.

Мышь - штука очень занятная. Она немного похожа на своего живого собрата. Это небольшая коробочка с плавными очертаниями, привязанная с помощью тонкого провода к ПЭВМ. Если посмотреть на нее снизу, то мы увидим на ее дне круглое отверстие, откуда выгляды вает тяжелый покрытый резиной шарик. Сверху у мыши есть две (реже три) кнопочки. Действует мышь следующим образом. Для того чтобы



вводить в машину информацию, достаточно катать мышь по любой твердой плоскости и нажимать на кнопочки.

У вас, должно быть, возник вопрос, как же машина понимает, что ей делать, когда мы катаем мышь по столу. Сама машина, естественно, не понимает. Строго говоря, машина вообще ничего не понимает. Понимает программа, написанная человеком. Поэтому машина, кроме мыши, должна иметь и соответствующую программу, называемую драйвером (от англ. driver - водитель). Такая программа делает процесс пользования машиной очень приятным. На экране дисплея отображаются яркие прямоугольники, внутри которых имеются надписи, характеризующие ту или иную программу (например, считывание с диска, вывод на печать и т. д.). Кроме этой информации на экране присутствует маленькая стрелка, положение которой меняется при перемещении мыши по столу. Для активизации выбранной программы необходимо загнать стрелку с помощью мыши в выбранный квадратик и нажать кнопочку. Машина сразу "поймет", что ей делать. Это, конечно, гораздо удобнее, чем бесконечное печатание на клавиатуре инструкций для ЭВМ: сделай то, подай это.

Мышь позволяет сделать процесс работы на ПЭВМ приятным и неутомительным. Особенно это касается игр. Но игра игре рознь. Если от вас требуется перемещать какие-либо объекты по экрану (наример, шахматные фигуры, самолеты, автомобили и т. д.), то мышь вполне с этой задачей справляется. Но бывают игры, в которых создается иллюзия того, что вы сами находитесь в кабине самолета, автомобиля, у перископа подводной лодки, - здесь мышь снижает наглядность игры и может отбить интерес и желание в нее играть. Для таких игр придумано устройство, называемое джойстиком (от англ. joystick - рукоятка для игры). Это устройство напоминает рукоять управления в самолете. Если вы совершаете полет в воображаемом самолете, то управление им с помощью джойстика очень похоже на управление настоящим самолетом.

Существуют еще бесчисленные изобретения, например "волшебные" палочки, позволяющие писать прямо на экране дисплея.

Отметим одну существенную деталь общения с ПЭВМ. Она может издавать различные звуки - от писка до имитации работы двигателя самолета и музыкальных инструментов. Разработчики ЭВМ и программисты, пишущие для них программы, доказывая, что не даром едят свой хлеб, буквально засыпали весь мир подобными игрушками. Чтобы использовать это богатство, ваша машина должна быть программно совместимой с другими ЭВМ. Это ясно понимали и создатели ПЭВМ Корвет. Его делали "по максимуму".

В мире существует огромное количество программ, рассчитанных именно на персональную специфику компьютера, которыми может воспользоваться владелец ПЭВМ Корвет, и нет таких периферийных устройств, которые нельзя было бы подключить к нему. При конструировании ПЭВМ Корвет был учтен мировой опыт разработки машин подобного класса. В результате всей этой деятельности получилась

машина, оказавшаяся столь удачной, что было принято решение о ее массовом производстве. После этого была проведена большая работа по внедрению опытного образца в серийное производство. Эта безусловно очень важная часть работ была выполнена на начальной стадии сотрудниками НИИ вычислительных комплексов, а затем НИИсчетмаш и БПО "Радиостроение" при постоянном творческом контакте с сотрудниками НИИ ядерной физики МГУ.

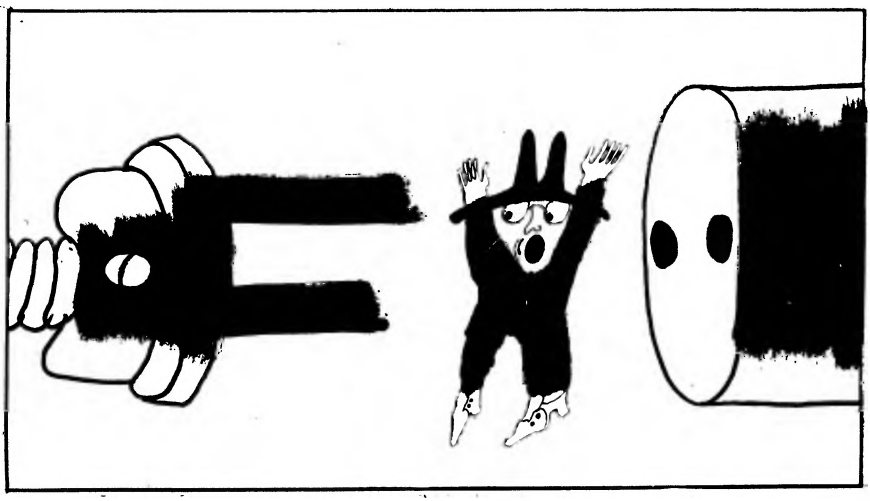
Теперь вы имеете представление о том, какая замечательная вещь стоит у вас на столе. Сознание предоставленных вам неисчерпанных возможностей наполняет сердце восторгом. Однако в глубине души шевелится страх и недоверие. Не волнуйтесь. Мы переходим, наконец, к процессу включения ПЭВМ Корвет в сеть.

3. ВКЛЮЧЕНИЕ ПЭВМ КОРВЕТ

Как овладеть искусством включения ПЭВМ, избежать при этом нервного расстройства и в конце концов возвыситься над непосвященным человечеством

Наконец настала долгожданная минута. Вы чувствуете себя достаточно подготовленным специалистом для того, чтобы решиться на ответственный шаг: мановением руки перевести тумблер СЕТЬ, расположенный на левой стороне блока питания, в положение ВКЛ. После того как по жилам ЭВМ потечет электрический ток, уже поздно менять местами неправильно подключенные провода. Возможно, придется вызывать мастера и в дальнейшем с трудом избавляться от комплекса неполноценности.

Поэтому мы начнем нашу беседу с рассмотрения правил соединения различных блоков ПЭВМ Корвет. Советуем вам обратить внимание на рис. 2. Отметим, что состав конкретной ПЭВМ может быть различ-



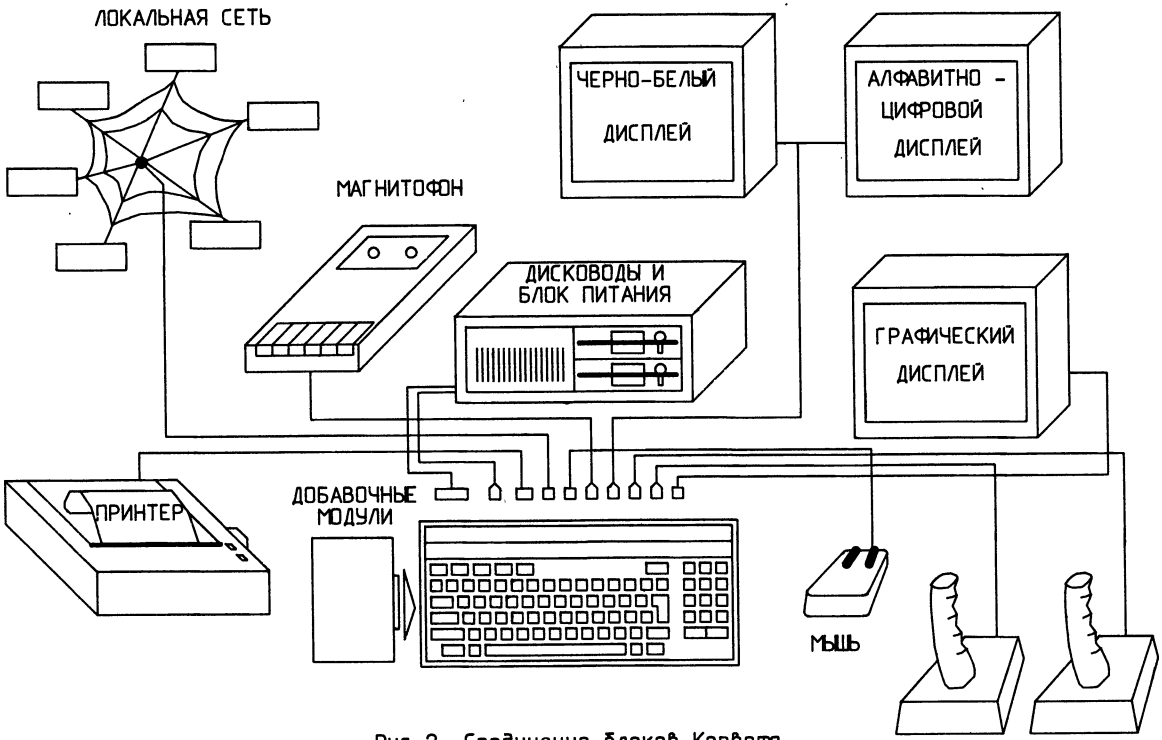


Рис.2. Соединение блоков Корвета

ным. (В таких случаях говорят, что машина может иметь различную конфигурацию.) Для уточнения имеющейся у вас конфигурации ознакомьтесь с прилагаемой к машине инструкцией по эксплуатации. Здесь мы остановимся на ряде особенностей, общих для любой конфигурации Корвета. Прежде всего найдите разъем подключения питания. Отметим, что попытки подключить питание к разъемам дисплея или локальной сети не приводят к оживлению ЭВМ, а, наоборот, надолго выведут ее из строя.

Итак, разъем питания. Возьмите блок клавиатуры в руки и поверните его задней стенкой вверх. Посмотрите на нее. Сопоставьте то, что вы видите, с рис. 3. Вы видите множество разъемов. Справа (если смотреть со стороны задней стенки) расположен длинный разъем с множеством контактов. Хотя он и выглядит очень внушительно, это не разъем питания. Через него подключают дисководы. Разъем питания находится следом за ним. Он содержит четыре контакта. Попытайтесь вставить в него ответную часть, имеющуюся на шнуре, связанном с блоком питания. Если получилось, поставьте себе оценку "отлично" и пропустите следующий абзац.

Если нет, то не отчаивайтесь. Попробуйте еще раз. Во-первых, убедитесь, что вы вставляете ответную часть разъема куда надо. Для этого перечитайте предыдущий абзац и повторите действия. Если опять неудача, то поверните ответную часть разъема на 180 градусов и попробуйте еще раз. После пятой попытки приляжте на диван и утешьтесь включением и выключением телевизора. Это укрепит вас в убежденности, что техника вам подвластна. На следующий день повторите все сначала, и так до тех пор, пока не получится.

Наконец получилось. Вздохните с облегчением и отдохните.

Следующее устройство - дисплей. Прежде всего выясните, какой из двух типов дисплеев вы собираетесь подключать. Рассмотрим черно-белый дисплей. Он подключается к машине с помощью пятиштырькового разъема, используемого обычно в бытовых магнитофонах, проигрывателях и т. п. Если вы ничего подобного в жизни не видели, обратитесь за помощью к знакомым.

Найдите теперь место подключения черно-белого телевизора к ПЭВМ Корвет. Продолжая держать машину задней стенкой вверх, находим два пятиштырьковых разъема. Они находятся в левом углу (рис. 3). Один из них (справа) - разъем для подключения кассетного магнитофона. Запомните это. Следующий - разъем для подключения черно-белого телевизора. Попробуйте вставить в него кабель, подсоединенный к телевизору. Если получилось, поздравьте себя с победой и пропустите следующий абзац.

Если возникли проблемы, не отчаивайтесь. Попробуйте повторить предыдущие действия. Для этого, во-первых, убедитесь, что ваш телевизор действительно черно-белый. Затем посмотрите на ответную часть разъема на соединительном кабеле. Этот разъем должен быть круглым и иметь пять штырьков. Если это не так, обратитесь в гарантийную мастерскую.

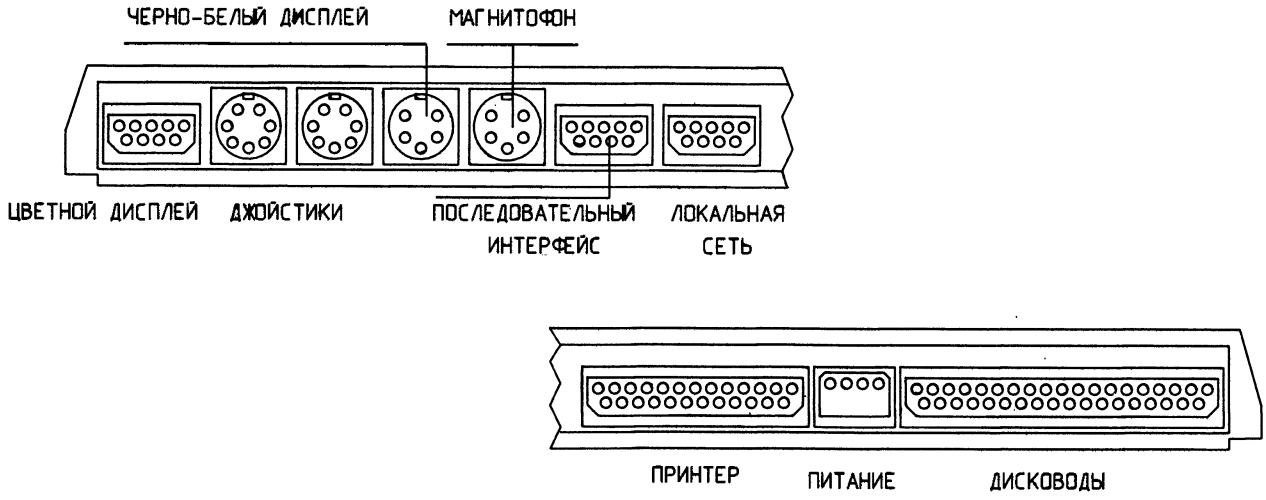


Рис.3. Задняя стенка Корвета

Рассмотрим процесс подключения цветного дисплея. Он снабжен кабелем с девятиштырьковым разъемом. На задней стенке ЭВМ ему соответствует крайний слева разъем. Попробуйте присоединить кабель цветного дисплея к ЭВМ. Ваши действия должны быть полностью аналогичны процессу подключения черно-белого дисплея.

Осталось подключить дисковод или магнитофон и принтер. Процесс подключения магнитофона аналогичен подключению черно-белого телевизора. Разъем кассетного магнитофона круглый и содержит пять штырьков. На задней стенке ЭВМ он первый справа в ряду круглых разъемов. Подключайте магнитофон так же, как если бы вы собирались записывать на него музыку.

Теперь дисковод. Разъем дисковода выделяется из всех своими размерами. Если не полениться и подсчитать, то получится, что на нем 37 контактов (рис. 3). Отметим, что на левой боковой стороне Корвета имеется похожий разъем. Он к дисководам никакого отношения не имеет, и его назначение будет объяснено позднее.

Принтер подключается через 25-штырьковый разъем, расположенный слева от разъема питания.

Теперь о некоторых дополнительных устройствах. Вы помните, что ПЭВМ обычно комплектуются мышью и джойстиками. Куда их подключать? Начнем с мыши. Мышь бывает различной конструкции и подключается к ПЭВМ по-разному. К сожалению, из всего семейства мышей к Корвету можно подключить только такие, которые работают через последовательный интерфейс.

Что это такое? Сначала разберемся с терминологией. В литературе, посвященной ПЭВМ, постоянно встречаются слова: интерфейс, адаптер, контроллер и т. д. Все эти термины в сущности означают одно и то же. Вы уже знаете, что периферийные устройства подключаются к ПЭВМ через разъемы. А к чему подключены сами разъемы? Естественно, к некоторым устройствам внутри ПЭВМ, которые обеспечивают связь с микропроцессором. Такие устройства и называются вышеперечисленными именами. Так, контроллер дисководов - это устройство для подключения дисководов, интерфейс принтера - устройство для подключения принтера, адаптер графического дисплея - устройство для подключения графического дисплея.

Теперь вернемся к последовательному интерфейсу. Вы, наверное, помните, как информация передается из машины в кассетный магнитофон? Мы передаем информацию бит за битом по одному проводу. Такой способ передачи называется последовательным. (Если передача или прием информации производится по нескольким проводам одновременно, такой способ называется параллельным. Примером интерфейса для параллельного способа передачи информации служит интерфейс принтера.) Последовательный интерфейс Корвета выполнен в международном стандарте RS232. К этому интерфейсу и подключается мышь. Давайте найдем соответствующий ей разъем. Обратите внимание на два девятиштырьковых разъема слева от разъема принтера. Первый из них -

разъем для подключения локальной сети, а второй разъем как раз и есть RS232.

Хорошо, с мышью разобрались. А что такое локальная сеть? Термин "локальная сеть" применяется для обозначения семейства устройств, соединенных друг с другом двумя проводами. Эти провода представляют собой замкнутый круг, от которого ответвляются провода, идущие к этим устройствам. По этим проводам циркулирует передаваемая информация. Внешне такой способ соединения устройств напоминает паутину, что и отражено на рис. 2. Устройствами, объединяемыми в локальную сеть, могут быть сами ПЭВМ или какие-либо приборы.

ПЭВМ Корвет создавалась как базовая машина для средних школ. Поэтому разработчики предусмотрели возможность объединения нескольких таких машин в учебный класс. Это делается через локальную сеть. Передача информации по локальной сети осуществляется последовательным способом.

В чем же отличие RS232 от локальной сети? В общих чертах в следующем. Через RS232 к ПЭВМ Корвет можно подключить (не принимая специальных мер) только одно устройство, например мышь. Это связано с тем, что у передаваемых через RS232 данных нет никакой пометки, откуда и куда они идут. В случае же локальной сети положение иное. Здесь данные можно уподобить электропоезду, на головном вагоне которого указана станция назначения. Любое устройство, подключенное к локальной сети, умеет распознавать свое имя и, обнаружив его, принимает посланную ему информацию. Передача информации в обратную сторону происходит аналогично.

Теперь займемся джойстиками. Два семиштырьковых разъема для них находятся слева от разъема черно-белого дисплея.

Остался один разъем на левой боковой стенке Корвета. Этот разъем имеет такой же вид, как и разъем дисководов. Через него к Корвету можно подключать самые разнообразные устройства, например программаторы ПЗУ, различные измерительные устройства и др. Для радиолюбителей укажем, что на этот разъем выведены все выходные ножки микросхемы КР580ВВ55А, один из каналов прерывания и один канал таймера. За подробностями обращайтесь к принципиальной схеме ПЭВМ Корвет.

Мы надеемся, что к этому моменту вы уже стали крупным специалистом по разъемам и кабелям и легко справитесь с поставленной в начале раздела задачей. Итак, все кабели подсоединены. Проверьте, не осталось ли лишних проводов. Если остались, то еще раз проверьте, все ли правильно подсоединено и не забыли ли вы подключить какое-либо устройство.

Наступает момент, неизбежный в жизни любого пользователя. Сейчас он перешагнет невидимый рубеж и окунется в неизвестность. Пока он точно знает, что ничего в ПЭВМ не сломал. Вот он протягивает руку к тумблеру СЕТЬ и включает машину. Она весело приветствует его миганием лампочек и похрюкиванием дисководов (если они есть). Каж-

дому ясно, каким ответственным делом является включение машины. Поэтому здесь мы дадим ряд рекомендаций.

Для начала убедитесь, что в дисководы не вставлены диски. Скорее всего, в новом добытом из коробки дисководе дисков нет. Если они вставлены, то попробуйте достать их оттуда. Методика работы с дисководами изложена ниже, в разделе 4. Затем включите дисплей и дайте ему прогреться. Это необходимо для контроля работы ЭВМ непосредственно с момента ее включения. Итак, телевизор за 1 - 2 минуты прогрелся. Теперь наберитесь храбрости и включите машину. В ответ на ваши действия машина должна весело пискнуть и при отсутствии дисков через 1 - 2 минуты написать на экране телевизора:

```
Бейсик Корвет в. 1
Москва 1986

OK
```

Это означает, что загружен язык Бейсик и ПЭВМ готова принимать любые команды на этом языке (детали этой или подобной надписи станут вам понятными в дальнейшем).

Поясним термин "загрузка". Для того чтобы понять его смысл, рассмотрим, как работает думающее устройство - микропроцессор.

Микропроцессор понимает и умеет выполнять определенные команды. Например, занесение в ячейку памяти числа (запоминание числа), сложение или сравнение двух чисел и т. п. Последовательность команд для микропроцессора называется программой. Программу пишет человек.

Итак, если мы хотим заставить процессор выполнять определенные действия, мы должны последовательно давать ему отдельные команды. Естественно, сделать это вручную невозможно из-за большого быстродействия ЭВМ около 600 тысяч операций в секунду). Поэтому процессор сделан таким образом, что умеет сам выбирать последовательность команд, составляющих программу. Наша задача - поместить программу в память ЭВМ (загрузить ее).

В процессоре есть специальное устройство, которое умеет считать команды (счетчик команд). Оно знает, в какой ячейке памяти лежит следующая команда. С помощью счетчика команд процессор всегда может определить, какую команду выполнять в следующий момент времени.

Что же происходит при включении ПЭВМ? Как вы помните, в состав ПЭВМ входит постоянное запоминающее устройство (ПЗУ). В нем содержится программа, написанная изготовителями машины. При включении ПЭВМ счетчик команд процессора автоматически настраивается на начало программы в ПЗУ, т. е. в него заносится номер ячейки, где лежит первая команда программы. Процессор читает первую команду и выполняет ее, читает следующую и т. д.

Что же происходит при выполнении программы? Прежде всего, это установка всех устройств ЭВМ в исходное состояние (инициали-

зация). Следующим важным моментом является тестирование (проверка работоспособности) различных узлов ЭВМ, т. е. первое, с чего машина начинает, - она сама себя проверяет. Если встречается ошибка, машина выводит сообщение о ней на экран и останавливается. Ошибки при проверках функционирования системного блока носят, как правило, фатальный характер. В этом случае в верхнем правом углу экрана обычно появляется сообщение:

СИСТЕМНЫЙ БЛОК НЕИСПРАВЕН

и машина останавливается. В этом случае необходимо осуществить гарантийное обслуживание ПЭВМ. Если тестирование прошло без ошибок, ПЭВМ переходит в состояние ожидания команд, вводимых вами с клавиатуры. При этом, естественно, нельзя набирать на клавиатуре любые команды, какие взбредут в голову. Попытки попросить ПЭВМ приготовить кофе или сходить за газетами ни к чему не приведут. Список разрешенных команд однозначно задается языком, который ПЭВМ понимает в данный момент. Включенная в сеть ПЭВМ понимает язык, записанный в ПЗУ. В Корвете это Бейсик.

Теперь вы можете поговорить с машиной. Делайте это безбоязненно. Нажимая клавиши на клавиатуре, сломать машину невозможно. Те, кому не интересно разбираться в работе внешних устройств типа дисководов и кассетного магнитофона, могут перейти сразу к изучению языка Бейсик, т. е. сразу начинать читать раздел 8. В дальнейшем будем именовать его просто Бейсиком, как старого знакомого, опуская титул "язык".

Многим пользователям, возможно, на всю жизнь хватит Бейсика. Действительно, вам не нужно вникать в тонкости работы тех или иных устройств, входящих в ЭВМ. Все уже сделано за вас, а вы общаетесь с ЭВМ с помощью простых и понятных команд Бейсика. Однако следует отметить, что существует огромное количество полезных программ, которые никакого отношения к Бейсику не имеют. Например, как приятно хранить картотеку вашей библиотеки на диске или кассете и в случае надобности разыскивать информацию о тех или иных книгах и статьях, вводя в ПЭВМ фамилии авторов или ключевое слово. Кроме того, если вы хотите писать программы, решающие сложные в математическом отношении задачи, то обнаружится, что Бейсик работает очень медленно. Поэтому, может быть, имеет смысл не останавливаться на Бейсике и продолжить процесс загрузки дальше?

Возникает вопрос, где взять соответствующую программу? Ответ: на внешнем носителе, т. е. на диске или кассете. Пользуясь инструкцией по эксплуатации, найдите системный диск или системную кассету. Термин "системный" означает, что на этом носителе записана операционная система. Операционная система - это программа, связывающая пользователя с ЭВМ. Она позволяет производить различные действия с устройствами, входящими в состав ЭВМ (дисководами, магнитофоном, принтером), выполнять различные программы, написанные пользователем не обязательно на языке Бейсик, и т. д. Вставьте, не выключая маши-

ны, системный диск в дисковод А (см. инструкцию по эксплуатации) или системную кассету в магнитофон. Дисководы обычно нумеруются буквами латинского алфавита: А, В, С и т. д.

Отметим, что с ПЭВМ Корвет могут работать различные операционные системы. Наиболее распространенными являются CP/M-80 (далее будем обозначать CP/M) и программно совместимая с ней МикроДОС. Мы в этой книге будем ссылаться на CP/M, как имеющую большее распространение в мире. Если вам по каким-либо причинам больше нравится МикроДОС, обратитесь к прилагаемой к Корвету "Инструкции оператора". Там эти вопросы освещены. Если вы сравните программы, входящие в эти операционные системы, то увидите, что они во многом похожи. Поэтому не пугайтесь, если при выполнении самостоятельных упражнений или при изучении тех или иных программ надписи, появляющиеся на экране, будут несколько отличаться. Это значит, что просто вам досталась иная версия. В целом же основные программы работают одинаково в обеих операционных системах.

Загрузить операционную систему с диска можно двумя способами. Первый - набрать команду SYSTEM языка Бейсик и нажать на большую клавишу, на которой изображена изогнутая влево стрелка. Эта клавиша называется ВК (Возврат Каретки), а также CR (от англ. Carriage Return). Ее нажатие всегда означает для ЭВМ ввод команды. Если в вашей версии резидентного Бейсика команды SYSTEM нет, остается второй способ: нажать на кнопку СБРОС, расположенную на правой боковой стенке блока клавиатуры. В результате этих действий на экране появится сообщение:

```
CP/M-80 v. 2.2
ОФП НИИЯФ МГУ BIOS
Ver. 1.2 (c) III 1988

A>
```

Это значит, что загружена операционная система CP/M-80, версия 2.2, и машина готова принимать команды из словаря CP/M-80, вводимые нами с клавиатуры. Отметим, что содержание третьей строки может изменяться в зависимости от даты выпуска машины.

Что же делать тем, у кого нет дисководов? В этом случае приходится работать с магнитофоном. В простейшем варианте магнитофон позволяет загружать и запоминать программы только на Бейсике, т. е. никакой операционной системы нет. Машина при включении загружает из ПЗУ Бейсик, и вы можете хранить свои программы на кассете. Это, конечно, не очень удобно, но что поделаешь.

Однако существует маленькая хитрость. Вы помните, что у Корвета довольно большая оперативная память. Так вот, существует программа, которая превращает часть этой памяти в аналог магнитного диска. Работа с памятью происходит так же, как и с обычным диском. Поэтому она называется электронным диском. Он довольно емкий и в некоторых модификациях Корвета может достигать 144 Кбайт. Если

вы купили Корвет, для которого есть возможность использовать электронный диск, то можно загрузить на него операционную систему СР/М. В результате удобства работы с такой машиной не будут сильно отличаться от варианта с дисководами. Придется лишь немного подождать (5 - 10 минут), пока с магнитофона загрузится операционная система.

Использование электронного диска определяется конфигурацией самой машины. Поэтому с этим вопросом разберитесь с помощью инструкции по эксплуатации. Если возникнут проблемы, обратитесь в гарантийную мастерскую или к поставщикам. В любом случае запомните, что каждая конфигурация Корвета может быть изменена (заменой нескольких микросхем памяти) так, чтобы использовать электронный диск.

Отметим, что прежде чем начинать нажимать на клавиши, необходимо снять копии с системных дисков или кассет. В противном случае вы можете случайно стереть с них операционную систему и машина превратится в безжизненную груду радиодеталей.

Итак, процесс загрузки окончен. Следующая ваша задача - скопировать системные диски. Как это сделать - объясняется в разделе 5, но прежде чем переходить к нему, полезно узнать основные принципы хранения информации на диске.

4. ХРАНЕНИЕ ИНФОРМАЦИИ НА ДИСКАХ И ЛЕНТЕ

О различных способах сохранения программ
на вечные времена

После того как вы включили ЭВМ, вам очень хочется поскорее попробовать на ней поработать. Вы воображаете себя асом-программистом. С легкостью, вызывающей зависть окружающих, вы решаете сложнейшие задачи. Нажав только вам известную комбинацию клавиш, вы элементарно выводите машину из тяжелого зависания. Для вас уже нет никаких проблем. Но вспомним печальный случай с системной лентой на БЭСМ-6. Поэтому подождем нажимать на кнопки и займемся нашими носителями информации, т. е. дисками и лентой. Сейчас мы с вами познакомимся с тем, как информация на них записывается и считывается с них.

Начнем с дисков, как наиболее удобных носителей информации. Принцип записи на магнитный диск аналогичен принципу записи в любом магнитофоне. Однако, в отличие от ленты, диск не имеет ни начала, ни конца. Поэтому возникает вопрос: как найти на нем записанную информацию? Единственный способ - это разметить диск, т. е. разделить его на отдельные области. Простейший путь разметки дисков был реализован при изготовлении граммофонных пластинок. На диск нанесена дорожка, имеющая вид спирали. Попав на нее, игла звукоснимателя неизбежно дойдет до ее конца, прочитав всю записанную на

дорожку информацию. Однако этот способ не очень удобен из-за необходимости проходить каждый раз всю длину дорожки, предшествующую тому месту, где записана интересующая нас информация.

Попробуем устранить этот недостаток. Сделаем на диске не одну дорожку в виде спирали, а много в виде концентрических окружностей. При этом сделаем так, чтобы головка дисковода могла перемещаться по радиусу диска скачками. Длина такого скачка будет равна ширине дорожки. Теперь, если при записи мы запомнили, на какую дорожку записана информация, то при считывании мы можем вывести головку в нужное место сразу, не читая информации, записанной на других дорожках. Дорожка на магнитном диске часто называется треком (от англ. track). Диск можно использовать с двух сторон. Поэтому в дисководах обычно ставят две головки - сверху и снизу. Разметка диска в этом случае приобретает еще одно измерение. Тогда дорожка часто называется цилиндром. На одну дорожку можно записать довольно много информации. Поэтому вводится более мелкая разметка: дорожка разбивается на секторы. На один сектор можно записать 128, 256, 512 или 1024 байт информации. Количество дорожек на диске может быть различным и определяется качеством диска и конструкцией дисковода.

Чем же определяется выбор количества дорожек при разметке диска? Прежде всего конструкцией дисковода: шириной его магнитной головки и диаметром диска. Принято три стандартных размера дисков: восьми-, пяти- и трехдюймовые диски. Сегодня во всем мире персональные компьютеры восьмидюймовыми дисками не комплектуются. Поэтому в дальнейшем восьмидюймовые диски мы рассматривать не будем. В зависимости от ширины магнитной головки дисководы для пятидюймовых дисков могут быть 40- или 80-трековые, т. е. при разметке диска может получиться 40 или 80 дорожек. Трехдюймовые диски обычно всегда размечаются на 80 треков.

Рассмотрим, от чего зависит количество секторов, помещающихся на одну дорожку. Прежде всего, количество секторов зависит от размера сектора. Это естественно. Если взять сектор длиной в 512 байт, таких секторов на одну дорожку поместится в два раза больше, чем секторов длиной 1024 байта. Но может возникнуть вопрос: а чем определяется максимально возможное количество секторов заданной длины на одной дорожке? Оно зависит от качества диска и быстродействия машины. Поскольку скорость вращения диска постоянна, плотность записи информации будет зависеть только от скорости ее передачи от ПЭВМ в дисковод.

Кроме того, у магнитного покрытия диска есть некоторый минимальный размер магнитной ячейки. Одну ячейку нельзя заставить принимать два состояния одновременно. Поясним это простым примером. Вы, конечно, хорошо знакомы с таким прибором, как компас. Представьте себе несколько необычный компас. Стрелка у него вращается не свободно, а может занимать только два положения - показывать на юг или на север. Поднося сильный магнит к стрелке, мы

можем переводить ее из одного состояния в другое. Такая стрелка и соответствует ячейке на диске. Другими словами, магнитное покрытие диска представляет собой большое количество таких стрелок, расположенных вплотную друг к другу.

Запись информации происходит при переворачивании стрелки из одного состояния в другое. Теперь легко понять, чем определяется плотность записи информации на магнитном диске. К моменту поступления очередного бита диск должен успеть повернуться настолько, чтобы под головкой оказалась следующая стрелка. Чем меньше размер стрелок, тем быстрее мы можем передавать информацию и тем больше ее мы можем записать на диск.

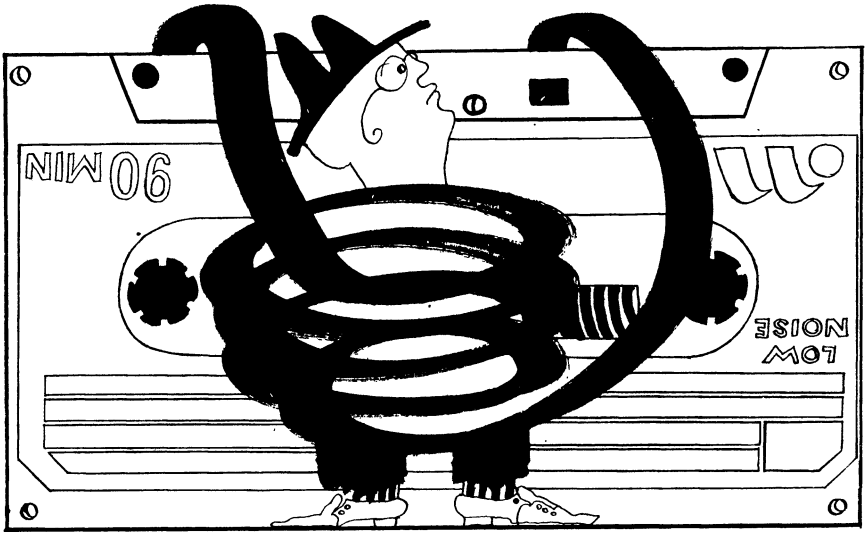
Что же реально происходит при записи информации Корветом на диск? Длина сектора в этой машине выбрана 1024 байта. Максимальное количество секторов, которое можно записать на одну дорожку пяти- и трехдюймовых дисков, равно пяти. Теперь легко подсчитать полную емкость таких дисков: для одностороннего диска она составляет 400 Кбайт, а для двустороннего - 800 Кбайт.

Итак, мы поняли, для чего надо размечать диск. Мы знаем, что для нахождения записанной информации необходимо запомнить, на каких треках, секторах и сторонах диска она записана. Поэтому на дисках выделяется область, где хранятся эти сведения. Эта область называется директорией (от англ. directory - каталог, оглавление). По существу, директория не отличается от оглавления в обычной книге. Если мы ищем программу с именем EXAMPLE1, то, просмотрев директорию, мы прочтем напротив имени программы информацию о том, где на диске она расположена.

Естественно, чтением директории и ее анализом занимается не сам пользователь, а соответствующая программа. Наша задача - только сообщить ей имя, под которым записана программа. Если мы хотим посмотреть, что записано на всем диске, мы можем попросить операционную систему вывести на экран содержимое директории. Для этого достаточно набрать команду DIR, после чего указать номер диска, поставить двоеточие и нажать на клавишу ВК, т. е. команда DIR A: означает, что мы хотим просмотреть директорию диска А. Попробуйте ознакомиться с содержимым вашего системного диска.

Как вы уже знаете, на один диск помещается довольно много информации. Если количество программ очень велико, то директория становится настолько большой, что ею неудобно пользоваться. Поэтому бывает удобно разбить диск на разделы. Такой раздел называется юзером (от англ. user) и нумеруется от 0 до 15. При использовании юзеров диск напоминает телефонный справочник, где вначале приводится грубое оглавление по разделам, а внутри каждого раздела имеется собственное оглавление. Для перехода из одного раздела в другой служит команда USER N (N - номер юзера).

Рассмотрим теперь вопрос хранения информации на магнитной ленте. Прежде всего отметим, что если вы решили сэкономить деньги и купили машину без дисководов, то это экономия на вашем собственном



здоровье. Компакт-кассета - вещь, конечно, очень хорошая, но придумана она не для записи программ. В сущности высокопродуктивная работа на ЭВМ, укомплектованной вместо дисководов кассетным магнитофоном, невозможна. Почему?

Во-первых, ненадежен носитель информации. Каждый, у кого есть магнитофон, неоднократно сталкивался с явлением "жевания" ленты. Если в обычной жизни "изжеванный" участок просто плохо воспринимается на слух, то при работе на ПЭВМ "жеваная" лента полностью непригодна.

Во-вторых, кассетный магнитофон придуман для воспроиведения и записи звука, а не программ. Поэтому мы не знаем точно, куда на ленту мы записали ту или иную программу. В результате при работе с лентами каждую кассету вам нужно снабжать бумажкой, на которой написано, какому числу на счетчике расхода ленты в магнитофоне соответствует начало программы. Естественно, что перемотка ленты автоматически не происходит, а выполняется самим пользователем. Поэтому процесс копирования программ с одной ленты на другую занимает большое время.

Учитывая, что человек, однажды работавший с дисками, ни за какие коврижки не согласится возиться с кассетами, мы в дальнейшем будем считать, что дисководы у вас есть.

При работе с лентой необходимо соблюдать технику безопасности. Как вы знаете, информация на кассету записывается последовательно. Одновременно вы сами на бумажке записываете показания счетчика, отмечая начало и конец программы. Потеря этой бумажки означает потерю всех программ на ленте. Это, вообще говоря, еще одно большое "удобство". Если бумажка потерялась, то понять, что записано на кассете, не потратив на это занятие несколько часов изнурительного труда, невозможно. Если вы хотите записать на место одной программы другую, вы должны быть уверены, что вторая про-

грамма короче первой. В противном случае сотрется программа, идущая следом.

Если вы еще недостаточно напуганы и не находитесь на пути в магазин, где продаются дисководы, приведем еще несколько соображений. Допустим, что вы сверхаккуратный человек и как зеницу ока храните пояснительные бумажки к кассетам. Предположим, что кассеты у вас сверхвысокого качества и магнитный слой с них не осыпается. Остается аппарат, который называется магнитофоном. Если этот аппарат значительно дешевле дисководов, то он плохой по качеству. Важным элементом является стабильность скорости движения ленты. Для вас "плывущий звук", плохой прижим ленты к головке и т. д. означает одно - гибель записанной информации.

Итак, мы познакомились с техническими принципами записи информации на магнитный диск и ленту. Вы же попробовали просмотреть директорию вашего системного диска. Набрав на клавиатуре команду DIR A: и нажав ВК, вы увидели на экране оглавление диска. Блоки информации, имена которых вы видите в директории, называются файлами (от англ. file). Файл может содержать какую-либо программу или просто текстовую информацию. Обратите внимание, что имена файлов выглядят несколько необычно. Естественно ожидать, что имя файла отвечает его назначению.

Имена файлов состоят из двух частей. Первая часть - собственно имя - может содержать от одного до восьми символов, которыми могут быть буквы латинского алфавита от A до Z, цифры от 0 до 9, различные специальные знаки (например, &, $, *, ? и т. д.). В принципе, имя может быть любым. Например, ABCDEF и PROGRAM1 - правильные имена.

Вторая часть имени, так называемое расширение (от англ. extention), содержит от нуля до трех символов и отделена от первой части имени точкой.

Символы, входящие во вторую часть, могут быть такими же, как и в первой. Обычно вторая часть имени используется для того, чтобы пояснить, что это за программа. Так, имена программ, написанных на языке Бейсик, имеют в качестве второй части расширение BAS. Фортрановские программы имеют имена, оканчивающиеся на FOR, ассемблеровские - на ASM, и т. д.

Может возникнуть вопрос, зачем делить имя на две части? Исключительно для удобства. Рассмотрим примеры правильных и неправильных имен файлов:

PROGRAM1.BAS - правильно
MYPROGRAM.BAS - неправильно (больше 8 символов в имени)
1$$$-25.FOR - правильно
EXAMPLE.THREE - неправильно (больше 3 символов в расширении)
PROGRAM1:EOM - неправильно (нет точки между частями имени)

Мы познакомились со сложной наукой, позволяющей легко ориентироваться в многочисленных файлах, хранящихся на диске. Теперь

самое время попробовать поработать с конкретными программами, и начнем мы с очень важного дела - копирования системного диска.

Заметим, что если вы достаточно опытны и хотите сделать это как можно быстрее, то можно из всего следующего раздела прочесть и выполнить только инструкции, выделенные в тексте.

5. ПОДГОТОВКА ДИСКОВ ДЛЯ ПОЛЬЗОВАНИЯ

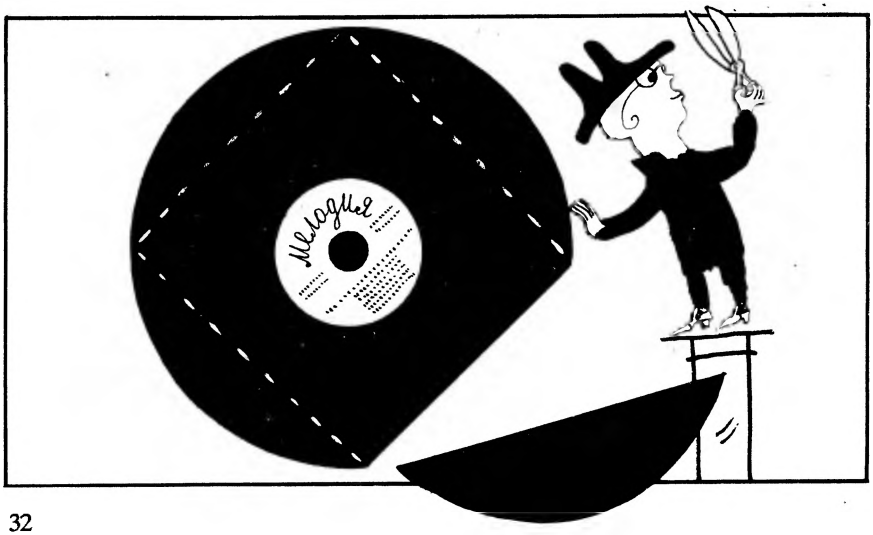
Пользователь учится самостоятельно выполнять процедуры разметки диска и генерации системы

Теперь, когда вы поняли основные идеи организации магнитной записи на диск, перейдем к практическому использованию полученных знаний.

Возьмите магнитный диск в руки и рассмотрите его внимательно. Прежде всего вы заметите, что, хотя он и называется диском, он имеет форму квадрата. Вы, наверное, догадались о причине такого несоответствия названия форме. Действительно, сам магнитный диск настолько нежное изделие, что его необходимо защитить от пользователя и спрятать в футляр.

В центре магнитного диска проделано большое отверстие. Оно предназначено для установки диска на вал двигателя, вращающего диск. Под этим отверстием имеется большая прорезь в футляре. В ней хорошо видна поверхность магнитного диска. С этим местом нужно обращаться исключительно осторожно. Нельзя царапать магнитную поверхность ножиком или писать на ней что-либо ручкой и даже касаться ее руками.

В правом верхнем углу футляра имеется квадратная прорезь. Она сделана так высоко, что сам магнитный диск в нее не попадает. Это очень полезная прорезь. Служит она для предотвращения стирания инфор-



мации с магнитного диска и называется "запрет записи" (от англ. write protect). Нужно отметить, что создатели дисков проявили заботу о пользователях. Так, для удобства они решили, что на восьмидюймовых дисках заклеенная прорезь будет означать, что можно записывать и стирать информацию с диска, а незаклеенная - запрет записи. В то же время на пятидюймовых дисках все сделано наоборот.

Мы сейчас проделаем очень важную операцию. Возьмите системный диск, прилагаемый к вашей машине (помните, вы с него уже загружали машину). Найдите в коробке с магнитными дисками маленькие отрезки клейкой ленты. Теперь определите, какого размера у вас диски. Это легко сделать, используя линейку, если вам лень почитать инструкцию по эксплуатации. Теперь защитите от случайного стирания ваш системный диск. Как это сделать? Если у вас пятидюймовые диски, то возьмите полоску клейкой ленты и заклейте прорезь write protect. На восьмидюймовом диске, наоборот, отклейте ленту, закрывающую прорезь.

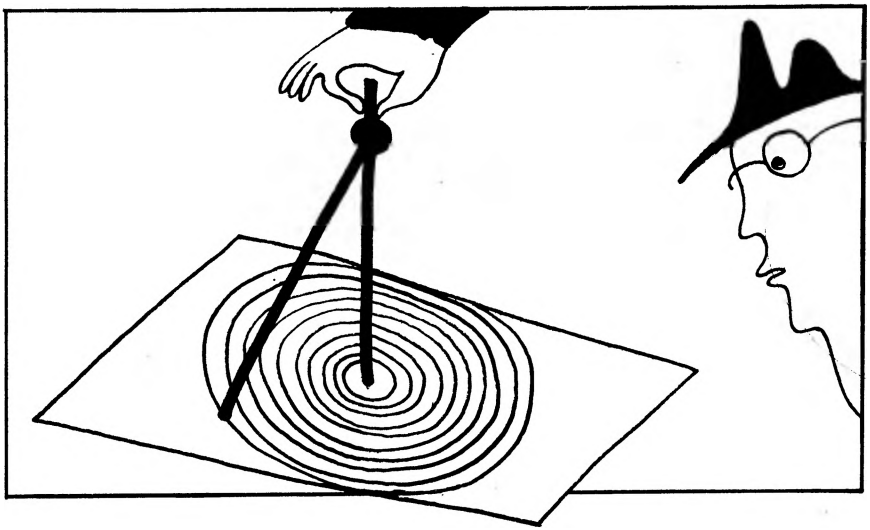
Теперь вы можете быть спокойны, никакими командами операционной системы вы уничтожить ваш диск не сможете. Остаются, конечно, механические методы. Так, магнитный диск можно разрезать на части, измять. Для немеханического уничтожения записанной на диск информации вполне подойдет достаточно сильный магнит. Надеемся, что все поняли шутку и теперь знают, что с диском можно делать и чего нельзя.

На диске имеется еще одна круглая прорезь - справа от большого отверстия. Вам, может быть, никогда не понадобится информация о том, для чего она нужна. Для любопытных скажем, что она необходима самой ПЭВМ для определения, вставлен ли диск в дисковод и вращается ли он. Так что постарайтесь случайно ее не заклеить. После этой операции включайте машину и загружайте операционную систему, как было описано в разделе 3.

Сейчас мы займемся подготовкой к использованию нового диска. Как вы помните, мы договорились, что прежде чем вы будете что-либо записывать на диск, вы должны его разметить. Но как это сделать? Если вы начали искать в коробках от машины дополнительное устройство для нанесения дорожек на диски и его не нашли, не волнуйтесь: таким устройством является сам дисковод, т. е., в отличие от граммофонной пластинки, разметка на магнитном диске не механическая. Как же она выглядит? Оказывается, достаточно записать на диск определенную последовательность нулей и единиц, содержащую номер дорожки, число байт в секторе и т. д. При считывании информации с диска машина прежде всего ищет эту последовательность. Если она ее не находит, значит диск неправильно размечен, и на экране дисплея появляется надпись:

BDOS ERROR:BAD SECTOR

Чтобы этого не происходило, необходимо размечать диски на дисководах, аналогичных тем, на которых в дальнейшем вы собираетесь работать. Так, бессмысленно пытаться прочесть содержимое диска, размечен-



ного на 80 треков, на 40-трековом дисководе. Поэтому, пользуясь инструкцией по эксплуатации машины, выясните, какие у вас дисководы.

Поскольку разметка диска производится путем записи на него служебной информации, для этой цели должна существовать специальная программа. Она действительно существует и называется FORMAT. Работа с этой программой осуществляется следующим образом.

Возьмите чистый диск. Убедитесь, что он не защищен от записи. Вставьте его в дисковод В. Проверьте еще раз, куда вставлены диски и правильно ли это сделано (раздел 3). Системный диск должен быть вставен в дисковод А, а чистый - в дисковод В. Наберите команду FORMAT и нажмите клавишу ВК. На экране дисплея появится текст:

ПРОГРАММА ФОРМАТИРОВАНИЯ ГИБКИХ ДИСКОВ
ВВЕДИТЕ ИМЯ ДИСКОВОДА (A, B, ...)

В это время машина ожидает ваших действий. Вы должны сказать ей, в каком дисководе находится форматируемый диск. В нашем случае это дисковод В. Для того чтобы указать дисковод, нажмите клавишу В (латинская буква). При этом клавишу ВК нажимать не нужно. На экране появится вопрос:

ФОРМАТИРОВАНИЕ, ПРОВЕРКА, ВОССТАНОВЛЕНИЕ? (Ф, П, В)

Вам предстоит выбрать, что вы будете делать с диском. Возможны случаи, когда в результате сбоев в работе машины портится одна или несколько дорожек на диске. Радикальный способ устранения этой неисправности - форматирование диска заново. Однако при этой процедуре теряется вся информация, записанная на диск. Поэтому разумнее найти испорченные дорожки и попытаться их восстановить. Для этого нужно выбрать ответ П или В. В нашем случае необходимо форматировать диск. Поэтому нажмите клавишу Ф. На экране после

вопроса появится сообщение: ФОРМАТИРОВАНИЕ. Этим машина подтверждает, что она нас правильно поняла и будет форматировать диск.

После этого на следующей строке появится новый вопрос:

ДИСКИ ПЯТИДЮЙМОВЫЕ ИЛИ ВОСЬМИДЮЙМОВЫЕ? (П/В)

Вы, наверное, уже поняли, о чем идет речь. Вам нужно ответить машине, какие у вас диски. Вообще говоря, можно выяснить размер диска из инструкции по эксплуатации либо при помощи линейки. Когда вы нажмете нужную клавишу, машина опять подтвердит, что она вас поняла, и ответит вам: ПЯТИДЮЙМОВЫЕ, если вы нажали клавишу П, и ВОСЬМИДЮЙМОВЫЕ, если вы нажали клавишу В.

Далее последует вопрос:

СОРОК ИЛИ ВОСЕМЬДЕСЯТ ДОРОЖЕК? (С/В)

Вам необходимо установить, пользуясь инструкцией по эксплуатации, какие у вас дисководы, и после этого нажать соответствующую клавишу. Машина опять ответит вам, что она приняла информацию, и на следующей строке появится новый вопрос:

ЗАГРУЗОЧНЫЕ ДОРОЖКИ ИЛИ ВЕСЬ ДИСК? (З/В)

Что это значит? Вы, наверное, помните, что операционная система хранится на диске. Под нее отведены первые дорожки (они называются загрузочными). Так вот, можно форматировать только эти дорожки, не трогая весь остальной диск. Это делается для сохранения информации, записанной на диск, при сбоях машины, в результате чего портятся дорожки, на которых записана система. В нашем случае мы форматируем весь диск, поэтому в ответ на вопрос необходимо нажать клавишу В. Машина подтверждает наш ответ сообщением ВЕСЬ.

Следующий вопрос касается количества сторон у диска:

ОДНОСТОРОННИЙ ИЛИ ДВУСТОРОННИЙ? (О/Д)

Если у вас двусторонние дисководы, то вы можете форматировать на них как двусторонние, так и односторонние диски (или двусторонние, но только с одной стороны). Эта возможность позволяет обмениваться программами с владельцами машин, у которых дисководы отличаются от ваших. Пользуясь инструкцией по эксплуатации, выберите нажатием на соответствующую клавишу нужный тип дисков и убедитесь, прочтя ответ машины, что она правильно вас поняла.

Затем последует вопрос:

ОДИНАРНОЙ ИЛИ ДВОЙНОЙ ПЛОТНОСТИ? (О/Д)

Что это значит? Вы, наверное, помните рассуждения о структуре магнитного покрытия диска. Мы рассматривали это покрытие как совокупность магнитных стрелок, расположенных вплотную друг к другу. Так вот, поперечный размер этих стрелок и определяет плотность записи информации. Она может быть двойной или одинарной. Плот-

ность информации обычно указывается на конверте диска. Так, этикетки импортных двусторонних дисков, на которых можно записывать информацию с двойной плотностью, помечены надписями DS (от англ. Double Side - двусторонний) и DD (от англ. Double Density - двойная плотность). На этикетках отечественных дисков также имеется информация о количестве сторон и плотности информации. Как правило, современные диски позволяют записывать информацию с двойной плотностью, поэтому на вопрос отвечаем: Д. Машина подтверждает ответ.

Вслед за этим машина выводит на экран итоговую информацию о параметрах диска и спрашивает, все ли параметры введены верно. Если какой-либо параметр ошибочен, необходимо нажать клавишу Н (Нет), и машина запросит все параметры заново. Если же всё в порядке, необходимо нажать клавишу Д (Да). При этом начинается процесс форматирования. На дисководе В загорается красная лампочка и на экране дисплея появляются на нижней строке точки. Каждая точка соответствует двум отформатированным дорожкам. После окончания процесса форматирования машина задаст вопрос:

ПОВТОРИТЬ ОПЕРАЦИЮ?

Если вы хотите отформатировать еще один диск, выньте из дисковода В только что отформатированный диск, вставьте туда новый и нажмите в ответ клавишу Д. Операция форматирования повторится. Если не хотите, нажмите клавишу Н, и машина прекратит форматирование дисков.

Вы, наверное, уже успели оценить преимущества диалогового режима работы с ПЭВМ. Машина разговаривает с вами простым и понятным языком, сама подсказывает вам варианты ответов, и в результате вы легко можете выполнить очень непростую процедуру форматирования диска.

Следующая наша задача - разместить на новом диске операционную систему. Для этой цели существует специальная программа SG (от англ. System Generation - создание системы). Наберите команду SG и нажмите клавишу ВК. На экране появится текст:

КОПИРОВАНИЕ ОС
ВВЕДИТЕ ИМЯ ИСТОЧНИКА (A, B, ...)

после чего машина будет ждать от вас ответа, на каком диске записана операционная система. В нашем случае системный диск вставлен в дисковод А. Поэтому необходимо нажать клавишу А.

После некоторой паузы, необходимой для считывания информации с системных дорожек диска А, появится следующий текст:

ВВЕДИТЕ ИМЯ ПРИЕМНИКА (A, B, ..., ESC)

Вам нужно ответить, на какой диск вы хотите записать систему. Если вы не хотите записывать систему, нажмите клавишу ESC (первая слева во втором ряду клавиш, на ней нанесена надпись ПРФ/ESC; ПРФ расшифровывается как прерывание функции). В нашем случае необходимо нажать клавишу В.

После паузы, необходимой для записи системы на диск B, повторится последняя надпись. Если вы хотите записать систему еще на один диск, смените диск в дисководе B и опять нажмите B. Если процесс копирования закончен, нажмите ESC.

Таким образом, после окончания работы программы SG вы располагаете по крайней мере еще одним системным диском и можете загрузить с него операционную систему. Но если вы это сделаете и потом попробуете набрать команду FORMAT или SG, то вместо, ожидаемого эффекта машина выдаст сообщение:

```
FORMAT?
SG?
```

всем своим видом давая вам понять, что она вас не понимает. В чем здесь дело? В том, что операционная система содержит две части. Первая из них загружается в память машины и находится там все время, пока машина работает. Эта часть системы называется резидентной. Вторая часть, гораздо большая по объему, хранится на системном диске и загружается по мере необходимости. К этой части и относятся программы FORMAT и SG.

Теперь наша задача - скопировать все файлы с системного диска A на новый системный диск B. Как это сделать? Для этого существует великое множество различных программ. Стандартной программой для операционной системы CP/M является программа PIP (от англ. Peripherial Interchange Program - программа обмена с периферийными устройствами). PIP позволяет производить обмен данными не только с дисками, но и с устройствами типа дисплея, принтера и т. п. В данном разделе нас интересуют только диски, поэтому рассмотрение остальных возможностей PIP мы продолжим в разделе 6.

Наберите команду PIP и нажмите клавишу BK. На экране под командой PIP появится символ *. Он означает, что программа PIP загружена и ждет команд пользователя. PIP существенно отличается по режиму общения с пользователем от предыдущих освоенных вами программ. С ней можно работать при соблюдении ряда правил. Так, команда, вводимая с клавиатуры, должна иметь следующий формат:

```
* D:FILENAME=S:FILENAME1 (BK)
```

где D - имя приемника информации (от англ. Destination - приемник), S - имя источника информации (от англ. Source - источник), FILENAME - имя файла, в который записывается принятая информация, FILENAME1 - имя копируемого файла. (Напомним, что для ввода любой команды необходимо нажать клавишу BK.) Например, если мы хотим скопировать программу с именем MYPROG.BAS с диска A на диск B, то достаточно набрать команду

```
* B:=A:MYPROG.BAS
```

В результате на диске B появится копия программы MYPROG.BAS под тем же именем. Если мы хотим, чтобы файл с копией программы на-

зывался по-другому, то после B: мы должны указать новое имя. Таким образом, последовательно набирая команду копирования различных файлов с диска A на диск B, мы сможем получить полную копию системного диска.

Однако если на диске A много файлов, то процедура окажется очень утомительной из-за необходимости многократно набирать команду на клавиатуре. Для облегчения этой процедуры предусмотрена возможность использования так называемых глобальных символов. Что это такое? Глобальными символами являются символы * и ?. Символ ? в имени файла или в его расширении означает, что эту позицию может занимать любой символ. Например, если на вашем диске A есть программы ABCDE.XYZ, ABJDE.XYZ и ABHDE.XYZ, команда PIP

```
* B:=A:AB?DE.XYZ
```

скопирует все программы с диска A на диск B. Символ * означает, что любые символы могут занимать эту и все последующие позиции в имени файла или в его расширении (в зависимости от того, где стоит глобальный символ). Например, если на диске A содержатся программы с именами ABCDE.XYZ, ABC327.XYZ, ABC$$$.XYZ и AB.XYZ, то команда PIP

```
* B:=A:AB*.XYZ
```

скопирует все четыре программы с диска A на диск B. Если вы хотите скопировать все файлы с диска A на диск B, наберите команду

```
* B:=A:*.*
```

В результате диск B будет содержать все программы, имеющиеся на диске A. Итак, теперь мы имеем полную копию системного диска и можем начинать знакомство с операционной системой.

6. ОПЕРАЦИОННАЯ СИСТЕМА ПЭВМ КОРВЕТ

Раздел, в котором читатель почерпнет массу полезных сведений об организации работы ПЭВМ, после чего сможет чувствовать себя системным программистом

Читателю могло показаться, что информация, данная ему в предыдущих разделах, носит несколько сумбурный характер. Действительно, вместо того чтобы последовательно знакомить с командами операционной системы и языками программирования, мы начали со сравнительно экзотических операций типа форматирования дисков. Но на это имеется веская причина. Изучать операционную систему ПЭВМ по книге, не имея перед собой компьютера и не пробуя тотчас же по прочтении ту или иную команду, бессмысленно. Поэтому мы предполагаем, что вы постоянно работаете с машиной. В целях безопасности вы скопировали системный диск, и теперь даже если вы слу-

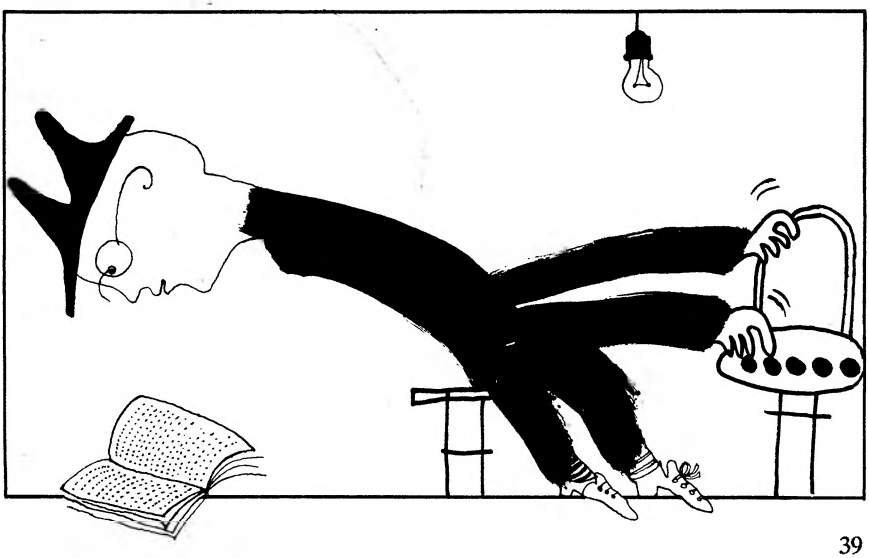
чайно сотрете что-нибудь с его копии, то ничего страшного не произойдет - у вас есть оригинал.

Перейдем теперь к знакомству с операционной системой Корвета. Операционная система (ОС) не есть свойство конкретной ЭВМ. Так, на нашей ПЭВМ могут работать различные ОС. Любая ОС является первой и наиболее важной программой в ЭВМ. Задачи, которые решает ОС, - это, в основном, управление различными частями компьютера. Операционная система определяет последовательность действий, в которых наиболее часто возникает потребность. Примерами могут служить определение содержимого магнитного диска, создание и удаление файлов, запуск программ, определение свободного пространства на дисках и т. д.

Таким образом, ОС выполняет следующие функции:

- управляет работой различных устройств ввода-вывода (дисплеями, принтером, накопителями на магнитных дисках и лентах и др.);
- управляет размещением файлов на магнитных дисках;
- обеспечивает загрузку в память и запуск стандартных программ ОС и программ, написанных пользователем.

Здесь мы рассмотрим операционную систему CP/M-80 (от англ. Control Program for Microprocessors - управляющая программа для микропроцессоров). Почему мы остановили свой выбор именно на этой ОС? Из-за ее широкой распространенности во всем мире. ОС CP/M-80 была разработана для ПЭВМ на микропроцессоре I8080 (отсюда число 80 в названии ОС) в 1975 г. Г. Килдэлом, который в 1976 г. основал фирму Digital Research. С того времени CP/M (далее для краткости будем называть ее так) стала одной из наиболее популярных и широко используемых ОС для 8-разрядных ЭВМ и признается многими фирмами "промышленным стандартом", поскольку она используется многими изготовителями ПЭВМ и обеспечивает работу буквально сотен различных программ, созданных разработчиками программного обеспечения для ПЭВМ.



Загрузка операционной системы CP/M

Знакомство с ОС CP/M начнем с процесса ее загрузки. Мы уже касались этой проблемы в разделе 3. Следуя изложенной там инструкции, загрузите ОС. Теперь попробуем более детально разобраться, что же в действительности происходит при загрузке ОС.

Как вы помните, загрузка системы означает размещение ее копии с системного диска в оперативной памяти машины. Существуют два типа загрузки: начальная загрузка и реинициализация системы.

Начальная загрузка ОС. В каждой ПЭВМ, работающей с ОС CP/M, имеется программа начальной загрузки, хранящаяся в ПЗУ. Поскольку вся ОС умещается на первых двух дорожках системного диска, программа в ПЗУ должна обеспечить считывание информации с этих дорожек и размещение ее в памяти. При включении ПЭВМ в сеть или при нажатии на кнопку СБРОС можно услышать звуковой сигнал, свидетельствующий о том, что информация с системных дорожек загружается. Какая именно информация записана на них?

Как мы знаем, ОС служит для управления различными устройствами, входящими в состав ПЭВМ. За это отвечает часть операционной системы, называемая BIOS (от англ. Basic Input-Output System - базовая система ввода-вывода). BIOS содержит подпрограммы управления периферийными устройствами. Поскольку организация периферийных устройств отличается на различных ПЭВМ, то и BIOS различен для разных ПЭВМ. Обычно BIOS пишется непосредственно разработчиками машины.

Оставшаяся часть ОС - общая для всех машин, которые могут с ней работать. Она не зависит от ЭВМ и ее периферийных устройств и работает с ними только через BIOS. Называется эта часть BDOS (от англ. Basic Disk Operating System - базовая дисковая операционная система). В ОС CP/M входит также программа CCP (от англ. Console Command Processor - процессор команд консоли[1]). В задачу CCP входит проверка состояния клавиатуры консоли на соответствие вводимых символов командам ОС CP/M и интерпретация команд.

Все три части ОС загружаются с системного диска. Процесс начальной загрузки называется "холодным стартом" системы. В результате "холодного старта" на экране дисплея появляется сообщение:

```
CP/M-80 v. 2.2
ОФП НИИЯФ МГУ BIOS
Ver. 1.2 (c) III 1988

A>
```

в котором ОС сообщает пользователю номера версий загруженной системы и BIOS. Последняя строка является стандартным сообщением

---

[1]) Консоль - стандартное устройство для взаимодействия человека с ПЭВМ. Обычно это телетайп или дисплей с клавиатурой.

системы о том, что она работает в данный момент с диском А и ждет команды пользователя (это сообщение называется промптом). Следует запомнить, что для нормальной работы машины необходимо, чтобы в дисководе А всегда находился диск, первые две дорожки которого содержат копию ОС CP/M.

Реинициализация системы ("теплый старт"). В результате этой процедуры с системного диска копируется в память ПЭВМ только часть ОС. Остальная память остается нетронутой. Обычно теплый старт используют при выходе из выполнявшейся в системе программы. Эта процедура выполняется ОС автоматически. Можно, однако, принудительно вызвать теплый старт, нажав одновременно на клавиши CTRL и C. (Клавиша CTRL находится слева на клавиатуре, на ней написано УПР/CTRL.) Это бывает необходимо в тех случаях, когда ОС не может реинициализировать себя сама, например при смене диска или при ошибках определенного типа. Если реинициализация системы не удалась, то необходимо выполнить полную перезагрузку системы, нажав на кнопку СБРОС.

Работа с дисками в ОС CP/M

После успешной загрузки операционной системы диск А автоматически становится "текущим" диском. Текущий диск или "диск по умолчанию" - это диск, обращение к которому подразумевается во всех командах, если явно не указано имя другого диска. Какой диск является текущим, можно определить по промпту ОС. Для изменения текущего диска необходимо набрать на клавиатуре имя нового диска, затем двоеточие и нажать клавишу ВК. Например, выбор текущим диском диска В можно осуществить следующим образом:

```
A>B:
B>
```

Двоеточие нужно для того, чтобы ОС CP/M восприняла символ В как имя нового диска. В противном случае она воспримет символ В как имя программы и будет искать на диске А командный файл B.COM. Если такого файла на диске не окажется, то появится сообщение об ошибке:

```
A>B
B?
A>
```

Перед тем как изменить текущий диск, необходимо вставить его в соответствующий дисковод. В противном случае клавиатура консоли будет заблокирована и потребуется реинициализация системы или появится сообщение об ошибке:

```
BDOS ERROR SELECT
```

после чего нужно нажать на любую клавишу.

Может оказаться, что и после помещения диска в дисковод попытка изменить текущий диск не даст результата. Это, как правило, является следствием несоответствия формата вновь вставленного диска формату, принятому в CP/M. Поэтому старайтесь пользоваться для форматирования дисков программой, имеющейся на вашем системном диске.

Если необходимо обратиться к файлу на другом диске, то вовсе не обязательно изменять текущий диск, достаточно перед именем файла указать имя диска, на котором он записан, отделив имя диска от имени файла двоеточием. Например, если необходимо просмотреть содержимое файла PROGRAM1.BAS, находящегося на диске в дисководе B, можно выполнить команду

```
A>TYPE B:PROGRAM1.BAS
```

Рассмотрим теперь основные команды ОС CP/M.

Команды управляющих символов

В операционной системе CP/M имеется набор специальных команд - "управляющих символов". Для ввода управляющего символа необходимо нажать одновременно клавишу УПР/CTRL и клавишу с требуемым символом. Приведем примеры управляющих символов:

CTRL-C - команда реинициализации системы (с этим управляющим символом мы познакомились выше);

CTRL-M - вызывает возврат каретки (эта команда эквивалентна действию клавиши ВК);

CTRL-J - вызывает перевод строки;

CTRL-H (или BACKSPACE) - возвращает курсор на одну позицию назад и стирает символ, стоящий в этой позиции (BACKSPACE находится в верхнем ряду клавиатуры справа и обозначается стрелкой, направленной влево);

DEL - стирает символ слева от курсора;

CTRL-V - удаляет уже набранную командную строку и перемещает курсор в начало следующей строки;

CTRL-X - удаляет текст командной строки и возвращает курсор в ее начало;

CTRL-E - используется для переноса командной строки на следующую строку экрана, пока не нажата клавиша ВК; когда клавиша ВК будет, наконец, нажата, все введенные с помощью CTRL-E команды будут восприняты CP/M как единая командная строка;

CTRL-R - используется для повторного вывода командной строки, содержащей удаленные при нажатии клавиши DEL символы; в результате из командной строки будут исключены все удаленные символы и в таком виде она будет выдана на следующую строку экрана;

CTRL-S - служит для задержки вывода текста на экран; при первом использовании этого управляющего символа вывод прекращается, при

следующем - возобновляется; подобную процедуру можно повторять столько раз, сколько потребуется;

CTRL-P - служит для вывода на принтер любого текста, появляющегося на экране; способ использования этой команды аналогичен команде CTRL-S.

Встроенные и транзитные команды

В операционной системе CP/M существуют еще два типа команд: встроенные и транзитные. Встроенные команды (они называются также резидентными) являются частью ОС и хранятся вместе с ней в одной и той же области оперативной памяти. Транзитные команды хранятся постоянно на диске в виде командных файлов (расширение имени у таких файлов - COM). Примерами встроенных команд являются:

DIR - команда вывода на экран дисплея или принтер данных о файлах, содержащихся на указанном диске (наиболее часто используемая команда ОС CP/M);

REN - команда изменения имени файла (от англ. REName - переименовать);

ERA - команда удаления файлов (от англ. ERAse - стирать);

TYPE - команда просмотра содержимого текстовых файлов;

USER - команда обращения различных пользователей к своим файлам, хранящимся на одном и том же диске;

SAVE - команда сохранения указанного числа страниц системной памяти в виде файла на диске.

Транзитных команд может быть очень много. По существу, любая программа в машинных кодах может являться транзитной командой. Все файлы, содержащие транзитные команды, имеют расширение COM. Примерами транзитных команд являются стандартные программы, поставляемые фирмой Digital Research в составе ОС CP/M:

STAT - предоставляет пользователю информацию о состоянии диска (от англ. STATus - состояние);

PIP - осуществляет обмен данными с периферийными устройствами (с этой программой мы уже встречались при копировании системного диска);

ED - позволяет пользователю запустить текстовый редактор (от англ. EDitor);

SUBMIT - предоставляет пользователю возможность запустить последовательность команд, указанных в заранее подготовленном файле;

ASM - позволяет произвести ассемблирование, т. е. перевод программы, написанной на языке Ассемблер при помощи редактора ED или какого-нибудь другого, в шестнадцатиричный файл с расширением HEX, который, в свою очередь, можно преобразовать в командный файл при помощи программы LOAD;

LOAD - используется для преобразования шестнадцатиричного файла в файл, содержащий машинные коды;

DDT - позволяет проследить за работой различных программ в машинных кодах и таким образом выявить имеющиеся в них ошибки;

DUMP - используется для вывода на экран дисплея или принтер в шестнадцатиричном формате содержимого командных файлов;

FORMAT - форматирует диски;

SG - копирует ОС с диска на диск (от англ. System Generation - создание системы).

Рассмотрим подробнее встроенные команды.

DIR - вывод на консоль оглавления диска. Команда DIR иллюстрирует способ ввода сообщений с клавиатуры для ОС CP/M. Сообщение состоит из команды и отделенных от нее пробелом одного или нескольких операндов. Операндами являются дополнительные сообщения системе о том, что конкретно от нее требуется. Например, использование просто команды DIR без операндов позволяет получить полное оглавление текущего диска:

```
A>DIR
A:MOVCPM  COM:STAT   COM:PIP    COM:SG    COM
A:FORMAT  COM:LOAD   COM:DDT    COM:ED    COM
A:SUBMIT  COM:XSUB   COM:ASM    COM:
A>
```

Как мы видим, при завершении выполнения команды DIR снова появляется промпт A>.

DIR может использоваться для вывода оглавления других дисков. В этом случае в качестве операнда нужно указать имя диска и поставить двоеточие, например:

```
A>DIR B:
B:TEXT    TXT:BASIC     COM:PIP     COM:SUBMIT    COM
B:PROG    BAS
A>
```

Заметим, что после выполнения команды DIR B: появился промпт A>, поскольку текущим диском остается диск A. Если в дисковод был вставлен чистый диск, совместимый с ОС CP/M, в результате выполнения команды DIR B: появится сообщение NO FILE (файлов нет).

Полное оглавление диска может быть очень длинным. В таком случае для задержки вывода оглавления на экран используйте команду CTRL-S.

Команда DIR позволяет выводить на экран не только все оглавление, но и имена отдельных файлов. Для этого в качестве операнда указывается имя файла. Оно может содержать глобальные символы * и ? для задания имен группы файлов, например:

```
A>DIR B:*.COM
B:PIP     COM:BASIC     COM:SUBMIT     COM
A>
```

Если указано имя файла, то оно выводится на экран. Если такой файл отсутствует на диске, то выводится сообщение NO FILE. В рассмотренном примере выводятся имена всех файлов, имеющих расширение COM.

REN - переименование файлов, имеющщихся в каталоге диска. Рассмотрим правила использования команды REN. Прежде всего разберемся с правилами написания операндов (синтаксисом). Казалось бы, проще всего ввести команду типа "заменить старое имя на новое". Однако обычно операционные системы выполняют это действие по-другому. Машина всегда идет логическим путем, т. е. присваивает новое имя старому файлу. Этот процесс имеет тот же вид, что и обычное алгебраическое выражение X=2, в результате выполнения которого X примет значение 2. Поэтому синтаксис команды REN имеет вид

```
A>REN NEW.TXT=OLD.TXT
```

Проверить, действительно ли мы переименовали файл, можно с помощью следующих команд:

```
A>DIR NEW.TXT
A:NEW TXT
A>DIR OLD.TXT
NO FILE
A>
```

Мы видим, что файл OLD.TXT исчез из оглавления диска, а NEW.TXT появился в нем. При попытке переименовать не существующий в оглавлении файл выдается сообщение NO FILE.

В операционных системах обычно устанавливается различный уровень защиты файлов. Это делается для того, чтобы предотвратить случайное стирание нужных файлов. Уровни защиты бывают двух типов: R/O (Read Only - только чтение) и R/W (Read Write - чтение и запись или свободный доступ). При попытке переименовать файл с уровнем защиты R/O выдается сообщение:

```
FILE IS SET R/O
```

(файл имеет такой уровень защиты, что он может быть только прочитан). Для появления промпта в этом случае необходимо нажать клавишу ВК.

Переименовывать файлы можно не только на текущем диске, но и на любом другом. Для этого в операнде первым должно стоять имя диска с двоеточием, например:

```
A>REN B:NEW.TXT=OLD.TXT
A>
```

Важно, чтобы имя диска присутствовало только один раз и помещалось в начале операнда перед новым именем. Допустимо указание имени диска также непосредственно перед старым именем, например:

```
A>REN B:NEW.TXT=B:OLD.TXT
A>
```

Приведем примеры ошибочного использования команды REN:

```
A>REN NEW.TXT=B:OLD.TXT
B:OLD TXT?
A>

A>REN B:NEW.TXT=A:OLD.TXT
A:OLD TXT?
A>
```

Отметим, что использование в операндах команды REN глобальных символов ? и * не допускается.

ERA- удаление хранящихся на диске файлов. Команда ERA опасна тем, что неправильное пользование ею вызовет стирание полезных программ. Поэтому пользуйтесь этой командой с осторожностью. Справедливости ради отметим, что по команде ERA реально ничего не уничтожается. Имя файла просто удаляется из каталога. При этом сам файл остается на месте. Однако после этого прочесть его стандартными средствами ОС CP/M невозможно. В случае, если вы случайно стерли что-нибудь необыкновенно ценное, обратитесь за помощью к более опытным системным программистам, и они помогут вам восстановить утраченный файл. Пользоваться командой ERA очень просто. Достаточно после имени команды указать имя стираемого файла. Проиллюстрируем сказанное примером:

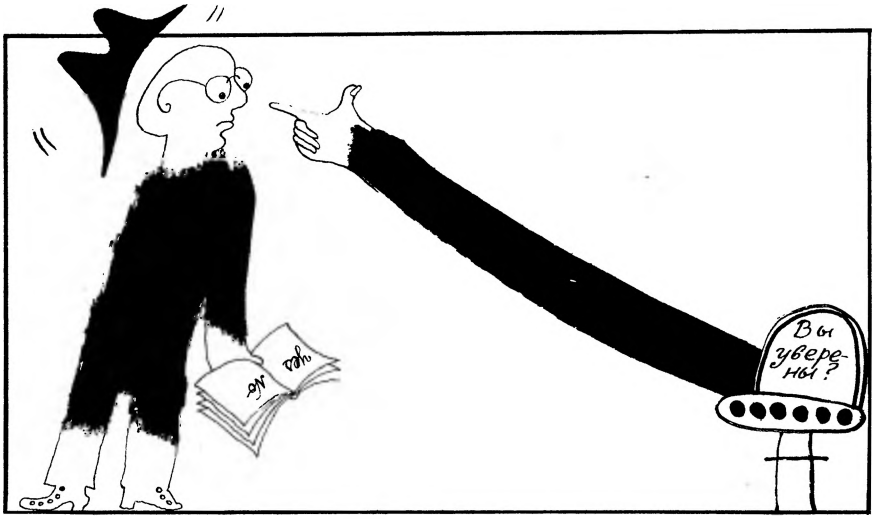
```
A>DIR TEXT.TXT
A:TEXT  TXT
A>ERA TEXT.TXT
A>DIR TEXT.TXT
NO FILE
A>
```

Любителям стирать файлы дадим несколько рекомендаций по технике безопасности. Не жалейте дисков. Храните резервные копии важных и ценных программ в недоступном для других и, в особенности, для себя месте. Прежде чем набрать команду ERA, трижды подумайте.

Стирать файлы можно не только с текущего, но и с другого диска. Для этого, как вы, наверное, уже догадались, перед именем файла нужно указать имя диска и поставить двоеточие, например:

```
A>ERA B:TEXT.TXT
A>
```

Перейдем к наиболее опасному по последствиям использованию команды ERA. Сейчас мы научимся уничтожать целые группы файлов, вплоть до стирания всей информации, записанной на диске. Для этой цели используются глобальные символы в имени файла.



Предположим, что вы хотите удалить файлы, имеющие расширение TXT. Для этого вы должны набрать команду ERA *.TXT. В результате все файлы с расширением TXT будут удалены. А как быть, если вам нужно удалить не все файлы с расширением TXT? В этом случае нужно либо удалять их по одному, не используя глобальных символов, либо переименовать те, которые вы удалять не собираетесь.

Рассмотрим, наконец, использование команды ERA для леденящей душу процедуры уничтожения всего, что есть на диске. Наберите:

A>ERA *.*

Когда вы нажмете ВК после ввода команды, машина в испуге попробует вас остановить, и на экране появится вопрос: ALL(Y/N)? Если вы ответите Y (Yes), машина, скрепя сердце, сотрет все содержимое диска. Если вы ответите N (No), машина с облегчением выведет на экран промпт и не будет ничего стирать. При попытке стереть несуществующий файл выводится сообщение NO FILE. Если стираемый файл имеет уровень защиты R/O, появляется сообщение FILE IS SET R/O, после чего необходимо нажать клавишу ВК.

Еще раз хочется напомнить: защищайте файлы и диски от случайного стирания и не доверяйте своих дисков случайным пользователям. Помните, что восстановление утраченных программ требует больших затрат времени, а в ряде случаев оказывается невозможным.

TYPE - вывод на дисплей содержимого текстовых файлов. Предположим, что вы написали большую и очень полезную программу или текст вашего выступления на всемирном конгрессе пользователей Корвета. Мысль о большой проделанной работе согревает вашу душу. Но время от времени закрадывается сомнение: а цело ли ваше творение? Не стерли ли его злоумышленники? Просмотр оглавления диска немного успокаивает. Да, вот оно, на месте, и имя тоже. А вдруг с

самим текстом что-нибудь приключилось? Для того чтобы убедиться в его сохранности, а также для просмотра других текстовых файлов служит команда TYPE. Заметим, что попытка просмотра с помощью команды TYPE файлов, не предназначенных для вывода на экран, например командных, приводит к непредсказуемым последствиям.

Для вывода на экран содержимого некоторого текстового файла, например содержимого новогоднего поздравления вашему дядюшке, хранимого на диске под именем UNCLE.TXT, достаточно ввести команду: TYPE UNCLE.TXT. В результате на экране появится:

```
A>TYPE UNCLE.TXT
Дорогой дядюшка! Поздравляю Вас с Новым годом
и желаю Вам всего наилучшего!
                                        Ваш племянник

A>
```

Машина распечатала текст вашего новогоднего послания, и в конце появился промпт A>. Если вы хотите напечатать ваше послание на бумаге, то перед нажатием клавиши ВК введите команду CTRL-P.

Использование глобальных символов в команде TYPE не допускается. Поэтому единовременно можно вывести на экран только один файл. Если вы просите вывести содержимое несуществующего файла, появится сообщение об ошибке в виде имени файла, сопровождаемого вопросительным знаком. То же самое будет выдано на экран при использовании глобальных символов. Если файл очень длинный, можно воспользоваться командой CTRL-S для задержки его вывода. Нажатие любых других клавиш на клавиатуре продолжает выдачу текста на экран, и в конце выдачи появляется промпт.

Избегайте вывода на экран содержимого файлов, к тому не предназначенных. В этом случае вы увидите последовательность бессмысленных символов, совершенно непонятных для вас. Попытка разобраться в них эквивалентна чтению китайских книг лицами, не разумеющими соответствующую грамоту. Кроме бесполезности эти действия чреваты зависанием системы, в результате чего ее приходится перезагружать.

USER - работа пользователей с областями на диске. Команда USER используется при работе многих пользователей на одном компьютере. Если бы у каждого программиста был на столе персональный компьютер и множество дисков, надобность в ней полностью отпала бы. Но это представляется лишь в мечтах. Обычно, хотя компьютер и персональный, с ним работают разные пользователи. У каждого есть свои программы, но не у каждого есть собственные диски. Для борьбы с путаницей, где чьи файлы, была придумана команда USER. В многопользовательском режиме работы с дисками они разбиваются на области с номерами от 0 до 15 (16 областей). Каждый пользователь имеет свою собственную область, в которой он хранит программы. Не следует думать, что такая организация дисков позволяет нескольким людям работать на машине одновременно. Операционная система CP/M - одно-

пользовательская. Все, что будет сейчас излагаться, относится только к архивизации информации.

После загрузки ОС CP/M текущей активной областью становится область с номером 0. При просмотре директории вы увидите только файлы, размещенные в текущей области. Для того чтобы добраться до файлов, расположенных в других областях, нужно сделать соответствующую область активной. При этом выбранная область делается текущей на всех дисках, вставленных в дисководы, независимо от того, с каким из них вы работаете в данный момент. Если вы захотите переслать какой-либо файл из одной области в другую, можете воспользоваться программой PIP. Об этом вы узнаете немного позже.

Попробуем теперь создать новую активную область на всех дисках, вставленных в дисководы вашей машины. Для этого необходимо набрать команду USER и через пробел ввести номер области в пределах от 0 до 15, например:

```
A>USER 1
A>
```

После ввода этой команды на экран не выводится никаких сообщений, кроме основного промпта. Однако кое-что все-таки произошло. При вводе команды DIR вы получите сообщение NO FILE. При этом вам окажутся недоступными все транзитные команды ОС CP/M и пользоваться можно будет только резидентными командами. При попытке ввода неправильного номера области появится сообщение об ошибке, например:

```
A>USER 16
16?
A>
```

Итак, во вновь созданной области 1 нет никаких файлов. Но вы хотите туда что-нибудь скопировать. Для этого прежде всего в вашу область необходимо перенести транзитную команду PIP, скопировав файл PIP.COM. Эту многозвенную процедуру мы рассмотрим позднее, поскольку для нее потребуются команды DDT, SAVE и STAT.

Еще одно важное замечание. С помощью команды PIP можно только копировать файлы из других областей в текущую область. В обратную сторону этот процесс выполнить нельзя.

Рассмотрим теперь некоторые транзитные команды.

STAT - получение и установка характеристик файлов и дисков. Забудем на время про пользователей и их территориальные притязания на дисках и займемся другими важными командами ОС CP/M. Вы, наверное, уже не раз вызывали на экран оглавление вашего диска. Замечено, что поскольку DIR является наиболее простой командой и вдобавок появляется на первых же страницах любой литературы, посвященной персональным компьютерам, все пользователи, впервые подойдя к машине, сразу эту команду набирают.

В действительности DIR дает очень куцее представление о вашем диске. Вы не видите ни размера файлов, ни имеющихся областей,

ни оставшегося свободного пространства. А эта информация чрезвычайно важна для нормальной работы с ЭВМ. Всю эту информацию вам предоставит команда STAT. Она, в отличие от предыдущих, является транзитной. Поэтому, чтобы использовать ее, убедитесь, что соответствующий файл STAT.COM имеется на вашем диске и находится в активной области. Для этого введите команду DIR. Если вы не обнаружите его в каталоге, попробуйте изменить активную область с помощью команды USER. Если опять не нашли, обратитесь за помощью к более опытным коллегам.

Команда STAT выполняет следующие функции.

1. Прежде всего она может выводить на экран объем свободного пространства на всех дисках, присутствующих в настоящее время в дисководах, и режим доступа к дискам. Напомним, что режим доступа бывает двух типов: R/O (Read Only - только чтение) и R/W (Read Write - чтение и запись). На диск с доступом R/O нельзя ничего записывать. Это значит, что вы не можете ни дополнить содержимое этого диска, ни удалить какие-либо файлы. С такого диска информацию можно только читать. Доступ R/W означает, что с диска можно читать и можно записывать на него информацию.

2. Команда STAT может выводить на экран характеристики как отдельных файлов, так и групп файлов. Для этого в качестве операнда нужно использовать соответствующее имя или, применяя глобальные символы, обозначить группу файлов.

3. Она позволяет защищать отдельные файлы и целые каталоги файлов, т. е. присваивать им статус R/O.

4. Команда STAT позволяет организовывать скрытые файлы. С помощью этой команды можно присвоить файлам некоторый специальный атрибут, который сделает так, что их имя не будет выводиться на экран по команде DIR из соображений секретности.

5. Она позволяет посмотреть номер активной области, определенный командой USER.

6. Обычно стандартным периферийным устройствам присвоены некоторые логические имена. Например, консоль - CON:, принтер (цифропечать) - LPT: или PRT: и т. д. По команде STAT вы можете узнать, какие логические имена присвоены, а также изменить их, если есть желание, или завести новые из списка разрешенных имен, который также выводится по команде STAT.

Разберем эти функции по порядку.

Вы уже убедились, что файл STAT.COM имеется в директории вашей активной области. Попробуем набрать команду STAT без операндов, увидим на экране:

```
A>STAT
A:R/W SPACE:140K
A>
```

Программа STAT вывела на экран сообщение о статусе диска R/W (или R/O) и размере свободного пространства. Если во все дисководы встав-

лены диски, подобная информация появится обо всех дисках. Отметим, что команда STAT будет смотреть остальные диски, только если к ним уже обращались с какой-либо командой или был выполнен теплый старт системы. Другими словами, ОС должна знать, что мы вставили диски в дисководы.

Если в качестве операнда у команды STAT используется имя какого-либо файла, мы можем получить информацию о его размерах и статусе. Например, если вы интересуетесь размером файла TEXT.TXT, то по команде STAT TEXT.TXT вы получите следующую информацию:

```
A>STAT TEXT.TXT
RECS  BYTES   EXT   ACC
110   14      1     R/W A:TEXT.TXT
BYTES REMAINING ON A:476K
A>
```

Первая строка этого сообщения - названия столбцов, вторая - содержание соответствующих столбцов. На третьей строке указывается свободное пространство на диске. В этом сообщении содержится информация о количестве записей в данном файле, его размере и уровне защиты. Если после имени файла добавить через пробел $S, то в сообщении появится еще одна колонка SIZE:

```
A>STAT TEXT.TXT $S
SIZE  RECS  BYTES   EXT   ACC
110   110   14K     1     R/W A:TEXT.TXT
BYTES REMAINING ON A:476K
A>
```

Поскольку колонка SIZE всегда дублирует RECS, то она, как правило, не используется.

Разберемся в том, какой смысл имеют выводимые на экран характеристики. В столбце SIZE печатается количество записей в файле. Эта характеристика совпадает с RECS и выводится только при использовании $S вслед за именем файла. Запись является единицей объема информации, хранимой на диске. В ОС CP/M длина записи равна 128 байт. Число, указанное в графе RECS, равно количеству 128-байтных записей в файле.

Число, указанное в графе BYTES, представляет собой округленный размер файла в килобайтах. Еще одной единицей измерения объема файлов в CP/M является экстент. Он равен 16K. Графа EXT содержит размер файла в экстентах. Приращение в этой графе может быть только целым. Как правило, эту цифру можно не принимать во внимание. Код, напечатанный в столбце ACC, представляет собой статус защиты файла, R/W или R/O.

Интерес для пользователя представляет практически только столбец BYTES, сообщающий приблизительный объем файла в байтах. Эта информация особенно важна при копировании файлов для определения необходимого свободного пространства на диске для записи вашего файла.

В команде STAT можно использовать глобальные символы * и ?. Так, по команде STAT *.* выводятся сведения обо всех файлах на текущем диске. Если файлов много и вы не успеваете прочитать выводимую информацию, можете воспользоваться командой CTRL-S для задержки вывода информации.

При использовании команды CTRL-P вы можете вывести на цифропечать информацию о вашем диске и после этого спокойно в ней разобраться.

Мы многократно упоминали, что файлы можно защищать. Но пока непонятно, как это делать. Сейчас пришло время овладеть этой премудростью. Для установки статуса защиты необходимо в качестве второго операнда команды STAT использовать $R/W или $R/0.

В качестве примера сделаем какой-либо файл защищенным от записи. Пусть этот файл называется TEXT.TXT. Тогда ваши действия должны быть следующими:

```
A>STAT TEXT.TXT $R/O
TEXT.TXT SET TO R/O
A>
```

Проверим, удалось ли нам изменить статус файла TEXT.TXT:

```
A>STAT TEXT.TXT
RECS  BYTES   EXT   ACC
110   14K     1     R/O A:TEXT.TXT
BYTES REMAINING ON A:476K
A>
```

Разумеется, все эти упражнения нужно проделывать с файлом, имеющимся на диске. В противном случае появится сообщение об ошибке.

Попробуем теперь стереть файл, имеющий защиту R/O. В результате вы получите сообщение:

```
A>ERA TEXT.TXT
BDOS ERR ON A:FILE R/O
```

После этого необходимо нажать клавишу ВК или CTRL-C. И тогда появится промпт системы.

Команда STAT позволяет временно защитить от записи весь диск. Эта защита сохраняется только пока диск вставлен в дисковод, например:

```
A>STAT A:=R/O
A>
```

Если мы теперь посмотрим объем диска, получим сообщение:

```
A>STAT
A:R/O 'SPACE 140K
A>
```

Для снятия защиты необходимо провести перезагрузку системы.

Разберемся еще с одной интересной функцией команды STAT. Предположим, что вы не хотите, чтобы кто-либо видел некоторые ваши файлы в директории. Команда STAT поможет вам спрятать их и сделать невидимыми. Это также бывает полезно, чтобы не выводить каждый раз в оглавлении имена системных файлов. Для того чтобы это сделать, используют операнд $SYS, как это сделано ниже:

```
A>STAT STAT.COM $SYS
STAT.COM SET TO SYS
A>
```

Теперь по команде DIR вы не увидите в оглавлении файл STAT.COM. Но на самом деле он существует.

Как же узнать, есть или нет на диске скрытые файлы? Оказывается, что все это упрятывание не распространяется на команду STAT. По этой команде вы увидите в оглавлении все файлы, однако имена скрытых файлов будут заключены в круглые скобки, чтобы отличить их от остальных, например:

```
A>STAT STAT.COM
RECS  BYTES   TEXT   ACC
41    6K      1      R/W A:(STAT.COM)
BYTES REMAINING ON A:476K
A>
```

Атрибут $SYS может быть назначен произвольному числу файлов при использовании глобальных символов * и ?. Для отмены статуса "скрытый" используется атрибут $DIR, например:

```
A>STAT STAT.COM $DIR
STAT.COM SET TO DIR
A>

A>DIR STAT.COM
A:STAT  COM
A>
```

Если вы используете разделение областей на диске при помощи команды USER, то возможен просмотр имеющихся областей и активной области:

```
A>STAT USR:
ACTIVE USER 0
ACTIVE FILES 0 1 2
A>
```

Это сообщение содержит информацию о том, что активная область на диске имеет номер 0 и существуют еще области 1 и 2, в которых есть файлы.

В разделе 4 мы говорили, что диски бывают разные по количеству сторон, плотности записи, длине сектора и т. д. Команда STAT дает

возможность посмотреть, что представляют собой диски, которыми вы пользуетесь. Для этого необходимо указать операнд DSK:.

В качестве примера рассмотрим сообщение о двустороннем диске с двойной плотностью записи при емкости 1024K на сектор:

```
A>STAT DSK:
   A:DRIVE CHARACTERISTICS
9600:128 BYTE RECORD CAPACITY
1200:KYLOBYTE RECORD CAPACITY
 256:32 BYTE DIRECTORY ENTRIES
 256:CHECKED DIRECTORY ENTRIES
 128:RECORDS/EXTENT
  16:RECORDS/BLOCK
   4:SECTORS/TRACK
   4:RESERVED TRACKS
A>
```

Поясним смысл каждого сообщения:

128 BYTE RECORD CAPACITY - указывает число 128-байтных записей, которое может храниться на диске; запись является единицей измерения объема данных; сделано это для повышения эффективности обработки файлов;

KILOBYTE RECORD CAPACITY - указывает максимальную емкость диска в килобайтах;

32 BYTE DIRECTORY ENTRIES - показывает максимальное количество файлов, которые можно хранить на диске; информация об одном файле, хранящемся на диске, занимает в каталоге пространство, равное 32 байтам;

CHECKED DIRECTORY ENTRIES - это значение, как правило, совпадает с предыдущим; его назначение - отслеживание системой смены дисков;

RECORDS/EXTENT - определяет максимальное число записей на один экстент, которому соответствует один вход в каталог; один экстент, как вы помните, занимает 16K, т. е. 128 записей; отдельные файлы могут занимать несколько экстентов;

RECORDS/BLOCK - указывает минимальный объем дисковой памяти, который можно предоставить для одного файла; для определения размера блока нужно умножить этот параметр на длину записи, т. е. на 128 байт; в нашем примере легко получить, что блок имеет длину 128 байт * 16 = 2048 байт = 2K;

SECTORS/TRACK - показывает количество секторов на одной дорожке;

RESERVED TRACKS - обозначает количество. дорожек, зарезервированных для системных нужд; обычно их две - нулевая и первая; на них хранятся BIOS, BDOS и CCP.

Закончим на этом с дисками и перейдем к другим периферийным устройствам. Этими устройствами являются консоль, другие устройства ввода-вывода и устройство печати. Этим устройствам соответ-

ствуют четыре логических имени: CON: - консоль, RDR: - устройство ввода данных, PUN: - устройство вывода данных, LST: - устройство печати.

Каждому логическому имени может быть назначено до четырех физических устройств. Ими могут быть дисплеи, перфораторы, считыватели, модемы и т. д. Для того чтобы посмотреть, какие физические устройства каким логическим именам назначены, достаточно ввести команду STAT DEV:

```
A>STAT DEV:
CON:IS CRT:
RDR:IS PTR:
PUN:IS PTR:
LST:IS LPT:
A>
```

Что же это за CRT, PTR, LPT и т. д.? Это имена логических устройств. Чтобы узнать, какие физические устройства могут быть назначены четырем логическим устройствам, а также правила работы с командой STAT, введите команду STAT VAL:

```
A>STAT VAL:
TEMP R/O DISK:D:=R/O
SET INDICATOR:D:FILENAME.TYP $R/O,$R/W,$SYS,$DIR
DISK STATUS:DSK:D:DSK
USER STATUS:USR:
IOBYTE ASSIGN:
CON:=TTY:CRT:BAT:UC1:
RDR:=TTY:PTR:UR1:UR2:
PUN:=TTY:PTR:UP1:UP2:
LST:=TTY:CRT:LPT:UL1:
A>
```

Первая часть сообщения не нуждается в комментариях, поскольку вы уже с ней встречались при опробовании разных вариантов команды STAT. Это просто напоминание для тех, кто забудет ключевые слова.

Вторая часть сообщения, озаглавленная IOBYTE ASSIGN, перечисляет имена всех физических устройств, которые при вводе-выводе могут быть назначены каждому логическому устройству:

TTY: - телетайп, низкоскоростное устройство (которое обычно осуществляет вывод информации на бумагу);

CRT: - высокоскоростное устройство (обычно это дисплей с клавиатурой);

BAT: - устройство пакетной обработки;

UC1: - консоль, назначаемая пользователем (обычно это устройство нестандартное и требует модификации секции BIOS);

PTR: - устройство чтения с перфоленты;

UR1:, UR2: - устройства ввода с номерами 1 и 2, определенные пользователем;

PUN: - устройство вывода на перфоленту;
UP1:, UP2: - определенные пользователем устройства вывода;
LPT: - цифропечать;
UL1: - устройство для печати, определенное пользователем.

Присваивание того или иного физического устройства логическому осуществляется при помощи команды STAT. Например, если мы хотим присвоить логическому устройству CON: физическое устройство CRT:, можно выполнить следующие действия:

```
A>STAT DEV:
CON:IS TTY:
RDR:IS PTR:
LST:IS LPT:
A>STAT CON:=CRT:
A>STAT DEV:
CON:IS CRT:
RDR:IS PTR:
LST:IS LPT:
A>
```

Мы видим, что в результате физическое устройство, присвоенное логическому устройству CON:, изменилось с TTY: на CRT:. При ошибочном присваивании на экране появится сообщение об ошибке.

Можно присваивать сразу много устройств разным логическим устройствам, например:

```
A>STAT CON:=CRT:, RDR=UR1:, PUN:=UP1:, LST=UL1:
A>
```

Пользуйтесь этой функцией команды STAT с осторожностью. Никогда не назначайте физических устройств, для которых нет программной поддержки. Особенно это касается устройств, определенных пользователями. Несоблюдение этого правила приведет к зависанию системы и к необходимости полной перезагрузки.

PIP - обмен информацией с периферийными устройствами. Вы, наверное, помните, как проходил процесс копирования вашего системного диска. В принципе, вам может надолго хватить сведений, которые вы почерпнули в ходе этого процесса. Однако команда PIP имеет куда больше возможностей, чем просто копирование файлов. Если вам интересно о них узнать, внимательно прочитайте все изложенное ниже о команде PIP.

Для того чтобы пользоваться командой PIP, необходимо иметь на вашем диске в активной области файл PIP.COM. Мы надеемся, что раз вы дочитали до этого места и по ходу чтения выполняли все описанные выше примеры, то файл PIP.COM на вашем диске имеется. В противном случае вы уже извели бы поставщиков машины еще на этапе копирования системных дисков. Вообще начинающие пользователи, балуясь с USER'ами, часто теряют свои файлы. После этого они начинают обвинять всех и вся и ругать как отечественную, так и зарубежную вычис-

лительную технику. Но мы надеемся, что вы не принадлежите к такой категории людей и не будете отчаиваться в такой ситуации, а просто проанализируете свои действия и найдете в них ошибку.

Перечислим функции команды PIP:

- создание копии файла на текущем диске;
- копирование одного файла или группы файлов на другие диски;
- объединение нескольких файлов в один файл;
- обмен данными между дисками и другими устройствами.

Команда PIP имеет еще много дополнительных возможностей, но мы не будем их рассматривать. Интересующихся отошлем к книге "Операционная система CP/M", написанной М. Уэйтом и Дж. Ангермейером. Перевод ее вышел в издательстве "Радио и связь" в 1986 г.

Начнем теперь практиковаться в работе с командой PIP. Сначала наберите PIP и нажмите клавишу ВК, после чего на экране появится:

```
A>PIP
*
```

Вы видите, что вместо системного промпта A> появился промпт *. Это так называемый промпт команды PIP. Его появление означает, что PIP загружен и ждет ваших указаний. Вспомним теперь, как копировать файлы. Сначала мы должны указать имя копии "приемника" (от англ. destination), затем имя "источника" (от англ. source). Если мы объединяем много файлов в один, их имена перечисляются одно за другим через запятую. В этом случае на экране должна быть набрана строка

```
* D:FILENAME=S:FILENAME1, FILENAME2, ...
```

Здесь D: - выходное устройство - может быть не только диском, но и логическим устройством. То же можно сказать и про входное устройство S:. Имена файлов могут быть опущены, однако если идет работа с дисками, хотя бы одно имя файла должно присутствовать. Эту малопонятную фразу не следует рассматривать как нечто непостижимое для понимания. Обычно, следуя простой логике, нетрудно сообразить, когда имя файла нужно указывать, а когда нет. При рассмотрении конкретных примеров мы вместе с вами постигнем эту премудрость.

Исследуем теперь по порядку упомянутые функции команды PIP.

Начнем с копирования файлов на одном и том же диске и с диска на диск. Как вы уже догадались, вам нужно сообщить программе имя копируемого файла. Рассмотрим эту процедуру на простом примере. Допустим, что у вас имеется файл UNCLE.TXT с уже упомянутым посланием вашему дядюшке. Вы очень дорожите этим файлом и хотите его записать на разных дисках, чтобы его случайно не испортить. Допустим, что диск у вас только один. Тогда вы можете создать под другим именем его копию. В этом случае в ответ на промпт команды PIP вы можете ответить:

```
* A:UNCLE1.TXT=A:UNCLE.TXT
```

В этом случае необходимо указывать имена двух файлов: источника (A:UNCLE.TXT) и копии (A:UNCLE1.TXT). Если диск А - текущий, его имя можно не указывать.

Для пользователей, имеющих много дисков, возможно копирование драгоценного послания с одного диска на другой. Формат записи команды тот же, только нужно указывать, с какого диска на какой идет копирование. Если ваш файл находится на диске А и этот диск текущий, вы должны набрать команду

```
* B:UNCLE1.TXT=UNCLE.TXT
```

Если вы не хотите изменять имя вашего файла, то равноправными командами являются:

```
* B:UNCLE.TXT=UNCLE.TXT
* B:=UNCLE.TXT
* B:UNCLE.TXT=A:
```

Теперь, когда вы научились копировать файлы, вас, наверное, волнует вопрос, как выйти из PIP и вернуть на экран милый сердцу промпт CP/M. Очень просто. Нажмите клавишу ВК.

Для копирования нескольких файлов можно использовать глобальные символы ? и *. Предположим, что в ходе вашей работы вы создали семейство файлов PROGRAM.FOR, PROGRAM.REL и PROGRAM.COM. После этого вы хотите их все скопировать на диск В. Ваши команды должны иметь вид

```
A>PIP
* B:=PROGRAM.*
```

В ходе копирования машина выдаст на экран сообщение:

```
COPYING
PROGRAM.FOR
PROGRAM.REL
PROGRAM.COM
*
```

Если вы явно указываете имя файла копии в левой части командной строки, ни в коем случае не используйте в нем глобальные символы ? и *. В случае нарушения этого правила вы получите сообщение об ошибке IMPROPER FORMAT. Если вы хотите скопировать несуществующий файл, PIP ответит вам сообщением NO FILE. При этом PIP все же создаст на диске рабочий файл с расширением .$$$. Поэтому не нужно пугаться, когда вы увидите в своем каталоге такие имена. Это значит, что какая-то программа типа команды PIP захотела что-то записать на диск, но почему-то не смогла этого сделать. Убрать эти файлы можно командой ERA.

Возможно, у вас возник вопрос: что будет, если файл, который вы указали как выходной, уже существует на диске? PIP просто запишет на то же место новую информацию. А как быть с защитой? Если

файл имел статус R/W, то он просто перепишется. Если же статус был R/O, в этом случае PIP сообщит, что файл уже существует и предназначен только для чтения. После этого вам нужно будет ответить на вопрос, стирать его или нет. Если вы ответите N, то копирования не произойдет и все останется, как было. ответив Y, вы сотрете старый файл и запишете на его место новый. Вообще, полезно внимательно читать то, что сообщается вам на экране дисплея. Иногда там говорятся полезные вещи.

Отметим еще одну важную особенность команды PIP. Она копирует только те файлы, которые можно увидеть в каталоге при помощи команды DIR. Поэтому если вы присвоили файлу статус SYS, своевременно измените его на DIR при помощи команды STAT.

Как вы помните, пользоваться командой PIP можно двумя способами. Вы либо сразу указываете в командной строке всю последовательность действий, либо, набрав PIP и нажав клавишу ВК, последовательно вводите их. В первом случае после самой команды через пробел вводятся ваши требования, например, копирования трех файлов в один:

```
A>PIP B:PROG.TXT=A:PROG1.TXT, PROG2.TXT, PROG3.TXT
A>
```

Обратите внимание, что в этом случае появляется промпт CP/M, а не PIP.

Команда PIP позволяет также выводить содержимое файлов на стандартные устройства, которые мы рассматривали при изучении команды STAT. Мы можем также упаковывать с помощью PIP информацию, вводимую со стандартных устройств, в некоторые файлы. Есть возможность производить обмен информацией между стандартными устройствами. В этом случае имена стандартных логических и физических устройств полностью равноправны с именами файлов. Например, если вы хотите ввести какие-либо данные с консоли в файл TEST.TXT, необходимо ввести команду

```
A>PIP TEST.TXT=CON:
```

После этого программа переходит к процессу обмена консоли с диском: все, что вы набираете на клавиатуре, записывается на диск. Для окончания этого процесса необходимо ввести символ CTRL-Z. Он является стандартным признаком конца файла. CTRL-Z оканчивает любой текстовый файл и поэтому всегда является последним символом в процессе передачи при помощи команды PIP.

Команда PIP имеет еще массу дополнительных возможностей. Рассмотрение их выходит за рамки данной книги, и мы отсылаем интересующихся к цитированной книге М. Уэйта и Дж. Ангермейера.

Наконец, вы стали достаточно подготовленным системным программистом. Вы с легкостью, вызывающей восхищение и восторженные возгласы родственников и знакомых, форматируете диски и копируете файлы. Со снисходительной улыбкой мага вы делаете файлы неви-

димыми. Вы можете все. Нажатие двух кнопок - и та информация, что выводилась на экран, пошла и на принтер. Короче, вы ас программистского дела. И это действительно так. Если вы внимательно ознакомились с содержимым данного раздела, вам, возможно, на всю жизнь хватит сведений об операционной системе CP/M.

Мы не рассматривали здесь такие программы, как DDT, ASM, LOAD, SAVE и т. д. Эти программы обычно нужны системным программистам, работающим на языке Ассемблер. Сейчас мы приведем только процедуру копирования программы PIP из одной области диска в другую. Эту процедуру выполняйте, как написано, не вдаваясь особенно в подробности:

```
A>DDT PIP.COM
-
CTRL-C
A>USER n
A>SAVE m-PIP.COM
A>
```

Здесь n - номер юзера. Для вычисления m определите с помощью команды STAT количество записей в файле PIP.COM, разделите это количество на два и округлите в большую сторону.

Если вы ничего из последнего абзаца не поняли, то либо прочитайте в книге М. Уэйта и Дж. Ангермейера о DDT и SAVE, или не разбивайте диск на области.

На этом мы заканчиваем знакомство с операционной системой ПЭВМ Корвет и займемся языками высокого уровня.

7. ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Читателю предлагается задуматься о пользе переводчиков при общении с малообразованными племенами тузелцев. Возможно, после этого он начнет уважать труд системных программистов

Давайте отвлечемся на минутку от забот, связанных с работой на вашем компьютере. Представьте себе, что вы являетесь разработчиком первого в мире компьютера. Вот вы достали, наконец, все необходимые компоненты и собрали первую в мире ЭВМ. Вы глядите на свое детище и думаете, что с ним делать дальше. Прежде всего хорошо было бы убедиться в его работоспособности. Вы можете сказать - проще простого. Вставили системный диск в дисковод А и загрузили операционную систему. Но, по условиям игры, ваш компьютер первый в мире и никаких операционных систем для него не существует. Тогда вы скажете, что программу загрузки можно поместить в ПЗУ. А как это сделать? Непонятно.

К сожалению, современная микроэлектроника еще не производит микропроцессоры, способные понимать команды на человеческом языке.

Как вы помните, микропроцессоры и все микро-ЭВМ в целом воспринимают только некоторые последовательности нулей и единиц, называемые командами. В распоряжении создателя первой в мире микро-ЭВМ имеется его детище и ручной программатор ПЗУ. Такие штуки бывают разных конструкций, но одна черта у них общая. Вы сначала тумблерами набираете адрес ячейки, потом то, что в нее нужно записать. После этого нажимаете кнопку ЗАПИСЬ, и если все в порядке, то необходимые данные попадут в ПЗУ. Вроде бы несложно. Но объем ПЗУ измеряется тысячами таких ячеек. Если вы ошиблись в записи предпоследней ячейки, вам придется записывать все ПЗУ заново, а это очень утомительно.

Первый способ программирования - "в кодах". Что это такое? Вы терпеливо составляете на бумаге список команд, которые будет выполнять ЭВМ. После этого, вообразив себя микропроцессором, вы с карандашом в руках проходите по всей программе и убеждаетесь в том, что она работает правильно. Затем вы вручную помещаете эту программу на носитель, с которым умеет работать ЭВМ. Если таких носителей нет, то программа размещается в ПЗУ. Можете поверить на слово - это каторжный труд.

Естественно, люди, помучившись с подобным способом программирования, начали думать, как же сделать этот процесс более приятным и легким. Прежде всего пришла в голову мысль назвать каждую команду понятным словом. Естественно, что запомнить слова, имеющие определенный смысл, легче, чем совокупность нулей и единиц. Тогда, имея под рукой словарь, в котором каждому такому слову ставится в соответствие код команды, легко перевести текст программы в последовательность нулей и единиц, доступных для понимания машине. В результате в штатных расписаниях вычислительных центров появились сотрудники, которые и выполняли роль переводчиков. Однако такое решение проблемы ничего в сущности не решило, а только добавило хлопот, связанных с тем, что эти сотрудники делали кучу ошибок.

Далее пользователи ЭВМ пришли к выводу, что проще и дешевле научить ЭВМ саму переводить программы на машинный язык. С этого момента разошлись дороги системных программистов и простых пользователей. Некоторые из пользователей решились на очень важное дело - написание первой программы-переводчика. Люди, которые пишут такие программы, стали называться системными программистами. Пользователи, получив в руки программу-переводчика и другие полезные программы, обслуживающие различные устройства в ЭВМ, сразу перестали интересоваться тем, как их программы в действительности воспринимаются машиной.

Итак, в мире ЭВМ появился переводчик - программа, переводящая наименования команд в соответствующие коды. Так появился первый язык программирования - Ассемблер (англ. Assembler). Этот язык по существу мало отличается от собственно машинного языка. Просто все договорились называть определенными словами различные команды микропроцессора. Так, например, команда вызова подпрограммы обычно называется CALL (вызов). Подобные слова называются мнемоникой. Придуманы они таким образом, чтобы их было легко запомнить. Естественно, что для различных типов микропроцессоров команды и мнемоника могут отличаться. Поэтому языки Ассемблер разные для различных типов ЭВМ.

Предположим, что вы написали программу на языке Ассемблер и хотите теперь, чтобы машина ее выполнила. Что для этого нужно сделать? Прежде всего перевести ее на машинный язык. Эта процедура называется компиляцией программы. Но этого мало. Как вы помните, при обсуждении процесса работы ЭВМ с программой мы говорили, что программа должна быть размещена в определенных ячейках памяти и процессор должен знать, какая ячейка первая. Если ваша программа не представляет собой монолитный блок, а состоит из различных частей, то нужно все их связать воедино и приязать к определенному начальному адресу. Этой проблемой занимается специальная программа LINK (компоновщик). Конкретное название этой программы может быть другим, например L80, в зависимости от того, какая фирма эту программу написала.

Обработав свою программу компоновщиком, вы получаете наконец окончательный перевод и можете ввести результат в машину. Для этой цели в любой операционной системе существует специальная подпрограмма-загрузчик. Эта подпрограмма является резидентной и пользователь ее явно не видит. Вызывается она автоматически, если с клавиатуры вы вводите какую-либо команду, не являющуюся резидентной для операционной системы. Сообразив, что вы не хотите активизировать резидентные команды, операционная система начинает искать на диске файл с именем введенной команды и с расширением .COM (транзитные команды). Если такой файл находится, начинает работать загрузчик. Он загружает содержимое файла в память ЭВМ и настраивает счетчик команд микропроцессора на первую ячейку памяти, занимаемую вашей программой. С этого момента начинается выполнение вашей программы.

Таким образом, люди получили в руки очень полезный инструмент, позволивший им писать программы хоть и на примитивном, но все-таки "человеческом" языке.

Дальше возник вопрос, а почему бы не написать программу-переводчика, которая переводила бы с более осмысленных фраз. Действительно, поскольку микропроцессор сам по себе не имеет таких команд, которые позволили бы производить сложные вычисления в одно действие, написание соответствующих подпрограмм на Ассемблере потребовало больших усилий и сделало сами программы довольно громоздкими. Программа-переводчик более высокого уровня может сама переводить сложные математические формулы в набор ассемблеровских команд. Так появились языки высокого уровня. Одним из первых был Фортран (от англ. FORmula TRANslator - переводчик формул). Этот язык оказался настолько удачным в применении к научным расчетам, что остается массовым даже сейчас, когда появилось множество других языков.

Появление языков высокого уровня стало возможным благодаря созданию еще одной программы-переводчика. Теперь вы с ее помощью сначала переводите текст своей программы на язык Ассемблер, а затем запускаете стандартные процедуры ассемблирования, компоновки и загрузки. Заметим, что многие фирмы объединяют программу-переводчика на Ассемблере с программой ассемблирования в одну, называемую транслятором или компилятором.

Для того чтобы пользоваться языком высокого уровня, вы должны иметь как минимум транслятор и компоновщик. Загрузчик всегда есть в операционной системе. Кроме этого, языки высокого уровня, как правило, комплектуются библиотеками, содержащими различные полезные подпрограммы, например вычисления синусов, косинусов и других математических функций, функций для работы с текстами и т. д. Необходимые для вашей программы функции выбираются из библиотеки компоновщиком автоматически.

Для того чтобы программы-переводчики и компоновщик правильно обрабатывали вашу программу, необходимо строго соблюдать правила (грамматику) того или иного языка. Обычно ошибки в грамматике обнаруживаются транслятором и выдаются на экран или принтер. При попытке использовать подпрограммы, отсутствующие в библиотеках или в вашей программе, появляются ошибки, обнаруживаемые компоновщиком. Так что попытки обмануть эти программы ни к чему не приведут. Лучше хорошо изучите грамматику и пользуйтесь только имеющимися функциями.

Мы разобрали процесс обработки программы, написанной на языке высокого уровня. Конечным продуктом его является файл, содержащий только коды команд процессора. Такой способ обработки программ очень удобен. Вы получаете перевод вашей программы, и, если вы в ней ничего не меняете, процесс ее запуска предельно упрощается. Быстрее запускаются только резидентные команды операционной системы. Однако если вы хотите что-либо изменить в вашей программе,

весь процесс перевода придется повторить заново. Поэтому процесс отладки программ остается довольно продолжительным по времени.

Программисты сообразно стилю программирования делятся на два класса. Первый класс - как правило, люди, прошедшие школу программирования в кодах или на языке Ассемблер. Они никогда не начнут вводить текст программы в машину, пока сами с карандашом в руках не пройдут по ней и не выловят многочисленные ошибки, неизбежно возникающие при написании программ. В результате они создают настоящее произведение искусства. С такими программами приятно работать.

Другой класс - начинающие пользователи, обычно страдающие манией величия. Они уверены, что любая их программа - это шедевр и поэтому, пользуясь тем, что персональная ЭВМ работает в диалоговом режиме, сразу начинают вводить в нее какие-то программы и без лишних размышлений компилировать их, компоновать и запускать на счет. Результатом этой деятельности является непроизводительная трата времени, поскольку каждая, пусть даже самая мелкая, ошибка заставляет вас заново компилировать программу. Однако это еще полбеды. Написанные таким способом программы являют собой монстров, которые чудом ухитряются подавать признаки жизни. Программисты этого класса нередко упрекают системщиков в том, что написанные ими компиляторы и компоновщики медленно работают. Вместо того чтобы с олимпийским спокойствием не обращать внимания на вопли ремесленников от программирования, системщики изобрели специальные языки высокого уровня специально для начинающих.

В основу этих языков положена простая мысль: переводить в машинные коды не всю программу, а только ту ее строку, с которой машина в настоящий момент работает. Теперь если объединить переводчика с программой редактора, которая позволит изменять текст любой строки, и присвоить определенной кнопке на клавиатуре функцию остановки программы в любом месте (название этой кнопки BREAK или СТОП), процесс отладки программ станет очень быстрым.

Действительно, при обнаружении ошибки переводчик сообщает вам, что в строке с таким-то номером обнаружена такая-то ошибка, прерывает выполнение программы и переходит в режим редактирования. Вы также можете сами, используя кнопку СТОП, оперативно вмешаться в процесс выполнения программы и изменить что угодно в любой строке. Языки с подобным способом организации перевода называются интерпретируемыми, а программа-переводчик - интерпретатором. Примером может служить известный вам язык Бейсик.

Казалось бы, чего еще желать? Удобнее не придумаешь. Но за комфорт нужно платить. В данном случае расплата - это время выполнения программы. Прежде чем что-нибудь выполнится, необходимо потратить время на перевод.

Действительно, в случае программ, написанных на компилируемых языках типа Фортран, отлаженная программа компилируется раз и навсегда (во всяком случае этот процесс не производится при каждом запуске). Для интерпретируемых языков процесс перевода повторяет-

ся каждый раз, когда вы запускаете программу. Поэтому выполняются такие программы значительно медленнее.

Справедливости ради отметим, что появились варианты интерпретируемых языков, которые могут создавать в конце концов файл из кодов команд. Однако даже в этом случае компилируемые языки являются более предпочтительными для программистов-профессионалов. Дело в том, что интерпретируемые языки обычно сильно упрощены в целях экономии времени, затрачиваемого на перевод. Однако для начинающих интерпретируемые языки типа Бейсика - лучшее, что можно порекомендовать.

В настоящее время существует масса гибридов компилируемых и интерпретируемых языков, которые сочетают удобства как одних, так и других. К сожалению, создать абсолютно совершенный язык, свободный от недостатков, обсужденных выше, невозможно. Поэтому начинающим мы рекомендуем работать с интерпретаторами типа Бейсика. По мере накопления опыта программирования, когда вы почувствуете, что больше времени уходит на ожидание результата счета, чем на исправление ошибок, переходите к компиляторам.

Прежде чем перейти собственно к Бейсику, поговорим еще об одном типе языков программирования. Дело в том, что языки типа Бейсика и Фортрана предназначены в основном для проведения различных вычислений. Однако область применения ПЭВМ не ограничивается только одними вычислениями. Существуют различные сферы человеческой деятельности, выходящие за рамки четырех арифметических действий.

Приведем несколько примеров. Допустим вы хотите завести электронную картотеку для своей библиотеки. Или вам крупно повезло, и ваша ПЭВМ подключена к телефонной сети. Естественно возникает желание научить ее самостоятельно разбираться с вашей телефонной книжкой. Для подобных задач созданы специальные пакеты программ, работающие с так называемыми "базами данных". Примерами могут служить программы DBASE II фирмы Aston Tate и MULTIPLAN фирмы Microsoft. Эти программы работают не с отдельными числами или символами, как, например, Бейсик, а со сложными объектами, которые могут содержать и числа, и целые куски текста. При этом в эти языки заложены как математические функции так и операции с текстами.

Поясним сказанное примером. Допустим, что мы являемся большими начальниками. Мы руководим предприятием и у нас много сотрудников. Мы хотим составить электронную записную книжку, в которой можно было бы хранить информацию о фамилиях сотрудников, их должностях, окладах и т. д. Книжка должна быть удобной. Это значит, что мы должны иметь возможность сортировать ее по фамилиям, должностям или окладам. Мы должны иметь возможность легко вставлять в нее новые фамилии и стирать старые. Эту задачу можно решить, используя язык Бейсик. Как это сделать изложено в разделе 22 нашей книги. Однако для серьезного решения данной проблемы обычно используют названные выше специальные языки.

С чем же в данном случае будет работать, например, DBASE II? Основной неделимой ячейкой информации будет объект, состоящий из

двух кусков текста - фамилии и должности и числа - оклада. Такой объект обычно называется записью (от англ. record).

Итак, запись имеет вид

Фамилия	Иванов
Должность	Инженер
Оклад	120

Каждая строка в записи называется полем. Программы, работающие с подобными объектами позволяют менять содержимое любого поля, сортировать всю базу данных по признакам указанным в различных полях, производить необходимые арифметические операции с численными полями и т. д.

Однако человеческая мысль на этом не остановилась. Появились языки, которые понимают простую человеческую речь. Работая на ПЭВМ Корвет, вы можете использовать один из таких языков uSIMP. При работе с этим языком, для нахождения, скажем, аналитических выражений для корней квадратного уравнения, вы просто пишете фразу типа

SOLVE ($Y=AX^2+BX+C$)

Слово solve означает - решить.

После этого машина сама находит метод решения такого уравнения и сообщает вам найденные корни. При этом корни представляются в аналитическом виде, т. е. в виде формулы.

На этом языке написан целый набор программ, называемый uMATH, который позволяет аналитически дифференцировать и интегрировать, решать алгебраические и дифференциальные уравнения, работать с матрицами и векторами, суммировать ряды и т. д.

Подобные языки являются шагом к созданию искусственного интеллекта.

Займемся теперь непосредственно Бейсиком.

8. ЗАГРУЗКА ЯЗЫКА БЕЙСИК

Читатель знакомится с процессом вызова переводчика и начинает разговаривать с машиной

Пришла, наконец, пора заняться настоящим делом. Сейчас мы научимся вычислять, сколько будет дважды два. Но для этого нам нужен переводчик. Где его взять? Тот, кто внимательно читал раздел 3, помнит, что если в дисковод не вставлен системный диск или дисководы в вашей машине вообще отсутствуют, то при включении машины на экране появится надпись[1]):

```
Бейсик Корвет в. 1
Москва 1986

OK
```

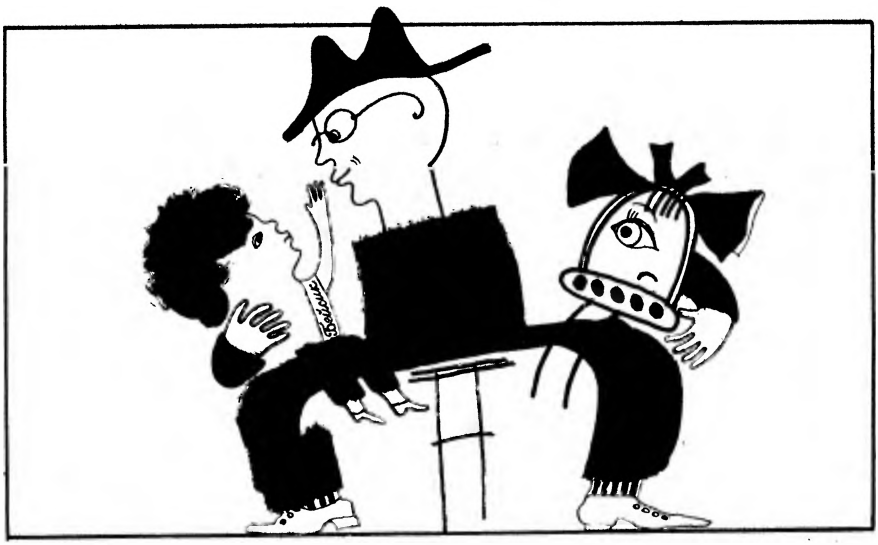
Эта надпись сообщает вам, что загружен интерпретатор Бейсика, версии 1. Слово OK, как вы, наверное, догадались, является промптом интерпретатора.

Вы, как разносторонне подготовленный пользователь, можете сразу сказать, откуда появился Бейсик. Правильно, основная часть интерпретатора хранится в ПЗУ. Поэтому для первых проб вам не нужны ни дисководы, ни магнитофон с кассетами. А если все-таки вы вставили системный диск и случайно загрузили операционную систему? Что же, вытаскивать его теперь и перезагружаться? Не обязательно. Для того чтобы запустить Бейсик из ОС, достаточно набрать команду BASIC и нажать клавишу ВК (естественно, что соответствующий файл BASIC.COM должен быть на текущем диске). После этой команды экран очистится и появится сообщение о загрузке Бейсика.

В чем разница между двумя способами загрузки Бейсика? Дело в том, что, хотя основная часть интерпретатора хранится в ПЗУ, специфические функции, касающиеся записи программ на диски и считывания с них, загружаются, только если ваша машина укомплектована дисководами. Поэтому правильной в этом случае является загрузка Бейсика из ОС.

---

[1]) Эта надпись может быть другой в зависимости от версии Бейсика.



Ну вот, Бейсик, наконец, загружен. Сначала изучим внимательно клавиатуру. Для этой цели удобно пользоваться рис. 1. Кроме кнопок с обозначениями букв русского и латинского алфавита, цифр и знаков препинания, имеются еще кнопки, на которых написаны слова. Познакомимся с функциями этих специальных кнопок. По ходу изложения нажимайте на обсуждаемые кнопки и изучайте произведенный эффект. Помните, что никакими действиями с клавиатурой машину вывести из строя невозможно.

Обратите внимание на верхний ряд кнопок из основного поля клавиш. На каждой из них присутствуют два символа, один над другим. Один из них цифра, а другой - некий знак, например 1 и !. Внимательно посмотрев на клавиатуру, вы увидите несколько подобных кнопок, например с символом точки и с символом >. Для того чтобы ввести верхний символ, достаточно просто нажать соответствующую кнопку. Для ввода нижнего символа необходимо одновременно с выбранной кнопкой нажать клавишу переключения регистра. Таких клавиш две. Расположены они справа и слева от края предпоследнего ряда кнопок. Они имеют надписи РГ и SHIFT. Кроме этого на них нарисована стрелка вверх-вниз. Попробуйте вводить с клавиатуры символы верхнего и нижнего регистров. Результат будет отображаться на экране.

Займемся русскими и латинскими буквами. Для переключения шрифтов служит кнопка, расположенная слева от самой большой клавиши - пробела. На ней написано АЛФ (рис. 1). Нажатие на нее приводит к тому, что с клавиатуры будут вводиться русские буквы. При этом эту клавишу постоянно нужно удерживать в нажатом состоянии. Попробуйте.

Для ввода маленьких букв используются кнопки РГ, уже известные вам. Если вы нажмете кнопку с какой-либо буквой, одновременно удерживая в нажатом положении РГ, то на экране дисплея появится изображение маленьких букв.

Но такой способ работы неудобен. Действительно, вам постоянно нужно удерживать в нажатом положении или АЛФ, или РГ. Для фиксации режима служит специальная кнопка-фиксатор. Расположена она слева от кнопки АЛФ и на ней нарисован кружок. Нажмите одновременно фиксатор и кнопку АЛФ. Теперь отпустите их и попробуйте ввести какую-либо букву. Вы видите, что на экране дисплея отображаются русские буквы. Нажатие фиксатора одновременно с АЛФ как бы удерживает последнюю в нажатом состоянии. Чтобы вернуться в режим ввода латинских букв, нажмите снова фиксатор и АЛФ. Потренируйтесь.

Наверное, вы уже догадались, что для фиксации верхнего или нижнего регистра нужно нажать одновременно фиксатор и РГ. При этом регистр переключится. Попробуйте теперь понабирать различные слова на экране на русском и английском языках, маленькими и большими буквами. В результате этих упражнений вы должны научиться вводить все символы, изображенные на кнопках.

Теперь мы надеемся, что у вас нет проблем с вводом различных символов. Займемся некоторыми другими важными кнопками. Прежде всего обратите внимание на самую крупную кнопку справа (рис. 1). На ней нарисована стрелка, изогнутая влево. Те, кто внимательно читал раздел 6, хорошо знают, что эта кнопка служит для ввода команды в машину. После нажатия этой кнопки все, что было до этого набрано на клавиатуре, вводится в машину как команда, которую она тут же будет пытаться выполнить. Эта кнопка имеет много названий, наиболее распространенные из них - ВК (Возврат Каретки), CR (Carriage Return) и, наконец, просто RETURN или ENTER. В дальнейшем будем называть ее ВК. Понажимайте на эту кнопку. Если вы перед этим набирали какие-либо сообщения на клавиатуре, то на дисплее скорее всего появится сообщение об ошибке, например:

```
Здравствуйте, я ваша ЭВМ Корвет!
ОШИБКА СИНТАКСИСА
ОК
```

Если ничего не было введено перед нажатием ВК, то курсор просто переместится на строку ниже.

Вы, наверное, помните, что, изучая работу интерпретатора языка Бейсик, мы говорили, что должна существовать кнопка, позволяющая в любой момент прерывать выполнение программы. Эта кнопка - самая красивая на клавиатуре. Она красного цвета и на ней написано СТОП и STOP. При нажатии на эту кнопку на экране появляется сообщение ^C и курсор перемещается на строку вниз.

Возможна ситуация, когда вы ввели какой-либо символ ошибочно. Как его убрать? Для этой цели существует специальная кнопка BACKSPACE - позиция назад. На нашей клавиатуре эта кнопка размещена прямо над клавишей ВК и на ней нарисована стрелка влево. При нажатии на эту кнопку курсор перемещается на одну позицию влево и стирает находящийся там символ.

Предположим, что вы хотите набрать какой-либо текст, состоящий из равных колонок, разделенных некоторым числом пробелов. Чтобы облегчить ввод подобных текстов, предусмотрена кнопка табуляции. Она расположена слева в третьем ряду сверху (рис. 1). На ней написано ТАБ. Понажимайте на эту кнопку и посмотрите на перемещение курсора. Под кнопкой ТАБ расположена знакомая знатокам ОС СР/М кнопка CTRL или УПР. Как ею пользоваться, вы уже знаете, если внимательно прочли раздел 6.

Теперь обратите внимание на кнопки, расположенные в самом верхнем ряду. Их пять. Это функциональные клавиши. При нажатии на них сразу вводятся определенные команды языка Бейсик. Функции этих кнопок мы разберем позднее. Если ваше любопытство разбужено, понажимайте на них. Ничего страшного не произойдет.

Остальные кнопки к Бейсику отношения не имеют.

Ну вот, наконец пришла пора поговорить с машиной. Вы уже, наверное, попробовали набирать различные предложения, ожидая осмысленной реакции от машины. Однако ее не последовало. Что же, вам попалась глупая машина? Не волнуйтесь. Машина у вас хорошая, и сейчас мы научимся с ней разговаривать.

До сих пор мы не требовали от машины многого. Мы нажимали на кнопки, а она просто выводила соответствующий символ на экран дисплея. Однако теперь наступило время заставить эту кучу радиодеталей работать. Пусть для начала ваша машина научится дружески приветствовать вас и говорить: "Здравствуйте, я ваша ЭВМ Корвет!"

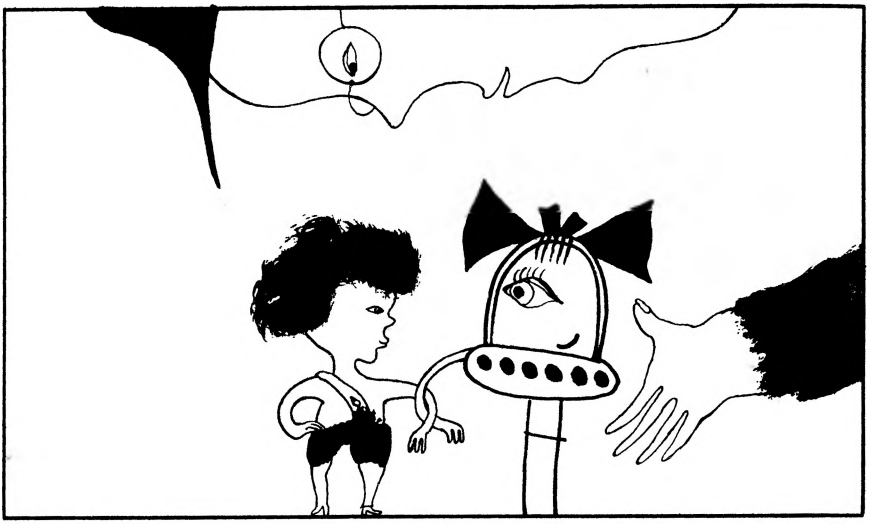
Сначала нажмите на клавишу СТОП. В результате в текущей командной строке вы уничтожите все, что уже набрали. После того как курсор переместится на следующую строку, наберите NEW. Это первая команда языка Бейсик, с которой вы познакомитесь. NEW в переводе с английского означает "новый". По этой команде интерпретатор Бейсика стирает из памяти ЭВМ все, что вы туда записали в ходе предыдущих упражнений. Теперь машина готова принимать ваши новые команды. Попробуем попросить машину вывести приветственную надпись на экран. Для этого служит команда PRINT. В переводе на русский язык это слово означает "печатать". Приблизительно так же воспринимает эту команду машина. Наберите на экране

```
PRINT "Здравствуйте, я ваша ЭВМ Корвет!"
```

Подождите нажимать ВК. Проверьте, что у вас получилось. Замеченные ошибки исправьте при помощи BACKSPACE (клавиши со стрелкой влево). Получилось? Прекрасно. Теперь нажмите ВК. Если все было сделано правильно, то на дисплее будет изображено:

```
Здравствуйте, я ваша ЭВМ Корвет!
OK
```

Если компьютер не вывел на экран то, что вы написали в кавычках (вместо этого может появиться сообщение об ошибке), попробуйте еще раз. Проверьте несколько раз, что вы набираете, перед тем как на-



жать ВК. Если все в порядке, потренеруйтесь в выводе различных надписей. Общее правило таково:

PRINT "Та надпись, которую вы хотите"

Обратите внимание: весь текст, заключенный в кавычки, выводится на экран в точности как написано. Исключение составляют сами кавычки, которые не выводятся на экран.

Теперь вы видите, что очень просто заставить машину выводить на экран все, что пожелаете. Это очень приятно. Но все-таки машина называется вычислительной. Как же заставить ее что-нибудь вычислить? Займемся, например, решением древней проблемы, сколько будет 2 плюс 2, т. е. 2+2=? Для начала убедитесь, что компьютер готов к приему новой команды. Если нет, нажмите СТОП. Теперь наберите команду

PRINT "2+2="

Если появилось сообщение об ошибке, попробуйте еще раз.

Прекрасно. Теперь задача поставлена перед компьютером. Почему же он ее не решает? Потому, что все, что заключено в кавычки, выводится на экран точно как написано. Так как же все-таки заставить компьютер вычислять? Можете догадаться сами?

Если вы набрали 2+2 и ничего не произошло, не удивляйтесь. Вы ничего не сломали. Просто, как во всяком языке, все, что заключено в кавычки, воспринимается как прямая речь. Вы попросили компьютер сказать "2+2=" - он это и сделал. Интерпретатор такие послания не обрабатывает. Теперь попробуем ввести ту же команду, только без кавычек и знака равенства:

PRINT 2+2

Ну как? На экране появилось: 4. Обратите внимание, что если после 2+2 вы поставите знак равенства, то появится сообщение об ошибке:

ОШИБКА СИНТАКСИСА

Вы, наверное, не раз его уже получали. Это означает, что вы пытаетесь общаться с машиной при помощи слов, которых она не понимает. Поэтому будьте внимательны.

Скептики могут сказать: "Подумаешь, 2+2. Это может каждый вычислить в уме". Хорошо. Попробуем что-нибудь посложнее. Например, разделим 17893 на 14.5. Для этого наберите команду

```
PRINT 17893/14.5
```

На экране появится: 1234. Обратите внимание, что знак деления - это косая черта, а не двоеточие, десятичные знаки отделяются от целой части точкой, а не запятой, как обычно. Это важно, запомните это хорошо.

Вы убедились, что вашему компьютеру по плечу сложные математические задачи. Но то, как он выводит результат на экран, вас, конечно, не удовлетворяет. Значительно удобнее было бы, если бы компьютер выводил на экран 2+2=4. Для этого существуют два способа. Первый способ - это в одной и той же команде PRINT поместить сообщение 2+2=, заключенное в кавычки, и отделить от него точкой с запятой саму задачу:

```
PRINT "2+2=";2+2
```

Можете даже расширить сообщение, добавив

```
PRINT "2+2=";2+2;"С моей точки зрения"
```

Попробуйте. Второй способ - использовать запятую вместо точки с запятой:

```
PRINT "2+2=",2+2
```

Попробуйте. Увидели, в чем разница между запятой и точкой с запятой? Если ваши три последних упражнения окончились успешно, вы, должно быть, обратили внимание, что в первых двух случаях сообщения выводились сразу одно за другим, а в третьем случае они выводились в определенной зоне. Можно использовать эти два способа совместно:

```
PRINT "17893/14.5=";17893/14.5,"17893/7.25=";17893/7.25
```

Запятая указывает компьютеру, что следующее сообщение должно быть выведено в следующей зоне. Зона имеет длину 16 символов.

Выполните упражнения. Попробуйте различные варианты приведенных выше вычислений, используя запятые, точки с запятыми. Внимательно изучите, как Корвет интерпретирует ваши команды.

Теперь поупражняйтесь в различных арифметических операциях. Используйте четыре арифметических действия: сложение (+), вычитание (-), умножение (*) и деление (/). Если ваша фантазия истощилась, проделайте следующие упражнения:

1. Определите десятичные выражения для простых дробей 1/3, 2/3, 7/8, 15/6; например, 1/4=0.25.

2. Решите более сложную задачу: 83+11/21-33.7.

3. Попробуйте использовать большие числа: 1234500*70010.10.

4. Научите компьютер сообщать, что за проблему он решает; например: 100+200=300.

Теперь немного об ошибках. Вам, скорее всего, уже неоднократно в ответ на ваши действия машина отвечала сообщениями о различных ошибках. Самая распространенная -

ОШИБКА СИНТАКСИСА

Попробуйте набрать на экране

PRRINT "Я думаю, что все в порядке"

Компьютер ответит: ОШИБКА СИНТАКСИСА. Это означает, что слова PRRINT в словаре Бейсика нет. Компьютер не понимает, что вы имели в виду команду PRINT.

Существует еще множество ошибок. Попробуйте, например, набрать на экране

PRINT 5/0

Вы получите сообщение

ДЕЛЕНИЕ НА 0

Что вы ожидали? Деление на нуль запрещено даже на персональном компьютере, установленном в вашем доме.

Попробуйте заставить компьютер вывести на экран число, состоящее из единицы и ста нулей. Не поленитесь, попробуйте. Вы получите сообщение ПЕРЕПОЛНЕНИЕ. Это значит, что число слишком большое.

Возможно, вы уже почувствовали, что можете решить огромное число проблем, используя PRINT. Однако не торопитесь. PRINT имеет большое количество "родственников" и "знакомых", обладающих возможностями даже большими, чем сам PRINT.

Для начала очистим экран от предыдущих упражнений. Наберите CLS и нажмите ВК. Кстати, вы узнали еще одно командное слово в словаре Бейсика - CLS (от англ. CLear Screen - очисть экран). Теперь попробуем вывести на экран 25 символов *. Первая, пришедшая в голову мысль - это набрать PRINT, затем в кавычках ввести 25 символов * и нажать ВК. Действительно, компьютер выведет 25 раз этот символ. Однако эту же процедуру можно проделать проще, не нажимая 25 раз на клавишу *. Для этого наберите

PRINT STRING$(25,"*")

Результат оказывается тем же. Этот способ характерен для многих языков высокого уровня. Можно работать не только с числами, но и с так называемыми строковыми переменными. Мы вернемся к ним в свое время. Сейчас просто поиграйте. Введите

PRINT STRING$(10,"0")

Первое число в скобках говорит, сколько символов нужно вывести. Это число может быть любым в диапазоне от 0 до 50. Второй символ в кавычках сообщает, чем нужно заполнить строку. Попробуем напечатать что-нибудь новенькое:

PRINT STRING$(40,149)

В чем разница между последними двумя командами? В первом случае у нас фигурировал нуль, заключенный в кавычки, во втором - просто число. В чем же дело? Оказывается, что каждому символу в машине ставится в соответствие некий код. Это естественно - как же иначе их отличать друг от друга. Система кодировки может быть различной. В Корвете принята система кодов ASCII. К этим кодам мы еще вернемся при обсуждении строковых переменных. Таким образом, для того чтобы объяснить машине, какой символ вы имеете в виду, нужно либо указать его заключенным в кавычках, либо указать его код. Число 149 говорит компьютеру, что нужно вывести специальный символ, который вы не можете ввести с клавиатуры.

Вы, наверное, обратили внимание, что происходит на экране, когда компьютер выполняет вашу команду. Появляется промпт OK и курсор перемещается на строку вниз. Когда курсор перемещается в самый низ экрана и не может уже двигаться ниже, в движение приходит весь текст на экране. Он начинает двигаться вверх. Верхние строчки исчезают, а снизу появляются новые. Эта процедура похожа на чтение древних свитков.

Неужели все, что вы набирали, компьютер забывает сразу после выполнения команды? К сожалению, да. Но не расстраивайтесь. Для того чтобы компьютер выполнял ваши команды много раз, необходимо собрать их вместе в программу. Но прежде чем перейти к программам, поупражняйтесь в выводе на экран надписей и результатов простых вычислений, после чего выключите компьютер и отдохните.

9. ЗНАКОМСТВО С ПРОГРАММИРОВАНИЕМ НА ЯЗЫКЕ БЕЙСИК

Первые шаги. Начинаем писать послания Корвету, называемые программами

Вы, наверное, обратили внимание, что машина выполняла ваши команды мгновенно. Каждый раз, когда вы нажимали клавишу ВК, машина смотрела, что вы набрали на экране, и пыталась выполнить то, что вы от нее хотели.

Как только ЭВМ выполняла операцию или выдавала сообщение об ошибке, она сразу же забывала об этом и ждала следующих инструкций. Единственным способом заставить ее повторить какую-либо опе-

рацию было набрать команду на экране заново и опять нажать клавишу ВК.

Очевидно, что это плохой способ работы с компьютером. Возможно, у вас уже созревает желание упаковать Корвет в коробку и послать его на завод-изготовитель вместе с гневным письмом. Но не торопитесь. Существует способ научить компьютер запоминать ваши команды. Сейчас мы узнаем, как это делается. Для начала убедитесь в том, что на экране имеется промпт Бейсика ОК и под ним белый прямоугольник курсора. Если нет, нажмите СТОП. Очистите машину от своих предыдущих экспериментов. Как вы помните, для этого нужно набрать команду NEW и нажать клавишу ВК.

Теперь наберите следующую инструкцию, не нажимая ВК:

10 PRINT "Здравствуйте, я ваша ЭВМ Корвет!"

Обратите внимание на число 10 слева, отделенное пробелом от команды PRINT. Если есть ошибки, исправьте их.

Теперь нажмите ВК и посмотрите, что произошло. Ничего не произошло. Что же случилось? Может быть, сломался компьютер? Но он же только что хорошо работал.

На самом деле произошло важное событие. Раньше, когда вы нажимали ВК, Корвет воспринимал вашу команду и выполнял ее. Теперь, когда он обнаружил в начале строки число 10, он подумал: "Ага! Это не обычная инструкция, которую нужно сразу же выполнить. Это инструкция многоразового пользования". Поняв это, он спрятал ее в память, чтобы выполнить по вашему требованию.

Как же сказать ему, что инструкцию нужно выполнить? Наберите команду RUN и нажмите ВК. Корвет обратился к вам с дружественным приветствием! Набирайте RUN и нажимайте ВК снова и снова. Компьютер будет каждый раз выполнять ...ту PRINT. Теперь вы, наверное,

поняли, что команда RUN заставляет компьютер выполнять инструкции, заложенные в его память.

Введите команду CLS. Экран очистится и всякие следы от ранее введенной вами команды PRINT исчезнут. Однако она останется в памяти Корвета. Не верите? Проверьте. Для того чтобы посмотреть, какие инструкции хранятся в памяти машины, существует команда LIST. Наберите LIST и нажмите ВК. Вы увидите инструкцию.

Теперь давайте попробуем написать более сложную программу. Язык Бейсик устроен так, что можно добавить дополнительные инструкции к уже имеющимся в памяти машины. Наберите на экране

```
5 CLS
```

Обязательно отделите 5 от CLS пробелом. Далее наберите

```
20 PRINT "Хозяин, как тебя зовут?"
```

Посмотрите с помощью команды LIST, что за программа у вас получилась. На экране должно появиться:

```
5 CLS
10 PRINT "Здравствуйте, я ваша ЭВМ Корвет!"
20 PRINT "Хозяин, как тебя зовут?"
```

Обратите внимание, что Корвет добавил к инструкции с номером 10 две новые, причем порядок их следования устанавливается по возрастанию номера слева в начале каждой строки.

Каждая строка в программе, написанной на языке Бейсик, должна иметь номер. Номер может быть от нуля до 65529. Компьютер выполняет команды по порядку и останавливается, когда достигает последней строки.

Попробуйте запустить вашу программу. Наберите для этого команду RUN. Компьютер очистит дисплей (строка с номером 5), напечатает при-

ветствие (строка 10) и спросит ваше имя (строка 20). В результате на экране дисплея появится текст:

```
Здравствуйте, я ваша ЭВМ Корвет!
Хозяин, как тебя зовут?
OK
```

Чего вы ждете? Набирайте: "Меня зовут..." и нажимайте ВК. У вас получится:

```
ОШИБКА СИНТАКСИСА
OK
```

Странно. Опять ошибка. Но ведь она сама спрашивала! Что же с ней случилось?

Здесь проявляется суровая правда жизни. Компьютер не может делать даже то, о чем сам спрашивает. Он выполняет только ваши инструкции. Вы попросили его задать вам вопрос, и он задал его. Вы не просили его воспринять ваш ответ - и он его не воспринял. Как только появился промпт Бейсика OK, это значит, что компьютер окончил выполнение вашей программы и теперь ожидает команду типа RUN, LIST, CLS, PRINT и другие, которые вам известны.

Остановитесь на минутку и дайте хорошо осесть в вашей голове мысли, что компьютер может выполнять только то, что вы ему сказали. Любая строка в любой программе, которую вы написали, должна в точности отражать то, что вы в данный момент от компьютера хотите. Не думайте, что компьютер сам догадается о том, что вы подразумеваете. Позже вы узнаете, что нужно сделать, чтобы компьютер воспринял ваше имя. Сейчас продолжим знакомство с процессом написания программ на Бейсике.

Теперь мы узнаем, как удалять отдельные строки из программы. Сначала проверим, сохранилась ли ваша программа. Вспомните, как пользоваться командой LIST. На экране должно появиться:

```
5 CLS
10 PRINT "Здравствуйте, я ваша ЭВМ Корвет!"
20 PRINT "Хозяин, как тебя зовут?"
```

Попробуем стереть строку с номером 10. Наберите 10 и нажмите ВК. С помощью команды LIST посмотрите, что стало с вашей программой. Вы видите, что строка с номером 10 исчезла. Когда вы вводите новую строку с номером, который соответствует ранее введенной инструкции, компьютер просто заменяет одну строку на другую. В нашем примере после числа 10 ничего не следовало. Именно на это "ничего" и была заменена старая строка 10. Иными словами, она исчезла. Этим способом можно исправлять ошибки в программе. Просто вводите исправленную строку с номером той строки, где были ошибки.

Как попросить компьютер забыть всю вашу программу? Для этого служит известная вам команда NEW. Теперь вы уже лучше должны понимать, что происходит по этой команде. Просто по этой команде

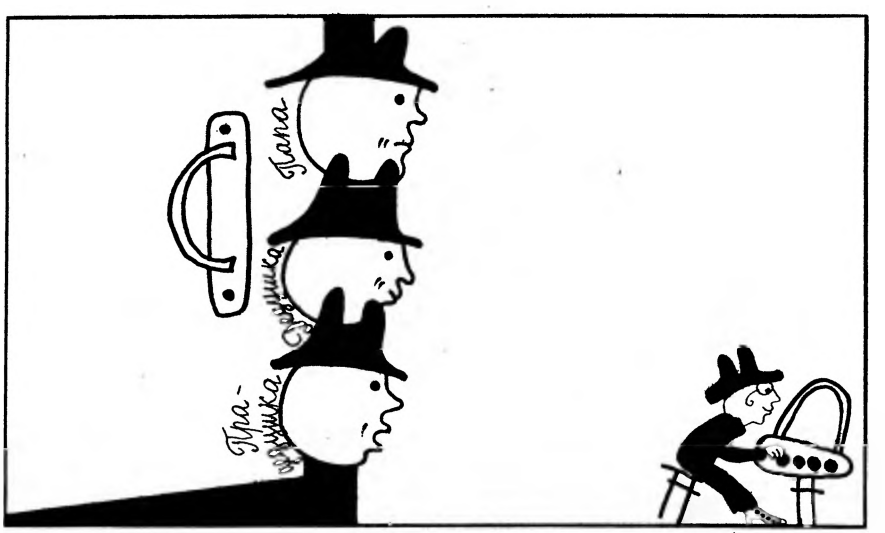
компьютер очистит место для ввода новой программы и сотрет из памяти старую программу.

Попробуйте теперь стереть вашу программу целиком. Наберите NEW и нажмите ВК. На экране появится промпт Бейсика. Проверьте с помощью команды LIST, действительно ли программа стерлась.

Теперь самое время поупражняться в программировании самостоятельно. Сейчас вам предстоит выполнить очень сложную задачу. Вы должны написать программу, которая очистит экран, выведет на экран строку, состоящую из 25 звездочек (*) и напишет "Я люблю звезды".

Будем исповедовать правильный стиль программирования. Возьмите лист бумаги и напишите на нем текст программы. Хорошенько проверьте, нет ли ошибок. Теперь аккуратно наберите текст на экране. Обратите особое внимание, чтобы каждая инструкция предварялась номером и была отделена от него пробелом. Мы рекомендуем выбирать приращение номера, равное 10. Так, первая строка имеет номер 10, вторая 20 и т. д. Когда вы введете последнюю строку (по нашему мнению, у вас их должно быть три), посмотрите, что у вас получилось, с помощью команды LIST. Исправьте замеченные ошибки с помощью переписывания соответствующей строки. Теперь, когда все ошибки исправлены, запускайте программу. Работает ли она? Если да, то самое время позвать всех своих родственников и знакомых, чтобы показать им, как вы умеете управляться с компьютером.

Возможно, у вас возник вопрос, почему мы рекомендуем нумерацию строк 10, 20, 30 и т. д., а не 1, 2, 3 и т. д. По очень простой причине. Вообразите, что вы хотите ввести еще одну строку между первой и второй. Если вы следовали нашей рекомендации, вы легко это сделаете, использовав, например, номер 15. Если вы следовали нумерации 1, 2, 3 и т. д., то вам придется переписывать всю программу после той строки, за которой вы собираетесь что-то вставить.



Что делать, если ваша программа не заработала? Прежде всего отдохните. Наиболее вероятная ошибка отображается на экране в виде

```
ОШИБКА СИНТАКСИСА В N
OK
N
```

(N - номер строки, в которой обнаружена ошибка). Обратите внимание, что компьютер сам перешел в режим исправления ошибки в этой строке. Чтобы вернуть промпт Бейсика, просто нажмите ВК. Компьютер выведет на экран содержимое строки с ошибкой. Теперь посмотрите внимательно на эту строку, найдите ошибку. Когда вы ее найдете, введите эту строку заново. Сначала наберите номер, а после этого через пробел саму инструкцию. После того как вы нажмете ВК, исправленная строка заменит старую. Просмотрите исправленную программу. Если все в порядке, запускайте ее. Если опять не получилось, посмотрите на правильный текст:

```
10 CLS
20 PRINT STRING$(25,"*")
30 PRINT "Я люблю звезды"
```

Это не единственный путь решения проблемы. Возможно, что вы придумаете что-то свое. Важен результат. Если компьютер выполнил задачу, значит все в порядке.

Теперь более сложное упражнение. Напишите программу для вычисления десятичного представления простых дробей 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9. Ваш результат должен быть представлен в виде 1/2=.5. Сделайте все так же, как и в предыдущем примере. Напишите текст программы на бумаге, проверьте, аккуратно введите в машину, просмотрите с помощью команды LIST, что получилось, исправьте ошибки и запускайте программу.

Если компьютер нашел ошибку, постарайтесь исправить ее самостоятельно. Надеемся, что вы не успели еще забыть, как это делается.

Сформулируем основные выводы, которые вы должны хорошо запомнить:

1. Программа - это перечень инструкций, хранящихся в памяти машины.

2. Каждая строка программы должна начинаться номером.

3. Для запуска программы используется команда RUN.

4. Чтобы стереть строку в программе, достаточно набрать ее номер и нажать ВК.

Теперь вы можете претендовать на звание начинающего программиста. Прежде чем продолжать совершенствоваться в этом деле, вернемся к специальным клавишам на клавиатуре (рис. 1). Обратите внимание на самый верхний ряд кнопок. На них нанесены надписи в виде буквы F и числа. Давайте с ними разберемся. Сейчас мы будем

нажимать их по очереди и смотреть, что появится на экране дисплея. Поскольку каждая из этих функций вводит некоторую команду Бейсика, то во избежание непредсказуемых результатов после нажатия на функциональную клавишу нажимайте клавишу СТОП.

Итак, нажимаем на функциональные клавиши и смотрим, что получается:

Клавиша	На экране появится
F1/F6	CLS - команда очистки экрана
F2/F7	LOAD - команда считывания файла с диска
F3/F8	EDIT - команда включения редактора
F4/F9	LIST - команда просмотра программы, хранящейся в памяти машины
F5/F10	RUN - команда выполнения программы

Возникает вопрос, почему на этих кнопках нанесены две надписи. Наверное, если воспользоваться кнопкой переключения регистра, то появятся новые команды. Действительно, переключите регистр (нажмите РГ и функциональную кнопку одновременно). Теперь повторите предшествующие упражнения:

Клавиша	На экране появится
F1/F6	CLS - команда очистки графического экрана
F2/F7	SAVE - команда записи файла на диск
F3/F8	AUTO - команда автоматической нумерации строк вводимой программы
F4/F9	RENUM - команда перенумерации строк программы
F5/F10	CONT - команда продолжения выполнения программы

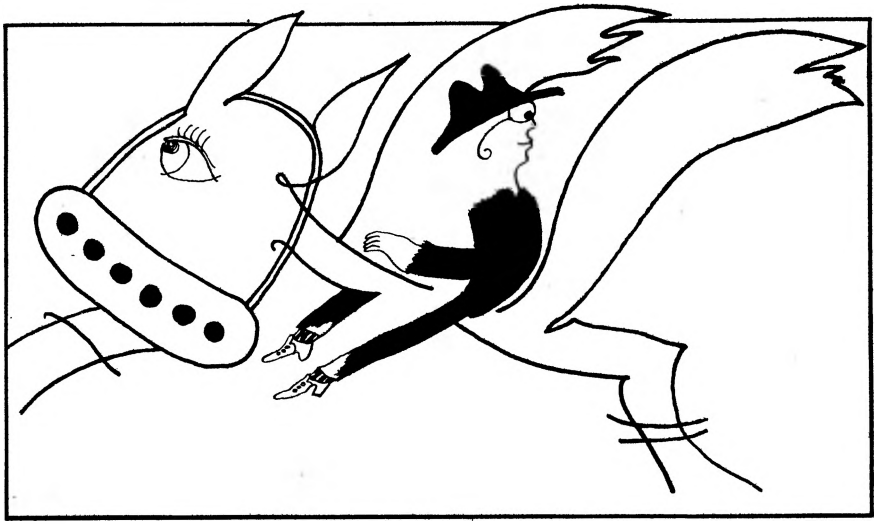
Смысл этих команд будет разъяснен в последующих разделах.

10. ПРОДОЛЖЕНИЕ ЗНАКОМСТВА С БЕЙСИКОМ

Начинаем знакомиться со словарем и грамматикой языка Бейсик

Как вы себя чувствуете? Ощущаете ли вы прилив сил? Вас переполняют мечты и великие проекты. Новые слова, идеи, открытия - все это дало вам знакомство с основами языка Бейсик. Готовы ли вы продолжать?

Да? Тогда двинемся дальше. Сначала мы познакомимся с некоторыми терминами. Вы помните, что мы говорили об инструкциях и командах. Добавим сюда еще один термин - оператор (не путайте с операндом!). Все эти термины обозначают одно и то же, но есть некоторая разница. Операторами обычно называются слова, используемые в программах, такие как уже знакомые нам PRINT и CLS.



Термин "команды" используется для обозначения тех операторов, которые используются вне программы. К ним относятся RUN, LIST, NEW и т. д. Вы, конечно, можете использовать их и в текстах ваших программ, но обычно это не делается.

Под инструкцией мы понимаем полное указание компьютеру, что ему делать. Так, PRINT "****" является примером инструкции. А вообще эти термины равноправны. Обычно их смысл ясен из контекста. Термин "оператор" является более научным.

Операторы Бейсика делятся на следующие основные классы:

1. Операторы присваивания. Эти операторы позволяют заносить какую-либо информацию под неким именем в память ЭВМ. В дальнейшем можно пользоваться этой информацией, обращаясь к ней по имени.

2. Операторы ввода-вывода. Они служат для общения с периферийными устройствами. В частности, PRINT может служить примером подобного оператора.

3. Операторы, контролирующие порядок выполнения программы. Эти операторы позволяют менять порядок выполнения различных строк программы. Например, вместо того чтобы выполнять сначала строку 10, потом 20, затем 30 и т. д., компьютер под действием этих операторов может выполнить сначала строку 20, потом 10, затем 30 и т. д.

4. Операторы спецификации. Вы знаете, что числа бывают целые, действительные, комплексные и т. д. Ясно, что, вообще говоря, это разные вещи. Поэтому машина должна всегда знать, какой тип числа вы используете. Для того чтобы объяснить это машине, и используются эти операторы.

5. Графические операторы. Эта группа операторов позволяет рисовать на экране дисплея различные картинки. Эти операторы будут подробно изучены в разделе 18.

Разберемся с перечисленными группами по порядку.

Операторы присваивания

Примером оператора присваивания является выражение A=2.5. Мы говорим компьютеру, что число 2.5 должно быть помещено в некоторую область памяти и имя этой области A. Теперь, когда нам понадобится это число, мы будем ссылаться на него как на A.

Наберите этот оператор, если вы этого еще не сделали. Вроде бы ничего не изменилось. Но вы помните, что происходило, когда вы вводили первую строку вашей первой программы? Правильно. Все, что произошло, произошло внутри компьютера. Чтобы проверить это, попросите компьютер вывести на экран содержимое области A. Для этого нужно набрать команду

PRINT A

Этим вы просто указали компьютеру распечатать содержимое области с именем A.

А зачем нам эти имена? Почему бы просто не оперировать числом 2.5 как именем области памяти? Хорошо, предположим, что 2.5 - это взносы в кассу взаимопомощи у вас на работе. Размер взносов может меняться. Если мы рассчитываем бюджет с помощью компьютера, нам необходимо всегда оперировать с текущим значением этих взносов, чему бы они ни равнялись. Для этого и служит A. Теперь, если это число равно 50, вы просто используете новый оператор присваивания

A=50

В результате компьютер заменит 2.5 на 50 в области A. Чтобы проверить, произошло ли изменение, используйте оператор PRINT:

PRINT A

Поскольку содержимое области памяти с именем A (часто его называют значением A) может изменяться, естественно назвать A переменной.

Содержимое, содержание - все эти слова наводят на мысль о коробках, банках, пакетах и т. д. Это как-то уводит в сторону от вычислительной техники. Поэтому мы будем в дальнейшем использовать более научные выражения: число A равно 50 или A=50, и т. д.

Итак, простейший оператор присваивания имеет вид

ПЕРЕМЕННАЯ=ЧИСЛО

Число помещается в память ЭВМ и машина называет это место переменной.

Сколько можно заводить переменных? Мы выбрали A просто потому, что это первое, что приходит в голову. Возможны еще сотни имен. Каждая новая переменная должна иметь новое имя. У разных переменных имена разные, например:

```
A1=100
B=21
H3=37
XX=15
```

Все эти имена - правильные. В любом случае число в правой части сохраняется в переменной левой части. Имена переменных могут начинаться с любой буквы от A до Z, второй символ может быть либо буквой, либо цифрой от 0 до 9. Единственное, чего нельзя делать, - это использовать в качестве имен переменных некоторые слова, используемые Бейсиком в своем словаре для других целей. Например, IF, GO и другие. Количество символов в имени переменной не может превышать двух. В качестве упражнения найдите правильные имена среди приведенных ниже:

AQ, 1J, MO, Q, 0E, 8Y, 2, HW, 24, 1B

Нашли? Действительно, это AQ, MO, Q и HW. Если осталось какое-то непонимание, вернитесь к предыдущему абзацу.

Так зачем же нам все-таки переменные? Сохраняя числа в памяти ЭВМ, вы можете производить любые арифметические действия с именами переменных, так же как и с собственно их значениями. Наберите следующие команды:

```
A=1
B=100
C=1357
```

Вы поместили три числа в памяти ЭВМ под именами A, B и C. Если не верите, проверьте с помощью оператора PRINT. Теперь попробуйте ввести операторы

```
PRINT A+B
PRINT B-C+A
PRINT B*C
PRINT C/B
```

Этим вы выполнили ряд арифметических операций и распечатали их результат, а ваши числа по-прежнему хранятся в памяти ЭВМ в неизменном виде.

Но это еще не все. Предположим, что вы хотите сохранить результат некоторых арифметических действий в памяти компьютера. Это просто: заведите новую переменную и присвойте ей значение результата, например:

```
D=A+B
```

Этот оператор говорит компьютеру: "Сложи A и B и скопируй результат в область памяти, занимаемую переменной D". Попробуйте теперь вывести содержимое D на экран с помощью оператора PRINT. Можете проверить, что A и B остались неизменными.

Итак, в общем случае оператор присваивания имеет вид

ПЕРЕМЕННАЯ=ВЫРАЖЕНИЕ

Компьютер вычисляет выражение и помещает результат в область памяти соответствующей переменной, стоящей в левой части.

Что же может служить выражением? Прежде всего последовательности чисел, математических символов (таких, как +, -, /, *) и переменных, например:

B+C, 1+A, 2*A/5, 1

Проверьте с помощью команды PRINT, что же получается в результате выполнения таких операций.

Пойдем дальше. Заметим, что последовательность операторов

```
A=1
A=A+1
```

имеет определенный смысл для Корвета. Попробуйте сами предсказать результат. Чему в результате будет равна переменная A? Проверьте ваши умозаключения с помощью компьютера, выполнив эту последовательность действий:

```
A=1
A=A+1
PRINT A
```

Вы, возможно, обнаружили, что одно и то же имя используется и в левой, и в правой частях оператора. Старое значение A используется для вычисления нового.

Итак, вы убедились, что Корвет, подобно электронному калькулятору, может очень быстро производить различные действия с числами: складывать, вычитать, делить и умножать. Но чем он тогда лучше калькулятора? Может быть, он быстрее вычисляет логарифмы, квадратные корни и т. д.?

Почему бы не попросить Корвет сохранить что-нибудь в своей памяти? Введите, например, оператор присваивания

N1="Корвет гораздо лучше любого калькулятора"

и нажмите ВК. Корвет вам ответит:

ОШИБОЧНЫЙ ТИП

Так вы, наверное, и думали. Ничем не лучше.

Однако не торопитесь с подобным выводом. Все дело в том, что переменные бывают не только численными. Оказывается, они могут быть еще такими, что им можно присваивать значения строк символов. Такие переменные называются строковыми. (В английской литературе они называются string.) "Корвет гораздо лучше любого калькулятора" представляет собой строковую величину. С ней, естественно, нельзя производить стандартные численные операции типа деления и умножения. И, конечно, ее нельзя спрятать в области памяти с именем N1, отведенной под численную величину.

Рассмотрим еще несколько примеров строковых величин:

"Иванов П.И. г. Москва"
"Красный, зеленый, желтый"
"1 января - день рождения дедушки"
"2*5=10"
"1 м=100 см"
">>>>---<<<<"

Ну и какой толк от этих величин? Вот если бы можно было их где-нибудь сохранить... Попробуем добавить символ $ к имени переменной N1. Наберите на экране

N1$="Корвет гораздо лучше любого калькулятора"

и нажмите ВК. Теперь посмотрите, что получилось:

PRINT N1$

Уловили идею? Переменные, имя которых оканчивается на $, являются строковыми. Строка с любыми символами, заключенная в кавычки (за исключением самих кавычек), является строковой величиной и может быть присвоена таким переменным. Рассмотрим примеры строковых переменных:

A1$, Y$, PQ$, BB$, F$, WV$

Добавляя к имени переменной $, вы создаете новую переменную:

A=100
A$="МГУ находится на Ленинских горах"

Это две разные переменные.

Необходимо сделать существенное замечание. Возможно, у вас возник вопрос, какова максимальная длина строковой величины? Попробуйте понабирать строки различной длины и присваивать их некоторым переменным. В случае очень длинной строки появится сообщение об ошибке:

НЕТ ПАМЯТИ ДЛЯ СТРОК

Что же делать? Сначала попробуем выяснить причину происходящего. Дело в том, что под строковые переменные отводится определенный объем памяти. Если вы ввели такую строковую величину, что превысили этот объем, то появляется сообщение об ошибке. Следовательно, чтобы исправить положение, необходимо попросить компьютер увеличить зарезервированную память. Для этого служит специальный оператор CLEAR. Наберите

CLEAR 1000

и нажмите ВК. Теперь попробуйте понабирать строковые величины разного размера.

Оператор CLEAR прежде всего говорит компьютеру о необходимости очистить память от всех численных и строковых переменных.

Затем компьютер резервирует в памяти под строковые переменные количество байтов, равное числу, следующему за CLEAR. Так, в нашем примере будет зарезервировано 1000 байт. Заметим, что при включении машины под строковые переменные автоматически резервируется 50 байт.

Теперь, когда мы разобрались с операторами присваивания, самое время перейти к другому классу операторов - операторам ввода-вывода.

Операторы ввода-вывода

Помните ли вы, как Корвет спросил ваше имя и не принял ответа? Наступило время научить его не только выводить на экран различные надписи, но и воспринимать ваши ответы, вводимые с клавиатуры. Вернемся к нашей старой программе.

Для начала сотрите все ваши предыдущие программы. Наберите NEW и нажмите ВК. Теперь вводите программу

```
10 CLS
20 PRINT "Здравствуйте, я ваша ЭВМ Корвет!"
30 PRINT "Как вас зовут?"
```

Наша задача состоит в том, как научить Корвет воспринять ваш ответ и сохранить его в памяти. Для начала поймем, какой тип переменной нужно использовать - численный или строковый. Поскольку ваше имя представляет собой набор букв, т. е. строку, то тип переменной, которой ваше имя присвоят, должен быть, естественно, строковым. Назовем эту переменную, если не возражаете, N$. Прекрасно. Мы зарезервировали место в памяти компьютера для вашего имени и назвали это место N$.

Теперь необходимо объяснить компьютеру, что он должен принять ваше имя, которое вы наберете на клавиатуре, и поместить его в эту область. Для этого существует специальный оператор языка Бейсик INPUT. Соответствующая строка в программе будет иметь вид

```
40 INPUT N$
```

Не пишите ваше имя прямо здесь. Помните, что строка 40 - это часть вашей программы. Она не будет выполняться, пока вы не скомандуете RUN. Но прежде этого добавим в вашу программу еще одну строку, которая поможет вам убедиться, что ваше имя воспринято правильно:

```
50 PRINT "Привет,";N$;"!" (ВК)
```

Просмотрите текст вашей программы с помощью команды LIST и найдите все ошибки. Если все в порядке, набирайте RUN и нажимайте ВК. На экране появится:

```
Здравствуйте, я ваша ЭВМ Корвет!
Как вас зовут?
?
```

Вопросительный знак в третьей строке означает, что компьютер дошел до строки с номером 40 в программе и готов принять ваше имя. Наберите его и нажмите ВК.

Теперь разберемся, как работают строки 40 и 50 программы.

Когда компьютер дошел до строки 40, он остановился и стал ждать ввода с клавиатуры вашего ответа. Как только вы нажали ВК, имя, которое вы набрали, было присвоено переменной N$. Строка 50 предписывает компьютеру вывести на экран надпись "Привет", затем содержимое переменной N$ и после этого добавить два восклицательных знака. Если вы не оставили пробела после слова "Привет" в строке 50, то оно сольется с вашим именем. Понимаете, как важен может быть пробел?

Теперь упражнение для самостоятельной работы. Научите ваш компьютер хорошим манерам. Пусть он спросит вас не только о вашем имени, но и о вашем возрасте в годах. После этого научите его сообщать вам, сколько вы прожили дней. Для этого вы должны научить компьютер воспринимать ваш возраст и хранить его в виде некоторой переменной (численной или строковой). Попробуйте достичь правильного результата.

Операторы, контролирующие порядок выполнения программы

Вы, наверное, помните упражнения по вычислению десятичного представления простых дробей. Предположим, что текст вашей программы имел вид

```
10 CLS
20 PRINT 1/2
30 PRINT 1/3
40 PRINT 1/4
50 PRINT 1/5
60 PRINT 1/6
70 PRINT 1/7
80 PRINT 1/8
90 PRINT 1/9
```

Не правда ли, повторение фактически одинаковых строк? Хотя эта программа отлично работает, вводить ее в машину было довольно утомительно. Но успокойтесь, существуют более удобные способы организации подобных программ, предлагаемые современной компьютерной наукой. Используя оператор присваивания, одну переменную и еще один оператор, можно свести эту программу к пяти строкам. В этом случае она будет иметь вид

```
10 CLS
20 A=1
30 PRINT 1/A
40 A=A+1
50 GOTO 30
```

В ходе обсуждения языков высокого уровня мы отмечали, что основной целью их создания было научить машину понимать человеческий язык. Попробуем прочесть эту программу по строкам просто так, не имея в виду Бейсик:

10: Очистить экран (CLS - от англ. CLear Screen)
20: Спрятать в области A число 1
30: Напечатать число, равное 1/A
40: Добавить единицу к числу в области A, и занести новое число в эту область
50: Перейти к выполнению строки 30

Попробуйте запустить эту программу. Предварительно сотрите старую программу. Наберите программу с 10-й до 50-й строки и просмотрите результат с помощью команды LIST. Если вы обнаружили ошибки, исправьте их. Наконец, с помощью команды RUN запустите программу.

Вот это да! На экране дисплея замелькали десятичные представления простых дробей. Они выводятся с такой скоростью, что трудно уследить за ними. Остановите выполнение программы. Для этого нажмите STOP. Обратите внимание на появившееся сообщение. Машина сказала вам, на какой строке программа была остановлена. Ниже появился промпт Бейсика.

Возможно, у вас возник вопрос, почему компьютер сам не остановился. Чтобы ответить на него, внимательно просмотрите текст программы, особенно строки 30, 40 и 50. Вы обнаружите, что в вашей программе вы нигде не говорите компьютеру: "Остановись". Поэтому он будет бесконечно выполнять строки с номерами 30, 40 и 50. Подобные блоки программ называются циклами.

Прежде чем дальше знакомиться с операторами Бейсика, вернемся к хорошо известному вам оператору PRINT. Вы успели заметить, что печать только одного числа на строке приводит к тому, что курсор быстро достигает низа экрана и все, что было выведено на него, начинает бежать вверх так быстро, что невозможно прочитать. Как же сделать так, чтобы использовать пространство экрана более рационально и, соответственно, уменьшить скорость перемещения его содержимого вверх? Догадались или нет? Помните, мы говорили, что для вывода сообщений на экран одного за другим существуют специальные символы - запятая и точка с запятой. Попробуйте использовать точку с запятой в конце строки 30:

```
30 PRINT 1/A;
```

Внесите это изменение в строку 30 и пустите программу. Теперь компьютер будет выводить на одну строку экрана столько символов, сколько поместится. И только после заполнения каждой строки будет переходить на следующую строку. Однако то, что вы видите на экране, не очень красиво. Действительно, числа разбросаны по экрану в беспорядке и понять, что где, очень трудно. Но вы помните, что если использовать вместо точки с запятой запятую, вывод на экран будет производиться только в определенные зоны, каждая длиной 16 символов. Сначала попробуйте ввести строку

PRINT 1/2,1/3,1/4,1/5,1/6,1/7,1/8,1/9,1/10

и затем -

PRINT "Зона 1","Зона 2","Зона 3","Зона 4"

Посмотрите, что получилось. Теперь исправьте свою программу, заменив в строке 30 точку с запятой на запятую:

30 PRINT 1/A,

Запустив программу, вы видите, что Корвет способен красиво располагать текст на экране не хуже машинистки. Это не единственный способ организации вывода сообщений на экран. С другими способами мы познакомимся позднее.

Теперь вернемся к программам. Как заставить компьютер делать паузу при выводе информации на экран, чтобы мы могли успеть прочесть ее? Вы знаете один способ - нажать клавишу СТОП. Но это прервет выполнение программы, и вам пока не известно, как продолжить его. Если вы после остановки программы наберете RUN, то программа начнет выполняться сначала.

Займемся воспоминаниями. Читали ли вы раздел 6? Если нет, или читали, но забыли, напомним, что для остановки вывода на экран служит управляющий символ CTRL-S. Запустите вашу программу и, нажав одновременно клавиши CTRL и S, посмотрите, что получится. Вывод на экран прекратится, но и не появится промпт Бейсика. Теперь для продолжения работы достаточно еще раз нажать одновременно CTRL и S. Компьютер опять начнет выводить информацию на экран. Обратите внимание, что эффект повторного запуска достигается также нажатием любой клавиши на клавиатуре.

GOTO - первый оператор контроля выполнения программы, который вы узнали. Нормальным для Корвета является последовательное выполнение программы - строка за строкой. Однако строка 50 говорит ему "Забудь нормальный порядок и выполняй строку 30". Оператор GOTO называется оператором безусловного ветвления. Когда компьютер находит его, он без лишних слов и долгих раздумий переходит к выполнению строки с номером, указанным после GOTO.

У вас, конечно, возник вопрос, есть ли способ останавливать программу без помощи клавиш СТОП или CTRL-S? Такой способ существует. Убедитесь, как всегда, что компьютер готов воспринимать

ваши команды (на экране имеется промпт OK и под ним белый прямоугольник курсора; если нет, нажмите СТОП). Добавьте теперь к вашей программе строку 45:

```
45 IF A=10 THEN END
```

Просмотрите программу с помощью команды LIST. Она должна иметь вид

```
10 CLS
20 A=1
30 PRINT 1/A,
40 A=A+1
45 IF A=10 THEN END
50 GOTO 30
```

Устраните ошибки переписыванием соответствующих строк и запустите программу. Заметьте, как удобно нумеровать строки 10, 20, 30 и т. д. Мы легко смогли вставить новую строку с номером 45. А если бы мы нумеровали их 1, 2, 3 и т. д.? Страшно подумать.

Займемся теперь строкой 45. Для начала переведем ее с английского на русский язык: "Если A равно 10, то окончить работу". Что компьютер и сделал.

А что делать, если A не равно 10? Мы этого не сказали. Однако компьютер в этом случае сам знает, что делать. Он будет выполнять следующую строку программы.

Оператор IF - оператор условного ветвления. Его основной формат записи:

```
IF УСЛОВИЕ THEN ЧТО ДЕЛАТЬ
```

Слова IF и THEN являются словами из словаря Бейсика. Если условие выполнено, компьютер будет выполнять то, что указано после THEN. Если нет, он перейдет к выполнению следующей строки. Обратите внимание, что оператор A=10 в строке 45 не является оператором присваивания. Переменная A не примет значения 10. Вместо этого компьютер просто проверит, равно ли A десяти, т. е. хранится ли в области памяти A число 10.

Теперь оператор END. О нем речь будет идти позднее, а сейчас просто запомните, что он позволяет простейшим способом окончить программу.

В качестве упражнения модифицируйте программу так, чтобы она вывела на экран десятичные представления следующих дробей 1/11, 1/12, ..., 1/19.

Уничтожьте ваши упражнения при помощи команды NEW и введите программу

```
10 CLS
20 PRINT "Привет, я твой Корвет - собеседник!"
30 PRINT "Как тебя зовут?"
40 INPUT N$
```

```
50 PRINT "Что ты делаешь сегодня вечером,";N$;"?"
60 INPUT N$
70 PRINT "Здорово... Скажи что-нибудь еще."
80 GOTO 60
```

Просмотрите программу, исправьте ошибки и запустите ее. Теперь побеседуйте с компьютером (только не сообщайте ему своих мыслей длиннее, чем в 50 букв и не ставьте знаки препинания). Как только вы наберете на экране очередную мысль, нажимайте ВК. Когда ваш запас мыслей истощится, нажмите клавишу СТОП и прервите бесконечный цикл. Иначе компьютер выпытает у вас все.

Попробуйте исправить программу так, чтобы как только вы ответите компьютеру "ХВАТИТ", он остановился. Придумали? Действительно, для этого можно использовать IF:

```
IF N$="ХВАТИТ" THEN END
```

Теперь подумайте, какой номер должен быть у этой строки. Ну, конечно, 65.

Как вы узнали, одним из наиболее часто используемых операторов (особенно в диалоговых играх) является оператор ввода чего-либо с клавиатуры. Этот оператор выводит на экран знак вопроса (его можно рассматривать как промпт), и после этого вы набираете ответ. Для примера введитѐ новую программу

```
10 PRINT "Как тебя зовут?";
20 INPUT N$
30 PRINT "Привет";N$;"!"
```

Запустите ее, чтобы потом можно было сравнить результат.

Попробуем теперь объединить строки 10 и 20. Для этого сотрем строку 10 и заменим строку 20 на

```
20 INPUT "Как тебя зовут?";N$
```

Запустите этот вариант. Сравните его с предыдущим примером. Можно использовать подобный способ для ввода нескольких переменных, например:

```
20 INPUT "Имя и возраст";N$,AG
30 PRINT N$;" ";AG;"лет".
```

Вы видите, что таким образом можно задавать вид промпта (любой вместо вопросительного знака) при общении с компьютером.

Итак, мы установили, что общим видом оператора INPUT является

```
INPUT "ПРОМПТ"; СПИСОК ПЕРЕМЕННЫХ
```

Переменных может быть сколько угодно. Они должны разделяться запятыми. Промпт должен заключаться в кавычки и после него обязательно должна быть точка с запятой.

Рассмотрим еще несколько полезных способов сравнения чисел. В любом IF...THEN-операторе мы должны указать проверяемое условие. Мы знаем, что можно проверить, равны ли два числа. Однако, может быть, мы интересуемся, какое число больше или меньше. Как поступить в этом случае? Помните программу о простых дробях:

```
10 CLS
20 A=1
30 PRINT 1/A
40 A=A+1
50 IF A=10 THEN END
60 GOTO 30
```

Предположим, что мы хотим остановить программу, когда 1/A будет меньше 0.01. Другими словами, мы не хотим выводить на экран дроби меньше 0.01. В этом случае нам нужно сказать компьютеру: "Если 1/A меньше чем 0.01, то конец". Для этой цели служат символы < и >. В качестве упражнения исправьте нашу программу.

Приведем полный список подобных символов:

< - меньше	<= (или =<) - меньше или равно
= - равно	>= (или =>) - больше или равно
> - больше	<> (или ><) - не равно

Эти символы называются операторами отношения чисел. Они могут быть использованы в любых условных операторах.

Операторы спецификации

До сих пор мы пользовались числами, не думая о том, как компьютер их воспринимает. В нашем примере, касающемся простых дробей, мы не говорили компьютеру, сколько десятичных знаков нам нужно или где располагать десятичную точку. Для того чтобы понять суть проблемы, попробуем поделить на бумаге в столбик 1700 на 11. Попробуйте проделать эту процедуру самостоятельно. Вы получите бесконечную десятичную дробь 154.545454(54). Вам предстоит решить, каким количеством десятичных знаков после запятой вы ограничитесь. Вы можете оставить один знак, округлив результат: 154.5. Можете оставить четыре десятичных знака: 154.5454. А может быть десятичные части вам вообще не интересны и вы оставите просто 154. Теперь поинтересуйтесь у Корвета, что он думает по этому поводу. Для этого наберите команду

```
PRINT 1700/11
```

Компьютер вам ответит:

```
154.545
```

В его ответе фигурирует всего шесть цифр. Откуда Корвет узнал, в каком месте прекратить деление? Или, другими словами, откуда он узнал, с какой точностью мы хотим получить ответ?

Корвет изначально предполагает, что вы являетесь человеком высокой души, а не сухарем, скрупулезно следящим за точностью самого последнего десятичного знака, даже если потребовалась бы вся жизнь для вывода его на бумагу. С другой стороны, Корвет, естественно, не считает вас легкомысленным субъектом, который безумно счастлив, увидев хоть какой-нибудь результат, даже если он состоит только из двух десятичных знаков.

В целях достижения средней степени аккуратности компьютер использует то, что в научной литературе называется одинарной точностью. Одинарная точность означает семь достоверных десятичных знаков числа. При этом не важно, где эти знаки стоят - до или после десятичной точки. В математической литературе для них существует специальный термин - значащие цифры.

Рассмотрим примеры чисел одинарной точности:

1, 1.234567, 3.14159, -25000000, 0.000123

Все эти числа имеют не больше семи значащих цифр. Теперь попробуем ввести эти числа в компьютер. Для этого напишем программу

```
10 INPUT "Введите число";X
20 PRINT X;"Хранится в памяти компьютера"
30 PRINT
40 GOTO 10
```

Когда программа начнет вас спрашивать, вводите числа, перечисленные выше. Обратите внимание, что произошло с числом 1.234567. Компьютер округлил его до шести знаков. Это произошло потому, что оператор PRINT так устроен. Он всегда округляет числа с одинарной точностью до шести знаков. При этом само число, хранящееся в памяти компьютера, содержит семь значащих цифр.

Попробуйте дать компьютеру число -25000000. Компьютер ответит вам:

-2.5 E+07 Хранится в памяти компьютера

Подобный результат получится, если ввести число 0.000123:

1.23 E-06 Хранится в памяти компьютера

Здесь вы сталкиваетесь с тем, какие цифры являются значащими. Компьютер очень экономен по своей натуре. Поэтому он заменяет длинные последовательности нулей сообщением о порядке числа. Порядок означает, что число нужно умножить на 10 в степени, равной порядку, т. е. сдвинуть десятичную точку влево или вправо (это зависит от знака порядка) на необходимое число позиций, заполняя свободные нулями. Попробуйте вводить числа в компьютер, используя этот формат.

Запомните, что десятичные числа, большие 1000000 и меньшие 0.01, по команде PRINT выводятся на экран в следующем формате:

X.YYYYY E ZZ

В этом же формате выводятся отрицательные числа, меньшие -999999 и большие -0.01. Старшая значащая цифра (самая левая) пишется слева от десятичной точки. Остальная часть числа YYYYY выводится справа от десятичной точки. Следующее за этим обозначение E ZZ заменяет собой выражение 10 в степени ZZ. Если число находится в интервале от -1 до +1, то ZZ отрицательно. Иными словами, запись X.YYYYY E ZZ означает X.YYYYY*(10 в степени ZZ).

Потренируйтесь с разными числами, записывая их в этом формате. Для контроля попросите компьютер вывести их на экран. При этом помните, что оператор PRINT оставляет только шесть значащих цифр.

Иногда, однако, оказывается, что представлять числа с десятыми долями не очень удобно. Действительно, непривычно звучал бы, например, отчет по статистике заболеваний в вашем институте, в котором было бы сказано, что в течение года в среднем ежемесячно болело гриппом 2.4 человека. Естественно, в таком случае появляется желание округлить число 2.4 до целого. Корвет сделает это округление за вас с помощью символа %. Как же использовать его? Помните, как мы использовали символ $? Он превращал обыкновенную переменную в строковую. Чтобы вы лучше могли представить себе смысл символа %, попробуйте завести новые переменные в памяти компьютера, для чего введите в машину следующие команды:

```
A%=1.334
B%=100.001
C%=-1.5
D%=-300.1
PRINT "Числа, которые хранятся в памяти";A%,B%,C%,D%
```

Вы увидите, что вместо 1.334 в памяти машины находится 1, вместо 100.001 число 100, вместо -1.5 число -2 и вместо -300.1 число -301. Что происходит с положительными числами, понятно. Вся дробная часть просто отбрасывается и остается только целая часть. Что происходит с отрицательными числами? Непонятно. Может быть компьютер сломался?

Добавление символа % к имени переменной изменяет ее тип из действительного числа с одинарной точностью в специальный тип, называемый целым числом. Целое число может принимать значения от -32768 до 32767. Когда вы присваиваете целой переменной нецелое значение, дробная часть отбрасывается так, что переменной присваивается максимальное целое число, не превышающее исходного. Для положительных чисел это просто их целая часть, для отрицательных это меньшее целое число. A%, B%, C% , D% являются целыми

переменными. Они абсолютно ничего общего не имеют с действительными переменными A, B, C , D.

Теперь познакомимся с тем, как написать программу для вывода на экран некоторой заданной последовательности чисел, скажем 1.2345, -1.01. Известный вам способ - это следующая последовательность операторов:

```
X=1.2345
PRINT X
X=-1.01
PRINT X
```

Попробуем включить все необходимые числа в один оператор, называемый DATA (данные). Чтобы считать эти числа, понадобится другой оператор READ (читать). Введите программу

```
10 READ X
20 PRINT X
30 DATA 1.2345
```

Запустите ее. Оператор READ прочтет величину, следующую за оператором DATA, и присвоит ее переменной X. Полезно запомнить, что оператор READ найдет последующий оператор DATA вне зависимости от того места в программе, где он расположен.

Модифицируйте вашу программу, добавив строку

```
27 GOTO 10
```

Отредактируйте строку 30 так, чтобы она имела вид

```
DATA 1.2345,2.12,3.23,4.34
```

Запустите программу. Вы увидите, что все величины в операторе DATA будут выведены на экран, после чего появится сообщение об ошибке:

```
ВНЕ ДАННЫХ 10
```

Если внимательно проследить за порядком выполнения программы, можно понять, что последнее сообщение на экране означает, что числа, указанные в операторе DATA, кончились, а вы тем не менее пытаетесь их считывать. Чтобы предотвратить это, обычно вводится специальное число, не повторяющееся нигде в операторе DATA, которое ставится в конце последовательности ваших чисел. При этом вы добавляете специальную строку, в которой следите, дочитали ли вы до конца. В нашем случае удобно выбрать это число равным нулю. Добавим контролирующую строку в программу:

```
15 IF X=0 THEN PRINT "КОНЕЦ ДАННЫХ":END
```

Исправьте строку 30 так, чтобы она имела вид

```
30 DATA 1.234567,2.12,3.23,4.34,0
```

Теперь запустите программу. Строка 15 каждый раз тестирует очередное считываемое число. Если оно не равно нулю, число выводится на экран и считывается новое.

Подчеркнем, что использование оператора DATA позволяет сохранять числа прямо в вашей программе. Они все должны разделяться запятой. Вы можете использовать любое количество операторов DATA в вашей программе. Компьютер будет рассматривать их, как если бы все они были перечислены в едином списке.

Содержимое оператора DATA может быть прочитано только оператором READ. Первой считывается самая первая величина в самом первом операторе DATA в программе. За один раз считывается одна величина. И каждая величина считывается только один раз, после чего считываются остальные по порядку. Оператор READ может содержать несколько переменных. В этом случае этим переменным присваиваются соответствующие величины в операторе DATA. Если их там окажется меньше, чем переменных в операторе READ, то появится сообщение об ошибке:

ВНЕ ДАННЫХ В ...

Предположим теперь, что после прочтения оператором READ нескольких величин из содержимого оператора DATA мы все-таки хотим вернуться к уже прочитанным величинам и использовать их еще раз. Такая возможность в Бейсике предусмотрена, и реализуется она с помощью оператора RESTORE. Введите программу

```
10 CLS
20 READ A$
30 DATA "Ваше имя и фамилия"
40 PRINT A$;
```

С помощью команды RUN запустите ее. Машина по команде READ прочла ваши имя и фамилию из содержимого оператора DATA и напечатала их на экране по команде строки 40. Предположим теперь, что вы хотите увидеть ту же надпись на экране, но напечатанную не один раз. Для этого, казалось бы, можно воспользоваться оператором ветвления GOTO, который после строки 40 возвращал бы машину к многократному прочтению данных. Попробуйте сделать это, добавив к вашей программе строку

```
50 GOTO 20
```

Запустите программу. Видите, что произошло? Машина, как и раньше, напечатала ваши имя и фамилию один раз, после чего сообщила вам:

ВНЕ ДАННЫХ В 20

Давайте разберемся, в чем причина такой реакции машины. Когда оператор GOTO в строке 50 вернул машину к строке 20, машина по команде READ хотела второй раз прочесть содержимое DATA, но

операторы READ и DATA устроены так, что при следующем прочтении машина хочет прочесть следующие данные. Следующих же данных, естественно, обнаружено не было, поскольку в программе их нет, о чем машина вас и проинформировала своим сообщением.

Как же заставить машину в очередной раз прочесть уже прочитанное содержимое оператора DATA? Для этого существует оператор RESTORE (восстановление). По этой команде машина восстанавливает все уже прочитанные данные и при следующем прочтении воспринимает содержимое оператора DATA так, как будто эти данные читаются впервые. Вставьте в вашу программу строку

```
45 RESTORE
```

Запустите ее. Теперь вы видите, что машина печатает ваши имя и фамилию безостановочно. Для того чтобы прекратить этот процесс, надо нажать клавишу СТОП.

Надеемся, что вы поняли смысл команды RESTORE. Эта команда может быть особенно полезна в диалоговых программах, когда вы предусматриваете возможность неоднократного прочтения одних и тех же данных.

Вернемся теперь к вопросу о точности вычислений. На первый взгляд кажется, что правильный результат с точностью до семи значащих цифр вполне достаточен. В наших повседневных хлопотах это действительно так. Вряд ли кто-либо из нас сталкивался с необходимостью достоверного знания восьмой значащей цифры при подсчетах семейного бюджета, решении школьных задач, подсчете необходимого при поездке из Москвы во Владивосток количества литров бензина и т. п. Для проведения только таких расчетов вам, конечно же, семи значащих цифр более чем достаточно, и если вы не собираетесь вести на Корвете научные расчеты, дальнейшие сведения о максимальной точности вычислений, на которую способна машина, можете прочесть разве что из любопытства.

Для тех же, кто собирается эксплуатировать Корвет для решения своих научных задач (наш опыт показывает, что это вполне разумное использование способностей машины), напомним, что Корвет может делать несколько сотен тысяч сложений или вычитаний в секунду. Поэтому, написав соответствующую программу, вы можете использовать машину тогда, когда вам нужно провести гораздо больше, чем сотни тысяч, элементарных арифметических операций.

А теперь представьте себе, что при каждом вычислении вы ошибаетесь всего лишь в восьмой значащей цифре. Предположите, что в ходе вычислений вы проводите арифметические действия над числами, отличающимися лишь в восьмом знаке, и расчет ведете недолго, всего лишь одну минуту. Тогда общее число элементарных арифметических операций, которые выполнит за это время машина, приблизительно равно

(100000 операций/с) * 3600 с = 360000000 операций

Ясно, что при проведении такого количества вычислений появляется вероятность получить ошибку и в первой значащей цифре. Поэтому максимальная точность вычислений, на которую способен Корвет при расчетах на языке Бейсик, чрезвычайно важна для проведения научных расчетов.

Как же попросить Корвет считать с большой точностью? Для этого существует символ #. Поставленный после имени переменной, он переводит ее из одинарной точности в двойную. Переменная двойной точности может содержать 17 значащих цифр. Числа, имеющие от 8 до 17 значащих цифр, также рассматриваются как числа с двойной точностью. Оператор PRINT выводит на экран только 16 значащих цифр, 17-я используется для округления.

Чтобы обучиться работе с числами разного типа, напишите программу

```
10 INPUT "Введите число";X#
20 X=X#
30 IF X#<-32768 OR X#>32768 THEN X%=0;GOTO 50
40 X%=X#
50 PRINT "Целое";"одинарная точность","двойная точность"
60 PRINT X%,X," ",X#
```

Попробуйте вводить с ее помощью различные числа. Обратите внимание, что если вы вводите очень большое число, скажем, 10 в 16-й степени, т. е. 1 и 16 нулей, число, соответствующее двойной точности, выводится в формате, подобном формату одинарной точности с той разницей, что вместо указателя степени E стоит D. D означает двойную точность. В остальном D-формат похож на E. Разница только в количестве выводимых значащих цифр. В D-формате их может быть 16.

Попробуйте написать программу, касающуюся вычисления простых дробей. Проблема состоит в том, чтобы вычислить 1/3 с двойной точностью и после этого умножить результат на 3. Ответ будет ошеломляющим: 1.000000029802322.

Да, компьютер преподносит сюрприз за сюрпризом. В чем же дело? Когда компьютер выполняет оператор A#=1/3, он сначала делит 1 на 3 и потом присваивает результат переменной A#. Но вычисление дроби 1/3 производится с одинарной точностью. Таким образом, результат вычисления 1/3 будет содержать только семь правильных значащих цифр. Когда это значение помещается на место, зарезервированное под A#, остальные значащие цифры (10 штук) заполняются чем угодно - своего рода цифровым мусором. Что же делать? Сотрите старую программу и напишите новую:

```
10 A#=1/3#
20 B#=A#*3
30 PRINT A#;"*3=";B#
```

Запустите ее, и вы увидите результат: .333333333333 * 3 = 1

Так получилось потому, что символ # после 1/3 указывает компьютеру, что 1/3 нужно вычислять с двойной точностью.

Надеемся, что вы поняли, как выбирать точность вычислений. Поупражняйтесь самостоятельно, а когда все окончательно прояснится, переходите к следующему разделу.

11. РЕДАКТИРОВАНИЕ ПРОГРАММ

О превращении процесса ввода и отладки программы в приятную и неутомительную процедуру

Альтернативное обозначение оператора PRINT

Сначала примем к сведению, что вместо команды PRINT можно с успехом использовать вопросительный знак. Попробуйте выполнить следующие упражнения:

```
? "Это сокращенное обозначение оператора PRINT"
? 1234/100
```

Можно использовать это сокращение и в программах, например:

```
10 ? "Как прервать бесконечную программу?"
20 GOTO 10
```

Когда вы будете просматривать текст программы по команде LIST, вы увидите, что Бейсик заменил знак "?" на PRINT. Запустите эту программу. Прервите ее действия, нажав на клавишу СТОП. Теперь выведите на экран текст программы. Обратите внимание, что произошло с сокращением. Также заметьте, что внутри текста в строке 10 символ ? не был ошибочно заменен на PRINT.

Многооператорные строки

Следующая возможность касается многооператорных строк. Что это такое? Для того чтобы выяснить это, рассмотрим программу

```
100 CLS
200 INPUT "Введите число, любое число";N
300 PRINT
400 PRINT "Да... Очень интересно..."
```

Номера строк в ней не играют никакой роли. Мы просто задали последовательность выполнения операторов, а на номера операторов нигде не ссылаемся. Компьютер в этом случае выполнит программу от первой до последней строки. Если номера строк не важны, зачем их использовать? Конечно, Бейсик требует, чтобы каждая строка имела номер. Но зачем давать номер каждому оператору? Вы можете

помещать на одной строке программы несколько операторов, разделяя их двоеточием:

```
100 CLS:INPUT "Введите число, любое число";N
200 PRINT:PRINT "Да... Очень интересно..."
```

Вы даже можете упаковать все четыре оператора в одну строку программы:

```
100 CLS:INPUT "Введите число, любое число";N:PRINT:
PRINT "Да... Очень интересно..."
```

Обратите внимание, что строка программы может занимать несколько строк на экране. Когда при наборе таких длинных строк курсор доходит до конца строки экрана, он автоматически перепрыгивает в начало новой строки.

Для чего же все-таки используют многооператорные строки? Прежде всего для экономии памяти. В результате вы можете вводить более сложные программы. Воспринимайте номера строк просто как заголовки. Они не влияют на эффективность выполнения программ. Другая причина - это возможность объединения простых связанных друг с другом операторов. Например, присваивание значений переменным, очистка экрана и вывод на него сообщений и т. д.

Сколько операторов можно поместить на одной строке программы? Их количество не ограничено. Ограничено только общее число любых символов (букв, специальных символов типа !, $ и т. д., цифр и пробелов) на вашей строке, т. е. ограничена длина строки. Одна строка программы на Бейсике может содержать до 255 таких символов. Вы можете ввести до 250 символов прямо с клавиатуры. Для ввода 5 остальных необходимо использовать метод, описанный ниже.

Какое количество операторов разумно использовать в одной программной строке? Вы, должно быть, заметили, что последняя ваша программа, содержащая четыре оператора в одной строке, получилась плохо читаемой. Действительно, в ней не так просто разобраться. Это одна из причин для того, чтобы не упаковывать в одну строку слишком много операторов.

Следующее ограничение - трудности в использовании операторов ветвления типа GOTO. Вы не можете сказать компьютеру, что нужно перейти в середину строки. Рассмотрим программу

```
100 INPUT "ДА или НЕТ";R$
200 IF R$="ДА" THEN 500
300 IF R$="НЕТ" THEN 600
400 GOTO 100
500 PRINT "Это положительно!"
550 GOTO 100
600 PRINT "Почему отрицательно?"
650 GOTO 100
```

В этом примере номера строк важны для описания последовательности выполнения операторов. Если вы ответили ДА, управление передается на строку с номером 500, потом 550 и затем 100. Если вы ответили НЕТ, управление передается на строку с номером 600, потом 650 и затем 100. После чего все повторяется сначала.

Теперь посмотрим, что получится, если мы объединим строки 200 и 300 в одну:

```
200 IF R$="ДА" THEN 500:IF R$="НЕТ" THEN 600
```

В этом случае, если вы отвечаете ДА, то управление переходит на строку с номером 500. Если вы отвечаете НЕТ, то выполняется следующая по порядку строка. Вы видите, что операторы IF R$="НЕТ" и THEN 600 вообще никогда не выполняются.

В качестве упражнения сократите количество строк в рассмотренной программе до пяти, используя многооператорные строки. При этом будьте осторожны. Обратите внимание на то, чтобы все операторы в вашей программе выполнялись.

Автоматическая нумерация строк программы

Следующим упрощением процесса ввода программ является возможность автоматической нумерации строк. Для этого сначала сотрите старую программу и введите команду

```
AUTO
```

Компьютер в ответ выведет на экран номер текущей строки (и изображение курсора):

```
10
```

Теперь наберите команду CLS и нажмите ВК. Компьютер напечатает номер следующей строки:

```
20
```

Наберите

```
PRINT "Это будет короткая программа"
```

и нажмите ВК. Появится следующий номер:

```
30
```

Если вы хотите прервать процесс ввода новых строк программы, нажмите клавишу СТОП. Появится промпт Бейсика OK. Теперь просмотрите содержимое программы. Почему компьютер начал нумеровать строки с номера 10 и увеличивал следующие номера на 10? Эта величина устанавливается по умолчанию, если вы не указали компьютеру другие параметры. Вы можете указать, каким должен быть первый номер и приращение номера между строками. Таким образом, команда AUTO имеет формат

AUTO НОМЕР ПЕРВОЙ СТРОКИ, ПРИРАЩЕНИЕ

Например, AUTO 100, 50 приведет к тому, что строки программы будут иметь номера 100, 150, 200, 250 и т. д.

При автоматической нумерации строк компьютер сам устанавливает номер строки. Все, что делаете вы, - вводите соответствующие операторы. Каждый раз, когда вы нажимаете ВК, эти операторы сохраняются в памяти машины и начинается ввод следующей строки. Для прекращения ввода используется клавиша СТОП. При этом текущая строка не сохранится. Если в процессе ввода компьютер обнаружит уже использованную в программе строку, он выведет символ * сразу же после номера строки, например:

30*

Символом * компьютер вам напоминает, что строка 30 уже есть и если вы сейчас введете новую инструкцию, старое содержание строки 30 пропадет. Если вы не хотите ее стирать, нажмите СТОП.

Просмотр текста программы на экране дисплея

Если ваша программа достаточно длинная и вся не умещается на экране дисплея, попытка просмотреть ее с помощью команды LIST окажется сложной задачей. Строки начнут быстро мелькать, и вы ничего не успеете прочитать. Вы, конечно, можете задержать вывод текста вашей программы, используя команду CTRL-S, или прекратить его вообще, нажав клавишу СТОП. Но это не лучший выход. Можно указать компьютеру, какие строки вы хотите посмотреть, с помощью одной из следующих команд:

Команда	Выполняемое действие
LIST 100	вывод на экран строки 100
LIST 100-110	вывод строк 100 - 110
LIST -190	вывод первых 190 строк
LIST 200-	вывод со строки 200 до конца программы
LIST	вывод последней введенной или выполненной строки

Общий вид команды LIST следующий:

LIST ПЕРВАЯ СТРОКА - ПОСЛЕДНЯЯ СТРОКА

Стирание строк программы

Теперь займемся стиранием ненужных строк. Вы уже знаете один из способов: просто набираете номер ненужной строки и нажимаете ВК. Если вы имеете в программе строки с номерами 100, 110, ..., 210 и хотите все их стереть, вы можете вводить последовательно

100
110
...
210

Существует более простой способ. Наберите команду

DELETE 100-210

Корвет автоматически сотрет строки программы с 100 по 210. Формат команды DELETE (удалить, стереть) аналогичен формату команды LIST:

DELETE ПЕРВАЯ СТРОКА - ПОСЛЕДНЯЯ СТРОКА

Комментирование текста программы

Предположим, что вы написали длинную программу с множеством переменных и сложной логикой. В результате вы создали проблему для себя. В такой программе, если вы попытаетесь по прошествии некоторого времени изменить ее, очень трудно разобраться. Вы уже успели забыть логику программы. Помочь в этом деле могут комментарии. Комментариями называются сообщения, вводимые в разных частях программы и поясняющие, что в этом месте вы хотели сделать. Комментарии не обрабатываются интерпретатором. Он просто не обращает на них внимания. Для того чтобы отделить комментарий от остальной программы, используется оператор REM (от англ. REMark - ремарка, замечание).

Попробуйте ввести команду

REM PRINT "Это комментарий"

Ничего не произошло! Это потому, что следующее за командой REM сообщение нужно игнорировать. Попробуйте использовать комментарий в программе:

10 REM... Это просто комментарий.

REM очень полезен, если он идет за оператором Бейсика, например:

20 Y=Y+B:REM увеличение Y на величину B

Существует сокращение для оператора REM - это апостроф. С его использованием команда выглядит следующим образом:

30 X=X+1 'Добавление 1 к счетчику

Обратите внимание на отсутствие двоеточия. При использовании сокращенной нотации REM оно не нужно.

Запомните, что нельзя использовать оператор REM в середине многооператорной строки, поскольку компьютер игнорирует все, что идет следом за REM.

Редактирование текста программы

Теперь займемся редактированием программ. Редактор позволяет вам изменять содержание любой строки в программе быстро и эффективно. Вам больше не потребуется вводить заново всю строку, чтобы исправить одну букву.

Прежде чем переходить к редактированию, приготовьтесь к неожиданностям!

Многие кнопки приобретут некоторые специальные функции. Как только вы их нажмете, компьютер сразу начнет что-то делать, не дожидаясь, пока вы нажмете клавишу ВК. Поэтому, чтобы избежать неприятностей, сделайте сначала все так, как мы вам изложим ниже. Кроме того, помните, что когда вы находитесь в режиме редактирования, все команды редактора будут правильно вводиться только при переключении клавиатуры в режим печати латинского шрифта.

Сначала заметим, что режим редактирования включается автоматически при обнаружении компьютером синтаксической ошибки.

Чтобы узнать, как компьютер работает в режиме редактирования, необходимо заставить его что-нибудь отредактировать. Поскольку редактировать правильно написанные строки неинтересно, введем программу со специально сделанными в ней ошибками. Наберите следующую строку в точности как она написана:

```
100 REM Уччимся ррработать с реедоктрром
```

Существуют два способа перевести компьютер в режим редактирования. Первый - запустить программу, и Бейсик немедленно выдаст сообщение об ошибке и запустит редактор. Однако есть и другой способ. Для этого служит специальная команда EDIT (редактировать), которая указывает компьютеру, что нужно начать редактирование. Наберите

```
EDIT 100
```

Этим вы сообщите компьютеру, что вы желаете редактировать строку с номером 100. При выполнении этой команды на экране появится номер редактируемой строки и изображение курсора. Поскольку в ходе изложения в данном разделе необходимо точно указывать положение курсора, мы будем обозначать его символом подчеркивания, хотя реально на экране дисплея он имеет вид белого прямоугольника. Итак, на экране дисплея вы видите

```
100 _
```

Теперь вы находитесь в режиме редактирования.

Запомним, что форматом команды EDIT является

```
EDIT НОМЕР СТРОКИ
```

Если при вводе этой команды вместо номера строки поставить точку, то начнется редактирование строки, которая была последней набрана или выполнена. В процессе редактирования Бейсик изображает

на экране рабочую копию нужной строки, показывая все изменения, которые вы в ней делаете. После того как вы нажмете ВК, эта копия заносится в память на место оригинала.

Сначала научимся передвигать курсор и таким образом просматривать содержимое строки. В первый момент курсор установлен в начале строки на месте самого первого символа. Для того чтобы подвинуть курсор на одну позицию вправо, необходимо нажать на клавишу пробела. Каждый раз, когда вы на нее нажимаете, изображается очередной символ и курсор передвигается на одну позицию вправо. При этом вы ничего не исправляете, а только просматриваете содержимое строки. Нажимая пробел раз за разом, вы постепенно достигнете конца строки. После этого нажатие на пробел не вызовет никакого эффекта. Курсор достиг конца строки, и двигаться дальше ему некуда. В результате на экране появится текст:

```
100 REM Уччимся ррработать с реедоктрром
```

Теперь, чтобы вернуться в начало строки, нажмите L. Если перед этим машина была переключена в режим работы с латинским шрифтом, то на экране появится надпись:

```
100 _
```

Команда L является сокращением команды LIST. Вы сказали компьютеру: "Выведи на экран всю строку". После этого будет заведена новая копия вашей строки и снова начнется процесс редактирования. Нажимите L еще раз и посмотрите, что происходит.

Теперь займемся редактированием ошибок. Сначала сотрем лишнюю букву ч в слове "уччимся". Для этого сотрем первую букву ч в этом слове: нажимая на пробел, подведем курсор к позиции, занимаемой этой буквой:

```
100 REM У_
```

Обратите внимание, что сама буква не видна. Теперь нужно сообщить Корвету, что эту букву нужно стереть. Для этого служит клавиша D (сокращение от команды DELETE). Нажмите D. (Опять напоминаем, что вы должны находиться в режиме работы с латинским шрифтом.) Стираемый символ будет изображен на экране заключенным в символы косой черты:

```
100 REM У/ч/
```

Символы / не будут добавлены к вашей строке. Они только изображаются на экране, чтобы вы точно видели, что вы стерли. Не двигайте сейчас курсор. Для того чтобы убедиться, что букву ч действительно стерли, нажмите L дважды.

Следующая ошибка касается трех р в слове "ррработать". Подведите курсор в нужное место:

```
100 Учимся _
```

Для того чтобы стереть две буквы, необходимо либо дважды нажать клавишу D, либо набрать более сложную команду 2D, т. е. сначала нажать цифру 2, а потом D. В результате будет напечатано:

100 Учимся /pp/_

Чтобы убедиться, что процесс стирания имел место, нажмите клавишу L дважды. Попробуйте самостоятельно удалить все повторяющиеся буквы, чтобы ваша строка приобрела вид

100 REM Учимся работать с редоктром

Теперь изменим неправильно введенный символ в слове "редоктром", т. е. первую букву о на а. Подведите курсор, нажимая на пробел, в нужное место:

100 REM Учимся работать с ред_

Изменение символа требует ввода двух символов одного за другим. Сначала нажмите на C (от англ. Change - изменять). (Опять напоминаем про латинский шрифт.) Потом введите тот символ, который вам нужен. Если вы вводите русскую букву (или буквы), то перед их введением надо, естественно, переключить клавиатуру в режим русского шрифта. Сразу же после введения, чтобы вернуться в режим редактирования, следует опять перейти к латинскому шрифту.

Попробуйте. Когда вы нажали клавишу C, ничего как будто не происходит. Однако в это время Корвет ожидает ввода нового символа. Когда вы его введете, он появится на экране и курсор переместится на одну позицию вправо:

100 REM Учимся работать с реда_

Нажмите L (опять напоминаем, что перед этим нажатием вам надо перейти к латинскому шрифту), чтобы увидеть остаток строки и опять начать процесс редактирования:

100 REM Учимся работать с редактром
100 _

Если вы хотите изменить несколько символов сразу, то вы должны ввести команду

NC НОВЫЕ СИМВОЛЫ (N штук)

где N - число символов, которые вы хотите поменять.

Следующая наша задача - научиться вводить дополнительные символы, например букву о между т и р в слове "редактром". Для этого, нажимая пробел, установите курсор в нужное место. Чтобы ввести один или несколько новых символов, необходимо использовать клавишу I (от англ. Insert - вставлять, вводить). Нажмите клавишу I. (Вы не забыли, что при вводе команды клавиатура должна быть латинской?)

Теперь вы можете вводить новые символы. Естественно, что если вы вводите русские буквы, то перед этим должна быть пере-

ключена клавиатура. Если вы ввели какой-либо символ ошибочно, верните курсор в нужную позицию, используя клавишу BACKSPACE, и введите нужный символ. При этом все введенные после этого символа знаки сотрутся. Чтобы прекратить ввод символов, нажмите клавишу ПРФ/ESC, расположенную во втором ряду под клавишей F1/F6. Заметьте, что в режиме ввода новых символов клавиши D, L, C и пробел теряют свои "командные свойства". Нажатие на них приводит только к появлению на экране соответствующих символов. После нажатия клавиши ПРФ эти командные функции восстановятся.

Теперь ваша строка приобрела, наконец, правильный вид:

```
100 REM Учимся работать с редактором
```

и вы готовы покинуть режим редактирования. Для этого существует несколько путей. Простейший из них - просто нажать клавишу ВК. Сделайте это. В результате появится промпт Бейсика. Теперь посмотрите, что стало с вашей строкой. Для этого введите команду LIST 100. Вы увидите:

```
100 REM Учимся работать с редактором
```

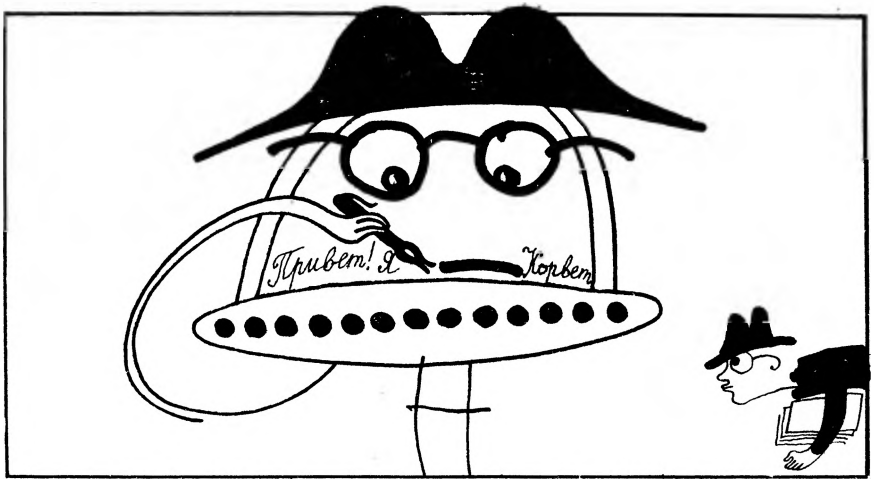
Итак, вы изучили три самые главные команды редактора:

ND - стирание символов
NC - замена символов (после C необходимо ввести N новых символов)
I - режим ввода новых символов (ввод прекращается нажатием клавиши ПРФ)

Предлагаем вам самостоятельно выполнить упражнение. С помощью редактора исправьте строку

```
10 PRINT "Привет!! Я ваш микрокомпьютер Корвет":GOTO 100
```

так, чтобы она имела вид



10 PRINT "Привет, я твой Корвет!":GOTO 10

Дополнительные команды редактора

Надеемся, что вы почувствовали удобство работы с редактором. Теперь самое время сделать редактирование еще более легким. Начнем с процесса перемещения курсора. Рассмотрим для примера какую-либо строку, содержащую ошибки:

100 PRINT "Коорвет лучший компьютер"

Перейдите в режим редактирования. Мы хотим сначала убрать лишнюю букву в слове "Корвет". Наберите цифру 8 и нажмите пробел. Вы видели? Курсор сразу перескочил в восьмую позицию:

100 PRINT "К_

Теперь можете стереть лишнюю букву.

Допустим, что вы хотите что-то добавить в самый конец строки. Для этого вы можете, естественно, нажимая на пробел, перевести курсор в конец строки и затем, нажав I (латинская буква), ввести необходимые символы. Существует и более простой способ. Нажмите клавишу X. Курсор сразу переместится на последнюю позицию и автоматически включится режим ввода новых символов. Таким образом, команда X полностью аналогична команде I с той лишь разницей, что ввод новых символов начинается в конце строки, а не в текущей позиции курсора. Все же остальное полностью аналогично.

Вы, наверное, помните, что строка Бейсика может содержать до 255 символов, из которых 250 можно ввести прямо с клавиатуры, когда вы вводите строку. Для ввода остальных 5 символов используются команды редактора I или X. Для того чтобы в этом убедиться, введите строку длиной в 250 символов. В конце ввода машина начнет пищать. Теперь перейдите в режим редактирования и нажмите X. После этого можете ввести еще 5 символов.

Предположим, что вы хотите стереть все символы, стоящие справа от курсора, и вместо них ввести новые. Для этого существует команда H (естественно, латинская буква). Она стирает остаток строки и в остальном приводит к тем же последствиям, что и команда X. Например, если у вас была строка

100 PRINT "Привет, я ваш Корвет"

то, перейдя в режим редактирования, установив курсор на следующую позицию после буквы ш в слове "ваш" и нажав клавишу H, можете ввести новое окончание:

100 PRINT "Привет, я ваш персональный компьютер Корвет"

Чтобы закончить редактирование, нажмите ВК.

Важной функцией редактора является возможность поиска нужных символов в редактируемой строке. Для того чтобы найти нужный

символ и установить курсор на то место в строке, где он располагается, служит команда S (от англ. Search - искать) с последующим указанием символа, который мы ищем. Рассмотрим, например, строку

```
100 PRINT "Здравствуйте, я Корвет"
```

и перейдем в режим редактирования:

```
EDIT 100
100 _
```

Теперь попробуем найти то место в строке, где первый раз встречается буква р. Для этого нажимаем клавишу S, переходим в режим русского шрифта и вводим букву р. На экране появилось:

```
100 PRINT "Зд_
```

Теперь, если хотите, можете изменить "найденный" символ, стереть его или что-нибудь добавить. Набрав еще раз команду поиска, находим местоположение второй буквы р:

```
100 PRINT "Здравствуйте, я Ко_
```

Функция поиска находит следующее положение указанного символа.

Существует способ найти сразу N-е положение символа. Для этого нужно указать N перед командой S. Так, в предыдущем примере, вместо того чтобы дважды набирать S и р, достаточно было набрать 2Sp и на экране сразу бы появилось:

```
100 PRINT "Здравствуйте, я Ко_
```

Когда вам нужно было стереть остаток строки, вы использовали команду H. А если вам нужно стереть часть строки с текущей позиции курсора до некоторого символа? Для этого служит команда K (от англ. Kill - убивать). Для того чтобы ею воспользоваться, вы должны нажать клавишу K и ввести тот символ, с которого должна начинаться отредактированная строка. Например, если вы хотите удалить из вашей строки, имеющей вид

```
100 CLS:PRINT "Корвет"
```

команду CLS, вы должны ввести команду KP. В результате получится:

```
100 /CLS:/_
```

т. е. на дисплее будет изображена стертая часть строки. Нажмите L, чтобы убедиться, что CLS: действительно стерто.

Вы можете стереть с помощью команды K все, что предшествует N-му появлению указанного в этой команде символа. Рассмотрим пример. Пусть ваша строка имеет вид

```
100 PRINT "LINE 1":PRINT "LINE 2":PRINT "LINE 3"
```

Перейдем в режим редактирования:

```
EDIT 100
100_
```

Сотрем все символы, предшествующие третьему оператору PRINT. Вы, наверное, думаете, что для этого надо набрать команду 3КР. Попробуйте. Как? Все стерлось. В чем же дело? Дело в том, что в начале редактирования курсор уже находился на первой букве P и поэтому компьютер начал счет не с нее, а со следующей буквы P. Таким образом, не найдя третьей P, он стер всю строку.

Исправим эту ошибку. Перепишите строку 100:

```
100 PRINT "LINE 1":PRINT "LINE 2":PRINT "LINE 3"
```

Перейдите в режим редактирования и наберите команду 2КР. В результате на экране появится:

```
100 /PRINT "LINE 1":PRINT "LINE 2":/
```

Теперь нажмите ВК и посмотрите, что получилось:

```
LIST 100
100 PRINT "LINE 3"
```

Это все, что осталось.

Итак, команда K работает следующим образом. Если вы ввели команду NK и символ, то отыскивается N-е положение символа, указанного после K, и все, что ему предшествовало, стирается. Если N-го положения нет, то стирается вся строка.

Теперь еще одно немаловажное замечание.

Что делать, если вы по ошибке стерли всю строку как в предыдущем примере, но спохватились об этом до окончания редактирования, т. е. до нажатия клавиши ВК? В Корвете предусмотрена возможность устранения подобных ошибок. Для этого служит команда A. По ней все ваши последние исправления игнорируются и процесс редактирования начинается снова. Попробуйте.

Предположим, что вы начали редактировать строку и затем решили этого не делать. Чтобы ее не испортить, вы можете нажать клавишу A и после этого ВК. Существует и более простой способ. Это команда Q. По этой команде Корвет также прекращает редактирование, оставляя строку неизмененной.

Нажатие клавиши ВК в любой момент завершает редактирование. Рабочая отредактированная копия строки записывается в память машины на место оригинала. Тот же эффект дает команда E.

Итак, вы овладели искусством редактирования программ. Советуем вам при дальнейшей работе с Корветом активно пользоваться редактором.

12. ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ ОБ АРИФМЕТИЧЕСКИХ ДЕЙСТВИЯХ И МАТЕМАТИЧЕСКИХ ФУНКЦИЯХ

Раздел, который покажет пользователю, что для того, чтобы вычислить пять в кубе, вовсе не обязательно умножать пять на пять и еще раз на пять

Вы, наверное, догадались, что в данном разделе речь пойдет о процедурах возведения в степень и извлечения корней. Естественно, вы можете вычислить, чему равно, например, 3 в степени 9. Для этого можно просто ввести команду

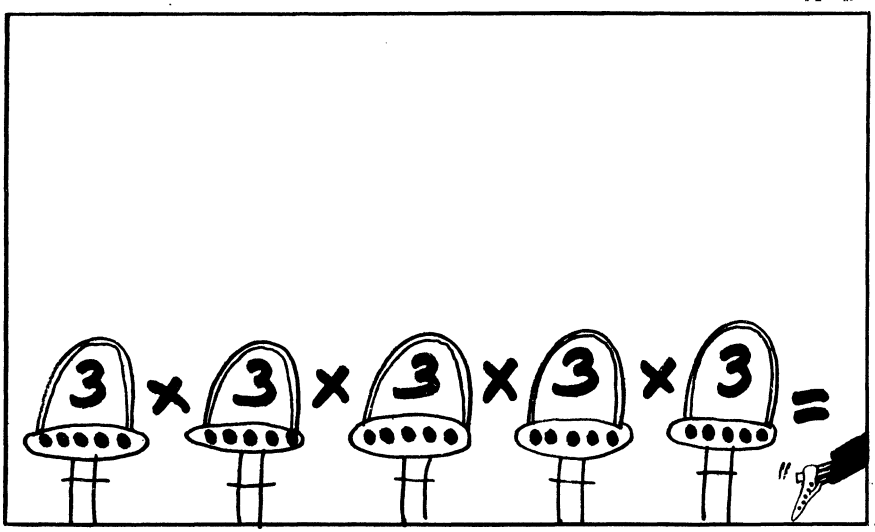
```
PRINT 3*3*3*3*3*3*3*3*3
```

Однако такой процесс очень утомителен. Слишком много работы с клавиатурой. Потом где-то в дальнем уголке сознания шевелится мысль: а что делать, если нам нужно вычислить 3 не в степени 9, а, например, в степени 1/137? Неужели такой современный аппарат, как Корвет, не имеет возможности помочь решить эту проблему?

Конечно, может. Для обозначения возведения в степень служит специальный символ ^. Если вы хотите возвести какое-либо число X в некоторую степень Y, достаточно использовать команду

```
PRINT X^Y
```

Возведение в степень является равноправным арифметическим действием. Оно может использоваться аналогично другим действиям типа сложения, умножения, деления и вычитания. А как возводить в дробную степень? Должны ли мы сначала вычислить показатель степени и лишь потом возводить в нее? Разберемся в этом вопросе подробнее. Допустим, мы хотим извлечь кубический корень из числа 27, или, что то же самое,



возвести число 27 в степень 1/3. Ориентируясь на предыдущий пример, вы вводите команду

```
PRINT 27^1/3
```

и видите ответ: 9. Поняли ли вы, в чем дело? Если да, то вы хорошо подготовленный математик.

До сих пор мы нигде не говорили, в каком порядке Корвет выполняет арифметические действия. Настало время внести окончательную ясность в этот вопрос. Возможно, логично было бы просто выполнять их слева направо. Но, к сожалению, исторически существует соглашение о приоритете одних арифметических действий над другими. Это значит, что одни действия выполняются раньше других.

Начнем с самых низкоприоритетных, но наиболее знакомых каждому из нас - сложения и вычитания. Они, к сожалению, выполняются в последнюю очередь. Несправедливо? Да, но кто-то должен замкнуть очередь. Перед этими действиями выполняются все умножения и деления. Перед ними, в свою очередь, возведение в степень и извлечение корней.

Для примера рассмотрим задачу о нахождении суммарной площади гипотетической квартиры, состоящей из трех комнат, имеющих размеры 5 на 6, 5 на 3 и 6 на 4 метров соответственно. Итак, нам нужно вычислить, чему равно

```
5*6+5*3+6*4
```

Компьютер будет работать следующим образом. Он сначала перемножит 5 и 6, 5 и 3, 6 и 4, а потом сложит полученные результаты. Поэтому по команде

```
PRINT 5*6+5*3+6*4
```

вы получите ответ 69. Если бы компьютер выполнял действия просто слева направо по очереди, то результат был бы иным.

Теперь вернемся к извлечению кубического корня из числа 27. Рассмотрим в свете только что полученной информации, как будет действовать Корвет. Сначала он выполнит возведение в степень, т. е. вычислит 27 в степени 1. Получится 27. После этого он перейдет к делению. Поделив 27 на 3, он получит 9, о чем и сообщит вам.

Что же делать? Один из известных вам способов - завести переменные и сохранять в них результаты промежуточных действий:

```
A=1/3
PRINT 27^A
```

После этих действий вы получите в следующей строке на экране число 3. На этот раз ответ правильный. Но что-то не нравится. Зачем для простого действия использовать переменные? Оказывается, есть другой способ. Вы можете указать Корвету, какие действия нужно выполнять в первую очередь. Для этого служат скобки.

Решим важную задачу о наполнении бассейна водой из пяти труб. По ним втекает 5, 6, 7, 8, 9 литров в секунду соответственно. Интересно, сколько воды будет в бассейне через 1 минуту? Любой школьник решит эту задачу. Сначала нужно сложить пропускную способность труб, чтобы определить общее количество воды, втекающее за 1 секунду в бассейн, а затем умножить результат на 60. Вы, естественно, понимаете, что просто выполнение команды

```
PRINT 5+6+7+8+9*60
```

приведет к ошибочному результату. Правильный результат можно получить, используя скобки:

```
PRINT (5+6+7+8+9)*60
```

Они указывают компьютеру, что сначала нужно выполнить те действия, что заключены в них, а потом уже все остальные. Внутри скобок действия выполняются по обычным законам.

Теперь вычислим, наконец, кубический корень из 27:

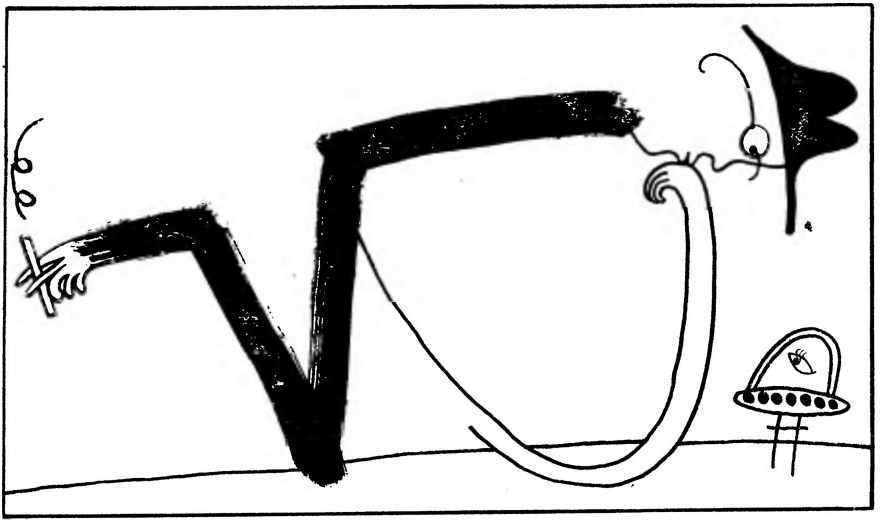
```
PRINT 27^(1/3)
```

Компьютер сначала выполнит действие в скобках, т. е. поделит 1 на 3, а потом возведет 27 в полученную степень. В результате будет получен правильный ответ: 3.

Теперь поупражняйтесь самостоятельно. Результатом ваших занятий должно быть умение свободно пользоваться скобками для определения порядка выполнения арифметических операций.

Вы теперь можете извлечь корень любой степени из любого числа. Однако существует частный случай корня. Это квадратный корень. Для его извлечения существует более простой и, следовательно, более быстрый алгоритм. Поэтому в Бейсике предусмотрена возможность извлекать квадратный корень, не прибегая к действию символа ^, т. е. возведению в степень. Эту задачу решает функция SQR(X) (от англ. SQuare Root - квадратный корень).

Вы, наверное, удивились, увидев выражение SQR(X). До сих пор вы ни с чем подобным не встречались. Действительно, сейчас мы познакомимся с новым для вас типом операторов Бейсика. Это функции. Возможно, вы припоминаете наши разговоры по поводу языков высокого уровня, где мы обсуждали возможность использования библиотек, составленных из наиболее распространенных и часто используемых программ. Такие программы подключаются к вашей программе и позволяют выполнять различные действия. Но как сообщить этой программе, что мы от нее хотим, и как получить от нее результат? Подобные программы определяются аналогично функциям в математике. Каждая программа имеет имя, по которому она разыскивается в библиотеке компьютером. Это делается автоматически при упоминании этого имени в тексте программы, когда машина дойдет до этого места. Если мы хотим что-нибудь передать для выбранной программы, мы используем аргументы. Их число может быть различным для различных функций.



Рассмотрим, например, функцию извлечения квадратного корня SQR(X). В данном случае SQR является именем функции, а X - аргументом. Когда компьютер найдет это выражение в тексте программы, он подумает: "Ага! Это не имя какой-то переменной, а имя функции.Ну-ка, поищем ее в библиотеке!" После успешного поиска компьютер вытащит эту программу из недр своей памяти, присвоит аргументу значение X и после выполнения необходимых для вычисления функции действий вернется к выполнению основной программы. Так, если выполнялся оператор

```
PRINT SQR(X)
```

то извлеченный корень будет выведен на экран. Если оператор присваивания имел вид

```
A=SQR(X)
```

то это значит, что компьютер присвоит A значения квадратного корня из X. Вместо переменной X можно использовать любые арифметические выражения, например:

```
PRINT SQR(2*3+5^(1/4)-21)
PRINT SQR(SQR(3))
```

Все это правильные операторы.

Приведем список математических функций, имеющихся в Бейсике:

Функция	Выполняемое действие
ABS(X)	вычисление абсолютного значения X
ATN(X)	вычисление арктангенса X
CDBL(X)	преобразование X в двойную точность
CINT(X)	преобразование X в целое число
COS(X)	вычисление косинуса X
CSNG(X)	преобразование X в одинарную точность

EXP(X)	вычисление значения e в степени X (экспоненты)
FIX(X)	усечение дробной части
INT(X)	преобразование X в целое число
LOG(X)	вычисление натурального логарифма
RANDOMIZE(X)	инициализация генератора случайных чисел
RND(X)	вычисление случайного числа от 0 до X
SGN(X)	определение знака числа: -1(X<0), 0(X=0), 1(X>0)
SIN(X)	вычисление синуса X
SQR(X)	вычисление квадратного корня из X
TAN(X)	вычисление тангенса X

Как видите, математических функций достаточно много. Пользуясь ими, можно производить довольно сложные вычисления. Попробуйте сами. Только имейте в виду, что у тригонометрических функций аргумент должен быть задан в радианах (3.14159 радиана соответствуют 180 градусам).

Теперь самое время вернуться к числам разной точности, целым и строковым переменным. Вы стали уже достаточно опытным программистом, чтобы чувствовать неудовлетворение от использования в именах переменных символов !, #, %, $. Вспомните, раньше мы упоминали об операторах, определяющих тип переменной. Теперь мы, наконец, до них добрались. Рассмотрим один из них - DEFINT. Этот оператор говорит компьютеру, что все переменные, имена которых начинаются с перечисленных за ним через запятую букв, - целые. Рассмотрим программу

```
10 DEFINT I
20 I1=2.567
30 I2=3.33333333
40 I1!=1.2345
50 I2#=3.33333333
60 I=-5.5
70 I$="Строковая переменная"
80 PRINT I1,I2,I1!,I2#,I,I$
```

Строка с номером 10 программы говорит компьютеру, что все переменные, имена которых начинаются с буквы I, - целые, т. е. I, I1, I2 - целые. Однако мы можем изменить тип переменной, добавив один из символов !, %, #, $.

Интересно, ругаете ли вы нас за забывчивость? Если нет, то вы невнимательный читатель. Нигде в предыдущем тексте мы не объясняли, что значит ! после имени переменной. Почему? Да потому, что ! означает, что эта переменная одинарной точности. Естественно, что пользоваться этим символом нужно только в случае, когда переменная определена как принадлежащая к какому-нибудь другому типу.

Аналогично DEFINT работают операторы DEFSNG, DEFDBL, DEFSTR - они определяют переменные одинарной точности, двойной точ-

ности и строковые соответственно. Форматом для всех этих операторов является:

DEFINT (СПИСОК БУКВ ЧЕРЕЗ ЗАПЯТУЮ)
DEFSNG (СПИСОК БУКВ ЧЕРЕЗ ЗАПЯТУЮ)
DEFDBL (СПИСОК БУКВ ЧЕРЕЗ ЗАПЯТУЮ)
DEFSTR (СПИСОК БУКВ ЧЕРЕЗ ЗАПЯТУЮ)

Если список содержит выражение типа A-D, это означает, что имеются в виду все буквы от A до D, например:

DEFINT A,B,W - все переменные, начинающиеся с A, B, W, являются целыми
DEFSNG F-H,X-Z - все переменные, начинающиеся с F, L, J, G, H и X, Y, Z, имеют одинарную точность
DEFDBL K-P,Q - все переменные, начинающиеся с K, L, M, N, O, P и Q, имеют двойную точность
DEFSTR R,S - все переменные, начинающиеся с R и S, являются строковыми

В любом месте программы вы можете переопределить имя переменной, добавив к нему символы !, #, %, $.

13. ЦИКЛЫ И ПОДПРОГРАММЫ

Плетение кружев и украшение сухих ветвей зеленой листвой

Вы, конечно, помните способ организации простейшего цикла:

```
10 A=1
20 PRINT 1/A
30 A=A+1
40 GOTO 20
```

Подождите немного. Прежде чем запускать эту программу, познакомимся с новыми операторами.

Операторы трассировки программы

Два новых оператора позволяют проследить выполнение программы компьютером. Один называется TRON (от англ. TRace ON - включение трассировки), другой - TROFF (от англ. TRace OFF - выключение трассировки).

Наберите команду TRON и нажмите клавишу ВК. Теперь компьютер знает, что нам очень интересно узнать, как он выполняет программу. Запускайте ее и посмотрите, что происходит. Вы увидите, что Корвет каждый раз начинает новую строку на экране с номера строки программы, заключенного в угловые скобки < >. Этим он говорит нам, что выполнена строка с указанным номером, что очень полезно при отыскании ошибок в плохо работающих программах.

Трассировка будет производиться до тех пор, пока компьютер не получит команду TROFF.

Операторы цикла

Циклы бывают трех типов: бесконечные циклы; циклы, оканчивающиеся по условию; циклы со счетчиком. Пример бесконечного цикла мы только что рассматривали. Вы также хорошо помните циклы с условиями. Они содержат операции типа IF...THEN, например:

```
10 A$="Эх раз, еще раз, еще много, много раз!"
20 PRINT A$
30 INPUT "Продолжать? (Д/Н)";R$
40 IF R$="Н" THEN END
50 GOTO 20
```

Обратите внимание, что вы не можете предсказать, сколько раз повторится этот цикл. Такие циклы очень удобны, если вам нужно вводить данные с клавиатуры или брать их из оператора DATA.

Третий тип циклов особенно удобен в случаях, когда вы точно знаете, сколько раз вам нужно его повторить. Такие циклы требуют специальной переменной счетчика, значение которой определяет выполнение цикла. Введите программу

```
10 A=1
20 PRINT 1/A
30 A=A+1
40 IF A=10 THEN END
50 GOTO 20
```

В данном случае A выполняет роль счетчика. Когда его значение достигнет 10, цикл закончится. Легко видеть, что для правильной работы программы необходимо указать начальное значение счетчика (A=1), его приращение (A=A+1) и конечное значение (A=10). Оказывается, что всю эту информацию можно указать компьютеру при помощи одного оператора. Он называется FOR и имеет формат

```
FOR ИМЯ СЧЕТЧИКА=НАЧАЛЬНОЕ ЗНАЧЕНИЕ
TO КОНЕЧНОЕ ЗНАЧЕНИЕ
STEP ШАГ
```

В конце цикла должен стоять замыкающий оператор NEXT (имя счетчика).

Рассмотрим, например, программу

```
10 FOR C=0 TO 59 STEP 2
20 PRINT "C=",C;
30 NEXT C
```

Прежде чем запускать эту программу, прочтите ее внимательно. Как вы думаете, что она будет выводить на экран? Теперь запустите ее и сопоставьте результат с вашими умозаключениями.

Мы не раз говорили, что языки высокого уровня похожи на человеческий язык. Исторически все эти языки ориентируются на английскую речь. Разберем с этой точки зрения оператор FOR. С английского FOR можно перевести как "для". Смысл первой части оператора

```
FOR C=0
```

можно передать словами: "Начальное значение счетчика цикла С положить равным нулю". TO означает "до", т. е. TO 59 значит "до 59". Теперь мы можем расшифровать всю фразу: "Выполнить цикл для значений счетчика С от нуля до конечного значения 59".

Остается еще слово STEP. Оно в переводе с английского означает ступень или шаг. Итак, STEP 2 означает "шаг 2" - значение счетчика должно изменяться каждый раз на 2.

А как компьютер узнает, где конец цикла? Для этого служит замыкающий оператор NEXT, что в переводе означает "следующий".

Рассмотрим теперь, как компьютер выполняет такой цикл. Встретив в программе слово FOR, компьютер думает: "Ага! Сейчас мне сообщат начальное и конечное значение счетчика и на сколько его изменять". Потом, запомнив все эти сведения, компьютер входит в цикл и начинает выполнять по очереди имеющиеся в нем операторы. Затем, дойдя до оператора NEXT, компьютер изменяет значение счетчика и возвращается к началу цикла. Если значение счетчика превысит его конечное значение, указанное в операторе TO, выполнение цикла будет прервано и будет выполнен оператор, следующий за оператором NEXT.

Заметим, что значения счетчика могут быть не только положительными, но и отрицательными. Так, следующие примеры являются правильными, с точки зрения Бейсика:

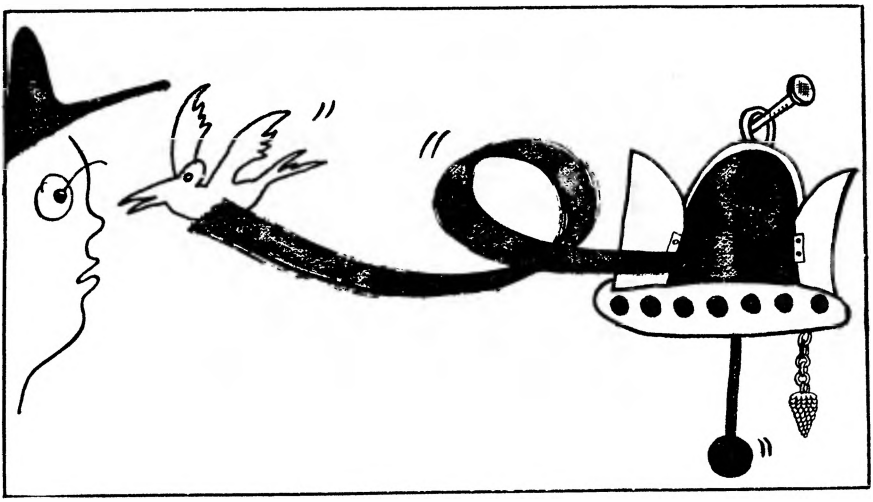
```
10 FOR I=0 TO -25 STEP -1
20 NEXT I

10 FOR I=25 TO 0 STEP -1
20 NEXT I

10 FOR I=-25 TO 0 STEP 1
20 NEXT I
```

Если приращение счетчика равно 1, оператор STEP можно опускать.

Мы познакомились с очень полезным оператором языка Бейсик. Теперь самое время написать какую-нибудь полезную программу. Для примера попробуем использовать компьютер в качестве настольных часов. Сначала попробуем соорудить секундомер, который бы мог считать секунды. Секунд в минуте 60. Следовательно, нам нужно научить машину считать до 60. Это можно сделать с помощью следующей программы:



```
10 FOR SEC=0 TO 59
20 CLS
30 PRINT SEC
40 NEXT SEC
50 GOTO 10
```

Теперь наберите RUN и нажмите клавишу ВК. Что вы видите? В верхнем углу экрана что-то мелькает, а секунд что-то не видно. Это потому, что часы у нас сильно спешат. Попробуем замедлить их бег. Для этого нажмите клавишу СТОП и добавьте в программу две новые строки:

```
14 FOR N=0 TO 450
16 NEXT N
```

Запустите программу. Что вы видите? Действительно, похоже на секундомер. Давайте теперь проанализируем нашу программу. Что дали нам строки 14 и 16? Вроде никаких операторов в этом цикле нет. Зачем же он нужен? Очень просто. Эти две строки заставляют машину внутри себя, не выводя ничего на экран, изменять N от 0 до 450 и тем самым обеспечивают необходимую задержку, чтобы цифры на экране появлялись точно через секунду. Время этой задержки, т. е. замедление или ускорение хода часов, можно изменять в широких пределах, варьируя конечное значение счетчика цикла. Возьмите проверенные часы и настройте ваш компьютер так, чтобы он правильно считал секунды. Заодно вспомните, как редактировать программы.

Теперь, когда у нас есть секундомер, можно сконструировать часы целиком с минутами и часами. Вы уже, наверное, догадались, как это сделать. Если нет, то ниже приводится текст программы (не забудьте перед вводом этой программы стереть старую):

```
10 CLS
20 INPUT "Введите часы";HR
30 INPUT "Введите минуты";MIN
40 INPUT "Введите секунды";SEC
50 FOR HR=HR TO 12
60 FOR MIN=MIN TO 59
70 FOR SEC=SEC TO 59
80 CLS:PRINT:PRINT:PRINT:PRINT
90 PRINT " ";HR;":";MIN;":";SEC
100 FOR N=0 TO 450
110 NEXT N
120 NEXT SEC:SEC=0
130 NEXT MIN:MIN=0
140 NEXT HR:HR=0
150 GOTO 50
```

Теперь у вас есть собственные электронные часы. На примере этой программы мы познакомились с правилами организации циклов со счетчиками. Как вы видите, в нашей программе используется четыре таких цикла. Заметьте, что некоторые из них помещены внутрь других циклов. Так, цикл, считающий минуты, находится внутри цикла, считающего часы. Счетчик секунд расположен внутри и того и другого. А цикл задержки входит во все три предыдущих цикла. Такие циклы называются вложенными. Организация подобных циклов подчиняется следующему правилу: оператор NEXT, закрывающий внутренний цикл, должен всегда стоять перед оператором, закрывающим внешний цикл, например:

Правильно	Неправильно
10 FOR J=1 TO 3	10 FOR J=0 TO 3
20 FOR J1=1 TO 2	20 FOR J1=1 TO 2
30 NEXT J1	30 NEXT J
40 NEXT J	40 NEXT J1

Во втором случае вы получите сообщение об ошибке:

```
NEXT БЕЗ FOR В 40
```

Это значит, что компьютер не понял, к какому оператору FOR относится NEXT J1. Закрыв внешний цикл по команде NEXT J, компьютер решил, что он закрыл все внутренние циклы.

Теперь поупражняйтесь в редактировании. Измените программу часов так, чтобы она показывала время не в 12-часовом, а в 24-часовом формате.

Рассмотрим еще одну очень распространенную ошибку. Мы надеемся, что вы уже наигрались с часами и без сожаления сотрете эту программу. Введите новую программу

```
10 CLS
20 A$="Эх раз, еще раз, еще много, много раз..."
```

```
30 FOR I=0 TO 5
40 PRINT A$
50 NEXT I
60 PRINT
70 INPUT "А что, может, и правда еще разок (Д/Н)";R$
80 IF R$="Н" THEN END
90 PRINT
100 GOTO 30
```

Эта программа является примером бесконечного цикла. Он будет выполняться, пока вы не нажмете Н. Попробуем переделать эту программу так, чтобы заменить оператор GOTO на оператор цикла. Это легко сделать. Добавим строку с номером 15:

```
15 FOR J=1 TO 100
```

и изменим строку 50:

```
50 NEXT J
```

Запустите программу снова. Все в порядке? Да. Но здесь и сидит ошибка. Дело в том, что если вы ответите Н, то цикл FOR J=1 TO 100 будет не закончен. Компьютер будет теперь ожидать команды NEXT J, чтобы продолжать цикл. Не верите? Проверьте. Введите команду

```
NEXT J
```

и нажмите ВК. Компьютер начнет выполнять строку 20! Поэтому будьте внимательны. Не выходите из циклов, не закончив их. А общим правилом является: не используйте циклов со счетчиками для организации бесконечных циклов. Пользуйтесь в этих случаях операторами GOTO и IF...THEN.

Теперь вы обладаете достаточными знаниями для того, чтобы писать различные программы. Вы вооружены такими мощными средствами, как вложенные циклы, условное и безусловное ветвление. Но Корвет имеет еще великое множество возможностей. Он дружески старается облегчить ваш труд и предлагает вам дополнительные программные средства.

Оператор ELSE

В переводе это слово означает "иначе". ELSE используется совместно с оператором IF...THEN. Таким образом, общий формат оператора IF принимает вид

```
IF УСЛОВИЕ THEN ОПЕРАТОР 1 ELSE ОПЕРАТОР 2
```

Прочитаем эту строку: "Если условие выполнено, то тогда выполняется оператор 1; в противном случае выполняется оператор 2".

Следующая программа иллюстрирует способ использования оператора IF:

```
10 INPUT "Введите два числа";A,B
20 IF A<B THEN PRINT A;"<";B:END
30 PRINT A;">=";B
```

Строки 20 и 30 можно свести в одну:

```
20 IF A<B THEN PRINT A;"<";B ELSE PRINT A;">=";B
```

Операторы, стоящие после THEN и ELSE, могут быть различными командами языка Бейсик. Мы можем научить нашу программу отличать еще случай A=B:

```
10 INPUT "Введите два числа";A,B
20 IF A<B THEN PRINT A;"<";B:STOP:ELSE IF B<A
   THEN PRINT B;"<";A:STOP
30 PRINT A;"=";B
```

Перевод строки 20 имеет вид: "Если A меньше B, то PRINT A<B и STOP, но если B меньше A, то PRINT B<A и STOP". Если оба условия не выполнены, то программа переходит к выполнению строки 30.

Оператор множественного ветвления

Рассмотрим программу, начинающуюся следующим образом:

```
10 PRINT "Введите 1 для выбора функции 1"
20 PRINT "Введите 2 для выбора функции 2"
30 PRINT "Введите 3 для выбора функции 3"
40 INPUT C
```

После того как вы введете значение C, программа должна найти ту функцию, которую вы выбрали:

```
50 IF C=1 THEN 1000
60 IF C=2 THEN 2000
70 IF C=3 THEN 3000
80 GOTO 10
1000 PRINT "Функция 1":END
2000 PRINT "Функция 2":END
3000 PRINT "Функция 3":END
```

Запустите программу. В каком случае выполняется строка 80? А может быть она вообще не нужна? Попробуйте ее выкинуть. Теперь сотрите строки 50 - 70 и замените их одной:

```
50 ON C GOTO 1000,2000,3000
```

Оператор ON является примером оператора множественного ветвления. Формат оператора следующий:

```
ON ТЕСТИРУЕМАЯ ВЕЛИЧИНА
GOTO СПИСОК НОМЕРОВ СТРОК
```

Тестируемая величина является числом или арифметическим выражением, которое может принимать значения от 0 до 256. Номера строк в списке должны разделяться запятыми.

Если тестируемая величина равна 1, управление передается на строку с номером, стоящим первым в списке. Если тестируемая величина равна 2, управление передается на строку с номером, стоящим вторым в списке, и т. д. Если тестируемая величина равна нулю или превышает число номеров строк в списке, выполняется следующая строка.

Подпрограммы и операторы организации подпрограмм

Подпрограммы обычно содержат некоторую последовательность действий, которую нужно многократно выполнять в различных местах основной программы. Организовывать доступ к подпрограммам можно разными способами: во-первых, мы можем использовать операторы условного ветвления

```
IF A<B THEN 100 ELSE 200
```

во-вторых, можно использовать оператор множественного ветвления

```
ON A GOTO 1000,2000,3000
```

Этот оператор, в зависимости от A, активизирует подпрограммы, начинающиеся со строк 1000, 2000, 3000. В конце подпрограммы должна стоять команда, которая позволит вернуться назад к выполнению основной программы.

Вернемся к нашей последней программе. Организуем ее так, чтобы использовать подпрограммы:

```
10 PRINT "Введите 1 для выбора функции 1"
20 PRINT "Введите 2 для выбора функции 2"
30 PRINT "Введите 3 для выбора функции 3"
40 INPUT C
50 ON C GOTO 1000,2000,3000
80 GOTO 10
1000 PRINT "Функция 1":GOTO 10
2000 PRINT "Функция 2":GOTO 10
3000 PRINT "Функция 3":GOTO 10
```

Строки 10 - 80 образуют основную программу. Подпрограммами являются строки 1000, 2000, 3000. Вроде бы все отлично. Но не спешите радоваться. Предположим, что подпрограмма должна выполнить что-нибудь полезное, а не просто сообщить, что машина в этом месте побывала. Например, подпрограмма может вычислять значение десятичного логарифма. В стандартной библиотеке есть только функция натурального логарифма. Для вычисления десятичного логарифма числа нам нужно поделить значение натурального логарифма этого числа на натуральный логарифм числа 10. Делать эту операцию каж-

дый раз, когда он нам нужен, утомительно. Поэтому естественно возложить выполнение этой процедуры на подпрограмму.

Здесь возникает вопрос: а как организовать возврат из подпрограммы в то место, где мы ее вызвали? В последней нашей программе с этим делом справлялся оператор GOTO 10. Но в этом случае мы точно знали, откуда подпрограмма вызывалась. Если мы попробуем ее вызвать из другой части программы, то после завершения подпрограммы управление будет передано опять на строку с номером 10, а вовсе не куда нужно.

Как избежать этой неприятности? Очень просто. Достаточно воспользоваться специальным оператором GOSUB языка Бейсик. Он специально придуман для вызова подпрограмм. Когда компьютер находит этот оператор, он действует почти так же, как в случае GOTO: управление передается на строку с номером, следующим за оператором. Однако есть маленькая разница. В отличие от GOTO, оператор GOSUB говорит компьютеру: "Запомни номер текущей строки!" Поэтому компьютер, послушный указанию программы, радостно запоминает номер строки, откуда вызывается подпрограмма. После этого начинается выполнение подпрограммы. Подпрограмма должна обязательно заканчиваться оператором RETURN, что в переводе на русский язык означает "возврат".

Итак, логически работа с подпрограммами организована очень просто. Мы вызываем ее оператором GOSUB и заканчиваем ее оператором RETURN, передавая тем самым управление на строку с номером, следующим за вызовом подпрограммы. Для примера рассмотрим, как можно организовать электронный секундомер, используя подпрограмму задержки:

```
10 PRINT "Демонстрация работы подпрограммы"
20 S=0
30 GOSUB 1000
40 S=S+1
50 IF S>59 THEN S=0
60 PRINT "Секунды=";S
70 GOTO 30
1000 PRINT "Подпрограмма задержки"
1010 FOR I=1 TO 450
1020 NEXT I
1030 RETURN
```

Рассмотрим правила, которые необходимо выполнять при организации подпрограмм. Во-первых, вход в подпрограмму должен осуществляться только по команде GOSUB. Если хотите посмотреть, что будет при пренебрежении этим правилом, сотрите строку 70 и запустите программу. Получите сообщение об ошибке:

```
RETURN БЕЗ GOSUB
```

Во-вторых, не выходите из подпрограммы каким-либо другим способом, кроме как при помощи оператора RETURN. Исключение составляет только вызов из подпрограммы других подпрограмм. Это значит, что в подпрограмме не должно быть операторов ветвления типа GOTO или ON...GOTO, которые передавали бы управление за пределы подпрограммы. Оператор же GOSUB можно применять, поскольку после выполнения подпрограммы управление автоматически будет возвращено в нужное место. Пренебрежение этим правилом может и не привести к сообщению об ошибке. Однако помните, что если компьютер не говорит вам, что нашел ошибку, то это не всегда означает, что программа работает правильно.

Зададимся вопросом: существует ли возможность множественного ветвления с помощью подпрограмм? Существует, и выглядит это почти так же, как и в случае GOTO:

```
ON ТЕСТИРУЕМАЯ ВЕЛИЧИНА
GOSUB СПИСОК НОМЕРОВ СТРОК
```

Действует этот оператор аналогично ON...GOTO. В зависимости от значения тестируемой величины вызывается соответствующая подпрограмма, например:

```
10 PRINT "Введите 1 для вывода квадратных корней"
20 PRINT "Введите 2 для вывода кубических корней"
30 PRINT "Введите 3 для вывода корней четвертой степени"
40 INPUT C
50 ON C GOSUB 100,200,300
60 GOTO 10
100 FOR I=1 TO 50
110 PRINT I;SQR(I),
120 NEXT I
130 RETURN
200 FOR I=1 TO 50
210 PRINT I;I^(1/3),
220 NEXT I
230 RETURN
300 FOR I=1 TO 50
310 PRINT I;I^(1/4),
320 NEXT I
330 RETURN
```

Попробуйте самостоятельно проследить ход выполнения этой программы. Обратите внимание на строку 60. Она предохраняет программу от попадания в подпрограмму, минуя оператор GOSUB. Попробуйте стереть эту строку и посмотрите, что получится.

14. МАССИВЫ

Раздел, из которого читатель узнает еще об одном способе хранения информации в памяти машины

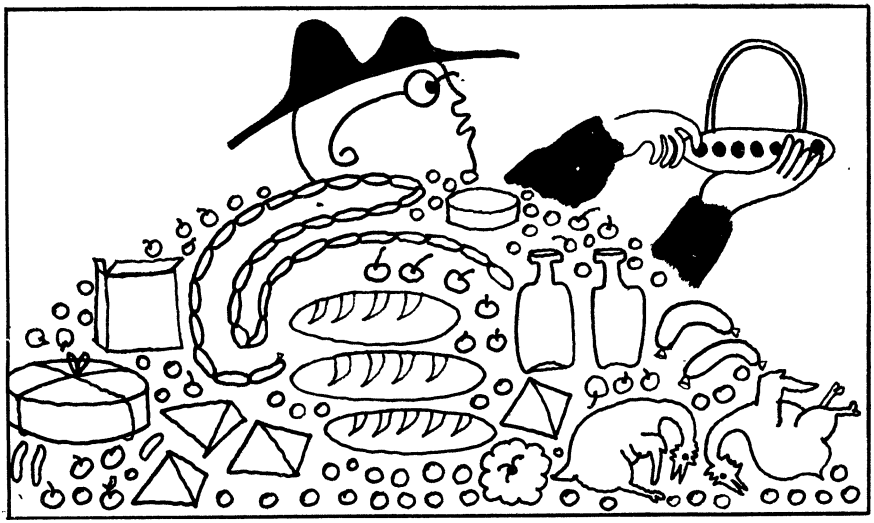
Допустим, что вы живете в многочисленной семье с большим числом родственников. Вся эта орава потребляет в месяц огромное количество самой разнообразной провизии. Вам, как наиболее образованному члену семьи, поручено заниматься учетом расходуемого провианта. Однако у вас есть и другие увлечения. Вы, возможно, любите рисовать или слушать музыку. Может быть, вы страстный болельщик. Поэтому нудный процесс учета продуктов отнимает у вас драгоценное время, которое вы бы с удовольствием потратили на любимое увлечение. В результате начинают возникать конфликты в семье. Как их избежать? Прекратить заниматься любимым делом и всецело отдаться процессу учета провизии? Конечно, нет. У вас есть надежный друг, который охотно возьмет на себя эту нудную обязанность. Это Корвет. Но для того чтобы он смог это сделать, необходимо выполнить подготовительную операцию - написать программу.

Сформулируем задачу. Ваши родственники употребляют следующие продукты: молоко, хлеб, мясо, овощи и сыр. Вам нужно уметь подсчитывать, сколько они съедают за месяц.

Простейшая программа, приведенная ниже, позволяет вам вести учет и всегда иметь требуемые данные под рукой:

```
10 INPUT "Сколько требуется молока (л)";MILK
20 INPUT "Сколько требуется хлеба (бух.)";BREAD
30 INPUT "Сколько требуется мяса (кг)";MEAT
40 INPUT "Сколько требуется овощей (кг)";VEGET
50 INPUT "Сколько требуется сыра (кг)";CHEESE
60 CLS
70 INPUT "Введите 1-молоко, 2-хлеб, 3-мясо, 4-овощи, 5-сыр";A
80 PRINT "Необходимо:"
90 IF A=1 THEN PRINT MILK;"л молока"
100 IF A=2 THEN PRINT BREAD;"бух. хлеба"
110 IF A=3 THEN PRINT MEAT;"кг мяса"
120 IF A=4 THEN PRINT VEGET;"кг овощей"
130 IF A=5 THEN PRINT CHEESE;"кг сыра"
140 PRINT:GOTO 70
```

Запустите эту программу и посмотрите, что получится. Вроде, все прекрасно работает. Но задумаемся, что было бы с нашей программой, если компонентов в меню было бы не 5, а 500. Страшно представить себе нашу программу, переделанную для такого случая. Получатся минимум 1000 строк. Такая работа под силу разве что Геркулесу.



Нет ли у Корвета какой-нибудь возможности, чтобы облегчить нашу жизнь? Оказывается, есть. Эта возможность называется массивами. Чем они отличаются от обычных переменных? В сущности ничем. Каждый элемент массива является полноправной переменной. Вся суть в том, что такие переменные пронумерованы. Поэтому для операций с ними нужно знать имя массива и номер элемента в массиве. Не очень понятно? Попробуем разобрать это на примере. Для этого напишем новую программу:

```
10 DIM FOOD(5)
20 PRINT "1-молоко, 2-хлеб, 3-мясо, 4-овощи, 5-сыр"
30 FOR I=1 TO 5
40 PRINT "Введите необходимое количество продукта";I
50 INPUT FOOD(I)
60 NEXT I
70 CLS
80 PRINT "Введите 1-молоко, 2-хлеб, 3-мясо, 4-овощи, 5-сыр"
90 INPUT I
100 PRINT "Вам нужно";FOOD(I)
110 PRINT
120 GOTO 90
```

Запустите программу. Вы видите, что она работает практически так же, как и предыдущая. Однако вместо перечисления продуктов появилась некая собирательная величина FOOD(I). Эта штука как раз и есть массив. Задается массив оператором DIM (строка 10) и имеет формат

DIM ПЕРЕМЕННАЯ (РАЗМЕРНОСТЬ)

DIM является сокращением английского слова DIMension, что в переводе означает "размер". Этот оператор говорит машине, что нужно зарезервировать в памяти место под переменные с именем, указан-

ным после оператора DIM. Число таких переменных указано в скобках. В нашем случае мы создали массив с именем FOOD. В этом массиве может содержаться 5 переменных. Обращение к элементам массива очень простое. Вы указываете имя массива и в скобках номер переменной.

Массивы могут состоять из переменных любого известного вам типа. Они могут быть целыми, действительными, одинарной и двойной точности и строковыми. Важно, что все элементы массива должны быть одного типа. Выражение, стоящее в скобках и определяющее номер элемента, может быть числом, переменной или арифметическим выражением.

Сравните только что написанную программу со старой версией. Все вроде бы одинаково, но первая программа комментировала то, что происходит на экране, а новая нет. Как ее научить этому? Очень просто. Нужно создать еще один массив, в котором собрать все сообщения. Попробуем это сделать. Прежде всего добавим в оператор DIM сообщение о новом массиве. Это будут строковые переменные:

```
10 DIM FOOD(5),MES$(5)
```

Теперь модифицируем строку 100:

```
100 PRINT "Вам нужно";FOOD(I);MES$(I)
```

А как сообщить машине, что должно быть в массиве MES$(I)? Существует несколько способов. И все они вам известны. Вы можете, например, использовать оператор присваивания:

```
MES$(1)="л молока"
MES$(2)="бух. хлеба"
```

и т. д. Но этот вариант не оптимальный. Если число статей расхода увеличится, то слишком много строк будет в вашей программе. Но теперь мы знаем, как бороться с повторяющимися операциями присваивания. Вспомнили? Как вам понравится такой вариант:

```
15 GOSUB 200
200 K=1:CLEAR 1000
210 READ A$
220 IF A$="I" THEN RETURN
230 MES$(K)=A$
240 K=K+1
250 GOTO 210
300 DATA "л молока","бух. хлеба","кг мяса","кг овощей","кг сыра"
310 DATA "I"
```

Это хороший случай, чтобы вспомнить старые уроки. Если вы не понимаете, как этот блок работает, то перечитайте разделы 10, 13.

В результате мы получили программу, которую легко модифицировать. Попробуйте самостоятельно изменить ее так, чтобы учесть

еще потребности в твороге, конфетах и чае. Не забудьте увеличить размерность соответствующих массивов.

Но пойдем дальше. Как научить машину составлять смету за год?

Мы можем, например, увеличить размерность массива FOOD так, чтобы он соответствовал году:

```
10 DIM FOOD(5*12),MES$(5)
```

Теперь мы можем попросить машину сообщить нам расход любого типа продуктов. Например, расход молока в апреле хранится в элементе массива FOOD(5*4-4). Почему так? Апрель - четвертый месяц года. Молоко же стоит первым в списке продуктов. А всего пять типов продуктов. Отсюда и формула. Если не верите, напишите все это на бумаге и подсчитайте номер элемента массива, содержащего молоко в апреле.

Этот способ очень хорош, он дает правильный результат. Но у него есть существенный недостаток: требуется определять формулу для вычисления номера элемента массива. Оказывается, можно обойтись без этого неудобства.

Дело в том, что массивы могут быть не только с одним измерением (одномерные). Они бывают двумерные и даже трехмерные. Если в одномерном массиве элементы располагаются цепочкой друг за другом, то в двумерном они уже занимают плоскость. Теперь элемент массива имеет не один номер, а два - по вертикали (номер колонки) и горизонтали (номер строки). Теперь решить нашу проблему проще пареной репы. Заведем двумерный массив

```
10 DIM FOOD(12,5),MES$(5)
```

А вся программа примет вид

```
5 CLEAR 1000
10 DIM FOOD(12,5),MES$(5)
15 GOSUB 200
16 FOR M=1 TO 12
18 PRINT "В течение месяца";M
20 FOR I=1 TO 5
30 PRINT "Введите необходимое количество";MES$(I);
40 INPUT FOOD(M,I)
50 NEXT I,M
60 CLS
70 INPUT "Какой месяц",M
80 PRINT "1-молоко, 2-хлеб, 3-мясо, 4-овощи, 5-сыр"
90 INPUT I
100 PRINT "Вам нужно в месяце";M;" ";FOOD(M,I);MES$(I)
110 PRINT
120 GOTO 70
200 K=1
210 READ A$
```

```
220 IF A$=" " THEN RETURN
230 MES$(K)=A$240 K=K+1
240 K=K+1
250 GOTO 210
300 DATA "л молока","бух. хлеба","кг мяса","кг овощей","кг сыра"
```

Итак, для того чтобы завести двумерный массив, необходимо указать в операторе DIM его имя, длину и ширину:

DIM ИМЯ ПЕРЕМЕННОЙ (РАЗМЕР 1, РАЗМЕР 2)

Естественно, что номера элемента массива не могут превышать первый - размера 1, второй - размера 2. Такой способ значительно удобнее, чем вычисление номера элемента по формулам.

Существует еще одно важное преимущество двумерного массива перед одномерным. Допустим, что мы интересуемся не только отдельными продуктами за месяц, но и некоторыми суммарными данными. Или мы интересуемся, в каком месяце выпили больше всего молока. Последний вариант программы легко модифицируется с учетом этого. Ниже приводится вариант программы, которая занимается анализом ваших данных. Попробуйте по тексту понять, что она будет делать. После этого запустите ее и сравните ваши умозаключения с результатами работы компьютера. Итак:

```
5 CLEAR 1000
10 DIM FOOD(12,5),MES$(5),FMAX(5),M(5)
15 GOSUB 200
16 FOR M=1 TO 12
18 PRINT "В течение месяца";M
20 FOR I=1 TO 5
30 PRINT "Введите необходимое количество";MES$(I);
40 INPUT FOOD(M,I)
50 NEXT I,M
60 CLS
65 GOSUB 400
70 INPUT "Какой месяц";M
80 PRINT "1-молоко, 2-хлеб, 3-мясо, 4-овощи, 5-сыр"
90 INPUT I
100 PRINT "Вам нужно в месяце";M;" ";FOOD(M,I);MES$(I)
110 PRINT
120 GOTO 70
200 K=1
210 READ A$
220 IF A$=" " THEN RETURN
230 MES$(K)=A$
240 K=K+1
250 GOTO 210
300 DATA л молока, бух. хлеба, кг мяса, кг овощей, кг сыра
400 FOR I=1 TO 5
```

```
410 FMAX(I)=0
420 FOR M=1 TO 12
430 IF FOOD(M,I)>=FMAX(I) THEN FMAX(I)=FOOD(M,I):MM(I)=M
440 NEXT M,I
450 FOR I=1 TO 5
460 PRINT "Максимальное потребление";FMAX(I);MES$(I);
465 PRINT "в месяце";MM(I)
470 NEXT I
480 RETURN
```

Надеемся, что все оценили удобство двумерного массива. Какой индекс изменяется первым - абсолютно все равно. Вы можете просматривать массив по строкам или столбцам.

Все сказанное о двумерных массивах относится и к многомерным массивам. Число измерений у массива может быть любым, лишь бы хватило памяти. Обратим внимание на один интересный момент. Пусть в программе существует строка

```
10 DIM A(1)
```

Сколько элементов в этом массиве? Один? Нет, два. Дело в том, что нумерация элементов массива ведется не с 1, а с 0. Другими словами, в массиве А имеется два элемента, А(0) и А(1). Не верите? Проверим:

```
10 DIM A(1)
20 A(0)=1
30 A(1)=2
40 PRINT A(0),A(1)
```

То же самое относится и к многомерным массивам. Любое измерение начинается с нуля. Так, массив А(4,15,8) содержит 5*16*9 элементов.

И еще одна важная деталь. Попробуйте выполнить операторы

```
A(1)=10
PRINT A(1)
```

Все получилось. Как же так? Мы ведь нигде не говорили, что хотим завести массив А. Однако Бейсик, когда находит оператор А(1), сам заводит соответствующий массив, не дожидаясь оператора DIM. Спрашивается, как он узнает размерность массива? Попробуйте следующие операторы:

```
A(5)=5
A(6)=6
A(7)=7
A(8)=8
A(9)=9
A(10)=10
```

До этого момента машина на ваши действия отвечала промптом OK. Теперь попробуем завести элемент с номером 11, вводя команду

```
A(11)=11
```

Получаем при этом сообщение об ошибке:

```
ИНДЕКС ВНЕ ДИАПАЗОНА
```

Когда Бейсик встречает оператор типа

```
A(1,2,3)=123
```

без предшествующего оператора

```
DIM A(N,M,K)
```

он автоматически заводит массив размером 11*11*11:

```
A(10,10,10)
```

Поэтому если вам нужен массив с максимальными размерами, не превышающими 10, его можно не определять в операторе DIM.

И последняя информация о массивах. Займемся вопросом, как элементы массива располагаются в памяти машины. Вы помните, что переменные бывают четырех типов.

1. Целая переменная занимает в памяти машины 2 байта. Поэтому массив целых переменных занимает объем памяти в байтах, равный произведению всех его измерений на два. Так, строка

```
100 DIM A%(3,7)
```

резервирует в памяти машины место для 4*8=32 целых чисел. Объем необходимой памяти равен 64 байтам.

2. Переменные одинарной точности требуют для себя 4 байта. Поэтому строка

```
100 DIM A!(3,7)
```

резервирует в памяти место объемом 4*4*8=128 байт.

3. Переменные с двойной точностью занимают 8 байт. Поэтому строка

```
100 DIM A#(3,7)
```

резервирует в памяти 8*4*8=256 байт.

4. Строковые переменные могут занимать различный объем памяти в зависимости от длины строки. В машине область памяти, отводимая под строковые переменные, ограничена и задается оператором

```
CLEAR M
```

где M - число байт, резервируемых под строковые переменные. Однако оператор

```
100 DIM A$(3,7)
```

резервирует первоначально по 3 байта на элемент массива, т. е. 3*4*8=96 байт памяти. Реальная длина каждого элемента массива будет добавлена позднее, при выполнении операций присваивания элементам массива.

Приведем общую формулу для подсчета памяти, резервируемой под произвольный массив. Массив A(N1,N2,...,NK) требует память размером

14 + (K*2) + M*((N1+1)+(N2+1)+...+(NK+1))

где K - число измерений массива, M равно количеству байтов, занимаемых переменной заданного типа.

15. ЛОГИЧЕСКИЕ ОПЕРАЦИИ

Раздел, который даст читателю информацию о принципиально новых приемах работы с компьютером

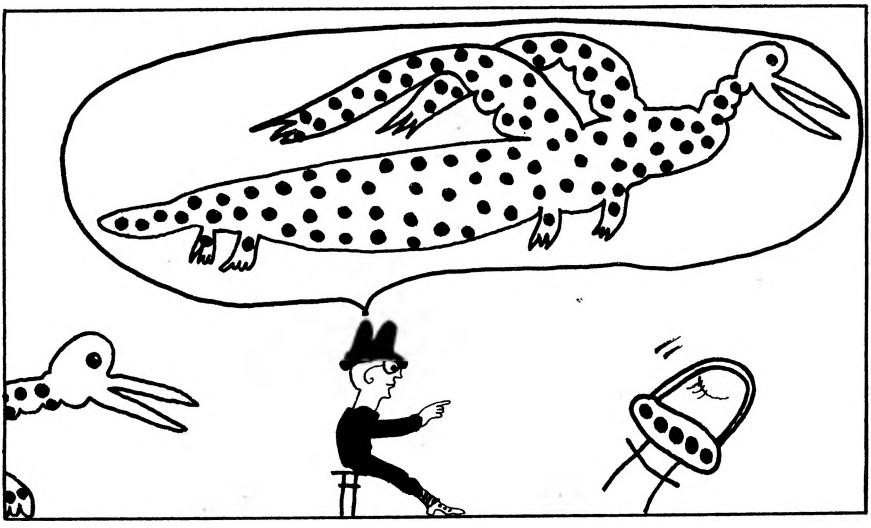
Может возникнуть вопрос, зачем нам логические операции? Мы умеем складывать и вычитать, умножать и делить. Нам не составляет никакого труда возводить числа в степень и вычислять синусы и косинусы. Чего еще желать?

Рассмотрим следующую ситуацию. Допустим, у вас завтра день рождения. Вы пригласили много друзей и теперь решаете проблему, чем их занять. Конечно, можно придумать массу развлечений вроде коллективного просмотра телевизионных программ или прослушивания записей популярных исполнителей. Но вы идете в ногу со временем. У вас имеется компьютер. Естественно, ваши друзья обязательно попросят вас показать, как он работает. Вы можете показать им, чему равно 2*2 или 5 в кубе. Но это не очень зрелищно. Вот бы научить компьютер развлекать ваших гостей! Для этого лучше всего подходит какая-нибудь игра.

Давайте научим ваш компьютер телепатии. Он будет угадывать, что загадали ваши гости. Естественно, телепатия - вещь сложная. Поэтому пусть присутствующие загадывают не что попало, а нечто определенное, скажем животных. Машина будет задавать им вопросы. На эти вопросы нужно отвечать "да" или "нет". В результате животное будет отгадано. Если случайно машина не сможет отгадать животное (задумали что-нибудь экзотическое), она попросит сообщить ей, что это за зверь и чем он примечателен.

Начнем писать программу. Она будет довольно длинная, поэтому наберитесь терпения. Для начала обратите внимание, что программа будет работать со строковыми переменными. Поскольку зверей в мире очень много, вы должны иметь в виду, что для хранения всей этой информации необходимо зарезервировать достаточно памяти. Поэтому первая наша строка будет иметь вид

```
10 CLEAR 10000
```



Далее вам нужно определить, где вы будете хранить названия зверей и их отличительные признаки. Для этого вам потребуется завести два массива. Один массив O$(50) - это названия животных. Другой массив Q$(50,50) - это вопросы, которые будет задавать машина. Поэтому следующая строка будет иметь вид

```
20 DIM O$(50),Q$(50,50)
```

Теперь очистим экран:

```
30 CLS
```

Далее сообщим имена первых двух животных. Пусть это будут слон и мышь. Отличаются они друг от друга размерами. Поэтому в массив вопросов введем слово "большое":

```
40 O$(2)="Слон":O$(1)="Мышь":Q$(1,2)="Большое"
```

Научим машину задавать вопросы:

```
50 M=2
60 PRINT "Загадай животное"
70 I=1
80 FOR J=I+1 TO M:IF Q$(I,J)="" THEN 110
90 PRINT Q$(I,J);
100 INPUT A$:IF A$="ДА" THEN (ПРОДОЛЖАТЬ)
110 NEXT J
```

Рассмотрим этот фрагмент программы. Вроде, все понятно. Поскольку сейчас машина знает только двух животных, то выбирать она может только из двух (M=2). Начинает она с первого вопроса (I=1). Если в массиве вопросов есть вопрос на соответствующем месте, машина выведет его на экран и попросит вас на него ответить. В случае положительного ответа программа будет продолжать работу.

Если вы ответите отрицательно, будет задан следующий вопрос. Он будет задан и в случае отсутствия текста вопроса в массиве Q$.

Теперь очень важное соображение. Как отвечать машине? Вы можете набрать ДА, Да или да. Все это утвердительные ответы. Но в том виде, в каком мы только что ввели строку 100, программа будет реагировать правильно только на ответ ДА заглавными буквами. Это явно неудовлетворительно. Ведь смысл ответа "да" не зависит от размера букв. Вы, конечно, сразу можете подсказать выход из положения. Добавим еще две строки:

```
102 IF A$="да" THEN (ПРОДОЛЖАТЬ)
104 IF A$="Да" THEN (ПРОДОЛЖАТЬ)
```

Ну что ж, это правильное решение, но оно не единственное. Можно объединить строки 100 - 104 в одну. Видоизмененная строка 100 будет иметь вид

```
100 INPUT A$:IF A$="Да" OR A$="ДА" OR A$="да"
    THEN (ПРОДОЛЖАТЬ)
```

Здесь мы применяем новый, пока еще не известный нам оператор OR. Те, кто знает английский язык, естественно, могут перевести OR как "или". Интересно отметить, что, изучая языки программирования, вы одновременно получаете некоторые сведения из английской грамматики. В свое время имела хождение фраза: "Знаю английский в пределах Алгола". Алгол - тоже язык программирования.

В качестве упражнения модифицируйте строку 100 так, чтобы машина понимала также и ответ Д.

Теперь нам предстоит продолжить программу. Что делать машине в зависимости от нашего ответа? Ведь она должна задавать нам не какие попало вопросы, а только те, которые ведут ее к цели (отгадке). Вы, наверное, обратили внимание, что массив вопросов двумерный. Поэтому по нему можно двигаться по разным траекториям. Первый индекс говорит нам номер шага, второй - номер вопроса на данном шаге. Учитывая это, продолжение программы может иметь вид

```
120 PRINT O$(I);:INPUT A$
125 IF A$="Да" OR A$="ДА" OR A$="да" THEN 130 ELSE 140
130 PRINT "Победа!!!" GOTO 60
140 PRINT "А что вы загадали?":INPUT O$(M+1)
150 PRINT "Чем отличается от того, что я сказала?"
155 INPUT O$(I,M+1)
160 M=M+1:CLS
170 GOTO 60
180 I=J
190 IF I<M THEN 80
200 IF I=M THEN 120
210 END
```

Далее следует исправить строку 100 так, чтобы она приняла вид

```
100 INPUT A$:IF A$="Да" OR A$="ДА" OR A$="да" THEN 180
```

Разберем действия машины. Если вы отвечаете на вопрос в строке 100 утвердительно, то машина переходит к выполнению шага с номером, соответствующим номеру вопроса. Если номер шага меньше числа известных машине животных, она продолжает спрашивать. Если все животные исчерпаны, она выводит на экран название того, которое нашла, и спрашивает, угадала или нет. Если да, наступает всеобщий восторг и удивление. Если нет, машина, огорчившись своей неудачей, попробует научиться отгадывать еще и это животное. Она включит его в список зверей (O$(50)) и попросит вас указать, чем именно этот зверь от подобных отличается. Эту информацию она потом использует в качестве вопроса. То же самое произойдет, если мы на каком-либо шаге исчерпаем все вопросы.

Итак, запускайте программу. Сначала вам покажется, что машина очень глупая. Но постепенно процесс обучения будет брать свое и машина будет все ближе и ближе к цели.

Теперь вернемся к логическим операторам. Как вы понимаете, их сфера деятельности лежит там, где что-нибудь с чем-нибудь сравнивается. Для логических величин существуют два значения: TRUE (правильно) и FALSE (неправильно). Подобно обычным числам, на логические величины действуют такие операторы, как сложение, умножение и инвертирование. Естественно, они несколько отличаются от аналогичных арифметических операций. В качестве сложения выступает операция OR. Как она работает? Вспомним строку 100 нашей программы:

```
100 INPUT A$:IF A$="Да" OR A$="ДА" OR A$="да" THEN 180
```

Условие, сформулированное в операторе IF,

```
A$="ДА" OR A$="Да" OR A$="да"
```

будет выполнено, если хотя бы одно из условий

```
A$="ДА"
A$="Да"
A$="да"
```

будет удовлетворено. Другими словами, мы имеем таблицу логического сложения:

```
TRUE OR FALSE=TRUE
TRUE OR TRUE=TRUE
FALSE OR FALSE=FALSE
```

Теперь займемся умножением. Умножением для логических величин является оператор AND ("и"). Таблица умножения имеет вид

```
TRUE AND TRUE=TRUE
FALSE AND TRUE=FALSE
FALSE AND FALSE=FALSE
```

Другими словами, если условие состоит из произведения различных условий, то оно будет выполнено, если будут выполнены все входящие в произведение условия.

Следующим оператором является оператор инвертирования. Его еще называют оператором отрицания. Он называется NOT, что в переводе означает "нет". Таблица отрицания имеет вид

NOT TRUE=FALSE
NOT FALSE=TRUE

Интересной особенностью логических операций является их взаимосвязь. Сравните таблицы сложения и умножения. Если поменять местами TRUE и FALSE, таблица сложения превратится в таблицу умножения. Такая замена производится оператором NOT согласно таблице отрицания. В этом факте заключена на самом деле важная идея. Какое состояние считать истинным? Вообще говоря, все равно. Таким образом, появляются два типа логики - "положительная" и "отрицательная". Они аналогичны, но в них друг по отношению к другу все наоборот.

В заключение приведем сводку правил по пользованию логическими операторами в случае условного ветвления программы, т. е. внутри оператора IF...THEN:

IF УСЛОВИЕ OR УСЛОВИЕ THEN ДЕЙСТВИЕ

- действие будет выполнено, если хотя бы одно из условий будет выполнено;

IF УСЛОВИЕ AND УСЛОВИЕ THEN ДЕЙСТВИЕ

- действие будет выполнено, если оба условия будут выполнены одновременно;

IF NOT УСЛОВИЕ THEN ДЕЙСТВИЕ

- действие будет выполнено, если условие не выполняется.

В качестве упражнения попробуйте предсказать, что появится на экране дисплея при выполнении команд

```
IF 2=5 OR 3=3 THEN PRINT "Выполнено"
IF 2=5 AND 3=3 THEN PRINT "Выполнено"
IF NOT 2=5 THEN PRINT "Выполнено"
IF NOT 2=5 AND 3=3 THEN PRINT "Выполнено"
IF NOT 2=5 AND NOT 1=2 THEN PRINT "Выполнено"
```

Проверьте ваши ответы с помощью компьютера.

16. СТРОКОВЫЕ ПЕРЕМЕННЫЕ

Автоматизация труда писателей, переводчиков, цензоров и редакторов

Вспомним, что такое строковая переменная. Она обычно содержит в себе часть самого обыкновенного текста. Это может быть предложение, отдельное слово, одна или несколько букв, любые символы и цифры. Чтобы отличать строковые переменные от остальных, за их именем ставят символ $. Попробуйте вводить строковые переменные в компьютер, используя программу

```
10 CLEAR 50
20 PRINT "Введите строку текста и нажмите (ВК)"
30 INPUT A$
40 PRINT A$;"В памяти компьютера"
50 PRINT
60 GOTO 20
```

Помните, зачем нужен оператор CLEAR? Попробуйте различные варианты текстов. Введите ваше имя и фамилию, математические формулы и т. д., например:

```
Моя фамилия Иванов
2+3=5
5>3, 4>2
```

А что это за сообщение появилось на экране:

```
? ЛИШНИЕ ДАННЫЕ
```

Что это, в машине завелся цензор? Нет, просто оператор INPUT принимает данные до запятой или точки с запятой. Все, что следует за ними, компьютер воспринимает как новую переменную. Если в операторе INPUT такая переменная не указана, появляется сообщение об ошибке. Как же тогда вводить запятую и точку с запятой? Очень просто. Заключите всю фразу в кавычки:

```
"5>3, 4>2"
```

Все в порядке?

Теперь займемся размерами строковых переменных. В ответ на запрос нашей программы введите фразу "Процесс работы с персональной ЭВМ Корвет доставляет мне массу удовольствия каждый день. Мне она очень нравится". А это что за сообщение появилось:

```
НЕТ ПАМЯТИ ДЛЯ СТРОК В 20
```

Это значит, что зарезервированная нами память под строковые переменные недостаточна для такой длинной строки. Тот, кто внимательно читал предшествующие страницы, легко модифицирует нашу программу. Для этого он изменит строку 20 следующим образом:

```
20 CLEAR 500
```

Итак, мы снова столкнулись с оператором CLEAR. В переводе с английского это слово означает "очистить". В соответствии с переводом этот оператор заставляет машину очистить все переменные и зарезервировать под строковые переменные число байт, равное выражению, стоящему после CLEAR. Поскольку оператор CLEAR очищает все
переменные, его нельзя ставить в программе где попало. Обычно это один из первых операторов программы. Если вы хотите просто очистить переменные, не изменяя величины пространства под строковые переменные, используйте просто CLEAR без числа или арифметического выражения.

Сколько памяти может быть выделено под строковые переменные? Сколько угодно в пределах общего объема оперативной памяти ЭВМ, отводимой Бейсику. Помните, однако, что кроме строковых существуют еще и обыкновенные переменные, не говоря уже о самой программе. Поэтому не отводите под строковые переменные места больше, чем нужно.

Теперь займемся строковыми переменными. Что нас может интересовать в этом вопросе? Ну, во-первых, хотелось бы знать длину строки. Для этого служит специальный оператор

LEN (СТРОКОВАЯ ПЕРЕМЕННАЯ ИЛИ ТЕКСТ)

Как вы догадались по его внешнему виду, это функция, подобная EXP, SIN и т. п. Разница в том, что LEN работает со строковыми переменными. Эта функция вычисляет длину любой строки, фигурирующей в качестве аргумента. Поскольку LEN является функцией, она не может использоваться сама по себе. Она должна быть включена в оператор присваивания, арифметическое выражение, использоваться совместно с оператором PRINT и т. д.

Попробуйте набрать команду

LEN ("Это строка")

В результате появится сообщение об ошибке:

ОШИБКА СИНТАКСИСА

Попробуйте теперь по-другому:

PRINT LEN ("Это строка")

Вы получите в ответ:

10

Теперь модифицируем нашу программу. Добавьте еще одну строку:

35 PRINT "Длина строки равна";LEN(A$)

Теперь мы будем точно знать длину любого сообщаемого машине текста. Между прочим, эта функция позволяет научить программу, а следовательно и машину, разбираться с тем, что в нее вводят. По-

пробуйте научить ее отличать 7-буквенные выражения и сообщать вам о них.

А сейчас пойдем дальше. До сих пор мы имели дело со строковыми переменными как с целыми кусками текста. Нельзя ли как-то разделить строковую переменную? Можно. Для начала заметим, что любая строка состоит из левой части, середины и правой части. В соответствии с этим существуют три функции, работающие с этими частями: LEFT (от англ. LEFT - левая), RIGHT (от англ. RIGHT - правая) и MID (от англ. MIDdle - середина).

Как этими функциями пользоваться? Ну, прежде всего, раз это функции, их нужно использовать совместно с операторами типа PRINT, присваивания и т. д. Как эти функции работают? Разберем их действия на примере. Введите в машину следующие команды:

```
A$="Я персональная ЭВМ Корвет"
PRINT LEFT$(A$,5)
```

На экране появится:

```
Я пер
```

Теперь попробуйте выполнить команду

```
PRINT RIGHT$(A$,5)
```

На экране появится:

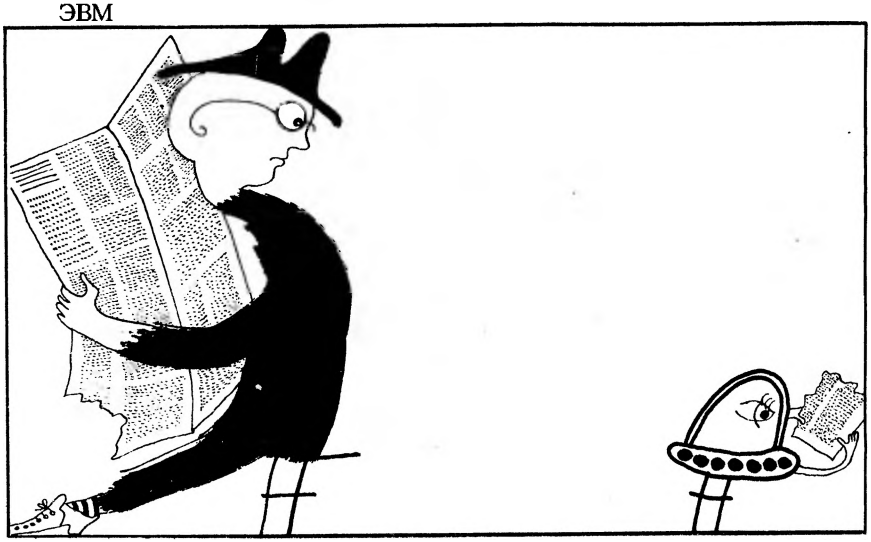
```
Корвет
```

Теперь попробуйте добыть что-нибудь из середины:

```
PRINT MID$(A$,15,4)
```

На экране появится:

```
ЭВМ
```



Вы, наверное, уже поняли, как используются аргументы у этих функций. Если нет, мы сейчас разберемся с этим вопросом.

Функции LEFT и RIGHT имеют формат

LEFT$ (СТРОКА, ДЛИНА)
RIGHT$ (СТРОКА, ДЛИНА)

Строка - это либо явно заданный текст типа "Привет, я ваша ЭВМ", либо имя строковой переменной. Число, фигурирующее в качестве длины, определяет, сколько символов слева или справа нужно взять из строки.

Функция MID$ имеет три аргумента. Это и понятно: нам нужно указать, откуда взять строку, с какого символа начинать и сколько их взять. Соответственно эта функция имеет формат

MID$ (СТРОКА, НОМЕР ПЕРВОГО СИМВОЛА,
ЧИСЛО СИМВОЛОВ)

Потренируйтесь с различными строками, чтобы устранить возможные неясности.

Может возникнуть вопрос, зачем нам все эти функции? Попробуем на него ответить. Допустим, мы хотим написать программу, которая обнаруживала бы в тексте строки, содержащие определенные ключевые слова. Как нам это сделать?

Рассмотрим пример программы, которая ищет слово "ЭВМ":

```
10 CLS
20 INPUT "Введите строку";A$
30 N=LEN(A$)
40 FOR I=1 TO N-3
50 IF MID$(A$,I,3)="ЭВМ" THEN MES$="В этой строке есть ЭВМ"
   ELSE MES$="В этой строке нет ЭВМ"
60 NEXT I
70 PRINT MES$
80 GOTO 20
```

Запустите эту программу. Вы видите, что Корвету можно поручать чтение писем и другой полезной информации. Понятно ли вам, как работает эта программа? Собственно поиском слова "ЭВМ" занят цикл 40 - 60. Машина рассматривает введенную строку через окошечко длиной в три символа, сдвигая это окошечко каждый раз на один символ вправо. Если нужное слово будет найдено, то вам об этом сообщат.

Существует еще одна функция, решающая подобную задачу. Она называется INSTR и имеет вид

INSTR (N, СТРОКА 1, СТРОКА 2)

Что эта функция делает? Она определяет, содержит ли строка 1 строку 2. Если не содержит, значение функции приравнивается нулю. Если содержит, значение функции приравнивается номеру позиции в

строке 1 символа, с которого начинается строка 2. Число N задает номер начальной позиции, с которой начинается поиск.

Примеры:

```
PRINT INSTR (1,"Я ЭВМ Корвет","ЭВМ")
3
PRINT INSTR (3,"Я ЭВМ Корвет","ЭВМ")
3
PRINT INSTR (1,"Я ЭВМ Корвет","ваша")
0
```

В последнем случае в строке "Я ЭВМ Корвет" слово "ваша" отсутствует. Тот же ответ (0) вы получите, если введете команду

```
PRINT INSTR (5,"Я ЭВМ Корвет","ЭВМ")
```

Это значит, что если начать поиск с пятой позиции, то в предложении "Я ЭВМ Корвет" слово "ЭВМ" также не обнаружится.

И еще одна полезная функция. Она позволяет создавать строковую переменную, состоящую из определенного числа пробелов. Называется эта функция SPACE$ (пространство). Она имеет один-единственный аргумент - требуемое число пробелов, так что ее формат следующий:

```
SPACE$(N)
```

где N - число пробелов. Введите, например, команды

```
A$="Я ЭВМ Корвет"
PRINT A$
```

На экране появится:

```
Я ЭВМ Корвет
```

А теперь введите

```
B$=SPACE$(12)
PRINT B$,A$
```

На экране появится та же надпись, но с отступом от начала строки на 12 пробелов.

Теперь перейдем к процедуре сложения строковых величин. Да, да, не удивляйтесь. В этом смысле строковые переменные полностью равноправны с целыми и действительными. Попробуйте ввести команды

```
A$="Я ЭВМ"
B$="Корвет"
C$=A$+" "+B$
```

Как вы думаете, чему равна C$? Это легко увидеть, если попросить Корвет вывести ее на экран:

```
PRINT C$
```

Увидим на экране:

Когда мы хотим добавить к одной строке другую, мы используем символ сложения. При этом вторая строка добавляется в конец первой. Можно добавлять как в явном виде текст, заключенный в кавычки, так и складывать различные строковые переменные и функции типа RIGHT$, LEFT$, MID$ и т. д.

Итак, в наших руках оказалось мощнейшее средство для работы с текстами. С одной стороны, для вырезания из строк нужных мест мы имеем функции типа RIGHT$, LEFT$, MID$, а с другой стороны, мы можем прибавить к строке любой текст. Давайте напишем программу, которая бы автоматически переводила слово computer на русский язык. Для этого сначала нужно ввести в машину текст. Это можно сделать, например, с помощью оператора INPUT. Однако удобнее для этой цели использовать операторы READ и DATA. Сначала введем пробный текст:

```
1000 DATA "Computer Корвет очень удобен для научных расчетов"
1010 DATA "Он представляет собой одноплатный computer"
1020 DATA "Персональный computer должен быть у каждого"
1030 DATA "I"
```

Мы не будем заниматься сейчас точным сопоставлением английской и русской грамматики. Все, что будет происходить, носит исключительно иллюстративный характер. Наша задача - научить машину находить в тексте слово computer или Computer и заменять его на "Компьютер". Приводим текст программы:

```
10 CLS
20 C=LEN ("Computer")
30 READ A$
40 IF A$="I" THEN END
50 N=INSTR(1,A$,"Computer")
60 IF N=0 THEN N=INSTR(1,A$,"computer")
70 IF N=0 THEN 120
80 IF N>1 THEN L$=LEFT$(A$,N-1) ELSE B$=" "
90 NA=LEN(A$)
100 IF N<NA-C THEN R$=RIGHT(A$,NA-N C+1) ELSE R$=" "
110 A$=L$+"Компьютер"+R$
120 PRINT A$
130 GOTO 30
```

Запустите программу и посмотрите, как Корвет справится с обязанностями переводчика. В качестве упражнения научите программу переводить еще слово COMPUTER, написанное заглавными буквами.

Теперь займемся представлениями различных букв в машине. Внимательные читатели помнят, что еще на заре нашей деятельности мы много раз говорили, что машина понимает только числа, причем представленные в двоичной системе счисления. А как же тогда Корвет научили понимать буквы и даже целые слова и предложения? Ответ простой. Каждому символу, нарисованному на клавишах клавиа-

туры, ставится в соответствие определенное число. Одним из стандартов являются так называемые ASCII коды. Расшифровывается эта аббревиатура как American Standart Code for Information Interchange - американский стандартный код для обмена информацией. Сейчас мы с ним познакомимся. Для этого напишем программу

```
10 REM Программа определения ASCII кодов клавиатуры
20 A$=INKEY$:IF A$="" THEN 20
30 PRINT A$, ASC(A$):GOTO 20
```

(Обратите внимание на то, что в строке 20 два последовательных знака кавычек стоят подряд, без интервала между ними.)

Что эта программа делает? Для того чтобы в этом разобраться, сначала поймем, как работают два новых для вас оператора INKEY$ и ASC(A$). Первый из них - оператор ввода с клавиатуры. В отличие от оператора INPUT он выдает только код нажатой клавиши. Если ни одна клавиша не нажата, то выдается нулевое значение. При вводе данных с помощью оператора INKEY$ не нужно нажимать ВК после нажатия требуемой клавиши. Второй оператор ASC(A$). Это функция, которая принимает значение, равное ASCII коду символа A$.

Таким образом, программа "зависает" на строке 20 до тех пор, пока не будет нажата какая-либо кнопка на клавиатуре (т. е. A$ не равно ""). После этого на экран выводится символ A$ и соответствующий ему ASCII код. Запустите программу. Попробуйте нажимать разные клавиши на алфавитно-цифровом поле. Попробуйте верхний и нижний регистры, латинский и русский алфавиты. Нажмите на ВК. Оказывается, этой клавише тоже соответствует код - 13.

А существует ли обратная функция, которая бы переводила число в соответствующий символ? Конечно. Называется она CHR$(N), где N - число от 0 до 255. Название этой функции происходит от английского character - символ. Для того чтобы посмотреть список ASCII кодов, напишите программу

```
10 FOR I=0 TO 255
20 PRINT I;" ";CHR$(I);" ";
30 NEXT I
```

Запустите ее. Вы видите, что на экране сначала что-то замелькало в верхней строке и лишь потом, начиная с I=32, начала выводиться регулярная информация. Дело в том, что символы с кодом I<32 используются как служебные. Они управляют положением курсора, вызывают стирание строк, всего экрана и т. д. Первый не служебный символ в ASCII таблице - пробел. Его код 32.

Теперь мы начинаем кое-что понимать. Раз машина воспринимает буквы как коды, т. е. просто числа, она, наверное, умеет их сравнивать. Совершенно верно. Более того, она может сравнивать целые строки. Для иллюстрации этого феномена вполне подходит следующая программа:

```
10 INPUT "Строка 1";S1$
20 INPUT "Строка 2";S2$
30 IF S1$>S2$ THEN PRINT S1$ ELSE PRINT S2$
40 GOTO 10
```

Эта программа вводит две строки и сравнивает их. Ту, которая больше, она выводит на экран.

Как происходит скрытый от нашего глаза процесс сравнения? Очень просто. Символ, имеющий больший код, считается старшим. Если мы сравниваем строки, то сравнение происходит символ за символом. Как только обнаруживаются несовпадающие символы в сравниваемых строках, то та строка, у которой первый из несовпадающих символов старше, считается большей. В отношении строковых переменных понятие больше - меньше не очень удобно. Тут чаще говорят предшествующий - последующий или старший - младший. Названия условны. Просто нужно раз и навсегда договориться, по каким критериям сортировать строковые переменные. Мы устанавливаем: по возрастанию ASCII кода.

Кстати, это не всегда совпадает с алфавитным порядком. Сравните, например, пользуясь последней программой, буквы Д и Ц. Вы получите, что Д старше. Почему? Это исторический факт. Дело в том, что поскольку ASCII коды придумали в Америке, то и порядок расстановки букв соответствует латинскому алфавиту. А в нем латинское C стоит перед D. Но это все дело привычки.

Осталась маленькая деталь. Вы знаете, что существует множество систем счисления. Наиболее актуальными для компьютеров являются двоичная, восьмеричная и шестнадцатеричная. Однако переводить числа из десятичной системы в эти три - дело, требующее определенного навыка. Вот бы машина сама могла это делать! Оказывается, может. Для этой цели существуют целых три функции:

```
BIN$(N), OCT$(N), HEX$(N)
```

где N - число, которое мы хотим преобразовать. Попробуйте набрать команду

```
PRINT "17", BIN$(17), OCT$(17), HEX$(17)
```

На экране появится:

```
17 10001 21 11
```

Потренируйтесь с другими числами.

В ряде случаев бывает необходимо переводить арифметические выражения в строковые переменные или, наоборот, строковые переменные, включающие в себя цифровые символы, в соответствующие числа. Для этого служат функции STR$ (арифметическое выражение) и VAL (строковая величина). Функция VAL превращает строковую величину в соответствующее число. Если же строковая величина является истинно текстовой и поэтому превращение ее в число довольно бессмысленно, то значение функции VAL приравнивается нулю, например:

```
PRINT 2*VAL("2")
4
PRINT "2+"+STR$(2)
2+2
PRINT VAL("1"+"."+"2")
1.2
PRINT VAL("Вася")
0
```

Вот и все, что нам необходимо знать о строковых переменных. Надеемся, что вы овладели ими в совершенстве.

17. ИСКУССТВО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ БЕЙСИК

Раздел, в котором читатель почерпнет массу полезных сведений о правильном стиле написания программ

Пришла, наконец, пора поговорить о том, как нужно писать программы. Вроде бы все в этом вопросе понятно, однако существует ряд соображений, которые полезно принимать во внимание.

С точки зрения методики написания программ можно условно разделить программистов на два класса. Одни из них (как правило, это начинающие программисты) пишут программы следующим образом. Допустим, что такому человеку требуется научиться вычислять какую-нибудь хитрую функцию, например гиперболический косинус. Он, потратив немного времени, сочиняет необходимую программу. Получив значение требуемой функции для различных значений аргумента, он может продолжать свою работу и использовать в ней полученные числа. Но тут ему приходит в голову, что разумнее было бы заставить машину саму выполнять те действия с вычисленной функцией, которые требуются.

В результате размышлений программа пополняется новыми операторами. Дальше - больше. Жизнь ставит перед вами все новые и новые задачи. Соответственно усложняется и ваша программа. Вы уже за давностью лет забыли, какие идеи закладывались в основу вашего программного продукта. Поэтому теперь вместо приятного занятия процесс работы с ЭВМ превращается в каторжный труд. Программа сделалась трудно управляемой, и вы сами с трудом в ней разбираетесь.

Этот метод программирования условно называется программированием снизу. Этот термин подчеркивает, что программист начинает написание программ с процедур и подпрограмм и только после этого пишет программу, которая объединяет эти разрозненные куски в одно целое. Если программа небольшая, возможно, она и будет работать. В случае больших программ такой путь неприемлем, поскольку логику работы программы очень трудно сделать единой для различ-

ных ее частей. В результате начинаются сбои и зависания программы. Отыскать ошибки в такой программе очень трудно, порой невозможно.

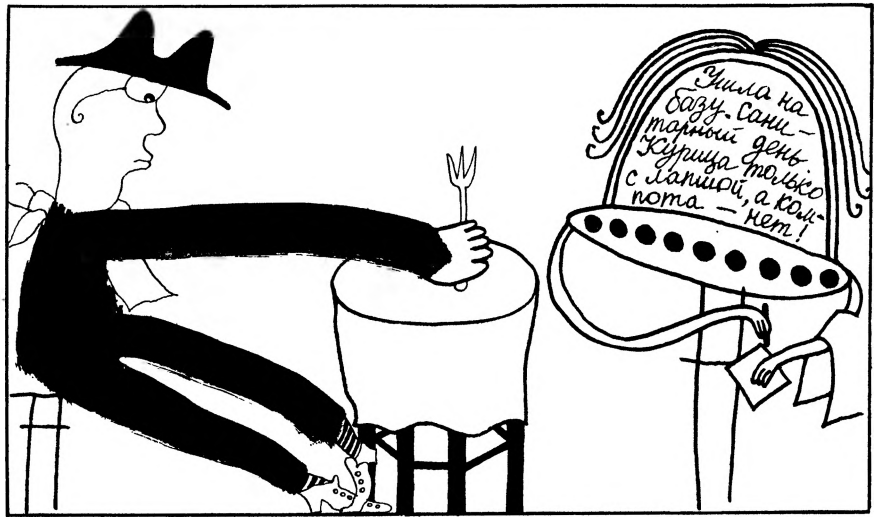
Существует другой способ написания больших программ. Он имеет условное название программирования сверху. Смысл этого термина в том, что подразумевается написание первоначально той части программы (будем называть ее управляющей), которая должна вызывать различные подпрограммы. После того как все связи основной программы с подпрограммами отлажены и проверены, можно приступать к написанию подпрограмм.

Чем такой способ удобнее? При отладке управляющей программы очень просто локализовать ошибки. В процессе отладки все подпрограммы можно заменить "пустышками", задача которых - лишь сообщать, что основная программа данную подпрограмму вызывала, например:

```
100 GOSUB 1000 'Вызов функции 1
1000 PRINT "Функция 1 вызвана"
1010 RETURN
```

Таким образом, если программа начинает плохо работать, это значит, что ошибка таится где-то в связи подпрограммы с основной управляющей программой. Для сравнения заметим, что при программировании снизу причина плохой работы программы может заключаться не только в этом, но и в любой из подпрограмм, что в свою очередь сильно расширяет возможное поле для поиска ошибки.

Проиллюстрируем стиль программирования сверху на простом и полезном для физиков примере. Допустим, что вам нужно рассчитывать вольт-амперные характеристики газового разряда. Вам принесли лист бумаги, на котором записаны показания приборов установки, изучающей низкотемпературную плазму. Перед вами стоит задача: во-первых, ввести эти данные в машину, во-вторых, исправить те ошиб-



ки, которые возникли в процессе ввода (они неизбежны), в-третьих, провести обработку данных.

Как разработать такую программу? Мы являемся опытными программистами и поэтому будем программировать сверху. При этом способе программирования напрашивается приятная аналогия. Вообразите себя сидящим в зале хорошего ресторана. У вас есть два способа добыть себе пищу. Первый способ - начать заказывать блюда по одному, спрашивая официанта: "А у вас это есть?", время от времени вспоминая, что вы хотели еще что-нибудь из закусок, хотя уже подали первое. Это - программирование снизу. Для программирования сверху в ресторанах предусматривается меню. Человек обстоятельный сначала его изучит, продумает схему обеда и только после этого позовет официанта.

Будем действовать в этом стиле, чтобы быть уверенными, что мороженое подадут на десерт, а не перед горячими блюдами. Итак, составим меню. Мы договорились, что в программе будут реализованы три функции. Соответственно начнем программировать:

```
5 CLS
10 PRINT "1.Ввод данных"
20 PRINT "2.Просмотр и редактирование"
30 PRINT "3.Вывод результата"
40 INPUT C
50 ON C GOSUB 1000,2000,3000
60 GOTO 10
```

Этот фрагмент программы играет роль управляющей программы. Она выводит нам для обозрения меню (строки 10 - 30) и просит выбрать из него необходимую строку (строка 40). После выбора управление передается на нужную подпрограмму. Как мы и договаривались, для начала заменим подпрограммы пустышками:

```
1000 PRINT "Функция 1"
1990 RETURN
2000 PRINT "Функция 2"
2990 RETURN
3000 PRINT "Функция 3"
3990 RETURN
```

Запустите программу и заставьте ее правильно работать. Теперь вы можете быть уверены, что связи управляющей программы с подпрограммами отлажены.

Займемся подпрограммами. Текст подпрограммы ввода данных:

```
1000 PRINT "Подпрограмма ввода данных":PRINT
1010 INPUT "Введите количество временных интервалов";N
1020 DIM A(4,N)
1030 INPUT "Введите временной интервал (с) между отсчетами";D
1040 FOR K=1 TO N
1050 PRINT "Напряжение (В) для T (";K;")=";:INPUT V
```

```
1060 A(1,K)=V
1070 PRINT "Ток (А) для T(";K;")=";:INPUT I
1080 A(2,K)=I
1090 NEXT K
1990 RETURN
```

Что делает эта подпрограмма? Вы помните, что вы хотели перенести данные о вольт-амперных характеристиках разряда в память машины. Для этого вы говорите машине, сколько точек по времени вы имеете, затем величину каждого временного интервала и после этого вводите вольт-амперную характеристику. Обратите внимание, что в операторе DIM мы использовали переменную N, а не число. Другими словами, Бейсик позволяет определять массивы переменного размера, т. е. размер массива может изменяться в ходе работы программы.

Теперь вторая подпрограмма. Вы помните, что она должна дать нам возможность исправить ошибки, если таковые обнаружатся. Вот ее текст:

```
2000 PRINT "Подпрограмма редактирования данных"
2010 PRINT "T","V","I"
2020 FOR T=1 TO N
2030 PRINT T,A(1,T),A(2,T)
2040 NEXT T
2050 PRINT:PRINT "Все правильно?(<Д>а/<Н>ет)":INPUT A$
2060 IF A$="N" OR A$="n" OR A$="Н" OR A$="н" THEN 2070
     ELSE RETURN
2070 INPUT "Для какого T нужно изменить данные?";K
2080 INPUT "Введите новое значение V";A(1,K)
2090 INPUT "Введите новое значение I";A(2,K)
2110 GOTO 2010
```

Как работает эта подпрограмма? В ней вы видите операторы ветвления, логические операторы, циклы, массивы - словом, все, что мы недавно изучали. Обратите внимание на строку 2060. Зачем столько условных операторов? Очень просто. В каком бы режиме ни находилась клавиатура, будь то верхний или нижний регистр, латинский или русский алфавит, программа все равно поймет ответ Н (НЕТ). В строке 2050 специально подчеркнуто, что правильным является ответ Д или Н, т. е. не надо отвечать полностью ДА или НЕТ.

Теперь займемся обработкой вольт-амперных характеристик. Эта подпрограмма имеет вид

```
3000 PRINT "Результаты расчета мощности и энерговклада"
3010 REM Расчет мощности
3020 FOR T=1 TO N
3030 A(3,T)=A(1,T)*A(2,T)
3040 NEXT T
3050 REM Расчет энерговклада
```

```
3060 A=0
3070 FOR T=2 TO N
3080 A=A+(A(3,T-1)+A(3,T))/2*D:A(4,T)=A
3090 NEXT T
3100 PRINT "T";"V","T","W","Q"
3110 FOR T=1 TO N
3120 PRINT T;" ";A(1,T),A(3,T),A(4,T)
3990 RETURN
```

Попробуем по тексту программы понять, что она делает. Одна из важнейших характеристик газового разряда - вкладываемая в него мощность - рассчитывается по формуле W=V*I. Этим расчетом занят цикл 3020 - 3040. Следующая характеристика разряда - энерговклад. Вообще говоря, для точного его определения нужно интегрировать вкладываемую мощность по времени. Однако можно оценить эту величину, просто просуммировав средние значения мощности, умноженные на временной интервал. Это делает цикл 3070 - 3090. Обратите внимание на строку 3080. Вспомните, что мы говорили о последовательности выполнения арифметических операций.

Итак, программа написана. Можно ее запустить и начать физические исследования на какой-нибудь плазменной установке. Попробуйте поработать с программой и убедитесь, насколько это удобно.

Но это еще не все. Предположим, что прошло время и мы хотим дополнить нашу программу еще какой-нибудь функцией. Например, мы заинтересовались сопротивлением разрядного промежутка. Для этого нам нужно вычислить его по формуле R=V/I. В программе для этой цели нужно добавить соответствующую процедуру. Сейчас вы увидите, насколько тривиальна эта процедура в случае программирования сверху. Добавьте в вашу управляющую программу следующую строку:

```
35 PRINT "4. Расчет активного сопротивления разряда"
```

и отредактируйте строку 50:

```
50 ON C GOSUB 1000,2000,3000,4000
```

Теперь осталось написать подпрограмму расчета сопротивления:

```
4000 PRINT "Расчет активного сопротивления"
4010 PRINT "T","R"
4020 FOR T=1 TO N
4030 R=A(1,T)/A(2,T)
4040 PRINT T,R
4050 NEXT T
4060 RETURN
```

Очень просто, не правда ли? Таким образом, программу, организованную с помощью меню, несложно пополнять различными подпрограммами. При этом мы не затрагиваем остальных процедур и функций и, следовательно, никак не нарушаем их работы. Надеемся, что вы оценили все преимущества данного стиля программирования.

В качестве упражнения попробуйте оценить значение напряженности электрического поля в вашем разрядном промежутке. Это делается по формуле E=V/L, где L - длина разрядного промежутка. Для решения этой задачи вам необходимо модифицировать подпрограмму ввода данных, чтобы иметь возможность ввести величину L и добавить в программу новую функцию. Надеемся, что ваши упражнения будут успешными. Если нет, то ниже мы приводим нашу версию.

Прежде всего модифицируем меню:

```
37 PRINT "5. Расчет напряженности электрического поля"
50 ON C GOSUB 1000,2000,3000,4000,5000
```

Затем модифицируем подпрограмму ввода данных. Для этого добавим строку

```
1015 INPUT "Введите длину разрядного промежутка";L
```

И, наконец, напишем соответствующую процедуру:

```
5000 PRINT "Расчет напряженности электрического поля"
5010 PRINT "T","E"
5020 FOR T=1 TO N
5030 E=A(1,T)/L
5040 PRINT T,E
5050 RETURN
```

18. РАЗЛИЧНЫЕ СПОСОБЫ ВЫВОДА ИНФОРМАЦИИ НА ЭКРАН. ГРАФИКА ВЫСОКОГО РАЗРЕШЕНИЯ

Изучаем новые свойства оператора PRINT, а также учимся рисовать графики и картинки

Управление выводом текстовой информации

Прежде всего научимся управлять экраном. Звучит странно, но в этом есть свой смысл. До сих пор мы отдавали вывод информации на экран на откуп оператору PRINT. Теперь попробуем вмешаться в этот процесс.

Для начала научимся изменять максимальное число символов в строке экрана с 64 до 32, а также инвертировать изображение на экране. Для этого наберите демонстрационную программу:

```
10 CLS:PRINT "ВВЕДИТЕ ЛЮБУЮ ЦИФРУ"
20 LOCATE 20,15:A$="0123456789":PRINT A$
30 B$=INKEY$:IF B$=" " THEN 30
40 PRINT CHR$(27);B$:GOTO 20
```

Запустите ее и по одной вводите цифры: 1 - ничего, 2 - ничего, 3 - экран как бы раздался вширь, и на нем остались только нечетные цифры, зато каждый символ стал в два раза шире. Это произошло потому, что мы ввели контрольные символы CHR$(27) и CHR$(51) (51 - ASCII код цифры 3). Продолжаем ввод: 4 - ничего, 5 - ничего, 6 - нижняя надпись проинвертировалась. Теперь она написана не белым по черному, а черным по белому. Продолжаем: 7 - опять произошла инверсия, 8 - ничего, 9 - ничего, 0 - ничего. Попробуем опять ввести 2. Экран снова принял привычный вид.

Итак, запомним контрольные символы:

CHR$(27);"3" - включение режима "32 символа в строке"
CHR$(27);"2" - включение режима "64 символа в строке"
CHR$(27);"6" - инвертирование строки
CHR$(27);"7" - обратное инвертирование строки

Что это за новый оператор LOCATE в строке 20? Слово LOCATE можно перевести с английского как "местоположение". Этот оператор управляет положением курсора. Он имеет формат

LOCATE X,Y

где X и Y - номера колонки и строки на экране, куда нужно установить курсор. Верхний левый угол соответствует координате (1,1), а нижний правый в режиме 64 символа - (64,16). Вы можете спросить, зачем это нужно? Позже мы ответим на этот вопрос, а сейчас займемся деталями оператора PRINT.

Оператор PRINT может иметь один из трех видов.

1. PRINT ВЫРАЖЕНИЕ или PRINT СПИСОК ВЫРАЖЕНИЙ, разделенных точкой с запятой или запятой. Этот вариант нам хорошо известен.

2. PRINT TAB (ПОЗИЦИЯ) ВЫРАЖЕНИЕ. Этот оператор выводит на экран выражение, отступив от текущего положения курсора число знакомест (это неделимая ячейка на экранной строке, которую может занимать выводимый символ), равное величине, указанной в графе "позиция".

3. PRINT USING "ФОРМАТ"; ВЫРАЖЕНИЕ. Этот оператор выводит информацию в виде, заданном в графе "формат".

Разберемся с последними двумя типами оператора PRINT по порядку. Начнем с PRINT TAB. Этот оператор особенно удобен для вывода таблиц. Попробуйте команды:

```
PRINT TAB(10)    "1" TAB(20)      "2" TAB(30)      "3"
                   1                2                3
```

Цифра 1 появится на 10-й позиции, 2 - на 20-й и 3 - на 30-й.

PRINT "ИМЯ" TAB(20) "НОМЕР ТЕЛЕФОНА" TAB(50) "ГОРОД"

На экране появится:

```
ИМЯ      НОМЕР ТЕЛЕФОНА      ГОРОД
```

Вы видите, что выражение TAB(N) применяется аналогично точке с запятой и запятой.

Теперь займемся оператором PRINT USING. Этот оператор позволяет задавать формат вывода на экран. Когда нам это нужно? Прежде всего при выводе десятичных чисел. В общем виде нам хочется выводить их в виде

```
##.#
```

где знак # стоит на месте значащей цифры, а точка стоит там, где мы хотим ее видеть.

Попробуем повозиться с числами:

```
PRINT USING "##.#";72563
72.6
PRINT USING "##.#";72
72.0
```

Понятна идея? Символы внутри кавычек в точности говорят машине, как выводить на экран информацию.

Попробуем теперь такой вариант:

```
PRINT USING "##.#";768.56
%768.6
```

Откуда взялся символ %? Это сообщение машины о том, что мы хотим вывести на экран число, несовместимое с заданным форматом. Несмотря на этот прискорбный факт, машина все-таки вывела это число, но предупредила нас. Попробуйте самостоятельно поупражняться с форматированием вывода положительных и отрицательных десятичных чисел. Обратите внимание на то, что знак минус равноправен со значащими цифрами.

Как быть, если мы хотим вывести несколько значений? Формат распространяется на все значения, которые после него указаны:

```
PRINT USING "####";123;456;789
123 456 789
```

Обратите внимание, что числа отделились друг от друга пробелами. Для сравнения попробуйте команду PRINT без #:

```
PRINT 123;456;789
123456789
```

В чем дело? Просто в первом случае мы указали в операторе PRINT, что нам нужно вывести четыре значащих цифры. В то же время наши числа трехзначные. Поэтому старший разряд остается свободным. Это очень удобно для формирования таблиц, поскольку правый край любого числа стоит на фиксированной позиции, что приятно для глаза.

Иногда, при выводе больших чисел типа миллионов и миллиардов, удобно разделять разряды по три. Есть такая возможность и в

операторе PRINT USING. Для этого служит запятая, помещенная в любом месте между первым символом # и десятичной точкой, например:

```
PRINT USING "#,######.#";1000000
1,000,000.0
PRINT USING "##,######";1000000
1,000,000
```

Какие еще могут быть форматы? Например, вы хотите поставить какой-нибудь символ слева от числа. Укажите это в формате:

```
PRINT USING "*###.#";12.34
*12.3
```

Если нужен другой символ вместо *, укажите его в выражении для формата. Если вы хотите заполнить все вакантные места символами *, то потребуется формат:

```
PRINT USING "**####.#";12.34
****12.3
```

Заметим, что последний способ представления относится только к символу *.

Есть еще один специальный символ, $. Если использовать его в формате типа

```
PRINT USING "$###.#";12.3
```

то он появится на первом месте слева от числа:

```
$12.3
```

Если указать в формате два символа $$, получится:

```
PRINT USING "$$####.#";12.3
$12.3
```

т. е. знак $ будет "плавать", так что он всегда будет напечатан непосредственно перед числом.

Вообще в формате можно указывать какой угодно текст, например:

```
PRINT USING "#### л молока";124
124 л молока
```

Для вывода на экран обязательного знака выражения необходимо перед первым # слева поставить знак плюс:

```
PRINT USING "+##.#";12.3
+12.3
PRINT USING "+##.#";-12.3
-12.3
```

Для того чтобы всем вводимым числам присваивался знак минус, нужно перед первым # поставить знак минус:

```
PRINT USING "-##.#";12.3
-12.3
```

Попробуйте ввести

```
PRINT USING "-##.#";-12.3
```

Обратите внимание, что в данном случае появится сообщение об ошибке в виде символа %. Этим сообщением машина напомнит, что вы излишне перестраховались, записав знак минус и перед числом. Если после последнего знака # поставить минус, то положительные числа будут печататься без знака, а отрицательные числа будут сопровождаться минусом справа от числа, например:

```
PRINT USING "##.#-";-12.3
12.3-
```

Мы знаем, что десятичные числа можно представить в экспоненциальном формате. Обычно они автоматически записываются в таком формате в случае очень больших или очень маленьких чисел. Можно, однако, принудительно заставить машину выводить числа в экспоненциальном формате. Для этого служит специальный символ ^:

```
PRINT USING "##.#^^";1234
1.2 E+03
```

Возможности оператора PRINT USING распространяются также и на строковые переменные. Для этого служат специальные символы ! и %%. Символ ! позволяет вывести на экран только первый символ в тексте:

```
PRINT USING "!";"Я ЭВМ Корвет"
Я
```

Для вывода определенного количества букв текста используются символы %%. Количество выводимых символов на экране будет равно числу пробелов между % плюс 2, например:

```
PRINT USING "% %";"Я ЭВМ Корвет"
Я ЭВМ
```

Как быть, если мы хотим на одной строке выводить различные переменные по разному формату? Можно указать формат для каждой переменной в отдельности. Разделить форматы можно любыми символами, которые не являются специальными для формата, например:

```
PRINT USING "A=##.# B=###.#";12.3;123.4
A=12.3 B=123.4
```

На этом закончим знакомство с оператором PRINT и вообще с отображением текстовой информации на экране.

Займемся теперь другой стороной Корвета, которая, быть может, является его самой привлекательной чертой, - графикой.

Для начала вернемся к отложенному вопросу о том, зачем нужны всякие фокусы на экране типа инвертирования надписей. Сейчас поймем. Вспомним нашу старую программу по обработке вольт-амперных характеристик разряда. Попробуем переделать управляющую программу так, чтобы она имела "фирменный" вид:

```
5 CLS:DIM A$(4):CN=1
10 A$(1)="1.Ввод данных"
20 A$(2)="2.Просмотр и редактирование данных"
30 A$(3)="3.Вывод результата"
35 A$(4)="4.Рисование графика"
40 FOR I=1 TO 4:LOCATE 10,5+I,0:PRINT A$(I):NEXT I:GOTO 100
50 B$=INKEY$:IF B$=" " THEN 50
60 IF B$=CHR$(26) THEN CS=CN:CN=CN+1:IF CN>4 THEN CN=1:
   GOTO 100
70 IF B$=CHR$(25) THEN CS=CN:CN-1:IF CN<1 THEN CN=4:
   GOTO 100
80 IF B$=CHR$(13) THEN ON CN GOSUB 1000,2000,3000,4000
100 LOCATE 10,5+CN:PRINT CHR$(27);"6";A$(CN)
110 LOCATE 10,5+CS:PRINT CHR$(27);"7";A$(CS):GOTO 50
```

Как работает эта программа? Ее цель - сделать процесс использования меню наиболее приятным. Выглядит это так. На экране изображается меню. При нажатии клавиш со стрелками вверх и вниз на правом поле белый прямоугольник бегает по строкам меню. При нажатии на ВК происходит выбор требуемой функции. Следует заметить, что на Корветах разного года выпуска могут различаться коды управляющих клавиш на правом поле. Поэтому в нашей программе, возможно, придется отредактировать строки 60 и 70.

Сначала установим коды управляющих клавиш. Для этого напишем программу (если вы уже набрали программу меню, то не стирайте ее)

```
10000 A$=INKEY$:IF A$=" " THEN 10000
10010 PRINT ASC(A$):GOTO 10000
```

Теперь наберите команду RUN 10000. Этим вы говорите компьютеру, что выполнение программы нужно начать со строки 10000. Нажмите по очереди на кнопку со стрелкой вверх и вниз. Посмотрите, какие числа появятся на экране. Вероятнее всего, для первой появится число 25, а для второй 26. Если результат будет отличен, замените в строках 60 и 70 в выражениях

IF B$=CHR$(26) и IF B$=CHR$(25)

числа 26 и 25 на то, что у вас получилось. После этой проверки сотрите строки 10000 и 10010 и запускайте программу. Не правда ли, работать стало несколько приятнее?

Обратите внимание на некоторые особенности использования операторов в этой программе. Так, у оператора LOCATE в строке 40 появился третий аргумент. Его смысл в том, что, ставя на его место 0 или 1, можно запрещать или разрешать отображение курсора на экране. Это не является обязательным. Просто, работая с таким красивым меню визуально, неприятно иметь где-то на экране еще и горящий курсор.

Следующим моментом является появление новой четвертой строки в меню: рисование графика. Как это делать? Сейчас мы с этим разберемся. Прежде всего убедитесь, что у вашего Корвета есть возможность пользоваться графикой высокого разрешения. Это станет ясно из изучения комплекта эксплуатационных документов. Если у вас графика есть, еще раз напомним, что она может быть цветной или черно-белой. В последнем случае цвета отображаются градациями яркости.

Итак, графика. Перво-наперво необходимо научиться зажигать точку любого цвета в любом месте экрана. Напомним, что экран у нас имеет 512*256 точек и каждая точка может гореть одним из 16 цветов. Для зажигания точки используется команда

PSET (X,Y),Z

где X, Y - координаты точки на экране по горизонтали и вертикали соответственно (полезно помнить, что 0<=X<=511, 0<=Y<=255); Z - цвет точки, который принимает значения от 0 до 7. (Здесь вы, ко тественно, спросите: а где же остальные 8 цветов? Как их использовать, будет объяснено чуть позже.)

Первая графическая программа, которую мы рассмотрим, называется "небо в алмазах". Это значит, что на экране зажигаются точки, расположенные случайным образом. Получается феерическая картина. Для работы программы нам потребуется специальная функция - генератор случайных чисел: RND(N). Она принимает значения, равные единице, деленной на случайным образом выбранное число из интервала от 1 до N.

Итак, "небо в алмазах":

```
10 CLS:PCLS
20 PSET(RND(511)*511,RND(255)*255),RND(7)*7
30 GOTO 20
```

Запустите программу. Вы увидите, как на экране зажигаются точки разного цвета. Однако скоро их становится очень много и зрелищность теряется. Хорошо бы научиться не только зажигать звезды, но и гасить их. Для этого служит функция

PRESET(X,Y)

где X, Y - координаты точки (0<=X<=511, 0<=Y<=255). Добавьте в программу еще одну строку:

```
25 PRESET (RND(511)*511,RND(255)*255)
```

Вводите RUN и наслаждайтесь эффектом.

Да, мы забыли разъяснить, что это за оператор PCLS в строке 10. Вы, наверное, и сами догадались. Нет? Очень просто. Вы помните, что у Корвета графический и алфавитно-цифровой дисплеи абсолютно независимы. Соответственно, нужен дополнительный оператор для очистки графического дисплея. Это и есть PCLS. Произошли эти буквы от сокращения английской фразы Picture CLear Screen - стирание картинки с экрана.

Теперь вы уже можете рисовать графики. Действительно, вы рассчитываете координаты точек, округляете их до целой величины (естественно, масштабируете так, чтобы уместить координаты в диапазоне 0<=X<=511 и 0<=Y<=255), задаете цвет и зажигаете необходимые точки. Попробуем таким способом построить график функции SIN(X). Сначала рассчитаем значение функции. Шаг по оси X выберем равным 1 (от 0 до 511):

```
10 CLS:PCLS
20 DIM SN(511)
30 FOR I=0 TO 511:SN(I)=SIN(6.28/512*I)
40 LOCATE 24,6:PRINT "Рассчитываем синус":LOCATE 30,8,0:
   PRINT I
50 NEXT I
60 CLS
```

Синус рассчитан. Нарисуем его по очереди всеми цветами:

```
70 FOR C=1 TO 7:FOR I=0 TO 511:PSET(I,127*(1-SN(I))),C:
NEXT I,C
```

Запустите программу и посмотрите, как на экране появляется изображение синуса, как оно перекрашивается. Очень красиво.

Одно плохо: у графика должны быть оси. Давайте их нарисуем. Для рисования линий существует специальный оператор LINE:

```
LINE (X1,Y1)-(X2,Y2),K,(F)
```

Здесь X1,Y1 и X2,Y2 - координаты точек, которые нужно соединить прямой линией, K - цвет линии, последняя буква должна быть вообще опущена, если рисуем линию. Если мы хотим нарисовать прямоугольник с диагональю X1,Y1-X2,Y2, то на этом месте мы ставим латинскую букву B. Если нам нужен закрашенный прямоугольник, то поставим вместо B букву F.

Сейчас нам нужны оси координат. Поэтому добавим в программу строку

```
65 LINE (0,0)-(0,255),7:LINE(0,127)-(511,127),7
```

Проверьте, что оси на самом деле нарисуются. Хотите проверить, что графический и алфавитно-цифровой дисплеи независимы? Тогда по окончании работы программы наберите команду LIST. Вы видите, что появившийся на экране текст не стер картинки.

Теперь займемся вопросом о скорости рисования. Рисовать точками хорошо, но медленно. Попробуем убыстрить процедуру. Это можно сделать, если рисовать не каждую точку, а проводить прямую линию между двумя точками, идущими не подряд, а с некоторым интервалом. Для этого модифицируем нашу программу:

```
10 CLS:PCLS
20 DIM SN(511)
30 FOR I=0 TO 511:SN(I)=SIN(6.28/512*I)
40 LOCATE 24,6:PRINT "Рассчитываем синус":LOCATE 30,8,0:
   PRINT I
50 NEXT I
60 CLS
70 LINE (0,0)-(0,255),7:LINE(0,127)-(511,127),7
80 FOR C=1 TO 7:PSET(0,127),G:FOR I=0 TO 511 STEP 16:
   LINE-(I,127*(1-SN(I))),C:NEXT I,C
```

Обратите внимание, что в функции LINE мы не указали координаты X1,Y1. В этом случае линия будет проводиться из последней нарисованной точки в точку X2,Y2.

Запустите программу. Вы видите, что скорость рисования резко возросла без заметного ухудшения качества картинки. Но это все двумерная графика. А если мы захотим изобразить трехмерный объект? Оказывается, что Корвет позволяет легко это реализовать. Для этой цели удобно использовать оператор смещения начала координат на экране

```
RELOC(X,Y)
```

где X, Y - смещения начала координат (-32768<X, Y<+32767). Рассмотрим пример программы, использующей этот оператор:

```
10 CLS:PCLS:DIM EX(72),X(72)
20 FOR N=1 TO 72:X=156-N*4:X=X*X/1024:EX(N)=EXP(-X):X(N)=
   100+4*N:NEXT N
25 FOR K=-8 TO 0:RELOC(5*K,-5*(8+K)):
   LINE(100,255)-(100,255),0
30 FOR I=1 TO 72
35 A=(8+K)/8
40 Y=127*(2-A*EX(I)):LINE-(X(I),Y)
50 NEXT I,K
```

Подумайте, зачем нужен в строке 25 оператор LINE? Запустите программу, посмотрите, что она делает, и после этого по тексту программы постарайтесь понять алгоритм. Это полезное упражнение.

С графиками все ясно. Теперь займемся живописью. Попробуем нарисовать какую-нибудь картинку, скажем дом. Дом рисовать очень просто. Это ведь, в основном, прямоугольники стен с квадратами окон:

```
10 CLS:PCLS:RELOC(0,0)
20 LINE(50,70)-(450,240),7,B
30 FOR I=0 TO 2:FOR J=0 TO 6:SX=50*J
40 ST=50*I:RELOC(SX,ST)
50 LINE(85,95)-(100,120),7,B
60 NEXT J,I
```

Эта программа нарисует нам стену дома с окошками. Теперь хорошо бы эту стену покрасить. Для этого существует специальный оператор

PAINT (X,Y),N,M

(от англ. PAINT - красить). Этот оператор закрашивает область, внутри которой находится точка, определяемая координатами X, Y. Цвет этой точки должен быть отличным от цвета границы. Цвет закрашивания задается числом N, цвет границы - M. Чтобы покрасить стену, вставим в программу строку

```
70 PAINT(60,60),4,7
```

Теперь стена покрашена. Нарисуем крышу. Это можно сделать, рисуя линии. Но можно и иначе. Существует еще один оператор

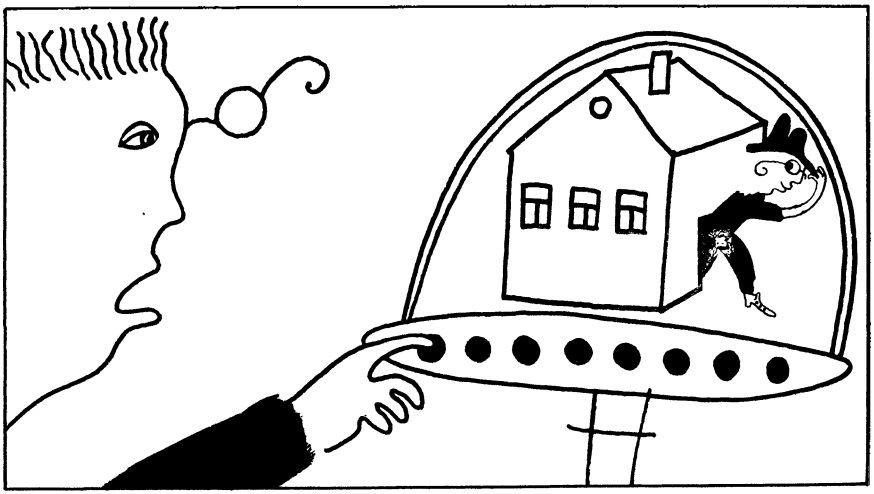
DRAW "СТРОКА", ЦВЕТ

(от англ. DRAW - рисовать). Строка задает перечень команд рисования. Они пишутся подряд, ничем не разделяясь:

UN - вверх на N точек
DN - вниз на N точек
RN - вправо на N точек
LN - влево на N точек
EN - вверх и вправо на N точек по горизонтали
FN - вниз и вправо на N точек по горизонтали
GN - вниз и влево на N точек по горизонтали
HN - вверх и влево на N точек по горизонтали
MX,Y - рисовать линию от текущей точки до точки с координатами X, Y (если перед X или Y стоит знак "+" или "-", то это относительные координаты).

Перед любой командой рисования могут стоять поясняющие операнды:

B - при этом будет нарисована только последняя точка
N - начертить линию и возвратиться обратно
A - задает поворот осей координат
A1 - на 90 градусов
A2 - на 180 градусов
A3 - на 270 градусов
A0 - вернуться в исходное положение



Кроме этого существуют дополнительные команды:

SN - задание коэффициента масштабирования; масштаб равен целой части N/4 плюс 1, т. е. если N=1, то масштаб равен 1
CN - задание цвета линии, N - номер цвета

Итак, рисуем крышу:

```
80 RELOC(0,0):PSET(50,70),0:DRAW"E30R340F30",7
90 PSET(255,40),0:DRAW"U20NM205,35M270,15",7
100 DRAW"NM240,11NM300,19":LINE(207,28)-(237,32):
    LINE (224,23)-(254,27)
110 PRINT "Ну, а дальше рисуйте и раскрашивайте сами!"
```

Запустите программу и поупражняйтесь в электронном рисовании.

А если мы хотим нарисовать что-нибудь посложнее? Можно или нет? Конечно, можно. Как вам понравится, скажем, круг или эллипс? Для рисования эллипсов и их частей служит оператор CIRCLE:

CIRCLE(X,Y),РАДИУС,ЦВЕТ,НАЧАЛО,КОНЕЦ,АСПЕКТ

где X, Y - центр эллипса; радиус - большая полуось; начало, конец - углы в радианах (для дуги); если перед ними стоит знак минус, то начало и конец дуги соединяются с центром эллипса; аспект задает соотношение между полуосями. Потренируемся в рисовании окружностей и эллипсов:

```
10 CLS:PCLS:RELOC(0,0)
20 INPUT "Введите число, большее 0, можно десятичную дробь";N
30 CIRCLE (255,127),50,7,,,N
40 LOCATE 30,15:PRINT "Чтобы продолжить, нажмите пробел"
50 A$=INKEY$:IF A$="" THEN 50 ELSE 10
```

Вы видите, что в этой программе вам предоставляется возможность посмотреть, как меняется форма эллипса в зависимости от аспекта.

Запустите программу и посмотрите, что получается, если N принимает значения 1, 1.25, 10 и 0.1.

Теперь займемся операцией перекрашивания картинок. Конечно, это можно делать, как вы уже поняли, с помощью команды PAINT. В качестве иллюстрации работы этого оператора введите программу

```
10 CLS:PCLS
20 FOR K=0 TO 50 STEP 10
30 CIRCLE(255,125),77+K,7
40 FOR I=0 TO 6
50 PAINT(255,200+K),I,7
60 NEXT I:NEXT K
70 GOTO 20
```

Запустите ее. Видите, как работает оператор PAINT? Хорошо видна скорость закрашивания. Довольно быстро, но не слишком. Можно ли увеличить эту скорость? Оказывается, можно с помощью следующего трюка, реализованного в Корвете. Дело в том, что в Корвете реальные цвета не закреплены жестко за каким-либо номером. Существует способ это соответствие изменять. Для этой цели придуман оператор

```
LUT A(N)
```

A(N) - целочисленный массив размерностью не менее N+15 (N может принимать любое значение от нуля и дальше). Этот оператор загружает в качестве таблицы цветов элементы массива A от A(N) до A(N+15). Заметьте, что здесь реализуются уже 16 цветов. Посмотрим, как это выглядит на практике:

```
10 CLS:PCLS:DEFINT L:DIM LT(31),L1(31):K=1
20 FOR I=0 TO 7:LT(I)=I+8:LT(I+8)=15:NEXT:LT(0)=0:
   LUT LT(0):FOR I=0 TO 7:L1(I)=I+8:L1(I+16)=I+8:
   L1(I+24)=I+8:NEXT I
30 FOR I=0 TO 127 STEP2:K=K+1:IF K=8 THEN K=1
40 LINE (255-I*2,127-I)-(255+I*2,127+I),K,B
50 NEXT I
```

Эта часть программы загрузит таблицу цветов, определенную первой частью массива LT, и нарисует нам разноцветные прямоугольники. Заметьте скорость рисования прямоугольников. Попробуем теперь их перекрасить не с помощью PAINT, а с помощью LUT, т. е. с использованием перекодировки таблицы цветов. Для этого дополните программу строкой

```
60 FOR I=0 TO 7:X=L1(I):L1(I)=0:LUT L1(T):L1(I)=X:NEXT I:
   IF INKEY$="" THEN 60 ELSE LUT LT(0):LOCATE,,1
```

Запустите эту программу. Смотрите, прямоугольники перекрашиваются гораздо быстрее, чем рисуются. Нажмите любую клавишу, чтобы остановить программу.

Теперь несколько заключительных замечаний по графике высокого разрешения. Прежде всего отметим, что в командах типа LINE, PSET, PAINT перед значением координат можно указать, что они являются не абсолютными координатами, а смещением от текущей точки. Для этого служит слово STEP, например:

```
PSET STEP (10,20),2
```

В результате зажжется точка, смещенная на 10 позиций по горизонтали и на 20 по вертикали от текущей точки. То же самое касается рисования линий; например, в результате выполнения команды

```
LINE STEP(10,20)-STEP(50,60),2
```

будет начерчена линия из точки, смещенной относительно текущей на 10 по горизонтали и 20 по вертикали, до точки со смещением 50 и 60 соответственно.

Полезной функцией является

```
POINT (X,Y)
```

(от англ. POINT - точка). Эта функция принимает значение, соответствующее номеру цвета точки с координатами X, Y.

Пойдем дальше в изучении графических возможностей Корвета. Каким цветом (или какой степенью черноты, если у вас монитор черно-белый) в данный момент светится экран перед вами? Нравится ли вам этот цвет? Если нет, вы можете изменить его с помощью оператора

```
COLOR M,N
```

(от англ. COLOR - цвет). Для того чтобы понять, как он работает, введите программу

```
10 PCLS:CLS
15 PRINT "COLOR M,N"
20 INPUT "Введите номер цвета линии";M
30 INPUT "Введите номер цвета экрана";N
40 COLOR M,N:PCLS:CLS
50 LINE(0,0)-(511,255):CIRCLE(255,127),50
60 GOTO 15
```

Запустите ее и отвечайте на вопросы, тем самым вводя цвета линии (M=0-7) и экрана (N=0-7) в операторе COLOR M,N. Видите, что этот оператор в сочетании с последующей командой PCLS меняет цвет экрана и цвет, которым будет осуществляться рисование, в том случае, когда в командах рисования цвет устанавливается по умолчанию, т. е. не задается его номер (см. строку 50).

И в заключение этого раздела следует сказать еще об одной графической возможности Корвета, которая реализуется лишь в том случае, если ваша машина укомплектована полной графической памятью. В этом случае вы можете рисовать и отображать четыре различные страницы памяти. Для переключения страниц используется оператор

SCREEN M,N

(от англ. SCREEN - экран, страница), где M определяет ту страницу, которая будет отображаться на экране, а N - ту страницу, с которой будет работать программа; значения M, N меняются от 0 до 3.

Используя страницы, можно, например, применять такой трюк. Вы рисуете сплошной объект на невидимой странице и потом практически мгновенно оператором SCREEN переключаете её в видимую область. Благодаря этому механизму на экране Корвета можно показывать мультфильмы собственного сочинения.

Мы надеемся, что богатые графические возможности Корвета не оставили вас равнодушным.

19. БЕЙСИК И ПАМЯТЬ ЭВМ

Пользователь учится добираться до чисел, хранимых в памяти ЭВМ, используя лишь команды языка Бейсик

Мы не раз говорили, что все переменные и сам текст программы размещены в оперативной памяти ЭВМ. Раз так, было бы удобно уметь находить, в каких именно ячейках памяти размещена та или иная переменная. Обычно это нужно при совместном использовании программы, написанной на языке Бейсик, и программы, написанной на языке Ассемблер. Подобный же прием может быть использован для связи с различными периферийными устройствами Корвета, поскольку они спроектированы как часть памяти машины.

Итак, память. У каждой ячейки памяти имеется адрес. Любая переменная занимает определенное своим типом количество ячеек памяти. Следовательно, если мы хотим что-либо сделать с переменной, минуя оператор присваивания, нам необходимо знать адрес первой ячейки. Вы, наверное, догадались, что для этого Бейсик имеет в запасе специальную функцию. Так оно и есть. Эта функция называется VARPTR:

VARPTR (ИМЯ ПЕРЕМЕННОЙ)

В скобках можно указывать имена любых переменных, в том числе и элементов массивов.

А как обращаться к ячейкам памяти? Для этой цели существуют специальные оператор и функция

POKE M,N и PEEK(M)

где M - адрес ячейки памяти. Оператор POKE записывает байт N по адресу M. Функция PEEK возвращает байт, записанный по адресу M.

Поэкспериментируем? Вы теперь получили инструмент, позволяющий копаться в памяти ЭВМ без всяких правил. Сейчас мы ей покажем! Пусть раскрывает свои секреты!

Внимание! Это очень опасное оружие. Не зная, что в машине по каким адресам находится, очень опасно пользоваться оператором POKE. Вы, сами того не подозревая, можете что-нибудь серьезно испортить.

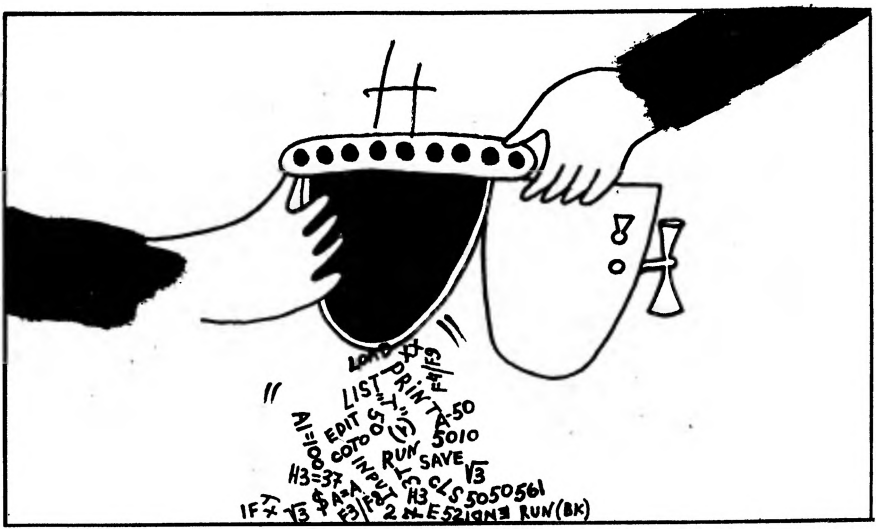
Поэтому начнем эксперименты с изучения распределения оперативной памяти в Корвете, или, что то же самое, с карты памяти. Как вы знаете, Бейсик бывает двух типов: резидентный в ПЗУ и загружаемый с диска. Естественно, что если вы загружаете Бейсик с диска, то ПЗУ вам не нужно. Поэтому логично отключать ПЗУ при работе с дисковым вариантом. Для дискового варианта Бейсика используется карта памяти

Адрес	Распределение памяти
0000H-5FFFH	Интерпретатор BASIC
6000H-C000H	ОЗУ, доступное для использования
C000H-F800H	Операционная система
F800H-FFFFH	Устройства ввода-вывода

Карта памяти для Бейсика, записанного в ПЗУ:

Адрес	Распределение памяти
0000H-5FFFH	ПЗУ - интерпретатор Бейсика
6000H-BEFFH	ОЗУ, доступное для использования
BE00H-BFFFH	Устройства ввода-вывода
C000H-FFFFH	ГЗУ - область памяти графического дисплея

Теперь попробуем пообщаться с памятью машины. Просто так это делать не интересно. Попробуем решить какую-нибудь интересную задачу. Например, научим машину пищать на разные лады. Для пищания и некоторых других целей в машине имеется специальная микросхема - таймер. Этот таймер имеет три независимых канала, один из которых подключен к пищалке. Попробуем добыть из машины звук.



В принципе можно добыть звук и без использования оператора POKE. В Бейсике существует специальный оператор

BEEP

По этой команде машина издает звук частотой 800 Гц и длительностью 1/4 секунды. Однако для сочинения музыкальных произведений этого явно мало.

Теперь займемся программированием таймера. Таймером ведают три ячейки памяти. Как именно устроен таймер, мы здесь разбирать не будем, поэтому запомните эти ячейки:

1. Ячейка с адресом FB03H - сюда заносится управляющее слово (в нашем случае 36H).

2. Ячейка FB00H - сюда последовательно заносятся два байта, определяющие частоту звука.

3. Ячейка FB32H - сюда заносится команда, включающая или выключающая таймер. Команда эта не какая-нибудь, а определенная: 8 - включение, 0 - выключение таймера.

Прежде чем переходить к программированию, выполним тест на внимательность. Вы поняли, что это за интересные числа приводятся в данном разделе? Например, что такое FB03H? Если понимаете - хорошо. Для остальных объясним. Вы помните, что бывают разные системы счисления. Для машины удобнее работать в ее родной, двоичной. Однако это не очень удобно для программистов. Уж очень длинные выражения будут получаться. Поэтому люди решили использовать вместо двоичной шестнадцатеричную систему. Она родственна двоичной, поскольку 16 равно двойке в четвертой степени. Так вот, все числа, приводимые в этом разделе, - шестнадцатеричные.

Прежде чем двигаться дальше, коснемся проблемы перевода чисел из одной системы в другую. Десятичная система имеет десять цифр:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

В двоичной системе - всего две цифры:

0, 1

В шестнадцатеричной их, естественно, шестнадцать:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Последние шесть цифр соответствуют числам

10, 11, 12, 13, 14, 15

в десятичной системе. Здесь они являются цифрами.

Так как же сопоставить числа в разных системах? Очень просто. Ответьте на вопрос: как представляются большие числа в десятичной системе, если их разложить по степеням числа 10? Если мы хотим представить число в некоторой системе счисления, мы должны разложить его по степеням основания этой системы, например:

321=3*100+2*10+1

Рассмотрим число 321 в шестнадцатеричной системе счисления. Прежде всего определим максимальную целую степень числа 16, не превышающую 321:

первая степень: 16<321.
вторая степень: 16*16=256<321.
третья степень: 16*16*16=4096>321

Итак, максимальная степень - вторая. Теперь вычислим, с каким весом входит вторая степень числа 16 в число 321. Очевидно, этот множитель равен 1. Проделаем то же самое с остатком, т. е. с числом, равным

321-16*16=321-256=65

Для этого числа максимальная степень равна 1, но получается другой множитель: 4. Теперь вычислим остаток:

65-16*4=65-64=1

Число 1 уже по степеням 16 не разлагается, поскольку, собственно, осталась одна степень: 0. В результате у нас получилось:

321=1*(16*16)+4*(16)+1

Мы выделили степени основания 16 в скобках для наглядности. Ну, а теперь нам легко получить представление числа 321 в шестнадцатеричном виде. Для этого мы пишем в строку наши множители:

321=141H

Буква H после числа как раз и говорит о том, что оно шестнадцатеричное.

Возможно, непривычность этих обозначений создает некоторый дискомфорт. Не беда. Пользуйтесь, если вам нравится, десятичной системой. Бейсику это все равно. Однако советуем вам все-таки овладеть этой премудростью. По мере накопления у вас опыта работы с ЭВМ вы оцените преимущества шестнадцатеричной системы.

Поскольку мы привыкли работать в шестнадцатеричной системе, то давайте поймем, как объяснить Бейсику, в каком виде мы вводим число. С десятичными числами до сих пор проблем не возникало. Что же касается шестнадцатеричных, то перед ними ставятся два символа: &H (H - латинская буква). Если мы хотим проверить, правильно ли мы перевели число 321 в шестнадцатеричную систему, введем команду

PRINT &H141

На экране появится ответ:

321

Все в порядке. Обратите внимание, что в тексте буква H, обозначающая шестнадцатеричное число, ставится после числа, а для Бейсика она должна стоять перед числом! Не путайте.

Но вернемся к музыке:

```
10 POKE &HFB03,&H36
```

Это команда программирования таймера, и здесь мы ее разбирать не будем. Далее идет собственно музыкальная часть:

```
20 INPUT "Введите число";X
30 POKE &HFB00,X AND 255
40 POKE &HFB00,INT(X/256)
50 POKE &HFB32,8
```

Если эту программу запустить, машина начнет непрерывно пищать на частоте, определяемой числом X. Обратите внимание на то, как мы разделили число X на два байта. Сейчас вы поймете преимущество шестнадцатеричной системы. Для начала проделаем эту операцию с десятичным числом 321, благо мы знаем уже его шестнадцатеричное представление.

Вспомним, что такое байт. Это восемь бит. Какое максимальное число может быть байтом? В двоичной системе это 11111111, т. е. все восемь бит равны 1. Что это такое в шестнадцатеричной системе? Напишем таблицу соответствия чисел:

Десятичная	Двоичная	Шестнадцатеричная
2	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Все очень просто, но заметьте, что число 15 соответствует 1111 в двоичной системе. По размеру это половина байта.

А чему равно десятичное 16? В двоичной системе это 10000, в шестнадцатиричной 10. Естественный результат. А чему равно десятичное число 255? Легко убедиться, что в шестнадцатеричной системе это FFH, а в двоичной 11111111, т. е. максимально возможное

значение байта. Теперь понятно, почему шестнадцатеричная система удобнее, чем десятичная. Перевод из нее в двоичную систему очень прост. Одна ее цифра (или разряд) соответствует в точности четырем разрядам двоичной системы. С этим ясно. Но зачем нам вообще нужна двоичная система? Бейсик отлично умеет обращаться с десятичными числами.

Давайте вспомним, с чего начался наш разговор. С разделения числа на байты. Удобнее всего сделать это с помощью логических операций AND и OR. При работе с числами они вызывают побитное умножение и сложение. Так,

```
1 AND 1=1    1 OR 1=1
0 AND 1=0    1 OR 0=1
1 AND 0=0    0 OR 1=1
0 AND 0=0    0 OR 0=0
```

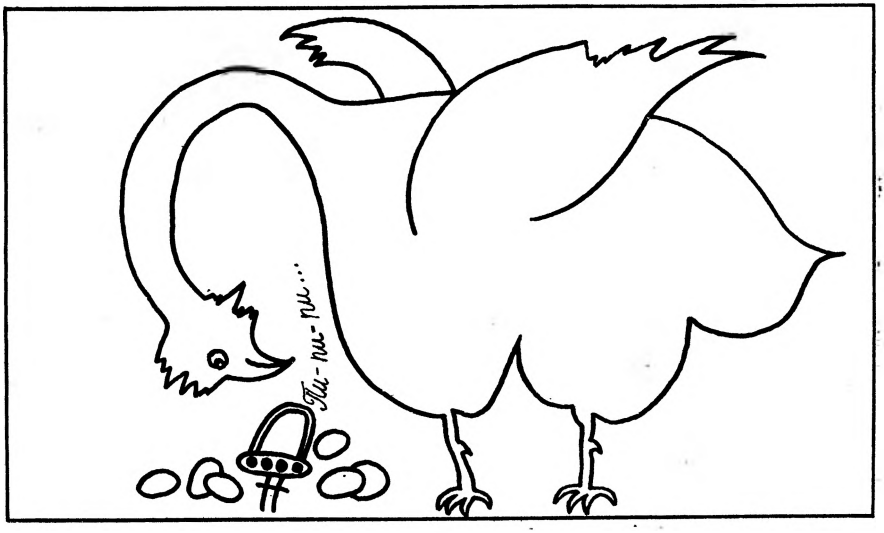
Выделим младший байт числа. Чтобы это сделать, необходимо все биты в старшем байте сделать равными нулю. Для этого проще всего наше число логически умножить на 00FFH (или на 255 в десятичной системе).

Возьмите число 321 и проделайте это вручную. Зная его представление в шестнадцатеричной системе, это очень просто сделать:

```
(321=141H=101000001) AND(255=FFH=011111111)=
=(65=41H=01000001)
```

Вы видите, что в десятичной системе непросто угадать результат при логических операциях. В случае шестнадцатеричной системы при некотором навыке эта процедура доступна для устного счета.

С младшим байтом все ясно. А как выделить старший? Для этого необходима процедура целочисленного деления. Раз старший байт - это та часть числа, которая говорит, сколько младших байтов в нем содержится, то для его определения воспользуемся функцией INT:



INT(321/256)

Вы помните, что функция INT вычисляет целую часть. Внимательный читатель, возможно, вспомнит, как мы переводили 321 в шестнадцатеричную систему. Просматриваете аналогию? Да, процедура разделения целого числа на старший и младший байты эквивалентна его представлению в 256-ричной системе счисления. Так вот, эту процедуру и делают строки 30 и 40 нашей программы.

Пока вы читаете эти строки, ваша машина беспрерывно пищит. Как бы ее остановить? Добавьте к программе строки

```
60 FOR I=0 TO 255:NEXT:REM Задержка
70 POKE &HFB32,0
80 GOTO 20
```

Строка с номером 70 прекратит писк.

Попробуйте поиграть со звуком. Чувствуете, как биты и байты воплощаются в физическую реальность?

Возможно, вещи, которые мы рассмотрели в этом разделе, не очень понятны или непривычны. Не беда. Когда вы начнете серьезно работать с машиной, вам неизбежно придется их освоить, и тогда, по мере накопления опыта, они станут совершенно естественными.

Прежде чем покончить с функциями POKE и PEEK, остановимся на одном примере. Попробуем управлять выдачей информации на дисплей без команды PRINT. Как это сделать? Чтобы это понять, остановимся на устройстве дисплея. Как он устроен? Допустим, что мы собираемся выводить некий символ на экран. Для этого мы обращаемся к команде PRINT. Она в свою очередь передает код требуемого символа дисплею. Куда он в действительности попадает?

Дело в том, что дисплей, оказывается, очень умное устройство. Он имеет собственную память. В процессе его работы специальное устройство проверяет, какие коды хранятся в этой памяти, и выводит соответствующие им символы на экран. Таким образом, каждой позиции на экране (она часто называется знакоместом) соответствует ячейка в памяти дисплея. Всего ячеек, как вы знаете, 64*16=1024. Теперь понятно назначение курсора. Он указывает, в какую ячейку памяти необходимо записать следующий символ.

Попробуем потренироваться. Для начала посмотрим, какие символы может выводить на экран Корвет. Основной набор содержит 256 символов, которым соответствуют коды от 0 до 255. Для того чтобы вывести эти символы на экран, необходимо записать их коды в память дисплея. Для этого нам нужно знать адрес ее начала. Стартовый адрес памяти дисплея равен FC00H. Теперь напишем программу:

```
10 CLS
20 FOR I=0 TO 255
30 POKE &HFC00+I,I
40 NEXT I
50 LOCATE 1,15
```

Запустите ее. Вы увидите полный набор символов, который понимает Корвет. Обратите внимание на третью строку символов. Это псевдографика. С их помощью можно рисовать на экране художественные произведения в стиле кубизма.

Давайте попробуем создать какую-нибудь "живую" картинку, например, с помощью следующей программы:

```
10 CLS
20 FOR I=0 TO 1023
40 POKE &HFC00+I,149
50 IF I>20 THEN POKE &HFC00+I-20,0
60 NEXT
70 LOCATE 1,10
```

То, что вы видите на экране, является прообразом игровых программ. Объект в них перемещается по экрану. Но в играх обычно бывают и препятствия. Как научиться их находить? Очень просто. При помощи оператора PEEK. Давайте посмотрим, что у нас находится в ячейке FC00+1023, т. е. в самой последней ячейке экрана:

```
PRINT PEEK (&HFC00+1023)
149
```

Точно так же можно посмотреть содержимое любой другой ячейки экрана. Сейчас мы не будем на этом останавливаться. Вы сможете изучить этот процесс подробнее, когда встретитесь с игрой Piton в разделе 23 нашей книги.

20. РАБОТА С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ

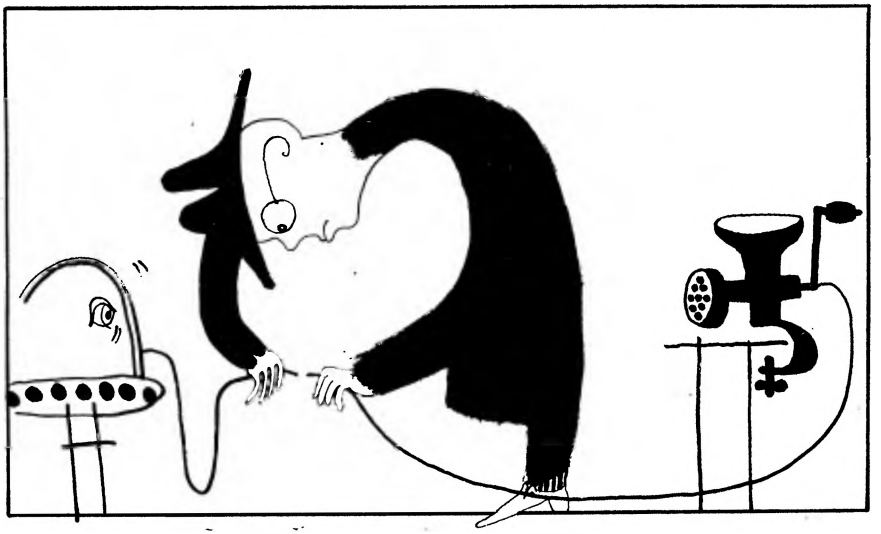
Читатель учится сохранять свои программы
на вечные времена с помощью внешних устройств
типа принтера, магнитофона и дисководов

Пришла пора подводить итоги. Вы уже являетесь квалифицированным программистом. Вам под силу писать сложные программы. Но до сих пор после выключения машины программа пропадала без следа. Это очень обидно. Давайте научимся спасать программы. Для начала научимся работать с принтером.

Вывод информации на принтер. Нас интересуют две вещи: как распечатать текст программы и как вывести на печать то, что вам насчитала машина. Все это делается очень просто. Нужно добавить букву L перед операторами, выполняющими эти действия на экране дисплея. Так, для распечатки текста программы служит оператор

```
LLIST
```

Если вам нужно распечатать не всю программу, а только ее часть, то аналогично выводу на экран нужно указать начальную и конечную строки. Например,



LLIST 100-200 означает: распечатать строки 100 - 200
LLIST 100- распечатать со строки 100 до конца программы

Для вывода результатов расчетов используются операторы, аналогичные операторам PRINT:

```
LPRINT "Здравствуйте, я ЭВМ Корвет"
LPRINT 2*2
LPRINT A$
LPRINT A,B,C
LPRINT TAB(20)
LPRINT USING "##.#" 12.386
```

и т. д. Подключите принтер к вашей машине и поупражняйтесь с операторами LPRINT подобно тому, как мы это делали с операторами PRINT в одном из предыдущих разделов.

Работа с магнитными дисками. Принтер хотя и позволяет получить изображение вашей программы и результатов счета на бумаге, к сожалению, не дает возможности считывать программу или какие-то цифры обратно. Поэтому если вы написали программу, распечатали ее и после этого выключили машину, то вам останется только любоваться напечатанным текстом. Для реанимации программы в машине вам придется заново ее ввести с клавиатуры. Для того чтобы избежать потери информации при выключении машины, и придуманы, как вы помните, магнитофоны и дисководы.

Для начала научимся работать с дисками. Те, у кого их нет, могут при чтении перескочить сразу на магнитофонные проблемы.

У нас есть две задачи, которые нам жизненно важно научиться решать. Во-первых, это сохранение и считывание программы. Для этой цели служат операторы

SAVE "ИМЯ ПРОГРАММЫ"
LOAD "ИМЯ ПРОГРАММЫ"
MERGE "ИМЯ ПРОГРАММЫ"

Оператор SAVE записывает на диск программу, находящуюся в памяти машины. Сейчас мы зададим вам вопрос на сообразительность. Как вы думаете, в каком виде хранится программа в памяти машины? Первое, что приходит в голову, - в виде текста. Предположим, что вы ввели строку

10 PRINT "Я ЭВМ Корвет"

Соответственно, в ячейках памяти хранится текст строки. Так, конечно, можно, но неудобно. Почему? Из-за времени выполнения программы. Ведь для определения типа оператора (а машина может это сделать, только сравнивая побайтно ваш текст со списком операторов) потребуется масса времени. Поэтому в Бейсике для ускорения процесса работы программы операторы кодируются. Каждому оператору ставится в соответствие один байт. Благодаря этому процесс интерпретирования программ существенно ускоряется.

С этим вопросом ясно. А если мы хотим все-таки получить полный текст программы без кодов и шифров? В этом случае необходимо в операторе SAVE после имени файла указать, что мы хотим записать на диск программу в виде текста

SAVE "ИМЯ ПРОГРАММЫ",A

Буква A как раз и делает это. Возможно, у вас возник вопрос: зачем, собственно, иметь на диске программу в виде текста? Ясно, что кроме увеличения пространства, занятого программой на диске, мы ничего не выиграем. Потерпите немного. Скоро мы сможем ответить и на этот вопрос.

Теперь займемся считыванием программы с диска. Для этого служит оператор LOAD. После того как программа загружена, ее можно начать выполнять, набрав команду RUN. Если вам не хочется лишний раз вводить команду RUN, можно сразу при загрузке сообщить машине, что программу нужно запустить на счет. Это достигается путем добавления к оператору LOAD буквы R:

LOAD "ИМЯ ПРОГРАММЫ",R

В этом случае программа будет загружена и без дальнейших вопросов запущена на счет.

Теперь несколько слов об имени программы. Оно должно существовать в оглавлении диска, т. е. программа на диск должна быть записана. Если вы хотите считать программу с текущего диска, то необходимо просто указать ее имя, например:

LOAD "EXAMPLE.BAS"
LOAD "PROBA.BAS",R

То же самое относится к записи программы на текущий диск:

```
SAVE "MYFILE.BAS"
SAVE "MYPROG.BAS",A
```

Обращаем ваше внимание, что если эти программы уже существуют на диске, то все их содержимое будет стерто и заменено на новую информацию. При работе с другими дисками в имени программы необходимо указать имя диска, например:

```
LOAD "B:EXAMPLE.BAS"
LOAD "B:PROBA.BAS",R
SAVE "B:MYFILE.BAS"
SAVE "B:MYPROG.BAS",A
```

Если имя программы имеет расширение .BAS, его в операторах SAVE, LOAD и MERGE можно не указывать. Машина и так поймет. Так, команды

```
LOAD "PROG.BAS"
LOAD "PROG"
```

приведут к одному и тому же результату.

Возможно, у вас возник вопрос: если у меня есть несколько программ, например две, и мне хочется их объединить, что для этого сделать?. Для этой цели служит оператор MERGE. Чтобы им воспользоваться, введите команду

```
MERGE "ИМЯ ПРОГРАММЫ"
```

Запомните, что для успешной работы этого оператора необходимо подсоединяемую программу записать в текстовом виде, т. е. с помощью команды

```
SAVE "ИМЯ ПРОГРАММЫ",A
```

В противном случае вы получите сообщение об ошибке:

```
НЕВЕРЕН РЕЖИМ ФАЙЛА
```

Проиллюстрируем сказанное примером. Допустим, у вас имеются две программы:

```
10 CLS
20 PRINT "Я ЭВМ Корвет"

30 INPUT "Как вас зовут?"N$
40 PRINT "Привет";N$;"!"
50 INPUT "Что нового?";A$
60 PRINT A$;"Это очень интересно"
70 GOTO 50
```

Мы хотим эти программы объединить. Сначала задумаемся, как происходит объединение программ. Предположим, что первая программа уже находится в памяти машины. При подсоединении вторая программа разместится в соответствии с номерами своих строк. В данном случае непосредственно после первой. А если какие-либо строки в

разных программах будут иметь одинаковые номера? В этом случае строки в первой программе, имеющие номера, аналогичные строкам второй программы, будут просто замещены. Вместо них будут фигурировать строки второй программы. Поэтому следите за тем, чтобы строки программ не перекрывались.

Теперь займемся экспериментом. Введите в память машины первую программу и после этого команду

```
SAVE "PROG1"
```

для записи ее на диск. Теперь введите команду

```
NEW
```

и вслед за этим наберите вторую программу. Запишите ее на диск:

```
SAVE "PROG2",A
```

Теперь соберем эти программы вместе:

```
LOAD "PROG1"
MERGE "PROG2"
```

Посмотрим, что получилось. Для этого введем команду LIST. Вы должны увидеть текст объединенной программы.

Теперь посмотрим, как объединяются программы с перекрывающейся нумерацией строк. Отредактируем вторую программу. Для этого выполните следующие действия:

```
LOAD "PROG2"
15 REM Программа, демонстрирующая операторы SAVE, LOAD,
   MERGE
```

Запишите эту программу:

```
SAVE "PROG2",A
```

и повторите процесс сшивания программ:

```
LOAD "PROG1"
MERGE "PROG2"
```

Теперь посмотрим, что получилось:

```
10 CLS
15 REM Программа, демонстрирующая операторы SAVE, LOAD,
   MERGE
20 PRINT "Я ЭВМ Корвет"
30 INPUT "Как вас зовут?"N$
40 PRINT "Привет";N$;"!"
50 INPUT "Что нового?";A$
60 PRINT A$;"Это очень интересно"
70 GOTO 50
```

Вы видите, что строка 15 второй программы попала между строками 10 и 20 первой.

А что будет, если в первой и второй программах будут строки с одинаковыми номерами? В этом случае строки первой программы будут заменены соответствующими строками второй программы. Из этого следует важное замечание. Следите за тем, чтобы объединяемые программы не пересекались с уже имеющимися в памяти машины.

Теперь вы уже многоопытный программист. Вами созданы сотни (а может быть и тысячи) программ на языке Бейсик. Они записаны на многих дисках, и, естественно, вы не помните, какая на каком. Если вы еще не запустили Бейсик, то посмотреть, есть ли нужный вам файл на диске или нет, очень просто. Для этого служит команда DIR. Можно ли посмотреть директорию диска прямо из Бейсика? Конечно, можно. Для этого существует специальный оператор FILES:

FILES "ИМЯ ДИСКА:ИМЯ ФАЙЛОВ"

Например:

FILES - выводит оглавление текущего диска
FILES "B:" - выводит оглавление диска B
FILES "B:*.BAS" - выводит информацию о наличии в оглавлении имен всех файлов с расширением BAS

Другими словами, синтаксис оператора FILES вполне аналогичен DIR, с той лишь разницей, что дополнительную информацию нужно заключать в кавычки.

Вы можете вывести информацию об оглавлении диска на принтер. Для этого используется оператор

LFILES "ИМЯ ДИСКА:ИМЯ ФАЙЛА"

Правила использования этого оператора те же, что и для оператора FILES.

Мы знаем, что в операционной системе CP/M есть специальный оператор для стирания файлов. Неужели Бейсик здесь уступит CP/M? Отнюдь нет. Такой оператор есть:

KILL "ИМЯ ДИСКА:ИМЯ ФАЙЛОВ"

Для тех, кто знает английский язык, этот оператор выглядит пугающе. Действительно, KILL в переводе на русский язык означает "убивать". Приведем примеры:

KILL "PROG.BAS" - удалить файл PROG.BAS
KILL "B:PROG1.BAS" - удалить файл PROG1.BAS с диска B
KILL "*.*" - удалить все файлы с текущего диска

Итак, мы научились записывать и считывать программы. Тем самым мы решили первую проблему, возникшую при работе с дисками. Теперь займемся второй проблемой. Ее можно сформулировать следующим образом. Допустим, что у нас есть две программы. Одна из них использует результаты, полученные при работе другой. Такие задачи возникают, например, в тех случаях, когда одна из программ работает очень медленно, но зато ее результатами можно пользовать-

ся в течение продолжительного времени. Чтобы постоянно не запускать такую программу, нужно просто перенести то, что она посчитала, в другую программу. Как это сделать?

Простейший способ - распечатать эти данные на бумаге и потом, по мере надобности, вводить их с клавиатуры. Но это хорошо, если таких данных не много. Реально не больше десятка чисел. А если больше? В этом случае выходом из положения может оказаться запись данных на диск одной программой и считывание их другой. Для того чтобы этот процесс реализовать, необходимо создать некий файл, который и будет содержать результаты расчетов.

Для работы с файлами прежде всего необходимо подготовить эти файлы к работе. Мы должны объяснить машине, как файлы называются, что мы хотим с ними делать и, наконец, пронумеровать их. Последнее необходимо для того, чтобы машина в дальнейшем оперировала с номером файла, а не с его именем. Для подготовки файла к работе служит оператор OPEN:

OPEN "РЕЖИМ", # НОМЕР ФАЙЛА, "ИМЯ ФАЙЛА"

Режим означает: "O" - запись в файл (OUTPUT), "I" - считывание из файла (INPUT). Отметим, что если задан режим "O", то все данные, бывшие в файле, будут уничтожены.

Теперь мы открыли файл. Что дальше? Прежде всего, поймем, как машина работает с файлом. Для того чтобы ориентироваться в содержимом файла, машина заводит указатель текущей позиции в файле. Если вам интересно, в каком месте в данный момент находится указатель, то вы можете воспользоваться функцией

LOC (НОМЕР ФАЙЛА)

Эта функция определяет номер позиции указателя. Если вас интересует число записей в файле (как вы помните, длина одной записи равна 128 байт), вы можете это узнать при помощи функции

LOF (НОМЕР ФАЙЛА)

Она вычисляет искомую величину.

Ну, а теперь самое интересное: как записать и считать данные?

Для записи данных служит уже знакомый нам оператор PRINT. Однако в данном случае он претерпевает небольшую косметическую операцию, чтобы не путать его с обыкновенным оператором PRINT:

PRINT # НОМЕР ФАЙЛА, СПИСОК ПЕРЕМЕННЫХ

Если вы хотите записывать данные в файл по определенному формату, то для этого, как вы догадались, служит оператор

PRINT # НОМЕР ФАЙЛА USING "ФОРМАТ#",
СПИСОК ПЕРЕМЕННЫХ

Правила работы с этими операторами те же, что и для обыкновенных операторов вывода информации на экран.

Далее: как считать данные из файла? Прежде всего, файл должен быть открыт на считывание (режим I). После этого текущая позиция устанавливается равной нулю, и вы можете последовательно считывать данные. Делают это с помощью операторов

INPUT # НОМЕР ФАЙЛА, СПИСОК ПЕРЕМЕННЫХ

Заметим, что поскольку в операторе INPUT не содержится никаких сведений насчет формата данных в файле, необходимо использовать переменные одинакового типа при записи и считывании информации. Для считывания строковых переменных используется особый оператор

LINE INPUT # НОМЕР ФАЙЛА, СТРОКОВАЯ ПЕРЕМЕННАЯ

Остановимся ненадолго на том, как происходит запись и считывание данных. Машина загружает в файл данные байт за байтом. Число записанных байтов определяется типом переменных. Таким же способом происходит считывание информации. Вообще говоря, записывать информацию можно двумя способами. Первый способ называется бесформатным. В этом случае мы записываем переменные в том виде, в каком они хранятся в памяти машины. При этом, если посмотреть на то, что получилось, нельзя прочитать такой файл, не зная, какие именно там числа. Действительно, мы знаем, что целое число занимает в памяти машины два байта, а число с одинарной точностью - четыре. Поэтому если мы записали в файл такое число и забыли, что оно действительное с одинарной точностью, то впоследствии оно может легко сойти за два целых. Таким образом, при использовании бесформатной записи необходимо четко помнить, какой тип переменных в файл записан, и еще полезно знать, сколько их там. Вы можете сказать - очень сложно. Да, но у этого способа есть одно существенное преимущество: получаемые таким образом файлы имеют малый размер. Почему? Это мы поймем, сравнив бесформатный способ записи с записью по формату.

Итак, второй способ - запись по формату. Что это такое? Допустим, что вы хотите записать в файл какое-нибудь целое число, скажем 12345. В случае бесформатной записи это будут два байта. Если мы производим форматную запись, мы должны записать в файл пять байтов, по байту на каждую значащую цифру. При этом записывается не сама цифра, а ее ASCII код. Как вы видите, проигрыш по объему файла существенный. Однако этот способ имеет ряд преимуществ. Вы можете легко распечатать такой файл и прочесть его содержимое. Далее его легко читать, даже забыв, что там за числа.

Еще одно полезное свойство. ASCII коды имеют некоторый диапазон своей величины. Часть кодов никогда не используется в текстах. Эти коды являются контрольными. Примером может служить, как вы помните, код клавиши ВК, он равен 13 (или 0DH в шестнадцатиричной системе). Один из контрольных кодов 1AH используется для обозначения конца файла (на клавиатуре он соответствует CTRL-Z).

Благодаря этому вы всегда можете проверить, достигли ли вы при чтении конца файла. Для этой цели используется функция

EOF (НОМЕР ФАЙЛА)

Она принимает значение 0, если конец файла не обнаружен, и -1, если обнаружен.

Теперь о том, что имеет место в Бейсике. Язык Бейсик производит только форматную запись файлов. Правила работы с операторами PRINT# и PRINT# USING аналогичны PRINT и PRINT USING.

Коснемся теперь строковых переменных. Как определить размер строки? Вы скажете: использовать функцию LEN. Совершенно верно. Займемся вопросом, как она работает. Искушенный программист скажет, что раз изобрели контрольный код для окончания файла, наверное, придумали также код, обозначающий конец строки. Совершенно верно. В качестве такого кода используют обычно 0, 0AH или 0DH. Последний вам хорошо знаком. Это код клавиши ВК. Код 0AH обозначает команду "перевод строки".

Таким образом, оператор считывания строковых переменных из файла

LINE INPUT #

считывает из вашего файла информацию, пока не встретит один из перечисленных выше кодов. В этом случае то, что считалось, будет присвоено строковой переменной.

После того как вы кончите работать с файлом, необходимо сообщить об этом приятном факте машине. По-научному это называется закрыть файл. Для этого служит специальный оператор

CLOSE НОМЕР ФАЙЛА

Можно закрывать не один, а несколько файлов одним оператором:

CLOSE 1,2,4,7

Это значит, что будут закрыты файлы с номерами 1, 2, 4 и 7. Если использовать оператор CLOSE без операндов, то будут закрыты все открытые файлы.

Ну, а теперь поупражняемся. Давайте поговорим с машиной о том, о сем и запишем наш диалог на память потомкам. Для этого нам потребуется программа разговора с машиной:

```
10 CLS
20 PRINT "Я ЭВМ Корвет"
30 INPUT "Как вас зовут?"N$
40 PRINT "Привет";N$;"!"
50 INPUT "Что еще скажешь?";A$
60 PRINT A$;"Говоришь... Очень интересно"
70 GOTO 50
```

Запустите эту программу и поговорите с машиной. Когда захотите записать ваш разговор, добавьте строки

```
15 OPEN "0",#1,"DIAL.TXT"
35 PRINT #1,N$
55 PRINT #1,A$
```

и модифицируйте конец программы:

```
70 INPUT "Продолжим (Да/Нет)";M$
80 IF M$="Да" OR M$="ДА" OR M$="да" THEN 50 ELSE CLOSE 1
```

Теперь вы можете быть уверены, что ваш диалог с машиной останется в целости, даже если вы выключите ее.

А как проверить, правильно ли записан ваш разговор? Для этого сочиним программу считывания:

```
90 OPEN "I",#1,"DIAL.TXT"
100 LINE INPUT #1,N$
110 PRINT "Привет";N$;"!"
120 PRINT "Только что вы мне сказали";
130 LINE INPUT #1,A$
140 PRINT A$
150 IF EOF(1)=0 THEN 130 ELSE CLOSE 1
```

Понятно ли вам, как работает эта программа? Обратите внимание на строку 150. Она проверяет, достигли ли мы конца файла. Если нет, программа считывает следующую фразу.

Надеемся, что теперь с дисками все ясно. Самое время заняться кассетным магнитофоном. Как вы помните, на кассете программы пишутся одна за одной и никакой возможности программно их разделить нет. Поэтому для определения, где какая программа записана, необходимо пользоваться счетчиком расхода ленты в магнитофоне. Для операций с магнитофоном служат операторы

```
CSAVE
CLOAD
MOTOR ON/OFF
```

Первый оператор записывает программу на кассету, второй ее считывает. Если вы вводите команду MOTOR ON, включается мотор в магнитофоне. Он выключается при подаче команды MOTOR OFF.

Последовательность работы с магнитофоном следующая:

- вы находите по счетчику то место на кассете, куда хотите записать программу;
- вводите команду MOTOR OFF;
- нажимаете клавиши воспроизведения или записи;
- вводите команду MOTOR ON;
- вводите команду CLOAD или CSAVE;
- выключаете мотор

При загрузке программ в некоторых версиях языка Бейсик в правом верхнем углу экрана появляется мигающий символ *. Если считывание или запись прошли успешно, то появится промпт Бейсика OK. В противном случае вы увидите сообщение об ошибке.

21. ПРОЩАНИЕ С БЕЙСИКОМ

Раздел, в котором мы завершим обсуждение языка Бейсик

Мы надеемся, что читатель уже стал профессионалом высшей квалификации в области программирования на языке Бейсик. Фактически мы изучили все операторы и команды этого языка. Но прежде чем перевернуть последнюю страницу и отправиться в самостоятельное плавание по безбрежному океану вычислительной науки, оглянемся назад и охватим взглядом картину, остающуюся за нашими плечами.

В действительности, хотя мы уже все про Бейсик знаем и все умеем, остается открытым вопрос о связи таких команд, как NEW, RUN, LIST, т. е. тех, которые мы вводили с клавиатуры, с теми операторами, которые мы использовали в наших программах. Попробуем составить представление о Бейсике как о едином целом. Бейсик представляет собой программу, которая умеет практически все. Он сам себе операционная система. В принципе для его функционирования ОС CP/M вообще не нужна. Бейсик сам умеет общаться с пользователем клавиатуры и дисплея, сам умеет обращаться к различным периферийным устройствам. Если вы полюбили Бейсик всем сердцем и ничего больше не желаете, то вам достаточно знать, как его загрузить, и больше ничего из ОС CP/M вам не нужно.

Что же делает Бейсик в то время, когда мы не запустили программу? Он ждет наших команд. Интерпретатор этого языка устроен так, что ему абсолютно безразлично, откуда он получил команду - с клавиатуры или из программы. Поэтому в смысле интерпретации все команды языка Бейсик абсолютно равноправны. Их можно вводить как с клавиатуры, так и вставлять в текст программы.

Давайте попробуем. Начнем с наиболее часто используемой нами команды RUN. Для демонстрации напишите программу

```
10 PRINT "Мне нравится работать"
20 RUN
```

Запустите ее. Вы видите, как на экране дисплея замелькала надпись, стоящая после оператора PRINT. Как работает эта программа, надеемся, ясно всем. Попробуем вставить в эту программу еще что-нибудь экзотическое, например:

```
15 NEW
```

Теперь запустим программу. На экране появится одно-единственное сообщение:

```
Мне нравится работать
```

и после этого промпт Бейсика OK. Естественный результат. Когда интерпретатор дошел до строки с номером 15, он стер старую программу. Программа стерла сама себя! После этого выполнять стало нечего и появился промпт, сообщающий о готовности интерпретатора

воспринимать ваши команды. Если вы теперь введете команду LIST, то никакой программы в памяти машины вы не обнаружите. Поупражняйтесь с другими командами типа LIST, LOAD, MERGE и т. д.

Теперь мы убедились в полном равноправии команд Бейсика. Зададим себе вопрос: вот мы прервали выполнение программы, нажав клавишу СТОП; что произошло с нашей программой? Самое интересное, что все осталось в целости и сохранности. Не только текст программы, но и значения всех переменных. Раз так, то, по-видимому, выполнение программы можно продолжить. Как это сделать? Сейчас мы узнаем, каким способом можно активно вмешиваться в работу программы. Рассмотрим пример:

```
10 A=2
20 PRINT A
30 IF A=2 THEN 20
```

Запустите эту программу. Легко понять, что происходит. Поскольку мы не изменяем величину А, то машина выполняет бесконечный цикл. Остановите программу, нажав клавишу СТОП. Вы можете проверить,чему равно значение переменной А. Для этого введите команду

```
PRINT A
```

На экране появится: 2. Машина все еще помнит значение А. Попробуем продолжить выполнение программы. Для этого придуман специальный оператор

```
CONT
```

Получив эту команду, машина продолжает выполнение программы, начиная с того места, где программа была прервана. Спрашивается, зачем прерывать программу, а затем вновь ее запускать? Очень просто. В тот момент, когда программа прервана, вы можете изменить значения любых переменных, фигурирующих в программе.

Итак, мы прервали нашу программу. Введите команду

```
A=3
```

После этого попробуйте продолжить выполнение программы:

```
CONT
```

Теперь на дисплее не появится мелькающая цифра 3. Естественно: условие в строке 30 окажется невыполненным и программа остановится. Можно проверить, чему равно значение переменной А:

```
PRINT A
```

На экране появится ответ: 3, т. е. машина помнит значения переменных и после окончания работы программы.

Теперь подумаем о том, все ли время машина помнит эти значения. Попробуем добавить в программу новую строку

```
25 PRINT A*2
```

Проверьте, чему теперь равно значение переменной A:

```
PRINT A
0
```

Забыла... А почему? Дело в том, что добавление строки в программу (равно как и ее удаление или редактирование) изменяет объем памяти, занятый программой. Вследствие этого происходит перераспределение памяти. Поэтому все адреса сбиваются и машина, чтобы не было недоразумений, записывает нули во все ячейки и ОЗУ, отведенные под переменные. Попробуем поуправлять программой самостоятельно. Введите команду

```
GOTO 20
```

На экране появится ответ:

```
0
0
```

Значит, машина выполнила строки 20, 25 и 30.

Попробуем следующую последовательность команд:

```
A=3
GOTO 20
```

На экране появится:

```
3
6
```

Теперь попробуем так:

```
A=3
RUN 20
```

Получим ответ:

```
0
0
```

Команду

```
RUN НОМЕР СТРОКИ
```

мы еще не рассматривали, но умудренный опытом программист уже, наверное, догадался, что она значит. Совершенно верно, мы указываем машине, с какой строки выполнять программу. Обратите внимание, что по команде RUN все переменные принимают значение, равное нулю.

Резюмируя вышесказанное, подчеркнем, что вмешиваться в выполнение компьютером программ имеет смысл только в тех случаях, когда это экономит машинное время, например при ошибочном вводе одного из параметров, если их вводится большое число и перед этим машина долго считала. В этом случае вы можете исправить ошибочно

введенную информацию и возобновить выполнение программы с нужной строки.

Теперь вернемся к вопросу о нумерации строк в программе. Вы помните, что хорошо выбирать приращение номера равным 10, 100 и т. д. Это целесообразно, чтобы при необходимости иметь возможность вставить новую строку в программу. Ну, а если мы вставляли строки и, в конце концов, исчерпали все свободное место? Что же, теперь набивать всю программу заново? Конечно, нет. В Бейсике для этого существует специальный оператор RENUM:

RENUM НОВЫЙ НОМЕР, СТАРЫЙ НОМЕР

По этой команде Бейсик перенумерует все строки в соответствии с полученной инструкцией. Строке с номером, указанным как старый номер, будет присвоен новый номер. Остальные идущие следом строки будут перенумерованы в соответствии с величиной приращения.

Займемся теперь еще одним важным вопросом. Вы неоднократно получали сообщения об ошибках. Часть из них имела фатальный характер, например ОШИБКА СИНТАКСИСА. В этом случае делать нечего, нужно исправлять соответствующую строку. Однако существуют ошибки, которые можно устранить, не погубив все предшествующие расчеты. Для этого существуют специальные функции обработки ошибок. Сейчас мы с ними познакомимся.

Для начала отметим, что все ошибки имеют код. Этот код присваивается специальной переменной ERR. Для примера сотрите все ваши предшествующие упражнения из памяти:

```
NEW
```

Введем какой-нибудь ошибочный оператор, скажем

```
NEXT
```

Естественно, мы увидим сообщение об ошибке:

```
NEXT БЕЗ FOR
```

Посмотрим на код этой ошибки:

```
PRINT ERR
1
```

т. е. код этой ошибки равен 1.

Раз все ошибки имеют код, то легко научить машину их устранять. Для этой цели служат команды

ON ERROR НОМЕР ОШИБКИ GOTO НОМЕР СТРОКИ
RESUME НОМЕР СТРОКИ

Эти операторы должны присутствовать в программе совместно. Первый оператор ставится обычно в начале программы и говорит машине, с какой строки начинается подпрограмма обработки ошибки. Второй оператор завершает подпрограмму обработки ошибки и говорит машине, с какой строки продолжать выполнение программы. Вы можете

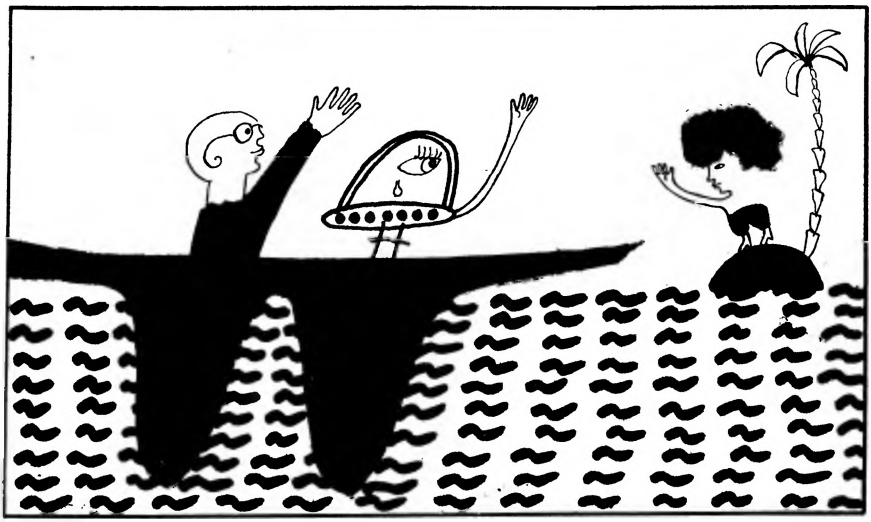
либо указать номер строки явно, либо воспользоваться следующими вариантами:

```
RESUME NEXT
RESUME 0
```

В первом случае выполнение программы продолжится, начиная с оператора, следующего за оператором, вызвавшим ошибку; во втором случае машина попробует снова выполнить оператор, вызвавший ошибку. Естественно, чтобы это сделать, мы должны проанализировать, что это за ошибка, и устранить ее.

Различных ошибок довольно много. С некоторыми из них мы уже встречались. Смысл того, где и какая была допущена ошибка, всегда ясен из соответствующих сообщений и из внимательного анализа строки, при выполнении которой появилось сообщение об ошибке. В общем случае дадим хотя и очевидный, но полезный совет: тщательно проверяйте программы. Тогда не будут появляться сообщения об ошибках. Приведем здесь сушествующие ошибки и их коды:

Код ошибки	Причина ошибки
1	NEXT БЕЗ FOR
2	ОШИБКА СИНТАКСИСА
3	RETURN БЕЗ GOSUB
4	ВНЕ DATA
5	НЕВЕРЕН ВЫЗОВ ФУНКЦИИ
6	ПЕРЕПОЛНЕНИЕ
7	НЕТ ПАМЯТИ
8	НЕОПРЕДЕЛЕННЫЙ НОМЕР СТРОКИ
9	ИНДЕКС ВНЕ ДИАПАЗОНА
10	ПЕРЕОПРЕДЕЛЕНИЕ МАССИВА
11	ДЕЛЕНИЕ НА НУЛЬ
12	ОШИБОЧНАЯ ДИРЕКТИВА
13	ОШИБОЧНЫЙ ТИП
14	НЕТ ПАМЯТИ ДЛЯ СТРОК
15	ДЛИННАЯ СТРОКА
16	ОЧЕНЬ СЛОЖНАЯ СТРОКА
17	ОШИБКА CONT
18	НЕТ ФУНКЦИИ ПОЛЬЗОВАТЕЛЯ
19	НЕТ RESUME
20	RESUME БЕЗ ОШИБКИ
21	НЕОПРЕДЕЛЕННАЯ ОШИБКА
22	НЕТ ОПЕРАНДА
23	БУФЕР СТРОКИ ПОЛНЫЙ
51	ВНУТРЕННЯЯ ОШИБКА
52	НЕВЕРЕН НОМЕР ФАЙЛА
53	ФАЙЛ НЕ НАЙДЕН
54	НЕВЕРЕН РЕЖИМ ФАЙЛА
55	ФАЙЛ УЖЕ ОТКРЫТ



57	ОШИБКА ВВОДА/ВЫВОДА ДИСКА
58	ФАЙЛ УЖЕ СУЩЕСТВУЕТ
61	ДИСК ПОЛНЫЙ
62	ВВОД ПОСЛЕ END
64	НЕВЕРНО ИМЯ ФАЙЛА
66	ПРЯМОЙ ОПЕРАТОР В ФАЙЛЕ
67	МНОГО ФАЙЛОВ

Отметим, что в различных версиях языка Бейсик могут встречаться и другие ошибки. Поэтому внимательно изучите прилагаемое к машине руководство оператора.

Ну, вот и все. Осталась последняя команда языка Бейсик. Это команда возвращения в лоно операционной системы. Наберите

SYSTEM

и... farewell, Бейсик!

22. ТЕКСТОВАЯ ИНФОРМАЦИЯ И БАЗЫ ДАННЫХ

Продолжаем разговор о строковых переменных
и их роли в повседневной жизни.
Как писать книги с помощью компьютера

Мы вступаем в совершенно новую область применения вычислительной техники. До сих пор мы учились записывать и считывать файлы, вычислять, сколько будет дважды два, и прочим полезным вещам. Сейчас, для расширения нашего кругозора, коснемся проблемы обработки текстов. При изучении языка Бейсик мы уже сталкивались с чем-то подобным, когда возились с текстовыми переменными. Тогда мы научились складывать строки, выделять их различные части.

Возможно, у вас в то время сложилось мнение, что это просто плод больной фантазии. Просто программисты не знают, чего еще напихать в Бейсик, и придумывают всякие глупости. Но это не так. Эти операции очень важны. Прежде чем мы поймем их значение, давайте пораскинем мозгами и подумаем, что такое строковая переменная с микроскопической точки зрения. Тут все ясно. Это набор букв. Но мы знаем, что набор элементов на языке Бейсик называется массивом. У массива есть индексы, по которым можно перебирать их элементы. Если вы вспомните все, что мы говорили в разделе 16, то вы легко сможете определить размерность массива с помощью оператора LEN, а также выделить отдельные символы с помощью оператора MID. Вы также помните упражнение с автоматическим цензором и переводчиком.

Это все элементарные ячейки, которые входят в состав программ-редакторов текстов. В данном разделе мы хотим познакомить вас с логикой и принципами их работы. По ходу изложения постараемся иллюстрировать упоминаемые функции короткими программами на языке Бейсик.

Что такое текст? Это набор строк, массив строковых переменных. Соответственно, первая функция редактора - это возможность просмотра текста построчно. Эту функцию в Бейсике выполняет команда LIST.

Задача редактора - отделять одну строку от другой. Конечно, когда все строки вашего текста аккуратно расфасованы в памяти ЭВМ и каждая снабжена индексом, нет ничего проще вывода на экран

любого количества строк в любом порядке. Следовательно, желательно иметь функцию, которая бы умела выделять в потоке информации строки.

Для отделения одной строки от другой используется, как вы знаете, специальный код - терминатор, который ставится в конце строки. Терминаторами обычно являются 0, 0AH (перевод строки) и 0DH (возврат каретки, ВК). Это вы знаете. Давайте теперь попробуем создать такую функцию самостоятельно. Отличие в нашем случае будет только в выборе терминатора. Его в учебных целях желательно иметь в наблюдаемом на экране виде. Выберем, например, в качестве терминатора символ %.

Сначала создадим модель нашего текста:

```
10000 DATA ДЕМОНСТРАЦИЯ ТЕКСТОВЫХ ПРОГРАММ.%
      СТРОКА1.% СТРОКА2.% СТРОКА3.%
      СТРОКА4.% СТРОКА5.%
```

Обратите внимание, что наши модельные строки текста разделены нашим терминатором %, а не запятой, как обычно в операторе DATA.

Попробуем теперь написать функцию, которая бы разделила этот текст на отдельные строки и соорудила бы из них массив. Мы, как программисты передовых взглядов, будем оформлять наши функции как отдельные подпрограммы. Поэтому сначала начнем создавать управляющую программу-меню:

```
10 CLS:DIM TX$(20):CLEAR 1000
20 READ A$
30 GOSUB 1000
900 END
1000 REM Подпрограмма, разделяющая текст на строки
1010 I=0:TX$(0)=" "
1020 FOR J=1 TO LEN (A$)
1030 LN$=MID$(A$,J,1)
1040 TX$(I)-TX$(I)+LN$
1050 IF LN$="%" THEN I=I+1:TX$(I)=" "
1060 NEXT J
1070 IMAX=I-1:RETURN
```

Надеемся, вам понятно, как эта программа работает и что делает. Сначала весь текст считывается в строковую переменную A$, а потом вступает в свои права программа-сортировщик. Она по одному символу перебирает весь текст и, найдя терминатор %, заполняет соответствующий элемент массива.

Если вы хотите проверить, заполнился ли наш массив, то после выполнения программы попробуйте посмотреть элементы массива, например:

```
PRINT TX$(3)
СТРОКА 3.%
```

Все в порядке. Теперь наш текст оформлен в виде массива.

Следующая наша задача - научиться выводить текст на экран. Казалось бы, чего проще. Вводим команду

```
PRINT TX$(I)
```

и порядок. Но это потому, что у нас есть Бейсик. А если мы создаем редактор без Бейсика? Что тогда делать? Естественно, писать программу, выводящую все символы строки без терминатора на экран:

```
2000 REM Отображение строки
2010 FOR I=0 TO IMAX
2020 GOSUB 2100 REM Программа отображения строки
2030 PRINT:NEXT I
2040 RETURN
2100 REM Программа отображения строки
2110 J=1
2120 C$=MID$(TX$(I),J,1)
2130 IF C$="%" THEN RETURN ELSE PRINT C$;:J=J+1:GOTO 2120
```

Как вы видите, эта подпрограмма состоит собственно из двух программ. Одна из них отображает строку, а другая весь текст и использует первую как подпрограмму. Добавим теперь в основную программу вызов этой функции:

```
40 GOSUB 2000
```

Теперь можно посмотреть, как наша программа работает с текстом.

Говоря научным языком, мы сейчас подготовили структуру данных (или базу данных) для хранения информации в текстовом редакторе. В данном случае такой структурой является простой массив. Теперь нам легко научить машину делать все что угодно с текстом. Например, мы хотим получить возможность вставить в текст строку. Это уже серьезная операция, которая при редактировании встречается часто. Поэтому для начала модифицируем основную программу:

```
50 PRINT "1.Вставка"
60 PRINT "2.Удаление"
70 PRINT "3.Запись на диск"
80 PRINT "4.Чтение с диска"
90 PRINT "5.Вывод на экран"
100 INPUT "Выберите функцию";FU
110 ON FU GOSUB 3000,4000,5000,6000,7000
120 GOTO 50
```

Теперь напишем соответствующие процедуры:

```
3000 REM
3010 INPUT "Введите номер строки";NS
3020 FOR I=IMAX TO NS STEP-1
3030 TX$(I+1)=TX$(I)
3040 NEXT I
```

```
3050 INPUT "Введите новую строку";TX$(NS)
3060 TX$(NS)=TX$(NS)+"%":IMAX=IMAX+1:RETURN
```

Попробуйте ввести новую строку и посмотрите весь текст на экране.

Подпрограмма удаления строки выглядит следующим образом:

```
4000 REM
4010 INPUT "Введите номер строки";NS
4020 FOR I=NS TO IMAX
4030 TX$(I)=TX$(I+1):NEXT I
4040 RETURN
```

Теперь уже можно сочинять любые тексты. Однако прежде чем писать что-либо ценное, научимся спасать наши произведения:

```
5000 REM
5010 INPUT "Введите имя файла";FN$
5020 OPEN "O",#1,FN$
5030 FOR I=0 TO IMAX
5050 PRINT #1,TX$(I)
5060 NEXT I:CLOSE 1
5070 RETURN
```

Раз мы записали текст на диск, давайте научимся его считывать:

```
5000 REM Чтение с диска
5010 INPUT
6020 OPEN "I",#1,FN$
6030 I=0
6040 IF EOF(1)<=>-1 THEN LINE INPUT TX$(I):I=I+1
6050 IMAX=I:RETURN
```

Итак, теперь мы можем работать с текстом. В реальном текстовом редакторе существует еще масса функций - для редактирования строк, вывода на экран определенных частей текста, поиска тех или иных слов и целых фраз и т. д. Важно понять, что, в основном, все редакторы устроены так же, как наша примитивная программа. В основе их лежит простой массив строковых переменных. Для работы с текстами вручную такая организация вполне приемлема. А для обработки более сложной информации, к сожалению, нет. Причина этого - слишком большие затраты времени на перелопачивание массива при изменении текста.

Прежде чем рассмотреть другие способы организации хранения и обработки информации, отметим, что в состав программного обеспечения, комплектующего Корвет, входит один из наиболее мощных современных редакторов текста Супертекст. Эта программа - аналог известного в мире редактора Final Word. Этот редактор реализует множество функций, необходимых для работы с текстами. Книга, которую вы в настоящее время держите в руках, была набрана целиком в этом редакторе и представлена в издательство на магнитном диске. Трудно в это поверить, но это правда.

Ну а теперь, когда с текстами имеется определенная ясность, займемся более совершенными методами обработки информации.

Часто нам требуется обрабатывать более сложную информацию, чем просто текст. Например, мы хотим завести в машине программу, которая обрабатывала бы каталоги. Примером такого каталога является штатное расписание. Если делать эту работу дедовским способом, то необходимо переписать всех сотрудников вашего подразделения на большой разграфленный лист бумаги, указать в соответствующих графах их должность, оклад, различные надбавки и т. д.

Как вы понимаете, такой способ составления штатного расписания аналогичен массиву строковых переменных, рассмотренному нами ранее. Такой путь вполне допустим. Но попробуйте внести в список новых сотрудников. В этом случае вам придется взять новый лист бумаги и переписать все штатное расписание заново. Не очень приятно. А как же машина только что перелопачивала весь текст и не жаловалась? Теперь мы поняли, как ей тяжело заниматься нудной и в общем бесполезной работой.

Давайте подумаем, как нам облегчить процесс сортировки. Человечество, начав писать книги, естественно, столкнулось с проблемой, где их хранить. Появились библиотеки. Пока в библиотеке имеется несколько книг, проблем с их поиском не возникает. А если их много тысяч? Конечно, все сталкивались с библиотечным каталогом. Это ящики, набитые карточками, на которых написана подробная информация о книге. Если у нас появилась новая книга, то мы просто поместим в нужное место новую карточку, не трогая весь остальной каталог. Это, конечно, гораздо удобнее и проще, чем предыдущий пример. Но с карточками возникает новая проблема - как бы их не перепутать. Простейший выход - связать их одну за другой веревочкой. Таким образом у нас получится база данных, называемая связанным списком.

Давайте попробуем создать подобную базу данных и для нашего штатного расписания. Сделаем оговорку. Поскольку в Бейсике ничего другого для хранения информации, кроме массивов, не придумано, будем их использовать необычным способом.

Поставим задачу. Что нам нужно знать о сотруднике?

1. Имя
2. Должность
3. Оклад

Конечно, возможны и другие сведения, но для простоты ограничимся пока этими. Соответственно заведем три массива:

```
NAME$(20)
JOB$ (20)
PAY (20)
```

Размерность 20 взята условно. Если хотите завести больше сотрудников, увеличьте ее.

Кроме этих массивов, элементы которых составляют содержимое карточек, нам необходима еще и веревочка, которая будет последовательно связывать карточки. Такой веревочкой служит еще один массив, задающий очередность карточек:

```
IND (20)
```

В этом массиве мы будем хранить номера карточек. Давайте теперь напишем программу, заполняющую нашу базу данных:

```
5 CLS:CLEAR 1000:I=0
10 DIM NAME$(20),JOB$(20),PAY(20),IND(20)
20 INPUT "Имя";NAM$(I)
30 INPUT "Должность";JOB$(I)
40 INPUT "Оклад";PAY(20)
50 IND(I)=I+1:
60 INPUT "Продолжать?<Д/Н>";A$
70 IF A$="Д" OR A$="д" THEN I=I+1:GOTO 20
80 IND(I)=-1:IMAX=I
```

Обратите внимание, что по мере заполнения карточек заполняется массив индексов. В нем каждый элемент указывает на номер следующей карточки. Когда ввод закончен, на "веревочке" должен быть завязан узелок. В нашем случае это число -1. Оно означает, что данный элемент последний и за ним больше ничего нет.

Заполним список для четырех сотрудников:

1. Иванов	инженер	100 рублей
2. Петров	механик	120 рублей
3. Сидоров	шофер	110 рублей
4. Федоров	дворник	90 рублей

Естественно, что массив индексов в настоящее время содержит просто последовательность чисел 1, 2, 3. Индекс, соответствующий последней карточке, равен -1, так что за ней ничего нет.

Рассмотрим теперь, как нам вставить в этот список новую фамилию. Например, мы взяли на работу Коновалова. В соответствии с алфавитом его карточка должна помещаться между карточками Иванова и Петрова. Как это сделать? В настоящее время индекс, соответствующий Иванову, равен 1, т. е. указывает на то, что за Ивановым следует элемент массива NAM$(1), т. е. Петров. Легко догадаться, как нам включить в список Коновалова. Он у нас будет введен пятым:

```
NAM$(5)="Коновалов"
JOB$(5)="Программист"
PAY$(5)=100
```

Теперь нам необходимо изменить массив IND:

```
IND(0)=5
IND(5)=1
```

Получается, что индекс Иванова указывает на пятую карточку, т. е. на Коновалова, а его индекс, в свою очередь, - на первую карточку, т. е. на Петрова.

Как просто. Вместо переставления элементов в трех массивах мы ограничились только двумя операциями присваивания. Вот что можно сделать, если немного подумать. Теперь давайте в соответствии с полученными знаниями займемся программой. Организуем ее, как обычно, в виде меню:

```
10 CLS:CLEAR 1000
20 DIM NAME$(20),JOB$(20),PAY(20),IND(20)
30 PRINT "1.Завести новый список"
40 PRINT "2.Добавить элемент к списку"
50 PRINT "3.Удалить элемент из списка"
60 PRINT "4.Распечатать список"
70 PRINT "5.Записать список на диск"
80 PRINT "6.Считать список с диска"
90 INPUT "Выберите функцию" N
100 ON N GOSUB 1000,2000,3000,4000,5000,6000
200 GOTO 30
1000 REM Создаем новый список
1010 I=0:CLS
1020 INPUT "Имя";NAM$(I)
1030 INPUT "Должность";JOB$(I)
1035 INPUT "Оклад";PAY(20)
1040 INPUT "Продолжать?<Д/Н>";A$
1050 IF A$="Д" OR A$="д" THEN IND(I)=I+1:I=I+1:GOTO 1020
1060 IND(I)=-1:IMAX=I:RETURN
2000 REM Добавляем элемент к списку
2010 INPUT "Номер элемента";N
2020 INPUT "Имя";NAM$(IMAX+1)
2030 INPUT "Должность";JOB$(IMAX+1)
2040 INPUT "Оклад";PAY(IMAX+1)
2050 IF N<50 IND(IMAX+1)=IND(N-1):IND(N-1)=IMAX+1
     ELSE IND(IMAX+1)=0
2060 IMAX=IMAX+1:RETURN
```

Теперь рассмотрим подпрограмму удаления элемента. Для этого необходимо произвести всего одну операцию присваивания. Допустим, что мы уволили сотрудника с номером K. Чтобы это сделать, необходимо указатель у сотрудника K-1 настроить на сотрудника K+1, т. е.

```
IND(K-1)=IND(K)
```

И все. Теперь сотрудник K потерян для нас навсегда. Правда, путешествовать по нашему списку можно только держась за веревочку. Попробуем посмотреть содержимое нашего списка. Процедура очень проста. Берем самый первый элемент. (Он должен быть всегда пер-

вым.) Смотрим, на какой номер указывает его индекс. Берем соответствующий ему элемент. И так далее, пока не найдем указатель, равный -1, что будет означать конец списка.

Давайте посмотрим, как будет выглядеть соответствующая программа:

```
4040 REM Вывод списка на экран
4010 CLS:I=0
4020 PRINT NAM$(I),JOB(I),PAY(I)
4030 I=IND(I)
4040 IF I=-1 THEN RETURN:ELSE GOTO 4020
```

Обратите внимание, что мы не знаем, в каком порядке заполняют массив наши карточки. Так, вторая карточка может соответствовать десятому элементу массива. Для нас это не важно. Поскольку такая база данных постоянно меняется, т. е. освобождаются одни номера и заполняются другие, она часто называется динамической.

Возможно, у вас возник вопрос, что делать с тем элементом массива, где находятся сведения об удаленном сотруднике? Таким образом, как мы написали программу, его использовать невозможно. Новые элементы будут добавляться в конец массива. Правильно. Само собой ничего не произойдет. Для утилизации свободных ячеек внутри массивов служит специальная процедура "сборки мусора". Ее задача - перелопачивать всю нашу базу данных на предмет собирания пустых ячеек. Тут уже без огромного числа операций присваивания не обойтись. Однако эту процедуру можно совместить с записью и считыванием с диска. Тогда все это будет происходить автоматически. Давайте напишем эти процедуры:

```
5000 REM Запись списка на диск
5010 INPUT "Введите имя файла";FN$
5020 OPEN "O",#1,FN$
5030 I=0
5040 PRINT #1,NAM$(I);JOB$(I);PAY(I)
5050 I=IND(I)
5060 IF I=-1 THEN CLOSE1: RETURN: ELSE GOTO 5040
```

Обратите внимание, что "веревочку" мы на диск не записали. Теперь читаем:

```
6000 REM Чтение списка с диска
6010 INPUT "Введите имя файла";FN$
6020 OPEN "I",#1,FN$
6030 I=0
6040 INPUT #1,NAM$(I),JOB$(I),PAY(I)
6050 IF EOF(1)=-1 THEN IND(1)=-1:IMAX=I:RETURN
6060 IND(I)=I+1:GOTO 6040
```

В результате массив опять выстроится по порядку номеров.

Отлично! У нас есть прекрасная база данных. Мы ее умеем пополнять, хранить, холить и лелеять. А зачем она нам нужна? Чтобы просто выводить на экран текст? С этим и редактор текста справлялся. Что же появилось нового? Давайте пристально всмотримся в элемент нашей базы данных. Это элементы трех массивов. Они, как видите, разного типа. Два из них строковые переменные, а один - число. В серьезных программах для работы с подобными вещами элемент базы данных называется записью, а входящие в него части - полями. Выражаясь по-научному, элемент нашего списка состоит из трех полей - имени, должности, оклада. Такое деление очень удобно. Скажем, мы хотим вычислить полную сумму фонда заработной платы:

```
10000 REM Вычисляем сумму
10010 I=0:SUM=0
10020 SUM=SUM+PAY(I)
10030 I=IND(I)
10040 IF I=-1 THEN RETURN ELSE 10020
```

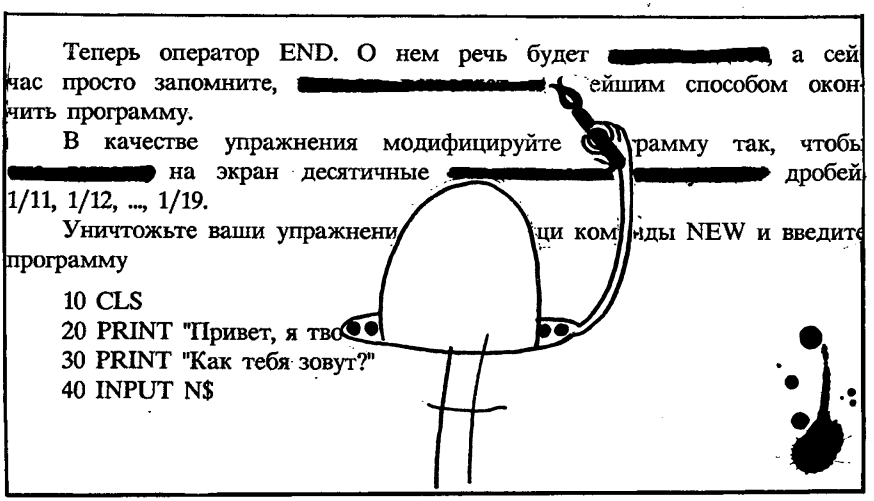
Это гораздо проще, чем работа с обычным текстом, поиск в нем цифр, перевод их в числа и т. д.

Но и это еще не все. Иногда возникает необходимость пересортировки каталогов. Например, мы делали наш список, располагая фамилии по алфавиту. Однако нам может потребоваться переставить список в порядке возрастания зарплаты или по каким-либо другим признакам. В нашем случае мы меняем только элементы веревочки - массива IND(I). При этом остальные массивы мы не трогаем.

Надеемся, что вам понятна идея организации связанных списков. Иногда бывает удобно иметь возможность ходить по списку туда и обратно. В этом случае заводят вторую веревочку, идущую из хвоста списка в его начало. Такие списки называются двусвязаными.

Ну, а что происходит в реальности? Трудно представить, что человечество до сих пор не реализовало такую хорошую вещь. Реализовало, конечно. В состав программного обеспечения Корвета входит специальная программа DBASE. В сущности это специальный язык программирования, придуманный специально для обработки баз данных. Она реализует не только упомянутые нами функции, но и многое другое.

Прежде чем закончить с этой темой, остановимся на одном моменте. В реальной программе, естественно, не заводят такую кучу массивов. Серьезные программы (пишутся они, естественно, не на Бейсике) обычно оперируют со специальными структурами, внутрь которых уже входят различные поля. Так вот, два поля в этих структурах хранят адреса ячеек памяти ЭВМ, где хранятся предшествующий и последующий элементы списка. Это необходимые атрибуты. Такой способ организации указателей удобнее, чем массив, поскольку в результате можно использовать под список сколько угодно свободной памяти ЭВМ, не декларируя это количество в начале работы. Это удобно.



Теперь оператор END. О нем речь будет [redacted], а сей час просто запомните, [redacted] ейшим способом окон чить программу.

В качестве упражнения модифицируйте [redacted] рамму так, чтобы [redacted] на экран десятичные [redacted] дробей 1/11, 1/12, ..., 1/19.

Уничтожьте ваши упражнени[redacted]ци ком[redacted]нды NEW и введите программу

```
10 CLS
20 PRINT "Привет, я тво[redacted]
30 PRINT "Как тебя зовут?"
40 INPUT N$
```

Ну, вот и все. Теперь доставайте Супертекст, DBASE и упражняйтесь самостоятельно.

23. ПРИМЕРЫ КОНКРЕТНЫХ ПРОГРАММ

Подсказки пользователю на тему о том, как и чем можно наполнить свою машину, чтобы было приятно жить и работать

Вы как опытный программист уже понимаете, что в компьютере самым важным являются программы. Вы имеете полезные вещи типа интерпретатора Бейсика, операционной системы и т. п. Однако этими программами не исчерпывается, естественно, то множество математического обеспечения, которое можно использовать на Корвете. Как сделать его доступным для всех? Самое простое - открыть магазины по продаже дисов и кассет с программами. Однако в настоящее время по ряду причин это трудно реализовать. Попробуем пойти по более трудоемкому, но зато доступному пути. Мы приведем здесь тексты некоторых программ, а вы, если заинтересуетесь, введете их в Корвет и будете ими пользоваться.

Начнем, естественно, с игр. Ниже приведены их тексты и краткие инструкции для пользователя.

PITON

Цель игры - отрастить питону хвост как можно большего размера.

Запустите игру и подождите несколько секунд, пока программа загрузит необходимую графику. Перед вами - поле, по которому может двигаться питон, у которого сначала есть только голова (знак $). По этому полю случайным образом разбросано девять камней, которые

препятствуют движению питона. Движение питона осуществляется при нажатии цифровых клавиш:

4 - влево, 6 - вправо	8 - вверх, 2 - вниз
7 - по диагонали влево-вверх	9 - вправо-вверх
1 - влево-вниз	3 - вправо-вниз
5 - остановка движения	

Каждый раз, когда питон "съедает" какую-либо цифру, его длина увеличивается на соответствующее цифре число знаков. Во время движения или остановки нажатием цифровой клавиши 5 длина питона не изменяется. Если же питон останавливается, упершись в препятствие (ими являются границы поля, камни и туловище самого питона), он стремительно укорачивается.

Слева наверху вы видите часы, которые стоят, когда питон двигается, и идут, когда он стоит, упершись в препятствие. Когда часы покажут число 100, игра кончится. Максимальная длина питона, которая была достигнута в процессе игры, показана наверху в центре. Наверху справа горят цифры, показывающие длину питона в текущий момент времени. Вы можете прервать игру в любой момент нажатием клавиши ПРФ. После появления соответствующей надписи решайте - начинать игру с начала или нет.

```
10 CLEAR 300:CLS:PCLS
20 DEFINT A-Z:PRINT CHR$(27);"3"
30 LOCATE 19,4,0:PRINT "<< П И Т О Н >>";
40 LOCATE 5,8
45 PRINT "О Д Н У  М И Н У Т У,  П О Ж А Л У Й С Т А !";
50 DIM A(16)
60 A(0)=2:A(1)=2:A(4)=2:A(6)=2:A(2)=2:A(3)=3:A(5)=5:A(7)=7
70 FOR I=1 TO 8:A(I+7)=15:NEXT I:LUT A(0)
80 FOR I=1 TO 177:LINE(168,35+I)-(343,35+I),6:NEXT
85 PAINT (200,100),3,0
90 DIM B(2000,1),C(29),D(19)
100 DIM XA(10),YA(10)
110 H=0:M=0:LM=0
120 LT=0
130 R1=0:R2=0:NR=0
140 S=1:X=0:Y=0
150 R=0
160 FOR I=0 TO 29:READ C(I):NEXT I
170 REM   0   1   2   3   4   5   6   7   8   9  ARRAY C
180 DATA  6,  4,  3, 32,  8,  2, 46, -1, -1, 140
190 DATA  4,  1, 151, 131, 171, 149, 170, 141, 140, 142
200 DATA 13,  0, 36,  5, -1, 64, 15,  9,  7,  0
210 FOR I=0 TO 19:READ D(I):NEXT I
220 REM   0   1   2   3   4   5   6   7   8   9  ARRAY D
230 DATA 40, 10, 20,  0,  0,  3,  0,  0,  0,  1
```

```
240 DATA  3,  0, 100,  0,  0,  0,  0,  0,  0,  0
250 FOR I=0 TO 10:READ XA(I):NEXT I:REM ARRAY XA
260 DATA -1, -1, -1, 0, 0, 0, 0, 0, 1, 1, 1
270 FOR I=0 TO 10:READ YA(I):NEXT I:REM ARRAY YA
280 DATA -1, 0, 1, 0, -1, 0, 1, 0, -1, 0, 1
290 FOR I=0 TO 10:READ ZA(I):NEXT I:REM ARRAY ZA
300 DATA 3, 8, 9, 10, 4, 5, 6, 0, 1, 2, 7
310 IF D(1)<3 OR D(1)>20 THEN D(1)=20
320 IF D(3)<1 OR D(3)+D(3)>D(0)-D(2)-2 THEN D(3)=(D(0)-D(2)-2)/2
330 D(4)=D(0)-D(2)-D(3)-1
340 D(6)=D(4)+D(2)+1
350 D(7)=D(5)+D(1)+1
360 D(8)=INT((64-D(0))/2):REM D(4)+INT(D(2)/2)-3
370 FOR I=0 TO 16:LINE((2*D(4)-5)*8,I)-((2*D(4)+1)*8,I),4:NEXT
380 FOR I=0 TO 16:LINE((2*D(6)-19)*8,I)-((2*D(6)-13)*8,I),1:NEXT
390 FOR I=0 TO 16:LINE((2*17-5)*8,I)-((2*17+1)*8,I),6:NEXT
400 CLS:PRINT CHR$(27);"2":FOR I=1 TO 7:A(I)=I:NEXT:LUT A(0)
410 FOR I=0 TO 8:A(I+8)=15:NEXT
420 W$=STRING$(D(0),C(3))
430 W1$=STRING$(D(2),C(6))
440 FOR I=14 TO 3 STEP-1
450 LOCATE D(8),I: IF I<>13 THEN PRINT W$ ELSE
    PRINT MID$(W$,1,D(0)-1)
460 LOCATE D(4)+D(8),I
470 IF I=D(5) THEN PRINT CHR$(C(12));STRING$(D(2),C(13));
    CHR$(C(14))
480 IF I=D(7) THEN PRINT CHR$(C(17));STRING$(D(2),C(18));
    CHR$(C(19))
490 IF I<=D(5) OR I>=D(7) THEN 510
500 PRINT CHR$(C(15));W1$;CHR$(C(16));
510 NEXT I
520 IF H>0 AND H<=2000 THEN 560
530 H=0:T=0
540 B(0,0)=D(5)+INT(D(1)/2)
550 B(0,1)=D(8)+D(4)+INT(D(2)/2)
560 I=H:L=1
570 LOCATE B(I,1),B(I,0):PRINT CHR$(C(22)):REM - HEAD
580 IF I=T THEN 630
590 I=I-1:IF I<0 THEN I=2000
600 LOCATE B(I,1),B(I,0):PRINT CHR$(C(25)):REM - BODY
620 L=L+1:GOTO 580
630 IF D(9)=0 THEN 720
640 J=INT(D(1)*D(2)/(D(9)*25))
650 FOR I=0 TO J
660 R1=D(5)+INT(RND(D(1))*D(1))+1
670 R2=D(4)+D(8)+INT(RND(D(2))*D(2))+1
680 IF PEEK(64511!+(R1-1)*64+R2)<>C(6) THEN 710
```

```
690 LOCATE R2,R1:PRINT CHR$(C(9)):REM - OBSTACLE
695 REM CIRCLE(R2*8-4,R1*16-8),5,0
700 REM PAINT(R2*8-4,R1*16-8),4,0
710 NEXT I
720 IF NR>=D(10) THEN 780 ELSE R=1+INT(RND(9)*9):RR=R+48
730 R1=D(5)+INT(RND(D(1))*D(1))+1
740 R2=D(4)+D(8)+INT(RND(D(2))*D(2))+1
750 IF PEEK(64511!+(R1-1)*64+R2)<>C(6) THEN 730
760 LOCATE R2,R1:PRINT CHR$(RR)
770 NR=NR+1:GOTO 720
780 IF L>LM THEN LM=L
790 LOCATE 17,1
800 PRINT USING " ### ";LT;
810 LOCATE 31,1
820 PRINT USING " ### ";LM;
830 LOCATE 46,1
840 PRINT USING " ### ";L;
850 IF LT<D(12) THEN 900
860 LOCATE1,15:PRINT "Нажмите <<пробел>>, чтобы начать снова,";
    "или <<ESC>>, чтобы кончить игру";
870 G$=INKEY$:IF G$<>" " AND G$<>CHR$(27) THEN 870
880 IF G$=" " THEN 890 ELSE 1170
890 RESTORE: GOTO 110
900 G$=INKEY$
910 F$=G$:G$=INKEY$:IF G$<>"" THEN 910
920 IF LEN(F$)<>1 THEN 970
930 IF F$=CHR$(27) THEN 860
940 F=ASC(F$)-48
950 IF F<1 OR F>9 THEN 970 ELSE F=ZA(F):GOTO 960
960 X=XA(F):Y=YA(F)
970 IF X=0 AND Y=0 THEN 900
980 P=B(H,0)+X: Q=B(H,1)+Y
990 G=PEEK(64511!+(P-1)*64+Q)
1000 S=0:R=G-48
1010 IF G=C(6) THEN S=1
1020 IF R>0 AND R<10 THEN S=2:M=M+R:R=0:NR=NR-1
1030 IF S=0 THEN S=3:LT=LT+1+M:M=0:GOTO 1100
1040 LOCATE Q,P:PRINT CHR$(C(22))
1050 LOCATE B(H,1),B(H,0):PRINT CHR$(C(25))
1060 H=H+1:IF H>2000 THEN H=0
1070 B(H,0)=P:B(H,1)=Q
1080 L=L+1
1090 IF M>0 THEN M=M-1:GOTO 1160
1100 IF L<2 THEN 1160
1110 LOCATE B(T,1),B(T,0):PRINT CHR$(C(6))
1120 T=T+1:IF T>2000 THEN T=0
1130 L=L-1
```

```
1140 IF L<2 THEN 1160
1150 LOCATE B(T,1),B(T,0):PRINT CHR$(127) :REM - TAIL
1160 IF S=2 THEN 720 ELSE 780
1170 A(0)=0:A(1)=1:LUT A(0):CLS:PCLS:END
1180 FOR I=10 TO 255
1190 PRINT I;CHR$(I),
1200 NEXT I
```

MATTEST

Цель игры - тренировка в устном счете. Машина контролирует как быстро вы считаете и как часто ошибаетесь.

После запуска программы на экране появится надпись:

ПРОВЕРКА МАТЕМАТИЧЕСКИХ НАВЫКОВ

Нажмите на любую клавишу. В ответ на это действие машина задаст вам вопрос из области арифметики и одновременно предложит варианты ответов. Нажмите номер ответа, который, по вашему мнению, правильный. Машина выдаст вам истинный ответ, сообщит, верно ли вы ответили или ошиблись. После этого она задаст вам следующий вопрос, и так 20 раз. После того как вы ответите на все 20 вопросов, машина сделает итоговое сообщение о том, сколько раз вы ошиблись при умножении, сложении, вычитании и делении, сколько времени вы думали над выполнением каждого из арифметических действий.

```
10 CLS
20 DIM TM(5)
30 GOSUB 1570
40 GOSUB 340
50 TIME=0
60 FOR I=0 TO 5
70 OA(I)=0
80 WA(I)=0
90 RA(I)=0
100 NEXT I
110 LOCATE 1,3,0
120 PRINT STRING$(64,"_")
130 LOCATE 1,11
140 PRINT STRING$(64,"_")
150 LOCATE 44,2:PRINT "ВРЕМЯ:       СЕК"
160 FOR OA=1 TO 20
170 A=INT(RND(1)*99):A$=STR$(A):A$="  "+A$:A$=RIGHT$(A$,2)
180 B=INT(RND(1)*99):B$=STR$(B):B$="  "+B$:B$=RIGHT$(B$,2)
190 GOSUB 930
200 C$="  "+C$
210 C$=RIGHT$(C$,4)
```

```
220 PRINT TAB(9) "????"
230 GOSUB 980
240 GOSUB 1060
250 LOCATE 10,9
260 PRINT TAB(9) C$
270 GOSUB 1270
280 NEXT OA
290 LOCATE 25,13:PRINT STRING$(12," ")
300 LOCATE 32,VK+4:PRINT "  "
310 GOSUB 1390
320 LOCATE 1,12,1
330 END
340 REM *INITIALIZE*
350 NA$(1)="УМНОЖЕНИЕ"
360 NA$(2)="СЛОЖЕНИЕ "
370 NA$(3)="ВЫЧИТАНИЕ"
380 NA$(4)="ДЕЛЕНИЕ"
390 NA$(5)="ВСЕГО"
400 RETURN
410 REM *MULTIPLY*
420 TA=1
430 OA(1)=OA(1)+1
440 LOCATE 8,2: PRINT NA$(1)
450 C=A*B: C$=STR$(C)
460 LOCATE 1,5
470 PRINT TAB(11)A$
480 PRINT TAB(9)"x"
490 PRINT TAB(11)B$
500 PRINT TAB(9)"----"
510 RETURN
520 REM *ADD*
530 TA=2
540 OA(2)=OA(2)+1
550 LOCATE 8,2: PRINT NA$(2)
560 C=A+B: C$=STR$(C)
570 LOCATE 1,5
580 PRINT TAB(11)A$
590 PRINT TAB(9)"+"
600 PRINT TAB(11)B$
610 PRINT TAB(9)"----"
620 RETURN
630 REM *SUBTRACT*
640 TA=3
650 OA(3)=OA(3)+1
660 LOCATE 8,2: PRINT NA$(3)
670 IF A<B THEN SWAP A,B:SWAP A$,B$
680 C=A-B
```

```
690 C$=STR$(C)
700 LOCATE 1,5
710 PRINT TAB(11)A$
720 PRINT TAB(9)"-"
730 PRINT TAB(11)B$
740 PRINT TAB(9)"---"
750 RETURN
760 REM *DIVIDE*
770 TA=4
780 OA(4)=OA(4)+1
790 LOCATE 1,5
800 PRINT TAB(11)"  "
810 PRINT TAB(9)"  "
820 PRINT TAB(11)"  "
830 LOCATE 8,2: PRINT NA$(4)
840 C=A*B
850 C$=STR$(C)
860 SWAP A,C : SWAP A$,C$
870 LOCATE 3,7
880 PRINT A$
890 LOCATE 9,7
900 PRINT "|"B$
910 PRINT TAB(9) "---"
920 RETURN
930 REM *CHOICE OF ACTION*
940 T=INT(RND(1)*3.9+1)
950 T0=TIME
960 ON T GOSUB 410, 520, 630, 760
970 RETURN
980 REM *WRONG & RIGHT ANSWERS*
990 NO=INT(RND(1)*4)
1000 X(NO)=C
1010 FOR I=1 TO 4
1020 IF I<>NO THEN X(I)=INT(C+(.5-RND(1))*C):
     IF C<=15 THEN X(I)=INT(RND(1)*20)
1030 NEXT I
1040 X(5)=X(0)
1050 RETURN
1060 REM *TABLE & CHOICE*
1070 FOR I=1 TO 4
1080 LOCATE 32,I+4
1090 PRINT "  "I". ";X(I)"  "
1100 NEXT I
1110 LOCATE 32,9
1120 PRINT "  5. HET OTBETA"
1130 GOSUB 1330:K$=INKEY$:IF K$="" THEN 1130
1140 VK=VAL(K$)
```

```
1150 IF VK<1 OR VK>5 THEN GOSUB 1200: GOTO 1130
1160 LOCATE 32,VK+4 :PRINT "=>"
1170 TM(TA)=TM(TA)+TIME-T0
1180 FOR II=1 TO 50 :NEXT II
1190 RETURN
1200 REM *ERROR*
1210 LOCATE 42,10
1220 PRINT "< ? >"
1230 FOR II=1 TO 50 :NEXT
1240 LOCATE 42,10
1250 PRINT "  "
1260 RETURN
1270 REM *ANALISES*
1280 IF X(VK)=C THEN LOCATE 25,13: PRINT "ВЕРНО !":
1285 RA(TA)=RA(TA)+1
1290 IF X(VK)<>C THEN LOCATE 25,13: PRINT "НЕ УГАДАЛ !":
1295 WA(TA)=WA(TA)+1
1300 FOR II=1 TO 700 :NEXT II
1310 LOCATE 25,13:PRINT STRING$(12," ")
1320 RETURN
1330 REM *TIMER*
1340 LOCATE 50,2
1350 PRINT TIME
1360 TIME=TIME+1
1370 FOR II=1 TO 323 :NEXT II
1380 RETURN
1390 REM *FINAL TABLE*
1400 LOCATE 1,8
1410 FOR I=1 TO 16: PRINT: NEXT I
1420 CLS
1430 LOCATE 15,2
1440 PRINT "ИТОГ ПРОВЕРКИ"
1450 PRINT TAB(13) STRING$(15,"~")
1460 PRINT "ДЕЙСТВИЕ   ЧИСЛО ПРИМЕРОВ   ЧИСЛО ОШИБОК
     ВРЕМЯ "
1470 PRINT STRING$(50,"_")
1480 FOR I=1 TO 5
1490 IF OA(I)<>0 THEN WP%=INT(WA(I)/OA(I)*100) ELSE WP%=0
1500 PRINT NA$(I),OA(I),WA(I)"("WP%"%)",TM(I)
1510 TM(5)=TM(5)+TM(I)
1520 OA(5)=OA(5)+OA(I)
1530 WA(5)=WA(5)+WA(I)
1540 IF I=4 THEN PRINT STRING$(50,"_")
1550 NEXT I
1560 RETURN
1570 REM *TRADE MARK*
1580 CLS:PCLS:RELOC(0,0):COLOR 7
```

```
1590 DEFINT L:DIM L(17):FOR I=0 TO 7:L(I)=I:L(I+8)=15:NEXT I:
     L(7)=15:LUT L(0)
1600 PRINT CHR$(27)"3"
1610 LOCATE 15,3: PRINT STRING$(32,CHR$(171))
1620 LOCATE 23,5,0: PRINT "П Р О В Е Р К А"
1630 PRINT TAB(16) "М А Т Е М А Т И Ч Е С К И Х"
1640 PRINT TAB(22) "Н А В Ы К О В"
1650 PRINT TAB(14) STRING$(32,CHR$(171))
1660 LOCATE 5,14:PRINT "С Н П О      К О Р С А Р"
1670 LOCATE 9,15:PRINT "( c )   1 9 8 7"
1680 LOCATE 13,16:PRINT "М Г У";:LOCATE 1,12
1690 FOR I=112 TO 114: PSET(I,230)
1700 DRAW "U15F5E5D7"
1710 NEXT I
1720 I=I+1: IF I>600 THEN GOSUB 1770
1730 IF INKEY$="" THEN A=RND(1): GOTO 1720
1740 PRINT CHR$(27)"2"
1750 CLS:PCLS
1760 RETURN
1770 LOCATE 35,15
1780 PRINT "Н А Ж М И Т Е   < В К >"
1790 FOR I=0 TO 400 : NEXT
1800 LOCATE 35,15
1810 PRINT STRING$(24," ")
1820 RETURN
```

REACT

Цель игры - проверка и тренировка вашей реакции. После запуска программы на экране появится название игры. Нажмите на любую клавишу и отвечайте на вопросы, которые вам задаст машина. Если вы играете впервые, попросите машину проинструктировать вас. В конце игры машина выдаст сообщение о вашей реакции и о характере ваших ошибок.

```
10 CLS:PCLS:GOSUB 630:C1=0:J=0:C2=0:R1=0:R2=0:K1=0:P=0:MS=0
20 PRINT TAB(19) "КТО ВЫ";:INPUT W$
30 PRINT TAB(10) "НУЖНА ЛИ ИНСТРУКЦИЯ (ДА - 1, НЕТ - 2)";
40 INPUT P:IF P=1 THEN GOSUB 530
50 IF P<>1 AND P<>2 THEN PRINT "ОШИБКА! НАБЕРИТЕ 1 ИЛИ 2.":
55 GOTO 30
60 PRINT "КАКОЙ УРОВЕНЬ";
65 PRINT "ВЫБИРАЕТЕ (ОТ 1 ДО 5 - СЛОЖНОСТЬ ВОЗРАСТАЕТ)";:
     INPUT G
70 IF G<1 OR G>5 THEN PRINT "ОШИБКА. НАБЕРИТЕ 1, 2 ИЛИ 3.":
     GOTO 60
80 PRINT "ЕСЛИ ВЫ ГОТОВЫ, НАЖМИТЕ ЛЮБУЮ КЛАВИШУ"
```

```
90 IF INKEY$="" THEN Z=RND(Z)*100:GOTO 90
100 IF G=1 THEN B=1:D=3
110 IF G=2 THEN B=2:D=2
120 IF G=3 THEN B=3:D=2
130 IF G=4 THEN B=5:D=2
140 IF G=5 THEN B=5:D=1.5
150 CLS
160 Z=400*RND(Z)+50:U=150*RND(U)+30
170 GOSUB 480
180 Z1=Z-5:Z2=Z+5:U1=U-4:U2=U+4:LINE(Z2,U2)-(Z2,U1),7:
185 LINE(Z2,U1)-(Z1,U1),7
190 LINE(Z1,U1)-(Z1,U2),7:LINE(Z1,U2)-(Z2,U2),7
200 X1=Z-20*D:Y1=U-20*DY:PSET(X1,Y1)
210 X=X1+B:Y=Y1+BY
220 PSET(X1,Y1),0:PSET(X,Y),7
230 X1=X:Y1=Y
240 IF INKEY$=" " THEN 270
250 IF X>Z+20*D THEN MS=MS+1:CLS:PCLS:GOTO 160
260 GOTO 210
270 R=SQR((Z-X)*(Z-X)+(U-Y)*(U-Y))
280 IF R<3 THEN J=J+1:GOTO 310
290 IF Z-X<0 THEN C1=C1+1:R1=R1+R:GOTO 310
300 C2=C2+1:R2=R2+R
310 K1=K1+1:IF K1<15 THEN PCLS:GOTO 160
320 PCLS:PRINT TAB(15) W$
330 PRINT
340 PRINT TAB(10) "ЧИСЛО ПОПЫТОК:";K1
350 PRINT TAB(10) "ПРОПУЩЕНО ПОПЫТОК:";MS
360 PRINT TAB(10) "КОЛИЧЕСТВО ТОЧНЫХ РЕАКЦИЙ:";J
370 PRINT TAB(10) "СУММАРНАЯ ВЕЛИЧИНА
    ОТКЛОНЕНИЙ:";(R1+R2)*.5;"ММ"
380 PRINT:
    PRINT TAB(10) "КОЛИЧЕСТВО ПРЕЖДЕВРЕМЕННЫХ
    РЕАКЦИЙ:";C2
390 IF C2=0 THEN M=0:GOTO 410
400 M=R2/C2
410 PRINT TAB(10) "СРЕДНЯЯ ВЕЛИЧИНА ПРЕЖДЕВРЕМЕННЫХ
    РЕАКЦИЙ:";M*.5;"ММ"
420 PRINT TAB(10) "КОЛИЧЕСТВО ЗАПАЗДЫВАЮЩИХ
    РЕАКЦИЙ:";C1
430 IF C1=0 THEN T=0:GOTO 450
440 T=R1/C1
450 PRINT TAB(10) "СРЕДНЯЯ ВЕЛИЧИНА ЗАПАЗДЫВАЮЩИХ
РЕАКЦИЙ:\";T*.5;"ММ"
460 PRINT:
    PRINT "ЕСЛИ ХОТИТЕ ПОВТОРИТЬ, НАБЕРИТЕ
    КОМАНДУ RUN"
```

```
470 L(7)=7:LUT L(0):LOCATE „1:END
480 RR=RND(Z)*100
490 IF RR<33 THEN BY=B:DY=D
500 IF RR>=66 THEN BY=-B:DY=-D
510 IF RR>=33 AND RR<66 THEN BY=0:DY=0
520 RETURN
530 CLS
540 PRINT TAB(15) "И Н С Т Р У К Ц И Я"
550 PRINT TAB(15) "----------"
560 PRINT "Перед Вами на экране появятся квадратик и"
570 PRINT "движущаяся точка. Ваша задача - остановить ее"
580 PRINT "в центре этого квадратика, нажимая клавишу ПРОБЕЛ."
590 PRINT
600 PRINT TAB(10) "ЖЕЛАЕМ УСПЕХА!!!"
610 PRINT:PRINT
620 RETURN
630 CLS:PCLS:RELOC(0,0):COLOR 7:LOCATE „0
640 DEFINT L: DIM L(17): FOR I=0 TO 7: L(I)=I: L(I+8)=15: NEXTI:
    L(7)=15:LUT L(0)
650 PRINT CHR$(27)"3"
660 LOCATE 15,5:PRINT STRING$(32,CHR$(171)):PRINT:
665 PRINT TAB(26) "Т Е С Т"
670 PRINT:PRINT TAB(20) "Н А  С К О Р О С Т Ь":
675 PRINT TAB(24) "Р Е А К Ц И И"
680 PRINT TAB(14) STRING$(32,CHR$(171))
690 LOCATE 5,14:PRINT "С Н П О     К О Р С А Р"
700 LOCATE 9,15:PRINT "( c )   1 9 8 7"
710 LOCATE 13,16:PRINT "М Г У";:LOCATE 1,12
720 FOR I=112 TO 114: PSET(I,230)
730 DRAW "U15F5E5D7"
740 NEXT I
750 I=I+1: IF I>600 THEN GOSUB 790
760 IF INKEY$=""THEN 750
770 PRINT CHR$(27)"2"
780 CLS:PCLS:RETURN
790 LOCATE 35,15
800 PRINT "Н А Ж М И Т Е  < В К >"
810 FOR I=0 TO 400 : NEXT
820 LOCATE 35,15
830 PRINT STRING$(24," ")
840 RETURN
```

WORRY

Цель игры - самооценка комфортности своего внутреннего мира.

После запуска программы на экране появится название игры. Нажмите на любую клавишу и отвечайте на вопросы, которые вам задаст машина. Если вы играете впервые, то попросите машину проинструктировать вас. Не стоит слишком серьезно относиться к результату самооценки. Однако если вы были искренни, то задумайтесь над ответом.

```
10 GOSUB 530:
20 PRINT:PRINT:PRINT:PRINT:PRINT:
25 PRINT:PRINT TAB(20) "Как Вас зовут ?":
   PRINT:PRINT:PRINT:PRINT:
26 PRINT:PRINT TAB(15):INPUT F$:PRINT:PRINT
30 INPUT "Вам нужна инструкция (ДА / НЕТ)";X$ :LOCATE „0
40 IF X$="НЕТ" OR X$="нет" OR X$="NET" OR X$="net" THEN 120
50 IF X$="DA" OR X$="da" OR X$="ДА" OR X$="да" THEN 70 ELSE 60
60 PRINT "Вы неправильно набрали ответ. Будьте внимательнее.":
65 LOCATE „1:GOTO 30
70 FOR I=0 TO 16:PRINT:NEXT:
75 LOCATE 1,3:PRINT "Сейчас Вам будет предложен";
76 PRINT "тест 'САМООЦЕНКА СИТУАТИВНОЙ":
   PRINT "ТРЕВОЖНОСТИ'.";
77 PRINT "Вам необходимо ответить на все вопросы теста,":
78 PRINT "нажимая в цифровом регистре клавишу с номером";
79 PRINT "выбранного ответа"
80 PRINT "из 4-х предложенных. Над ответами долго";
82 PRINT "не задумывайтесь, т.к.":
   PRINT "правильных и неправильных ответов нет.":PRINT:PRINT
90 PRINT TAB(10) "Если все ясно, нажмите любую клавишу":PRINT
100 IF INKEY$="" THEN 100
110 FOR I=0 TO 200: NEXT
120 LOCATE 1,10:FOR J=0 TO 5:PRINT STRING$(64," "):NEXT J:
    FOR I=0 TO 16: PRINT:FOR J=0 TO 10:NEXT:NEXT
130 GOTO 170
140 GOSUB 430
150 LOCATE 9,7:
155 PRINT "С А М О О Ц Е Н К А  С И Т У А Т И В Н О Й":
156 PRINT TAB(18) "Т Р Е В О Ж Н О С Т И"
160 RETURN
170 DIM C(21):DATA "Я спокоен","Мне ничто не угрожает"
172 DATA "Я нахожусь в напряжении","Я испытываю сожаление"
173 DATA "Я чувствую себя свободно","Я расстроен"
174 DATA "Меня волнуют возможные неудачи"
175 DATA "Я чувствую себя отдохнувшим","Я встревожен"
180 DATA "Я испытываю чувство внутреннего удовлетворения"
```

```
181 DATA "Я уверен в себе","Я нервничаю",
    "Я не нахожу себе места"
182 DATA "Я взвинчен","Я не чувствую скованности, напряжения"
183 DATA "Я доволен","Я озабочен","Я слишком возбужден"
185 DATA "Мне радостно","Мне приятно"
190 FOR J%=1 TO 20
200 READ A$:FOR I=0 TO 16:PRINT:GOSUB 460:NEXT
210 LOCATE 1,2:PRINT STRING$(63,"Е")"Д":PRINT:GOSUB 460
220 PRINT TAB(10) A$:PRINT:GOSUB 460
230 PRINT TAB(15) "1. Нет, это совсем не так":GOSUB 460
240 PRINT TAB(15) "2. Пожалуй, нет":GOSUB 460
250 PRINT TAB(15) "3. Похоже, что так":GOSUB 460
260 PRINT TAB(15) "4. Совершено верно":GOSUB 460
270 PRINT:GOSUB 460:PRINT:GOSUB 460:PRINT:GOSUB 460:
    PRINT STRING$(63,CHR$(166)):GOSUB 460
280 V$=INKEY$: IF V$="" THEN 280 ELSE
    IF V$<>"1" AND V$<>"2" AND V$<>"3" AND V$<>"4"
    THEN GOSUB 470:GOTO 280:
285 FOR I=0 TO 16 :PRINT:NEXT: GOTO 210
290 C(J%)=VAL(V$):GOSUB 500:NEXT: FOR I=1 TO 16:
    PRINT:GOSUB 460:NEXT:PRINT:PRINT:PRINT:PRINT:
295 PRINT TAB(15) "Оценка ситуативной тревожности:":PRINT
300 PRINT TAB(15);F$:PRINT
310 X=C(3)+C(4)+C(6)+C(7)+C(9)+C(12)+C(13)+C(14)+C(17)+C(18)
320 Y=C(1)+C(2)+C(5)+C(8)+C(10)+C(11)+C(15)+C(16)+C(19)+C(20)
330 Z=X-Y+35
340 PRINT "Показатель уровня тревожности:";Z:PRINT
350 IF Z<25 THEN 380
360 IF Z>=25 AND Z<=45 THEN 390
370 PRINT TAB(15) "(Высокий уровень":GOTO 400
380 PRINT TAB(15) "(Низкий уровень":GOTO 400
390 PRINT TAB(15) "(Средний уровень"
400 PRINT TAB(13) "состояния тревожности)"
410 PRINT:PRINT "Если желаете повторить, наберите RUN":PRINT
420 LOCATE „1:END
430 LOCATE 10,4:PRINT STRING$(40,CHR$(171))
440 LOCATE 10,10:PRINT STRING$(40,CHR$(171))
450 RETURN
460 FOR II=1 TO 10:NEXT:RETURN
470 FOR IJ=1 TO 3
480 LOCATE 24,11:PRINT"< ? >":GOSUB 460:GOSUB 460:
485 LOCATE 24,11:PRINT"      "
490 NEXT IJ :RETURN
500 IF C(J%)>0 AND C(J%)<5 THEN LOCATE 13,5+C(J%):PRINT "=>"
510 FOR I=1 TO 50:NEXT : LOCATE 13,5+C(J%):PRINT "   "
520 LOCATE 1,13:RETURN
530 CLS:PCLS:RELOC(0,0):COLOR 7:LOCATE „0
```

```
540 DEFINT L:DIM L(17):FOR I=0 TO 7:L(I)=I:L(I+8)=15:NEXTI:
    L(7)=15:LUT L(0)
550 PRINT CHR$(27)"3"
560 GOSUB 140
570 LOCATE 5,14:PRINT "С Н П О     К О Р С А Р"
580 LOCATE 9,15:PRINT "( с )   1 9 8 7"
590 LOCATE 13,16:PRINT "М Г У";:LOCATE 1,12
600 FOR I=112 TO 114: PSET(I,230)
610 DRAW "U15F5E5D7"
620 NEXT I
630 I=I+1:IF I>600 THEN GOSUB 670
640 IF INKEY$="" THEN 630
650 PRINT CHR$(27)"2"
660 CLS:PCLS:RETURN
670 LOCATE 35,15
680 PRINT "Н А Ж М И Т Е  < В К >"
690 FOR I=0 TO 400: NEXT
700 LOCATE 35,15
710 PRINT STRING$(24," ")
720 RETURN
```

KALAH

Играют двое - вы и компьютер. Игровое поле состоит из 14 ячеек: 12 маленьких и 2 больших. Большие называются "Калах". Ваши ячейки с 1 по 6 и правый Калах. У партнера-противника ячейки с 7 по 12 и левый Калах. Перед началом игры в каждой пронумерованной ячейке лежит по шесть фишек. Цель игры - собрать как можно больше фишек в свой Калах. Правила игры приведены ниже.

1. Игроки ходят поочередно. Ход заключается в том, что из одной своей ячейки (любой, кроме Калаха) вынимаются все фишки и по одной раскладываются в следующие ячейки, включая и Калах (свой и противника). Например, вы берете фишки из ячейки 1 и начинаете раскладывать по одной в ячейки 2, затем 3, 4, 5, 6 и последнюю фишку кладете в следующий за ячейкой 6 свой Калах. Таким образом, после вашего хода в ячейке 1 пусто, в ячейках 2 - 6 по 7 фишек, в вашем Калахе 1 фишка. Если бы в ячейке 1 было 8 фишек, вы обязаны положить далее по фишке в ячейки 7 и 8, т. е. в ячейки вашего противника.

2. Если последняя фишка в ходе пришлась на ваш Калах, вы имеете право на внеочередной ход. Поскольку рассмотренная в первом правиле ситуация именно такая, то сделаем, например, ход из ячейки 6. Вынем все фишки из нее, первую положим в свой Калах, там их станет две, и остальные 6 разложим в ячейки противника с 7 по 12. Таким образом, там станет по 7 фишек. В рассмотренном случае ход переходит к противнику.

Если у вас, предположим, в ячейке 4 лежат 16 фишек, то при своем ходе из этой ячейки вы проходите полный круг, не забывая положить фишку также в Калах противника, и заканчиваете ход в своем Калахе, получая право на дополнительный ход.

3. И последнее правило. Если последняя фишка попадает в вашу пустую ячейку, то из нее и из ячейки противника, расположенной напротив вашей, вынимаются все фишки и кладутся в ваш Калах. Пусть, например, ваш ход и в ячейке 5 пусто, в ячейке 6 лежат 12 фишек, в ячейке 8 - 10 фишек. Тут надо, не задумываясь, хватать фишки из ячейки 6 и быстро-быстро раскладывать. В результате при отходе одну фишку вы сразу кладете в свой Калах, а затем раскладываете фишки в ячейки противника, его Калах, и снова в свои ячейки 1 - 4 и, наконец, в пустую пятую. Поскольку вы положили последнюю фишку в свою пустую ячейку, то вы забираете ее, а также 11 фишек из ячейки с номером 8 и все кладете в свой Калах. Заметим, кстати, что следующий ход - вашего противника.

Вот и все.

Выигрывает тот, у кого окажется больше фишек в Калахе. Если у вашего противника в ячейках нет фишек и ход его, то игра заканчивается, подсчитываются фишки в Калахах, а вы к фишкам в своем Калахе добавляете фишки из ячеек.

```
10 REM COPYRIGHT С.Доленко KALAH-GAME
20 REM Версия 7.5 для ПК-8020 создана А.А.Девятовым 23.02.88
30 CLEAR 500:DEFSNG A,B:DEFINT C-Z:
35 DIM P(12),P0(13),P1(12),P2(12),K(2),K0(2),K1(2),K2(2),S1(12),
   S2(12),S3(12),J0(6),HS(2,5,3),H$(2,5),D(15)
40 FOR I=0 TO 7:D(I)=I:NEXT:FOR I=8 TO 15:D(I)=10:NEXT:LUT D(0)
50 ON ERROR GOTO 60
60 IF ERR=53 THEN CLOSE:OPEN"O",#1,"KALAH.DAT":CLOSE
70 CLS:GOSUB 1640: GOSUB 1700: GOSUB 1890
80 I$="":N1=0:N2=0:T1=0:T2=0:DL=0:D$="Ваша":CLS:
85 LOCATE „1:INPUT "Уровень (1 - простой ... 3 - сложный) ";LV
90 IF LV<>2 AND LV<>3 THEN LV=1
100 CLS:TT=8:FOR I=1 TO 12:P(I)=6:NEXT I:K(1)=0:K(2)=0
110 RANDOMIZE(32767):GOSUB 910:
112 PRINT"Сейчас ";D$;" очередь начинать игру.":
115 IF D$<>"Ваша" THEN D$="Ваша":GOTO 380
120 D$="моя"
130 IF P(1)+P(2)+P(3)+P(4)+P(5)+P(6)=0 THEN 690
140 GOSUB 1600:LOCATE „1:INPUT "Ваш ход: ";J:Z=0:
145 IF J>0 AND J<7 THEN 190
150 IF J<0 THEN 80
160 IF J<>0 THEN 180
170 IF K(1)>36 OR K(2)>36 THEN 710
180 GOSUB 1600:BEEP:BEEP:
185 PRINT"Запрещенный ход. Попытайтесь снова":GOTO 140
```

```
190 IF P(J)=0 THEN 180 ELSE GOSUB 200:IF Z<>1 THEN 380 ELSE 130
200 GOSUB 1530:I0=P(J):P(J)=0:I=J:GOSUB 1380:IF J>6 THEN 290
210 IF I0+J>6 THEN 230
220 FOR I=J+1 TO I0+J:P(I)=P(I)+1:GOSUB 1380:NEXT I:M=I0+J:
225 GOTO 280
230 IF J=6 THEN 250
240 FOR I=J+1 TO 6:P(I)=P(I)+1:GOSUB 1380:NEXT I
250 K(1)=K(1)+1:GOSUB 1510:
255 I0=I0+J-7:IF I0=0 THEN 350 ELSE IF I0>6 THEN 270
260 FOR I=7 TO 6+I0:P(I)=P(I)+1:GOSUB 1380:NEXT I:GOTO 360
270 FOR I=7 TO 12:P(I)=P(I)+1:GOSUB 1380:NEXT I:
275 I0=I0-6:J=0:GOTO 210
280 IF P(M)>1 THEN 360 ELSE IF P(13-M)=0 THEN 360 ELSE LL=1:
285 GOSUB 1540:GOTO 360
290 IF I0+J>12 THEN 310
300 FOR I=J+1 TO I0+J:P(I)=P(I)+1:GOSUB 1380:NEXT I:M=I0+J:
305 GOTO 340
310 IF J=12 THEN 330
320 FOR I=J+1 TO 12:P(I)=P(I)+1:GOSUB 1380:NEXT I
330 K(2)=K(2)+1:GOSUB 1520:
335 I0=I0+J-13:IF I0=0 THEN 360 ELSE J=0:GOTO 290
340 IF M<7 THEN 360 ELSE IF P(M)>1 THEN 360 ELSE IF P(13-M)=0
    THEN 360 ELSE LL=2:GOSUB 1540:GOTO 360
350 Z=1
360 RETURN
370 GOSUB 1600:BEEP:PRINT "Конечно, мой ход: ";J:GOSUB 200
380 IF P(7)+P(8)+P(9)+P(10)+P(11)+P(12)=0 THEN 700 ELSE J=12
390 IF P(J)+J=13 THEN 370 ELSE IF P(J)+J=26 THEN 370 ELSE J=J-1:
    IF J>6 THEN 390
400 FOR I=1 TO 12:P0(I)=P(I):NEXT I:K0(1)=K(1):K0(2)=K(2)
410 FOR L1=7 TO 12
420 FOR I=1 TO 12:P(I)=P0(I):NEXT I:K(1)=K0(1):K(2)=K0(2):J=L1
430 GOSUB 990:IF LV=1 OR ABS(K(2)-K(1))>100 THEN 610 ELSE
    IF P(1)+P(2)+P(3)+P(4)+P(5)+P(6)>0 THEN 440 ELSE S1(L1)=200:
435 GOTO 620
440 J=6
450 IF P(J)+J=7 THEN 430 ELSE IF P(J)+J=20 THEN 430 ELSE J=J-1:
    IF J>0 THEN 450
460 FOR I=1 TO 12:P1(I)=P(I):NEXT I:K1(1)=K(1):K1(2)=K(2)
470 FOR L2=1 TO 6
480 FOR I=1 TO 12:P(I)=P1(I):NEXT I:K(1)=K1(1):K(2)=K1(2):J=L2
490 GOSUB 990:IF LV=2 OR ABS(K(2)-K(1))>100 THEN 570
    ELSE IF P(7)+P(8)+P(9)+P(10)+P(11)+P(12)>0 THEN 500
    ELSE S2(L2)=-200:
495 GOTO 580
500 J=12
510 IF P(J)+J=13 THEN 490 ELSE IF P(J)+J=26 THEN 490 ELSE J=J-1:
```

```
    IF J>6 THEN 510
520 FOR I=1 TO 12:P2(I)=P(I):NEXT I:K2(1)=K(1):K2(2)=K(2)
530 FOR L3=7 TO 12
540 FOR I=1 TO 12:P(I)=P2(I):NEXT I:K(1)=K2(1):K(2)=K2(2):
    J=L3:GOSUB 990:S3(L3)=K(2)-K(1):NEXT L3:S2(L2)=S3(7)
550 FOR I=8 TO 12:IF S3(I)>S2(L2) THEN S2(L2)=S3(I)
560 NEXT I:GOTO 580
570 S2(L2)=K(2)-K(1)
580 NEXT L2:S1(L1)=S2(1)
590 FOR I=2 TO 6:IF S2(I)<S1(L1) THEN S1(L1)=S2(I)
600 NEXT I:GOTO 620
610 S1(L1)=K(2)-K(1)
620 NEXT L1
630 FOR I=1 TO 12:P(I)=P0(I):NEXT I:K(1)=K0(1):K(2)=K0(2):
    J0(1)=7:C=7:N5=1
640 FOR I=8 TO 12:IF S1(I)<S1(C) THEN 660 ELSE IF S1(I)=S1(C)
    THEN 650 ELSE J0(1)=I:C=I:N5=1:GOTO 660
650 N5=N5+1:J0(N5)=I
660 NEXT I:RANDOMIZE(32000):A=RND(0):B=N5*A:I1=1
670 IF B<=I1 THEN 680 ELSE I1=I1+1:GOTO 670
680 J=J0(I1):GOSUB 1600:BEEP:PRINT "Мой ход:";J:GOSUB 200:
685 GOTO 130
690 K(2)=K(2)+P(7)+P(8)+P(9)+P(10)+P(11)+P(12):GOTO 710
700 K(1)=K(1)+P(1)+P(2)+P(3)+P(4)+P(5)+P(6)
710 T1=T1+K(1):T2=T2+K(2):
712 IF K(1)>K(2) THEN 720 ELSE IF K(1)=K(2) THEN 740 ELSE TT=16:
    GOSUB 1600:
    PRINT "Вы проиграли со счетом";K(1);" : ";K(2);" = ";
    K(1)-K(2):N2=N2+1:GOTO 730
720 TT=16: GOSUB 1600:
    PRINT"Вы выиграли со счетом";K(1);" : ";K(2);" = ";
    K(1)-K(2):N1=N1+1
730 GOSUB 1600:PRINT "Счет в матче стал";N1;" : ";N2:GOTO 750
740 TT=16:GOSUB 1600:PRINT "Игра закончилась вничью.":
745 PRINT "Счет в матче остался прежним:";N1;" : ";N2
750 GOSUB 1180:IF N1<5 THEN 790 ELSE PRINT "Вы выиграли матч."
751 IF NV<3 THEN 770 ELSE IF N2<2 TNEN 752 ELSE 900
752 PRINT "Это прекрасный результат!";
753 PRINT "Я объявляю Вас чемпионом игры в калах.";
754 PRINT "Примите мои поздравления!"
760 GOTO 810
770 IF N2<2 THEN
    PRINT "Это хороший результат.";
    "Я советую Вам сыграть на следующем уровне."
    ELSE PRINT "Это не так плохо,":
    PRINT "хотя я думаю, что Вам";
    "следует сыграть на этом уровне еще раз."
```

```
780 GOTO 810
790 IF N2<5 THEN 850 ELSE PRINT "Вы проиграли матч.":
    IF NV>1 THEN 800 ELSE PRINT "Не расстраивайтесь."
795 PRINT "Сыграйте снова, может быть,";
    "на этот раз у Вас выйдет лучше.":GOTO 810
800 IF N1<P3 THEN
    PRINT "Я советую Вам сыграть на предыдущем уровне." ELSE
    PRINT "Не расстраивайтесь.":
    PRINT "Сыграйте снова, может быть, на этот раз у Вас";
    "выйдет лучше."
810 IF HS(LV-1,4,0)+HS(LV-1,4,1)=0 THEN 1200 ELSE
    IF N1<HS(LV-1,4,0) THEN 840
820 IF (N1=HS(LV-1,4,0) AND N2>HS(LV-1,4,1)) THEN 840
830 IF N1=HS(LV-1,4,0) AND N2=HS(LV-1,4,1) AND
    T1-T2<=HS(LV-1,4,2)-HS(LV-1,4,3) THEN 840 ELSE GOTO 1200
840 LOCATE „1:END
850 GOSUB 1600:PRINT:
855 PRINT "Вы будете продолжать матч? (Д)а/(Н)ет/(С)начала"
860 Z$=INKEY$:IF Z$="D" THEN 100
870 IF Z$="N" THEN 890 ELSE IF Z$="S" THEN 80
880 GOTO 860
890 LOCATE „1:END
900 GOSUB 1600:PRINT "Это замечательный результат.";
    "Если Вы будете играть так дальше,";
    "то обязательно станете чемпионом!":END
910 CLS:LOCATE 13,2,0:PRINT CHR$(168);:FOR I=0 TO 7:
    PRINT STRING$(4,CHR$(140));CHR$(172);:NEXT
920 FOR J=0 TO 1
925 FOR I=0 TO 5:LOCATE 19+I*5,1,0:NN=12-I:PRINT NN;:NEXT
930 FOR I=0 TO 8:LOCATE 13+I*5,3+J*2,0:PRINT CHR$(170);:NEXT
940 FOR I=0 TO 5:LOCATE 20+I*5,3+J*2:PRINT "6";:NEXT
950 NEXT
960 LOCATE 13,4,0:PRINT CHR$(170):LOCATE 53,4,0:PRINT CHR$(170):
    LOCATE 18,4,0:PRINT CHR$(170);
970 LOCATE 15,4,0:PRINT "0";:LOCATE 50,4,0:PRINT "0";
974 LOCATE 19,4,0: FOR I=0 TO 5:
    PRINT STRING$(4,CHR$(143));CHR$(175);:NEXT
976 LOCATE 13,6,0:PRINT CHR$(130);STRING$(40,131)
978 FOR I=0 TO 5:LOCATE 19+I*5,7,0:NN=1+I:PRINT NN:NEXT
980 RETURN
990 I0=P(J):IF I0=0 THEN 1020 ELSE P(J)=0:IF J>6 THEN 1100
1000 IF I0+J>6 THEN 1040
1010 FOR I=J+1TOI0+J:P(I)=P(I)+1:NEXT I:M=I0+J:GOTO 1090
1020 IF J>6 THEN K(2)=-200 ELSE K(2)=200
1030 GOTO 1170
1040 IF J=6 THEN 1060
1050 FOR I=J+1 TO 6:P(I)=P(I)+1:NEXT I
```

```
1060 K(1)=K(1)+1: I0=I0+J-7: IF I0=0 THEN 1160 ELSE
     IF I0>6 THEN 1080
1070 FOR I=7 TO 6+I0:P(I)=P(I)+1:NEXT I:GOTO 1170
1080 FOR I=7 TO 12:P(I)=P(I)+1:NEXT I:I0=I0-6:J=0:GOTO 1000
1090 IF P(M)>1 THEN 1170 ELSE IF P(13-M)=0 THEN 1170
     ELSE K(1)=K(1)+P(M)+P(13-M):P(M)=0:P(13-M)=0:GOTO 1170
1100 IF I0+J>12 THEN 1120
1110 FOR I=J+1 TO I0+J:P(I)=P(I)+1:NEXT I:M=I0+J:GOTO 1150
1120 IF J=12 THEN 1140
1130 FOR I=J+1 TO 12:P(I)=P(I)+1:NEXT I
1140 K(2)=K(2)+1:I0=I0+J-13:IF I0=0 THEN 1170 ELSE J=0:GOTO 1100
1150 IF M<7 THEN 1170 ELSE IF P(M)>1 THEN 1170 ELSE IF P(13-M)=0
     THEN 1170 ELSE K(2)=K(2)+P(M)+P(13-M):P(M)=0:P(13-M)=0:
     GOTO1170
1160 Z=1
1170 RETURN
1180 GOSUB 1600:PRINT "Общий счет стал";T1;":";T2;" = ";T1-T2:
     GOSUB 1600:RETURN
1190 REM Запись пьедестала почета на диск
1200 GOSUB 1890:CLS:PRINT "Примите мои поздравления!!!":
1205 PRINT:PRINT:PRINT "Вы вошли в историю!!!":PRINT:PRINT
1210 LOCATE „1:INPUT "Введите Ваше имя";N$
1220 FOR I=3 TO 0 STEP -1
1230 IF (HS(LV-1,I,0)+HS(LV-1,I,1)=0) THEN 1270
1240 IF N1<HS(LV-1,I,0) THEN 1280 ELSE IF N1>HS(LV-1,I,0)
     THEN 1270
1250 IF N2>HS(LV-1,I,1) THEN 1280 ELSE IF N2<HS(LV-1,I,1)
     THEN 1270
1260 IF T1-T2<=HS(LV-1,I,2)-HS(LV-1,I,3) THEN 1280
1270 NEXT I
1280 FOR I2=4 TO I+1 STEP -1:H$(LV-1,I2+1)=H$(LV-1,I2)
1290 FOR I3=0 TO 3:HS(LV-1,I2+1,I3)=HS(LV-1,I2,I3):NEXT I3:
     NEXT I2
1300 H$(LV-1,I+1)=N$:HS(LV-1,I+1,0)=N1:HS(LV-1,I+1,1)=N2:
1305 HS(LV-1,I+1,2)=T1:HS(LV-1,I+1,3)=T2
1310 OPEN"O",#1,"KALAH.DAT"
1320 FOR I1=0 TO 2:FOR I2=0 TO 4:FOR I3=0 TO 3:
     PRINT #1,HS(I1,I2,I3):NEXT I3:PRINT #1,H$(I1,I2):NEXT I2:
     NEXT I1:CLOSE
1330 GOSUB 1640:GOSUB 1700 :REM Распечатка пьедестала почета
1340 PRINT "Хотите сыграть еще раз?  (Д)а/(Н)ет";
1350 A$=INKEY$:IF A$=""THEN 1350
1360 IF LEFT$(A$,1)="D" THEN 80 ELSE 840
1370 REM Конец подпрограммы записи на диск
1380 ON I GOTO 1390,1400,1410,1420,1430,1440,1450,1460,1470,
     1480,1490,1500
1390 LOCATE 19,5,0:PRINT "    ";:GOSUB 1590:
```

```
1395 LOCATE 19,5,0:PRINT P(1);:GOSUB 1590:RETURN
1400 LOCATE 24,5,0:PRINT "   ";:GOSUB 1590:
1405 LOCATE 24,5,0:PRINT P(2);:GOSUB 1590:RETURN
1410 LOCATE 29,5,0:PRINT "   ";:GOSUB 1590:
1415 LOCATE 29,5,0:PRINT P(3);:GOSUB 1590:RETURN
1420 LOCATE 34,5,0:PRINT"   ";:GOSUB 1590:
1425 LOCATE 34,5,0:PRINT P(4);:GOSUB 1590:RETURN
1430 LOCATE 39,5,0:PRINT "   ";:GOSUB 1590:
1435 LOCATE 39,5,0:PRINT P(5);:GOSUB 1590:RETURN
1440 LOCATE 44,5,0:PRINT "   ";:GOSUB 1590:
1445 LOCATE 44,5,0:PRINT P(6);:GOSUB 1590:RETURN
1450 LOCATE 44,3,0:PRINT "   ";:GOSUB 1590:
1455 LOCATE 44,3,0:PRINTP(7);:GOSUB 1590:RETURN
1460 LOCATE 39,3,0:PRINT "   ";:GOSUB 1590:
1465 LOCATE 39,3,0:PRINT P(8);:GOSUB 1590:RETURN
1470 LOCATE 34,3,0:PRINT "   ";:GOSUB 1590:
1475 LOCATE 34,3,0:PRINT P(9);:GOSUB 1590:RETURN
1480 LOCATE 29,3,0:PRINT "   ";:GOSUB 1590:
1485 LOCATE 29,3,0:PRINT P(10);:GOSUB 1590:RETURN
1490 LOCATE 24,3,0:PRINT "   ";:GOSUB 1590:
1495 LOCATE 24,3,0:PRINT P(11);:GOSUB 1590:RETURN
1500 LOCATE 19,3,0:PRINT "   ";:GOSUB 1590:
1505 LOCATE 19,3,0:PRINT P(12);:GOSUB 1590:RETURN
1510 LOCATE 49,4,0:PRINT "   ";:GOSUB 1590:
1515 LOCATE 49,4,0:PRINT K(1);:GOSUB 1590:RETURN
1520 LOCATE 14,4,0:PRINT "   ";:GOSUB 1590:
1525 LOCATE 14,4,0:PRINT K(2);:GOSUB 1590:RETURN
1530 FOR EE=1 TO 3:I=J:GOSUB 1380:NEXT EE:RETURN
1540 FOR EE=1 TO 3:I=M:GOSUB 1380:I=13-M:GOSUB 1380:NEXT EE
1550 K(LL)=K(LL)+P(M)+P(13-M)
1560 P(M)=0:I=M:GOSUB 1380
1570 P(13-M)=0:I=13-M:GOSUB1380
1580 ON LL GOSUB 1510,1520:RETURN
1590 FOR HH=0 TO DL:NEXT HH:RETURN
1600 TT=TT+1:IF TT>14 THEN LOCATE 1,8,0:FOR II=1 TO 7:
     PRINT STRING$(64,32):NEXT:TT=8
1610 LOCATE 1,TT,0:RETURN
1620 PRINT:PRINT:
1625 PRINT "Чтобы продолжить игру, нажмите любую клавишу.":S$=""
1630 S$=INKEY$:IF S$=""THEN 1630 ELSE RETURN
1640 REM
1650 OPEN"I",#1,"KALAH.DAT"
1660 IF EOF(1) THEN 1680
1670 FOR I1=0 TO 2:FOR I2=0 TO 4:FOR I3=0 TO 3:
     INPUT #1,HS(I1,I2,I3):NEXT I3:INPUT #1,H$(I1,I2):NEXT I2:
     NEXT I1
1680 LOCATE „0:CLOSE:RETURN
```

```
1690 REM Распечатка пьедестала почета
1700 CLS:
     PRINT "******  П Ь Е Д Е С Т А Л     П О Ч Е Т А  *****"
1710 PRINT TAB(12);"Уровень 1";TAB(41);"Уровень 2"
1720 PRINT "     Имя      Счет     Мячи ";
1725 PRINT TAB(30);"     Имя      Счет     Мячи"
1730 FOR I2=0 TO 4: FOR I1=0 TO 1
1740 PRINT TAB(30*I1);MID$(STR$(I2+1),2,1);". ";
1750 IF HS(I1,I2,0)+HS(I1,I2,1)=0 THEN 1780
1760 PRINT USING "\          \"; H$(I1,I2);: PRINT" ";:
     PRINT MID$(STR$(HS(I1,I2,0)),2,1);":";
     MID$(STR$(HS(I1,I2,1)),2,1);"    ";
1770 PRINT USING "#####";HS(I1,I2,2)-HS(I1,I2,3);
1780 IF I1=1 THEN PRINT
1790 NEXT I1:NEXT I2
1800 PRINT TAB(12);"Уровень 3"
1810 PRINT "     Имя      Счет     Мячи"
1820 FOR I2=0 TO 4
1830 PRINT MID$(STR$(I2+1),2,1);". ";
1840 IF HS(2,I2,0)+HS(2,I2,1)=0 THEN PRINT:GOTO 1870
1850 PRINT USING"\          \";H$(2,I2);:PRINT " ";:
     PRINT MID$(STR$(HS(2,I2,0)),2,1);":";
     MID$(STR$(HS(2,I2,1)),2,1);"    ";
1860 PRINT USING "#####";HS(2,I2,2)-HS(2,I2,3)
1870 NEXT I2: RETURN
1880 REM Конец подпрограммы распечатки
1890 PRINT "Нажмите любую клавишу, чтобы продолжить.";:S$=""
1900 S$=INKEY$:IF S$="" THEN 1900 ELSE RETURN
1910 PRINT:PRINT K(2);"   ";:FOR I=12 TO 7 STEP-1:PRINT P(I);:
     NEXT:PRINT "   ";K(1)
1920 PRINT "      ";:FOR I=1 TO 6:PRINT P(I);:NEXT
1930 KK$=INKEY$:IF KK$=""THEN 1930
1940 RETURN
```

На этом мы закончим примеры игровых программ и приведем одну из программ для научно-технических расчетов - программу построения графиков функций.

ПОСТРОЕНИЕ ГРАФИКОВ

На практике довольно часто встречаются функциональные зависимости, поступающие к нам в виде таблиц. Чтобы наглядно представить себе вид такой функции, можно построить график на миллиметровке. При этом необходимо верно угадать масштаб по осям, чтобы график, как говорят, смотрелся. Часто приходится выделять наиболее интересную часть графика в укрупненном масштабе. Построение графика на бумаге требует значительного времени, аккуратной и до-

статочно тупой работы. Приведенная ниже программа GROS позволяет строить на экране вашего дисплея график функции, заданной таблично, менять масштабы по осям координат, выделять и укрупнять наиболее интересные участки. Чтобы при помощи программы GROS построить график функции, необходимо занести в нее исходную информацию. При работе программа использует оператор READ для чтения данных, представленных в виде последовательностей чисел в операторах DATA. Если функциональная зависимость F(U) задана в виде последовательности точек $F_i=F(U_i)$, где i=0, 1, ..., n, ее следует занести в операторы DATA в таком порядке:

U_0, F_0, ..., U_i, F_i, ..., U_n, F_n, S,

причем $U_0<=U_i<=U_n$, а $S<=U_n$ - признак конца ввода данного графика. В операторы DATA можно занести сразу несколько функциональных зависимостей, отделенных друг от друга соответствующими признаками S, а затем выводить их последовательно на одну и ту же или на разные координатные плоскости. Пример ввода последовательностей точек при помощи операторов DATA приводится в конце распечатки программы.

Программа определяет области задания аргумента и функции и предлагает построить оси координат в выбранном ею самой масштабе, выдавая рекомендуемые значения:

X_{origin} - место по оси абсцисс крайней левой точки графика на экране дисплея;

X_{length} - длину оси абсцисс графика ($X_{origin}+X_{length}<511$);

Y_{origin} - место на экране верхней точки графика;

Y_{length} - длину оси ординат графика ($Y_{origin}+Y_{length}<255$);

U_{min}, U_{max} - диапазон вывода аргумента функции, заданной таблично;

F_{min}, F_{max} - диапазон вывода значений функции.

Если предлагаемые компьютером значения устраивают, можно на каждый вопрос просто отвечать нажатием клавиши ВК, в противном случае следует ввести другие данные.

Программа построит оси координат, разбивая каждую из них на 4 - 6 частей, и оцифрует их. При этом она может немного увеличить диапазоны задания аргумента и функции, чтобы по осям были "более удобные" числа, а также несколько уменьшить длины осей координат.

При построении табличной функции на координатную плоскость сначала наносят точки, соответствующие значениям функции при заданных значениях аргумента, а затем соединяют их достаточно гладкой кривой. Получение промежуточных значений функции между заданными точками называют интерполяцией. Если точки нанесены на бумагу, а в руках у вас карандаш, вы можете соединить их (т. е. проинтерполировать функцию) "на глаз", можете воспользоваться лекалами или гибкой стальной линейкой. А как при этом поступает компьютер?

Он задает вопросы о том, какой вид интерполяции вас устраивает, как маркировать точки и каким цветом выводить график. Если вывод данного графика не нужен, на вопрос о типе графика следует ответить отрицательным числом. Если вы ответите нулем, на график будут выведены только табулированные значения в виде точек (маркер 0) или различных маркеров размером 7*5 экранных точек.

Программа позволяет произвести кусочно-линейную (тип графика 1) или кусочно-кубичную (тип графика от 2 до 8) интерполяцию. Если вы укажете тип графика 9, на экране продемонстрируются все виды интерполяции, заложенные в программу (в два раза меньшем масштабе по оси ординат).

Если точки задания (узлы) функции заданы достаточно часто, между ними можно провести отрезки прямых линий (кусочно-линейная интерполяция), причем график будет выглядеть достаточно гладким. Если же точки заданы редко, их лучше соединить отрезками кривых, представляющих собой куски графиков кубичных многочленов, сшитых в узлах по значениям функции и наклону, т. е. по первой производной. Поскольку в узлах заданы только значения функций, значения производных можно выбрать достаточно произвольно, а затем вычислить необходимые четыре коэффициента при различных степенях аргумента кубичной функции.

Значения первой производной проще всего получить в узлах, если через три рядом стоящие точки провести параболу и тангенс угла наклона ее касательной в средней точке считать производной функции в этой точке. Этот вид интерполяции программа реализует, если выбрать тип графика 2.

Более естественным является случай, если в узлах интерполяции будут сшиты не только значения функции и ее первые производные, но и вторые производные. Этот вид интерполяции называют интерполяцией кубичными сплайнами, и иллюстрировать его можно поставленной на ребро гибкой стальной линейкой, согнутой по гвоздикам, "вбитым" в узлы интерполяции. Коэффициенты кубичных многочленов в этом случае вычисляются решением системы линейных уравнений с трехдиагональной матрицей методом прогонки. Этот случай программа реализует при типе графика 3.

Однако встречаются функциональные зависимости, когда функция резко меняется от точки к точке. При описанных выше видах кусочно-кубичной интерполяции в этом случае могут возникать дополнительные максимумы и минимумы функции (переколебания). От этого эффекта можно избавиться, если в случае монотонной зависимости функции от аргумента выбирать производную как минимум модуля тангенса наклона прямых слева и справа от узла при кусочно-линейной интерполяции, а в экстремумах приравнивать ее нулю. Этот вид интерполяции производится при типе графика 4.

Остальные типы графиков дают промежуточные результаты между описанными выше и тремя основными типами выбора производных в узлах интерполяции кусочно-кубичными многочленами. Во всех слу-

чаях кусочно-кубичной интерполяции для крайних точек графика выбрано естественное условие равенства нулю второй производной в этих точках.

После вывода графика программа выходит на оператор STOP. Команда CONT позволяет продолжить работу с программой, причем можно работать с той же самой функциональной зависимостью (не вводить новый график), либо со следующей. Если вы ответите отрицательно на вопрос о построении новых координатных осей, новый график будет выводиться рядом с уже нарисованными.

```
10 :REM   Program GROS
20 :REM
30 :REM   Построение графиков по заданным точкам с применением
40 :REM   кусочно-линейной или кусочно-кубичной интерполяции
50 :REM   различных типов.
60 :REM   Массивы точек заносятся в программу операторами DATA
70 :REM   (в конце распечатки программы) в виде
80 :REM   последовательностей U0,F0, ...,Ui,Fi, ...,Un,Fn, S, где
90 :REM   аргументы U0<...Ui<...Un, а S<Un является
100 :REM  признаком конца данных одного графика. Разные
110 :REM  графики могут выводиться последовательно на одном
120 :REM  или нескольких рисунках.
130 :REM
140 DEFINT I-N
150 DEF FNF3(X)=((A3*X+A2)*X+A1)*X+A0
160 DEF FNP3(X)=(3!*A3*X+A2+A2)*X+A1
170 DEF FNSL(Y)=INT((XP-XM)*(Y-YM)/(YP-YM)+XM+.5)+X0%
180 DIM U(100),F(100),O(4,100)
190 WG$="N"
200 X0%=50 : XL%=431
210   GOTO 270
220 CLS
230 PRINT "Ввод нового массива <Y-N>?"
240 W$=INKEY$ : IF W$="" THEN 240
250 IF W$ <> "Y" AND W$<>"y" THEN 380
260 :REM Ввод массивов Ui и Fi, подсчет точек WN%=n
270 WN%=0
280 READ U(0),FM
290   F(0)=FM : FP=FM
300   READ UI
310    IF UI<=U(WN%) THEN 380
320    WN%=WN%+1 : U(WN%)=UI
330   READ FI
340    F(WN%)=FI
350    IF FI>FP THEN FP=FI
360    IF FI<FM THEN FM=FI
370   GOTO 300
```

```
380 CLS
390 PRINT "Число точек введенного массива ="; WN%+1
400 PRINT "Umin=";U(0);"  Umax=";U(WN%)
410 PRINT "Fmin="; FM ;"  Fmax="; FP
420 IF WG$<>"Y" THEN 460
430 PRINT "Построение новых осей <Y-N> ?"
440   W$=INKEY$ : IF W$="" THEN 440
450   IF W$<>"Y" AND W$<>"y" THEN 690
460    X0%=40 : XL%=470
470    Y0%=5 : YL%=239
480    UB=U(0) : UF=U(WN%)
490    FB=(FM-FP)*.05+FM
500    FF=(FP-FM)*.05+FP
510    WT%=3 : WM%=8 : WC%=6
520 :REM Построение осей
530   WG$="Y"
540 PRINT X0%;"   ";XL%,
550 INPUT "ВВОД Xorigin,Xlength";X0%,XL%
560 PRINT Y0%;"   ";YL%,
570 INPUT "ВВОД Yorigin,Ylength";Y0%,YL%
580 PRINT UB;"   ";UF,
590 INPUT "ВВОД Ubeg,Ufin";UB,UF
600 PRINT FB;"   ";FF,
610 INPUT "ВВОД Fmin,Fmax";FB,FF
620 PRINT "   ";WC%;
630 INPUT "ЦВЕТ ОСЕЙ";WC%
640 COLOR WC%,0
650 CLS
660 PCLS
670 GOSUB 1910          :REM Subr. AXES
680 :REM Пересылка функции в рабочие массивы
690   YL=YL% : XL=XL%
700 WF=YL/(FF-FB)
710 WU=XL/(UF-UB)
720 FOR I=0 TO WN%
730    O(0,I)=(U(I)-UB)*WU
740    O(1,I)=(FF-F(I))*WF
750    NEXT I
760 PRINT "   -1 - вывод графика игнорируется"
770 PRINT "    0 - точки"
780 PRINT "    1 - линейная интерполяция"
790 PRINT "    2 - кубичная, F' - параболлическая"
800 PRINT "    3 -   - " - , сплайны"
810 PRINT "    4 -   - " - , |F'| - минимальная"
820 PRINT "    5 -  случаи   2 + 3
830 PRINT "    6 -  - " -   2 + 4 "
840 PRINT "    7 -  - " -   3 + 4 "
```

```
850 PRINT "    8 - - " -   2 + 3 + 4"
860 PRINT WT%;WM%;
870 INPUT "Тип графика <-1 - 8> и маркер <0 - 8>";WT%,WM%
880 PRINT "    ";WC%;
890 INPUT "Цвет графика <1 - 7>";WC%
900 COLOR WC%,0
910 CLS
920   IF WT% <= 8 THEN 970
930  FOR I=0 TO WN%
940    O(1,I)=O(1,I)*.5
950    NEXT
960 :REM Вывод точек
970 IF WT%<0 THEN 220
980  FOR I=0 TO WN%
990    XX%=O(0,I)
1000    IF XX%<0 OR XX%>XL% THEN 1050
1010    YY%=O(1,I)
1020    IF YY%<0 OR YY%>YL% THEN 1050
1030    PSET(X0%+XX%,Y0%+YY%)
1040    ON WM% GOSUB 5130,5140,5150,5160,5170,5180,5190,5200:
          REM Subr.MARKERS
1050    NEXT
1060 ON WT% GOSUB 3130,1120,1170,1220,1270,1340,1410,1600,1710:
          REM Тип графика
1070 STOP
1080 GOTO 220
1090 :REM   * * * * * * * * * * * * * * * * * * * * *
1100 :REM   Тип = 2
1110 :REM   * * * * * * * * * * * * * * * * * * * * *
1120   GOSUB 270          :REM  PSQUARE
1130    GOTO 1490
1140 :REM   * * * * * * * * * * * * * * * * * * * * *
1150 :REM   Тип = 3
1160 :REM   * * * * * * * * * * * * * * * * * * * * *
1170   GOSUB 3950         :REM  SPLINE
1180    GOTO 4580         :REM  INTERCUBE
1190 :REM   * * * * * * * * * * * * * * * * * * * * *
1200 :REM   Тип = 4
1210 :REM   * * * * * * * * * * * * * * * * * * * * *
1220   GOSUB 4430         :REM  PABSMIN
1230    GOTO 1490
1240 :REM   * * * * * * * * * * * * * * * * * * * * *
1250 :REM   Тип = 5
1260 :REM   * * * * * * * * * * * * * * * * * * * * *
1270   GOSUB 3950         :REM  SPLINE
1280   GOSUB 1830         :REM  COPYP
1290   GOSUB 4270         :REM  PSQUARE
```

```
1300    GOTO 1450
1310 :REM   * * * * * * * * * * * * * * * * * * * * *
1320 :REM   Тип = 6
1330 :REM   * * * * * * * * * * * * * * * * * * * * *
1340    GOSUB 4430          :REM  PABSMIN
1350    GOSUB 1830          :REM  COPYP
1360    GOSUB 4270          :REM  PSQUARE
1370     GOTO 1450
1380 :REM   * * * * * * * * * * * * * * * * * * * * *
1390 :REM   Тип = 7
1400 :REM   * * * * * * * * * * * * * * * * * * * * *
1410    GOSUB 3950          :REM  SPLINE
1420    GOSUB 1830          :REM  COPYP
1430    GOSUB 4430          :REM  PABSMIN
1440 :REM Усреднение производных в бинарных случаях
1450    FOR I=0 TO WN%
1460      O(2,I)=(O(2,I)+O(3,I))*.5
1470      NEXT
1480 :REM Коэффициенты кубичной интерполяции
1490    FOR I=1 TO WN%
1500      OH=O(0,I)-O(0,I-1)
1510      A2=(O(1,I)-O(1,I-1))/OH
1520      A3=O(2,I)+O(2,I-1)-A2-A2
1530      O(3,I-1)=(A2-A3-O(2,I-1))/OH
1540      O(4,I-1)=A3/(OH*OH)
1550      NEXT
1560     GOTO 4580          :REM  INTERCUBE
1570 :REM   * * * * * * * * * * * * * * * * * * * * *
1580 :REM   Тип = 8
1590 :REM   * * * * * * * * * * * * * * * * * * * * *
1600     GOSUB 3950         :REM  SPLINE
1610     GOSUB 1830         :REM  COPYP
1620     GOSUB 4270         :REM  PSQUARE
1630    FOR I=0 TO WN%
1640      O(4,I)=O(2,I)
1650      NEXT
1660.    GOSUB 4430         :REM  PABSMIN
1670    FOR I=0 TO WN%
1680      O(2,I)=(O(2,I)+O(3,I)+O(4,I))/3!
1690      NEXT
1700      GOTO 1490
1710 :REM   * * * * * * * * * * * * * * * * * * * * *
1720 :REM   Тип = 9
1730 :REM   * * * * * * * * * * * * * * * * * * * * *
1740    FOR WT%=1 TO 8
1750     IF WT%<5 THEN COLOR WT%+2 ELSE COLOR WT%-2
1760     OQ=YL/16!
```

```
1770     FOR I=0 TO WN%
1780       O(1,I)=O(1,I)+OQ
1790       NEXT I
1800     ON WT% GOSUB 3130,1120,1170,1220,1270,1340,1410,1600:
           REM Тип графика
1810       NEXT WT%
1820   RETURN
1830 :REM   * * * * * * * * * * * * * * * * * * * * *
1840 :REM   Subroutine COPYP
1850 :REM   * * * * * * * * * * * * * * * * * * * * *
1860   FOR I=0 TO WN%
1870     O(3,I)=O(2,I)
1880     NEXT
1890   RETURN
1900 :REM   * * * * * * * * * * * * * * * * * * * * *
1910 :REM   Subroutine AXES
1920 :REM   * * * * * * * * * * * * * * * * * * * * *
1930 ZB=UB
1940 ZF=UF
1950 ZL%=XL%
1960 GOSUB 2710
1970 GOSUB 2710
1980   UB=ZB
1990   UF=ZF
2000   UH=ZH
2010   XN%=ZN%
2020   XM%=ZM%
2030   XL%=ZL%
2040   XH%=ZH%
2050 ZB=FB
2060 ZF=FF
2070 ZL%=YL%
2080 GOSUB 2710
2090 GOSUB 2710
2100   FB=ZB
2110   FF=ZF
2120   FH=ZH
2130   YL%=ZL%
2140   YN%=ZN%
2150   YM%=ZM%
2160   YH%=ZH%
2170 LINE(X0%,Y0%)-STEP(XL%,YL%),,B
2180 QH%=XH%\2
2190 FOR I=1 TO YN%
2200   FOR J=1 TO YM%-1
2210     QZ%=(I*YM%-J)*YH%+Y0%
2220     LINE(X0%,QZ%)-STEP(QH%,0)
```

```
2230      LINE(X0%+XL%,QZ%)-STEP(-QH%,0)
2240    NEXT J,I
2250 FOR I=1 TO YN%-1
2260    QZ%=YM%*I*YH%+Y0%
2270    LINE(X0%,QZ%)-STEP(XH%,0)
2280    FOR J=3 TO XN%*XM%
2290      PSET STEP(XH%,0)
2300      NEXT J
2310    LINE -STEP(XH%,0)
2320    NEXT I
2330 QH%=YH%\2
2340 FOR I=1 TO XN%
2350    FOR J=1 TO XM%-1
2360      QZ%=(I*XM%-J)*XH%+X0%
2370      LINE(QZ%,Y0%)-STEP(0,QH%)
2380      LINE(QZ%,Y0%+YL%)-STEP(0,-QH%)
2390      NEXT J,I
2400 FOR I=1 TO XN%-1
2410    QZ%=I*XM%*XH%+X0%
2420    LINE(QZ%,Y0%)-STEP(0,YH%)
2430    FOR J=3 TO YM%*YN%
2440      PSET STEP(0,YH%)
2450      NEXT J
2460    LINE -STEP(0,YH%)
2470    NEXT I
2480 XI=CINT(UB/UH)
2490 XX%=X0%
2500 XD%=XH%*XM%
2510 FOR I=0 TO XN%
2520    DI=UH*XI
2530    YY%=Y0%+YL%+11
2540    DS%=0
2550    GOSUB 2920
2560    XI=XI+1!
2570    XX%=XX%+XD%
2580    NEXT
2590 YI=CINT(FF/FH)
2600 YY%=Y0%+3
2610 FOR I=0 TO YN%
2620    DI=FH*YI
2630    XX%=X0%-7
2640    DS%=-1
2650    GOSUB 2920
2660    YI=YI-1!
2670    YY%=YY%+YH%*YM%
2680    NEXT
2690 RETURN
```

```
2700 :REM   * * * * * * * * * * * * * * * * * * * * *
2710 :REM   Subroutine AXIS
2720 :REM   * * * * * * * * * * * * * * * * * * * * *
2730 IF ZF > ZB THEN 2760
2740   IF ZF = 0 THEN ZF=1!
2750   IF ZF > 0 THEN ZB=0! ELSE ZF=0!
2760 ZD=ZF-ZB
2770 ZR=LOG(ZD)/LOG(10!)-.0792
2780 ZR=10!^INT(ZR)
2790 ZI=CINT(ZD*.625/ZR+.5)
2800 ZH=ZR+ZR
2810 ZM%=4
2820 IF ZI<5! THEN ZH=.25*ZI*ZR : ZM%=5
2830 ZB=ZH*INT(ZB/ZH)
2840 ZF=ZH*INT(ZF/ZH+.999)
2850 ZN%=INT((ZF-ZB)/ZH+.5)
2860 ZH%=ZL%\(ZN%*ZM%)
2870 ZL%=ZH%*ZN%*ZM%
2880 RETURN
2890 :REM   * * * * * * * * * * * * * * * * * * * * *
2900 :REM   Subroutine PRVALUE
2910 :REM   * * * * * * * * * * * * * * * * * * * * *
2920 DA$=STR$(DI)
2930 DL%=LEN(DA$)
2940 DD$=RIGHT$(DA$,4)
2950 IF ASC(DD$)<58 THEN 2980
2960   DA$=LEFT$(DA$,DL%-4)+RIGHT$(DA$,3)
2970   DL%=DL%-1
2980 IF ASC(DA$)<43 THEN DL%=DL%-1
2990 DX%=XX%
3000 IF DS%=0 THEN DX%=DX%-7*DL%\2
3010 IF DS%<0 THEN DX%=DX%-7*DL%
3020 DJ%=POINT(DX%-3,YY%)
3030   DA$=RIGHT$(DA$,DL%)
3040   ON ASC(DA$)-42 GOSUB 5250,5250,5270,5290,5290,
          5310,5330,5350,5370,5390,5410,5430,5450,5470,5490
3050   DL%=DL%-1
3060   IF DL%>0 GOTO 3030
3070 RETURN
3080 :REM   * * * * * * * * * * * * * * * * * * * * *
3090 :REM   Subroutine INTERLINE
3100 :REM   * * * * * * * * * * * * * * * * * * * * *
3110 :REM   Кусочно-линейная интерполяция
3120 :REM
3130 IF O(0,0)>XL OR O(0,WN%)<0! THEN RETURN
3140 :REM   * * *  Первая точка  * * *
3150    I=0
```

```
3160    I=I+1
3170 IF O(0,I)<=0! THEN 3160
3180    XM=O(0,I-1)
3190    YM=O(1,I-1)
3200 IF XM=>0! THEN 3240
3210 :REM   * * *  XM < 0  * * *
3220    YM=(O(1,I)-YM)*XM/(XM-O(0,I))+YM
3230    XM=0!
3240    XM%=INT(XM+.5)+X0%
3250    YM%=INT(YM+.5)+Y0%
3260    IF YM>YL THEN YM%=YL%+Y0%
3270    IF YM<0! THEN YM%=Y0%
3280 :REM   * * *  Следующая точка (продолжение)  * * *
3290  XP=O(0,I)
3300  YP=O(1,I)
3310 IF XP<=XL THEN 3360
3320 :REM   * * *  XP > XL  * * *
3330  YP=(YP-YM)*(XL-XM)/(XP-XM)+YM
3340  XP=XL
3350 :REM   * * *  Прямая  * * *
3360    GOSUB 3540
3370 :REM   * * *  Следующая точка  * * *
3380 IF O(0,I)=>XL THEN RETURN
3390    I=I+1
3400 IF I<=WN% THEN 3290 ELSE RETURN
3410 :REM   * * * * * * * * * * * * * * * * * * * * *
3420 :REM   Subroutine GRLINE
3430 :REM   * * * * * * * * * * * * * * * * * * * * *
3440 :REM
3450 :REM   Рисует на экране прямую линию от XM до XP
3460 :REM   в пределах площади графика
3470 :REM   Ограничения по оси X не проверяет
3480 :REM   По окончании подпрограммы конечная точка
3490 :REM   становится начальной
3500 :REM   Исходные данные:
3510 :REM     по начальной точке: XM, YM, XM%, YM%
3520 :REM     по конечной точке: XP, YP
3530 :REM     по графику: YL, X0%, Y0%
3540 IF YP<0 THEN 3620
3550 IF YP>YL THEN 3710
3560 :REM   * * *  0 <= YP <= YL  * * *
3570    XP%=INT(XP+.5)+X0%
3580    YP%=INT(YP+.5)+Y0%
3590 IF YM<0 THEN 3770
3600 IF YM>YL THEN 3680 ELSE 3790
3610 :REM   * * *  YP < 0  * * *
3620    YP%=Y0%
```

```
3630 IF YM<=0! THEN 3810
3640 :REM   * * *  YP < 0, YM > 0  * * *
3650    XP%=FNSL(0!)
3660 IF YM<=YL THEN 3790
3670 :REM   * * *  YP <= YL, YM > YL  * * *
3680    XM%=FNSL(YL)
3690       GOTO 3790
3700 :REM   * * *  YP > YL  * * *
3710    YP%=YL%+Y0%
3720 IF YM>=YL THEN 3810
3730 :REM   * * *  YP > YL, YM < YL  * * *
3740    XP%=FNSL(YL)
3750 IF YM>=0! THEN 3790
3760 :REM   * * *  YP => 0, YM < 0  * * *
3770    XM%=FNSL(0!)
3780 :REM   * * *  Прямая  * * *
3790 LINE(XM%,YM%)-(XP%,YP%)
3800 :REM   * * *  Конец  * * *
3810    XM=XP
3820    YM=YP
3830    XM%=XP%
3840    YM%=YP%
3850  RETURN
3860 :REM   * * * * * * * * * * * * * * * * * * * * *
3870 :REM   Subroutine SPLINE(O,NO)
3880 :REM   * * * * * * * * * * * * * * * * * * * * *
3890 :REM   DIM O(4,NO)
3900 :REM   O(0,i) - Xi, i=0,1,...,NO
3910 :REM   O(1,i) - Yi
3920 :REM   O(2,i) - Bi
3930 :REM   O(3,i) - Ci
3940 :REM   O(4,i) - Di
3950 NO=WN%
3960 O(3,0)=0!
3970 O(4,0)=0!
3980 IF NO=0 THEN O(2,0)=0!:RETURN
3990 OH=O(0,1)-O(0,0)
4000 O(2,0)=(O(1,1)-O(1,0))/OH
4010 O(3,NO)=0!
4020 O(4,NO)=0!
4030 IF NO=1 THEN O(2,1)=O(2,0):RETURN
4040 FOR I=1 TO NO-1
4050    OS=OH
4060    OH=O(0,I+1)-O(0,I)
4070    OQ=(2!-O(4,I-1))*OS+OH+OH
4080    O(2,I)=(O(1,I+1)-O(1,I))/OH
4090    O(3,I)=(O(2,I)-O(2,I-1)-O(3,I-1)*OS)/OQ
```

```
4100   O(4,I)=OH/OQ
4110   NEXT
4120 O(2,NO)=O(2,NO-1)+O(3,NO-1)*OH
4130 FOR I=NO-1 TO 0 STEP -1
4140  OH=O(0,I+1)-O(0,I)
4150   O(3,I)=O(3,I)-O(4,I)*O(3,I+1)
4160   O(2,I)=O(2,I)-(O(3,I+1)+O(3,I)*2!)*OH
4170   O(4,I)=(O(3,I+1)-O(3,I))/OH
4180   O(3,I+1)=O(3,I+1)*3!
4190   NEXT
4200 RETURN
4210 :REM   * * * * * * * * * * * * * * * * * * * *
4220 :REM   Subroutine PSQUARE
4230 :REM   * * * * * * * * * * * * * * * * * * * *
4240 :REM   Вычисляет значения производных на основании
4250 :REM   квадратичной интерполяции по трем точкам
4260 :REM
4270   OH=O(0,1)-O(0,0)
4280   OF=(O(1,1)-O(1,0))/OH
4290   O(2,0)=1.5*OF
4300   FOR I=1 TO WN%-1
4310    OS=OH
4320    OQ=OF
4330    OH=O(0,I+1)-O(0,I)
4340    OF=(O(1,I+1)-O(1,I))/OH
4350    IF OH=OS THEN O(2,I)=(OQ+OF)*.5
          ELSE O(2,I)=(OQ*OH+OF*OS)/(OS+OH)
4360     NEXT
4370   O(2,0)=O(2,0)-O(2,1)*.5
4380   O(2,WN%)=(OF-O(2,WN%-1))*.5+OF
4390   RETURN
4400 :REM   * * * * * * * * * * * * * * * * * * * *
4410 :REM   Subroutine PABSMIN
4420 :REM   * * * * * * * * * * * * * * * * * * * *
4430 OF=(O(1,1)-O(1,0))/(O(0,1)-O(0,0))
4440 O(2,0)=OF*1.5
4450 FOR I=1 TO WN%-1
4460   OQ=OF
4470   OF=(O(1,I+1)-O(1,I))/(O(0,I+1)-O(0,I))
4480   IF SGN(OQ)=-SGN(OF) THEN O(2,I)=0! : GOTO 4500
4490   IF ABS(OQ)<ABS(OF) THEN O(2,I)=OQ ELSE O(2,I)=OF
4500   NEXT
4510 GOTO 4370
4520 :REM   * * * * * * * * * * * * * * * * * * * *
4530 :REM   Subroutine INTERCUBE
4540 :REM   * * * * * * * * * * * * * * * * * * * *
4550 :REM   Строит график на основании кусочно-кубичной
```

```
4560 :REM   интерполяции
4570 :REM
4580 FOR I=0 TO WN%-1
4590   XF=O(0,I+1)
4600   IF XF<0 THEN 4690
4610   IF XF>XL THEN XF=XL
4620   XB=O(0,I)
4630   IF XB=>XL THEN RETURN
4640   A0=O(1,I)
4650   A1=O(2,I)
4660   A2=O(3,I)
4670   A3=O(4,I)
4680   GOSUB 4830      :REM Subr. GRCUBE
4690   NEXT
4700 RETURN
4710 :REM   * * * * * * * * * * * * * * * * * * * * *
4720 :REM   Subroutine GRCUBE
4730 :REM   * * * * * * * * * * * * * * * * * * * * *
4740 :REM   График кубичного полинома
4750 :REM   Рисует на экране отрезок кубичной
4760 :REM   функции от XB до XF
4770 :REM   Исходные данные:
4780 :REM     XB, XF, A0, A1, A2, A3, где
4790 :REM     Y(X)=A0+A1*(X-XB)+A2*(X-XB)^2+A3*(X-XB)^3
4800 :REM   Использует функции AF3 и AP3
4810 :REM
4820 IF XF<0! OR XB>XL THEN RETURN
4830 X1=0!
4840 X2=0!
4850 A4=A3*3!
4860 IF ABS(A3)>1E-09 THEN 4890
4870   IF ABS(A2)>1E-06 THEN X1=XB-.5*A1/A2
4880     GOTO 4940
4890  ZD=A2*A2-A1*A4
4900  IF ZD<=0! THEN 4940
4910    ZD=SQR(ZD)
4920    X1=XB-(A2+ZD)/A4
4930    X2=(ZD-A2)/A4+XB
4940 YM=A0
4950 XM=XB
4960 IF XB=>0! THEN 4990
4970   XM=0!
4980   YM=FNF3(-XB)
4990  XM%=CINT(XM)+X0%
5000  YM%=CINT(YM)+Y0%
5010  IF YM<0 THEN YM%=Y0%
5020  IF YM>YL THEN YM%=Y0%+YL%
```

```
5030      XP=CINT(XM+1!)
5040      IF XP>XF THEN XP=XF
5050      IF XM<X1 AND XP>X1 THEN XP=X1
5060      IF XM<X2 AND XP>X2 THEN XP=X2
5070      YP=FNF3(XP-XB)
5080      GOSUB 3540          :REM Subr. GRLINE
5090      IF XP<XF THEN 5030 ELSE RETURN
5100 :REM   * * * * * * * * * * * * * * * * * * * * * *
5110 :REM   Subroutines MARKERS
5120 :REM   * * * * * * * * * * * * * * * * * * * * * *
5130 DRAW"R1F2L1H4L1F2G2R1E4R1G2":RETURN
5140 DRAW"BH2D4BE4D4L5U4R6D5":RETURN
5150 DRAW"E1L3BH1R7BG3L3BG1R7":RETURN
5160 DRAW"L1G2U4F2R2E2D4H2":RETURN
5170 DRAW"BL3E2R2F2G2L2H2R2BR3":RETURN
5180 DRAW"NL3R3U2BL2L3BG1D3BR2R3BE1BL3BU2":RETURN
5190 DRAW"NG3E3BD1BL3D5BE3L7BE2F5BU1BL6BU4":RETURN
5200 DRAW"BE2F1U1G3H3R1G2BF1F1L1E3F3L1E2":RETURN
5210 :REM   * * * * * * * * * * * * * * * * * * * * * *
5220 :REM   Subroutines PRDIGITS
5230 :REM   * * * * * * * * * * * * * * * * * * * * * *
5240 :REM   + (43)
5250 DRAW"BE3ND3U2R1D5BH3R6C0D3":RETURN
5260 :REM   - (45)
5270 DRAW"BE3NL3R4C0D3":RETURN
5280 :REM   . (46)
5290 DRAW"BR4H1D1E2C0F2":RETURN
5300 :REM   0 (48)
5310 DRAW"BE1U4E1NR4D6R3U5R1D4F1":RETURN
5320 :REM   1 (49)
5330 DRAW"BR1R2U6G2R1E2D6R3":RETURN
5340 :REM   2 (50)
5350 DRAW"BE1E2R2E2BL6R1U1R3D1R1G5R5U1F1":RETURN
5360 :REM   3 (51)
5370 DRAW"BE1R1D1R3U6L4BD1R2BD2R1E2D2BD1D1F1":RETURN
5380 :REM   4 (52)
5390 DRAW"BR4NU5R1U6G4D1R5F1":RETURN
5400 :REM   5 (53)
5410 DRAW"BE6L5D2NE2R4D4L3U1L2BR6NU3F1":RETURN
5420 :REM   6 (54)
5430 DRAW"BR2R3U4L3H1D4R1U5R4BD3D2F1":RETURN
5440 :REM   7 (55)
5450 DRAW"BR3U1E3U2L5D1R1BR3D3H1D3R1C0BR2":RETURN
5460 :REM   8 (56)
5470 DRAW"BR2R3U6L3D5L1U2BU1U2BF3NL2BE2D2BD1D1F1":RETURN
5480 :REM   9 (57)
5490 DRAW"BR2R3U6L3D4NR3H1U2R1BR4D4F1":RETURN
```

```
5500 :REM   Исходные данные для графиков!
5510 :REM   F(x)=sin(Пx)
5520 DATA -2.5, -1, -2.25, -.7071, -2, 0, -1.75, .7071,
          -1.5, 1, -1.25, .7071, -1, 0, -.75, -.7071
5530 DATA -.5, -1, -.25, -.7071, 0, 0, .125, .3827,
           .25, .7071, .375, .924, .5, 1, .625, .924
5540 DATA  .75, .707, .875, .383, 1, 0, 1.125, -.383P,
          1.25, -.707, 1.375, -.924, 1.5, -1, 1.625, -.94
5550 DATA 1.75, -.707, 1.875, -.383, 2, 0, 2.125, .383,
          2.25,.707,2.375,.924,2.5,1,0
5560 :REM   F(x)=sin(Пx)/(Пx)
5570 DATA -2.5, .127, -2.25, .1, -2, 0, -1.75, -.129,
          -1.5, -.212, -1.25, -.18, -1, 0, -.75, .3
5580 DATA -.5, .636, -.25, .9, 0, 1, .125, .974,
           .25, .9, .375, .784, .5, .637, .625, .471
5590 DATA  .75, .3, .875, .139, 1, 0, 1.125, -.108,
          1.25, -.18, 1.375, -.214, 1.5, -.212, 1.625, -.181
5600 DATA 1.75, -.129, 1.875, -.065, 2, 0, 2.125, .057,
          2.25, .1, 2.375, .124, 2.5, .127, 0
5610 :REM Единичные перепады. Зависимость от длины фронта.
5620 DATA 0, 0, 1, 0, 2, 0, 8, 1, 9, 1, 10, 1, 11, 1, 16, 0,
          17, 0, 18, 0, 19, 0, 23, 1, 24, 1, 25, 1, 26, 1
5630 DATA 29, 0, 30, 0, 31, 0, 32, 0, 34, 1, 35, 1, 36, 1,
          37, 1, 38, 0, 39, 0, 40, 0, 0
5640 :REM Единичные перепады. Зависимость от длины ступеньки.
5650 DATA 0, 0, 1, 0, 2, 0, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1,
          9, 1, 10, 1,11, ,0, 12, 0, 13, 0, 14, 0, 15, 0
5660 DATA 16, 0, 17, 0, 18, 1, 19, 1, 20, 1, 21, 1, 22, 1,
          23, 1, 24, 0, 25, 0, 26, 0, 27, 0, 28, 0
5670 DATA 29, 1, 30, 1, 31, 1, 32, 1, 33, 0, 34, 0, 35, 0,
          36, 1, 37, 1, 38, 0, 39, 1, 40, 1, 0
```

КОМАНДЫ ЯЗЫКА БЕЙСИК

AUTO Автоматическая нумерация строк.

```
AUTO
AUTO 150, 20
AUTO , 5
```

BEEP Вызывает звук.

```
BEEP
```

CIRCLE Рисует эллипс или его часть.

```
CIRCLE (10, 12), 6
```

CLEAR Резервирует память под строковые переменные; обнуляет все переменные.

```
CLEAR
CLEAR 75
CLEAR 0
```

CLOAD Загружает программу с кассеты.

```
CLOAD
```

CLOSE Закрывает открытые файлы.

```
CLOSE
CLOSE 1, 2
```

CLS Очищает экран.

```
CLS
```

COLOR Задает цвет экрана.

```
COLOR 2, 6
```

CONT Продолжает выполнение программы после нажатия клавиши СТОП или выполнения команды STOP.

```
CONT
```

CSAVE Запись программы на кассету.

```
CSAVE
```

DATA Хранит данные считываемые командой READ.

```
DATA 1234, "ВАСЯ", "Москва"
```

DEFDBL Определяет переменные двойной точности.

```
DEFDBL W, A-D
```

DEFINT Определяет целые переменные.

```
DEFINT W, A-D
```

DEFSNG Определяет переменные одинарной точности.

```
DEFSNG W, A-D
```

DEFSTR Определяет строковые переменные.

```
DEFSTR W, A-D
```

DELETE Стирает строки программы.

DELETE 1000
DELETE -80
DELETE

DIM Задает размерность одного или нескольких массивов.

DIM A(6), B(11)
DIM QQ$(20, 30)
DIM K%(12, 13, 14)

DRAW Рисует объект, определяемый строкой.

DRAW "U5R7D6L8"

Перечень команд для оператора DRAW

U Вверх.

U10

D Вниз.

D5

R Вправо.

R7

L Влево.

L9

E Вверх и вправо.

E15

F Вниз и вправо.

F10

G Вниз и влево.

G16

H Вверх и влево.

H5

M Рисует линию от текущей точки до указанной в команде.

M16, 20

B Изображается только последняя точка.

B

A Вращение осей координат.

A0
A1
A2
A3

S Задание коэффициента масштабирования.

S8

C Определяет цвет линии.

C5

EDIT Включает режим редактирования.

```
EDIT 100
EDIT .
```

END Оканчивает выполнение программы.

```
END
```

ERROR Имитирует ошибку номер . . .

```
ERROR (1)
```

FILES Выводит на экран каталог диска.

```
FILES "A:*.BAS"
```

FOR . . . TO . . . STEP/NEXT Открывает циклическую программу.

```
FOR I=1 TO 8 (. . .) NEXT I
FOR C!=8 TO 16 STEP 4 (. . .) NEXT C!
```

GOSUB Передает управление подпрограмме.

```
GOSUB 9876
```

GOTO Передает управление указанной строке.

```
GOTO 654
```

IF . . . THEN . . . ELSE Проверяет условное выражение.

```
IF P=A THEN 200
IF V>M THEN 240 ELSE M=M+D
```

INPUT Вводит данные с клавиатуры.

```
INPUT A, B, C
INPUT C#
INPUT "ВАСЯ"; M$
```

INPUT # Считывает данные с диска.

```
INPUT #1, A, B, C
INPUT #6, B!
```

KILL Стирает файл с диска.

```
KILL "A:PROG.BAS"
```

LFILES Выводит на принтер каталог диска.

```
LFILES "A:*.BAS"
```

LINE Рисует линию или прямоугольник.

```
LINE (10, 20) - (200, 100), 6
LINE (10, 20) - (200, 100), 6, B
LINE (10, 20) - (200, 100), 6, F
```

LINE INPUT # Считывает строковую переменную с диска.

```
LINE INPUT #3, B$
```

LIST Выводит на экран текст программы.

```
LIST
LIST 100-1000
```

LLIST Выводит на принтер текст программы.

```
LLIST
LLIST 100-1000
```

LOAD Загружает программу с диска.

```
LOAD "PROG1.BAS"
LOAD "B:PROG2.BAS", R
```

LOCATE Устанавливает курсор на заданную позицию.

```
LOCATE 15, 2, 1
LOCATE 20, 5, 0
```

LPRINT Выводит данные на принтер.

```
LPRINT C, V, B
LPRINT "2+2="; 2+2
```

LPRINT TAB Вывод данных на принтер начиная с указанной позиции.

```
LPRINT TAB(25) "ВАСЯ"
```

LPRINT USING Вывод данных на принтер по указанному формату.

```
LPRINT USING "####,"; 1234
```

LUT Переопределение цветов.

```
LUT A(15)
```

MERGE Объединяет программу на диске с программой размещенной памяти.

```
MERGE "A:PROG2.BAS"
```

MOTOR Управляет включением мотора магнитофона.

```
MOTOR ON
MOTOR OFF
```

NAME Переименовывает файлы.

```
NAME "PROG1.BAS" AS "PROG2.BAS"
```

NEW Стирает программу из памяти; обнуляет все переменные.

```
NEW
```

ON ERROR GOTO Передает управление программе обработки ошибок.

```
ON ERROR GOTO 300
```

ON ERROR GOTO 0 Выключает программы обработки ошибок.

```
ON ERROR GOTO 0
```

ON . . . GOSUB Оператор множественного ветвления.

```
ON Y GOSUB 50, 60, 70, 80
```

ON . . . GOTO Оператор множественного ветвления.

```
ON Y GOTO 50, 60, 70, 80
```

OPEN Открывает файл.

```
OPEN "I", #1, "DATA1"
OPEN "O", #2, "DATA2"
```

PAINT Закрашивает область.

```
PAINT (100, 50), 6, 5
```

PCLS Очищает графический экран.

```
PCLS
```

PSET Зажигает точку на экране.

```
PSET (100, 50), 7
```

POKE N, M Помещает байт M в ячейку памяти N.

```
POKE 12345, 123
```

PRINT Выводит данные на экран.

```
PRINT A+B
PRINT "ВАСЯ"
```

PRINT # Записывает данные на диск.

```
PRINT #1, A, B, C
```

PRINT TAB Вывод данных на экран начиная с указанной позиции.

```
PRINT TAB(25) "ВАСЯ"
```

PRINT USING Вывод данных на экран по указанному формату.

Тип формата

\# Форматирует числа.

```
PRINT USING "#####"; 18.5
```

. Положение десятичной точки.

```
PRINT USING "###.##"; 18.5
```

, Печатает запятую после каждой третьей цифры.

```
PRINT USING "#####,"; 12345
```

** Заполняет пустые позиции звездочками.

```
PRINT USING "**####"; 55.6
```

$$ Плавающий знак доллара.

```
PRINT USING "$$####"; 55.6
```

**$ Плавающий знак доллара; заполняет пустые позиции звездочками.

```
PRINT USING "**$#.###"; 13.24
```

^ Экспоненциальный формат.

```
PRINT USING "###.#^^^^"; 12345678
```

\+ Печатает знак числа.

```
PRINT USING "+###"; 123
```

\- Печатает знак отрицательного числа.

```
PRINT USING "-###"; -12
```

! Печатает первый знак строковой переменной.

```
PRINT USING "!"; "ВАСЯ"
```

% пробелы % Поле строковой переменной; длина поля равна числу пробелов плюс 2.

```
PRINT USING "%   %"; "ВАСЯ"
```

READ Читает данные, записанные в операторе DATA.

```
READ C
READ V$
READ BB%, S#
```

RELOC Смещает начало координат.

```
RELOC (20, 30)
```

REM Коментарий (' - сокращенное обозначение REM).

```
REM Это коментарий
' Это тоже коментарий
```

RENUM Перенумеровывает строки программы.

```
RENUM 100, 10, 20
```

RESTORE Устанавливает указатель данных на первый символ в первом операторе DATA.

```
RESTORE
```

RESUME Оканчивает выполнение программы обработки ошибок.

```
RESUME
RESUME 40
RESUME NEXT
```

RETURN Возврат из подпрограммы к следующей после GOSUB строке.

```
RETURN
```

RUN Выполняет программу или ее часть.

```
RUN
RUN 150
```

SAVE Запись программы на диск.

```
SAVE "PROG1.BAS"
SAVE "PROG2.BAS", A
```

SCREEN Переключает графические страницы.

```
SCREEN 1, 2
```

STOP Прерывает выполнение программы.

```
STOP
```

SWAP Обменивает значения двух переменных одного типа.

```
SWAP A, B
```

SYSTEM Передача управления операционной системе.

```
SYSTEM
```

TROFF Включение трассировки.

```
TROFF
```

TRON Выключение трассировки.

```
TRON
```

WIDTH Устанавливает размер экрана.

```
WIDTH 20
```

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

Книги

Уэйт М., Ангермейер Дж. Операционная система CP/M: Пер. с англ. - М.: Радио и связь, 1986.

Пул Л. Работа на персональном компьютере: Пер. с англ. - М.: Мир, 1986.

Персональный компьютер в играх и задачах. - М.: Наука, 1988.

Дьяконов В.П. Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ. - М.: Наука, 1987.

Уэйт М., Прата С., Мартин Д. Язык Си: Пер. с англ. - М.: Мир, 1988.

Брябрин В.М. Программное обеспечение персональных ЭВМ. - 2-е изд. - М.: Наука, 1989.

Фодор Ж., Бонифас Д., Танги Ж. Операционные системы для IBM PC: Пер. с англ. - М.: Мир, 1989.

Журналы

В мире персональных компьютеров.

Радио.

Микропоцессорные средства и системы.

Информатика и образование.

В мире науки.

Научно-популярное издание

АХМАНОВ Сергей Александрович
ПЕРСИАНЦЕВ Игорь Георгиевич
РАХИМОВ Александр Турсунович
РОЙ Николай Николаевич
СКУРИХИН Александр Васильевич

ЗНАКОМЬТЕСЬ:
ПЕРСОНАЛЬНАЯ ЭВМ КОРВЕТ

Заведующий редакцией *Л.И.Гладнева*

Редакторы *В.Н.Задков, В.А.Григорова*.
Художник *Д.А.Крымов*
Художественный редактор *Т.Н.Кольченко*

Корректор *Е.Ю.Рычагова*

ИБ № 32848

Компьютерный набор. Подписано в печать с оригинал-макета 07.09.89.
Формат 60x90/16. Бумага книжно-журнальная. Гарнитура Таймс. Печать офсетная. Усл. печ. л. 15. Усл. кр.-отт. 15,25. Уч.-изд. л. 17,99.
Тираж 175 000 экз. Заказ № 805 . Цена.75 к.

Издательско-производственное и книготорговое объединение
"Наука". Главная редакция физико-математической литературы
117071 Москва В-71, Ленинский проспект, 15

Отпечатано в Четвертой типографии ИПКО "Наука"
630077 г. Новосибирск-77, ул. Станиславского, 25

NAUKA PUBLISHERS

Main Editorial Board for Literature
on Physics and Mathematics

15, Leninsky prospect, Moscow 117071, USSR

INTODUCTION TO PERSONAL COMPUTER KORVET

Sergei A. AKHMANOV, *Cand. Sc. (Phys. & Math.)*
Alexander T. RAKHIMOV, *D. Sc. (Phys. & Math.)*
Igor G. PERSIANTSEV, *D. Sc. (Phys. & Math.)*
Nikolaj N. ROY
Alexander V. SKURIKHIN

Institute of Nuclear Physics, Moscow State Vniversity

1989, 240 pages. ISBN 5-02-014207-7

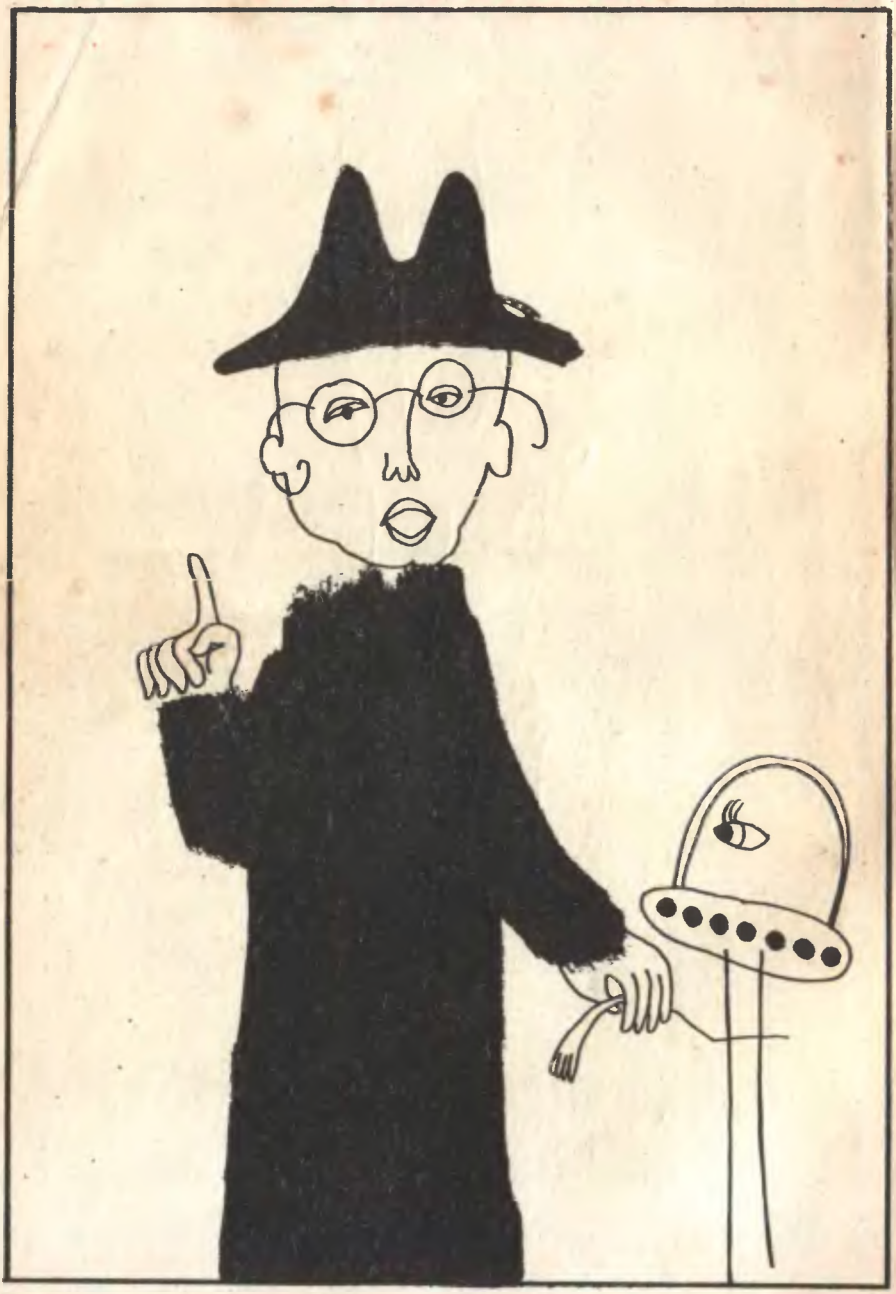
READERSHIP: Students of secondary and higher schools, scientists and all other peoples which start working with personal computers.

The BOOK: This book is the user's guide for the beginers. We start with the detailed information about preparing of the Korvet to work. We consider the main functions of the operating system CP/M-80. In this book we carefully describe the BASIC language. We use a lot of examples of the BASIC language programs as an illustration of the Korvet's possibilities.

CONTENTS: What is the personal computer. The possibilities of the PC Korvet. Installing PC Korvet. Magnetic tape and diskettes. Preparing diskettes for using. Operating system PC Korvet. Languages. Start with BASIC. Introduction to BASIC. More about BASIC. Program's editor. Arythmetic statements and functions. Loops and subroutines. Arrays. Logical statements. How to be an artist with BASIC. String variables. Print statement and graphycs. Memory addressing. Peripherial devices. Text redactors and data bases. Examples of programs.

75 коп.

ТЫ НАУЧИЛСЯ РА-



БОТАТЬ НА КОРВЕТЕ?