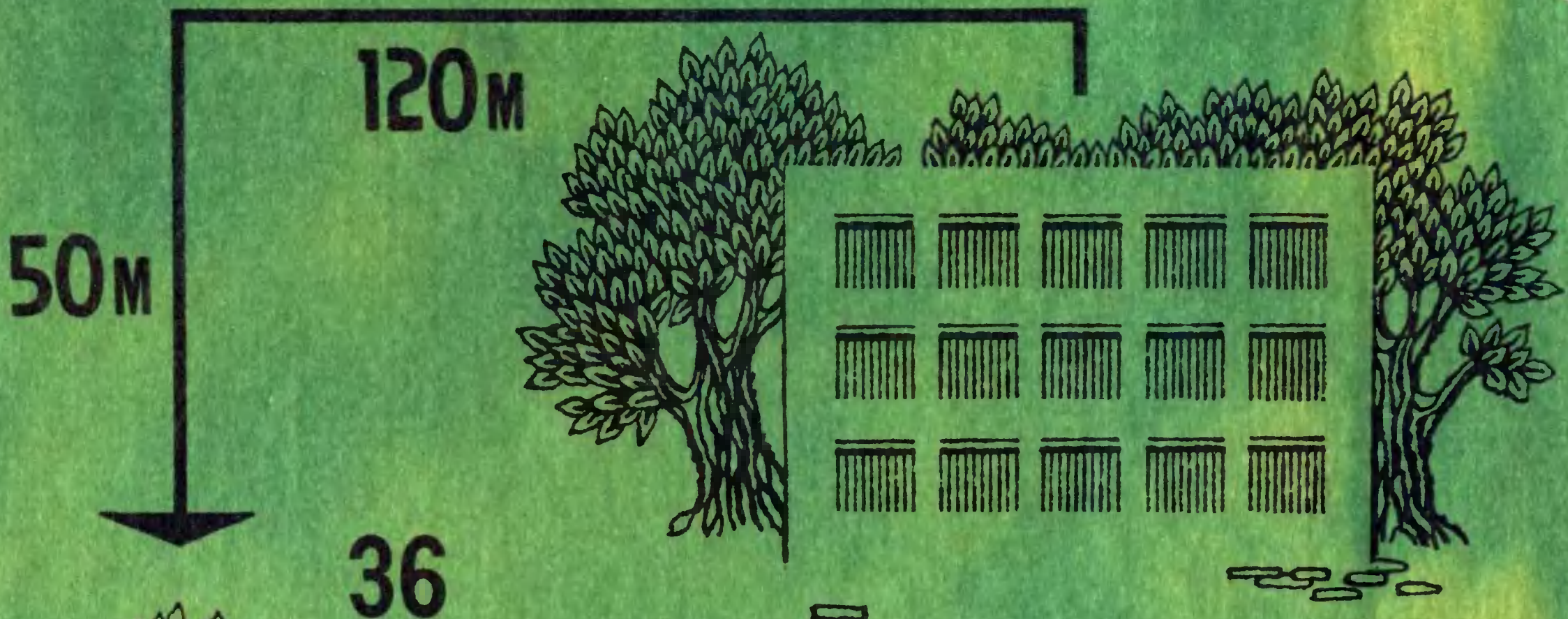


Б. Я. МАРТУЗАН



ПОДАРУЖИСЬ
СО МНОЮ,
КОМПЬЮТЕР!





38



СХЕМА АЛГОРИТМА

ВЫЙТИ ИЗ ШКОЛЫ, ПОВЕРНУТЬ НАЛЕВО,
ПРОЙТИ 120м

ПОВЕРНУТЬ НАЛЕВО
И ПРОЙТИ 50м

ПОСМОТРЕТЬ
НАЛЕВО

БЕГАЕТ ЛИ
СОБАКА БАРМАЛЕЙ?

НЕТ

ПЕРВЫЙ
ВЫХОД
ИЗ ЦИКЛА

ОБОЙТИ ДОМ 36 СЛЕВА
И НАПРАВИТЬСЯ К ДОМУ 38

ДА

ПОСМОТРЕТЬ
НАПРАВО

ИГРАЕТ ЛИ
БОРЬКА-ХУЛИГАН?

НЕТ

ВТОРОЙ
ВЫХОД
ИЗ ЦИКЛА

ОБОЙТИ ДОМ 36 СПРАВА
И НАПРАВИТЬСЯ К ДОМУ 38

ДА

Б.Я. МАРТУЗАН

ПОДРУЖИСЬ
СО МНОЮ,
КОМПЬЮТЕР!

КНИГА ДЛЯ УЧАЩИХСЯ

МОСКВА • ПРОСВЕЩЕНИЕ • 1991

**ББК 32.973
М29**

Рецензент:

учитель информатики школы № 57 Москвы В. Р. Лещинер

Мартузан Б. Я.

**М29 Подружись со мной, компьютер!: Кн. для учащихся.—
М.: Просвещение, 1991.—256 с.: ил.— ISBN 5-09-002704-8.**

В книге в занимательной форме излагаются основные понятия и примеры программирования. После подробного рассмотрения понятия алгоритма и краткого изложения устройства ЭВМ подробно описан язык программирования Бейсик, широко применяемый для работы на мини- и микроЭВМ.

Приводимых в книге сведений достаточно для практического освоения программирования на этом языке. Может быть интересна учащимся средних и старших классов.

**М 4306020000—554 225—90 [заказ по КБ-11]
103(03)—91**

ББК 32.973

ISBN 5-09-002704-8

© Мартузан Б. Я., 1991

О ЧЕМ РАССУЖДАЛИ ТОНЯ И ПЕТЯ, ПРОЧИТАВ ВСЕ ПОСЛЕДУЮЩЕЕ

— Ну и книжка! Надо же было додуматься описывать наши с тобой разговоры.

— А я тебе, Тоня, скажу, что книжка получилась ничего. . .

— Ну конечно! Ведь ты там такой умный, такой умный, прямо дрожь берет! А я? Понадобилась глупышка, так сразу: куда эта Соображалкина опять запропастилась?!

— Не преувеличивай, совсем не такой уж глупой ты там изображена. А что касается книги, то вряд ли кто станет утверждать, что по ней легко можно будет научиться программировать, но какое-то представление о программировании книжка должна дать. . .

— Конечно, должна, конечно, даст! Таким, как ты, кто уже программирует и даже других обучает. А тем, кто начинает с самого начала? Тем трудные места придется по несколько раз изучать.

— А кое-где простые вещи мусолятся и мусолятся. . .

— Скажу откровенно: такую привычку ты имеешь, так что в этом отношении все правильно, раз уж описывать наши беседы.

— Между прочим, мне кажется, что совсем необязательно долго корпеть над непонятным местом. Можно попробовать читать дальше, вдруг что-то напутано и то, что непонятно будет объяснено потом. И вообще в программировании хорошо понять начало можно только тогда, когда понимаешь конец, а конец, естественно, нельзя понять без начала. Поэтому, читая середину, есть смысл иногда полистать и начало. А если во время чтения встречаешь такое понятие программирования, которое уже успела забыть, то посмотри в конце книги предметный указатель, там ты узнаешь, на какой странице об этом понятии шла речь.

— А какой толк от всего этого чтения? Ведь фактически школьнику на электронной вычислительной машине удастся поработать так редко. Приведут его в единственную школу

города с ЭВМ и начинают: «Вот, милые дети, это наш замечательный класс э-вэ-эм, посмотрите хорошенько, но ничего не трогайте!»

— Пожалуй, ты близка к истине. Но скоро в учреждениях и школах машин будет много, и даже в магазине их можно будет купить.

— Нашелся прорицатель! Что же такое произойдет в ближайшее время?

— Вот увидишь!

— А сейчас? Кому нужны такие книжки, если свою программу сможешь пропустить только через несколько лет, и притом совсем не на тех машинах, о которых здесь идет речь? И вообще, разве можно научиться программировать только по книжкам?

— Мне трудно судить, так как я сам попал на машину с самого начала. Но все-таки мне кажется, что в программировании не самое важное сама машина. И даже не так уж важно знать, как на ней работать. Самое важное — сообразить, что вообще можно и нужно делать на ней.

— Собразить, что вообще надо делать. . .

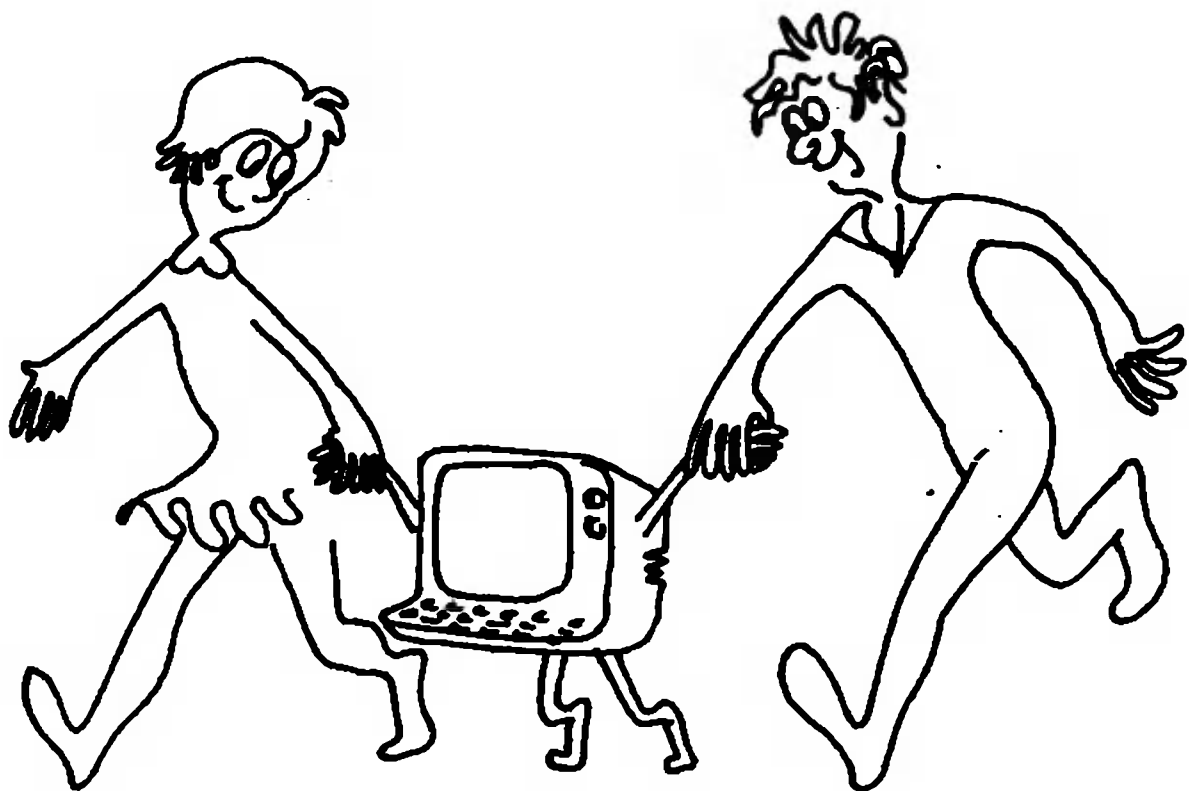
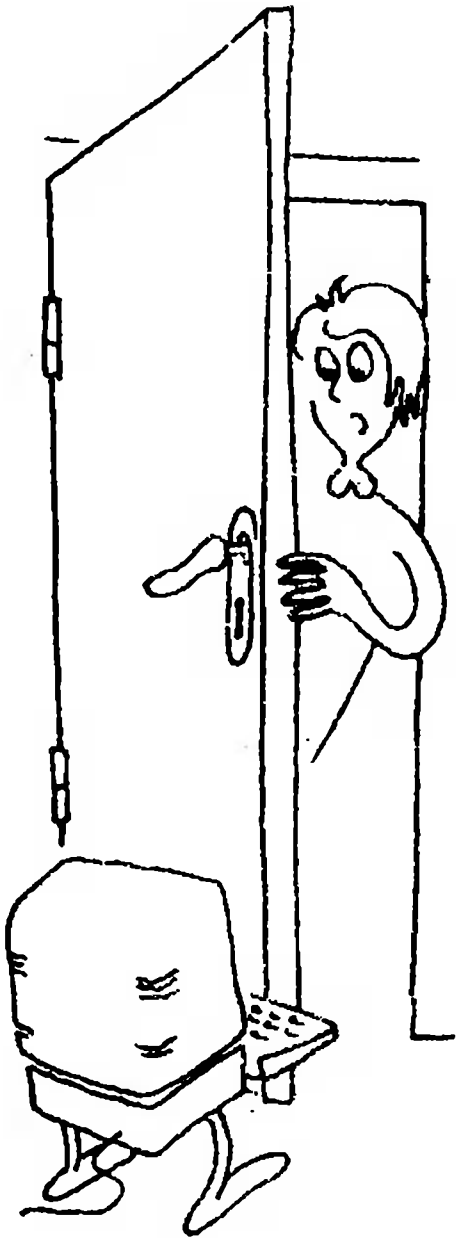
— Ну да! И для этого никакая машина не нужна. Сиди и соображай, пока не сообразишь.

— Вполне возможно. Мы же тоже не только на машине работали. Еще и дома обсуждали алгоритмы и изучали готовые программы.

— А программы для упражнений ты ведь писала спокойно сидя за столом.

— Вот именно. И после каждой написанной программы я понимала намного больше, чем после твоего объяснения, которое так старательно здесь воспроизведено.

— Ясное дело, понимала. Но не забудь, что до этого ты слышала мои пояснения, как бы ты их ни называла! И они тебе помогли, даже если ты сама этого не заметила. Также и



изучение этой книжки поможет и тому, кто на машину попасть пока не может. Самое важное, чего, не пропуская программу на ЭВМ, нельзя будет понять,— это то, как исправлять ошибки, которых немало получается, когда работаешь.

— Да... Эти ошибки... Существует целая наука по вылавливанию ошибок в программах, и то нельзя быть уверенным, что в большой программе их не осталось. Этими вопросами я немного интересовалась, так как, кажется, пойду на физмат. Могу тебе открыть такой секрет.

— Я-то еще не решил окончательно. Физмата я немножко боюсь. Что-нибудь попроще.

—...И полегче. А зачем тебе программирование, если легкой жизни хочется? Между прочим, это высказывание в книжке цитируется как твое.

— Я от него не отказываюсь. Но ведь избежать легкой жизни можно в любой профессии и всюду нужен будет компьютер. Ты же понимаешь, что он скоро станет обыденным орудием труда, первым помощником и советчиком каждого человека, в особенности взбалмошного и несобранного.

— Похоже, камушек в мой огород. А ты что, заманив меня на физмат, сам собираешься в писатели... Хочешь стать этим, ну... инженером человеческих душ?

— Да нет... Мне хочется программировать, но только я сомневаюсь, что смогу всю жизнь этим заниматься. А вдруг надоест!

— Нечего сомневаться, смело вперед на физмат!

1. КАК ВСЕ НАЧАЛОСЬ

Петя Бейсиков шел по улице и спокойно размышлял. И казалось, ничего не предвещало каких-либо неожиданностей.

И вдруг... Тоня Соображалкина из параллельного класса! Идет навстречу! Петя быстро взглянул направо. Стена дома! Обычная стена и ни одного входа в магазин или в подъезд. Налево и смотреть было нечего. Там была улица, ехали автомашины, а Петя еще с детского сада презирал людей, переходящих улицу в неположенном месте.

Делать нечего, надо продолжать идти. И Петя пошел! Конечно, чуть склонил голову, наморщил лоб и вообще напустил на себя вид человека, глубоко задумавшегося. Может быть, удастся проскочить... Не удалось.

— Чао, Бейсиков! — услышал Петя. Пришлось вздрогнуть, вроде бы от неожиданности:

— А-а! Это ты? Привет!



— Куда путь держишь?

— Я?.. Я вот... Да я в школу иду.

— Что с тобой случилось? В школу? Так поздно!

Не собирался Петя Бейсиков Тоню Соображалкину в свои планы посвящать. Соврать что ли? Петя смог бы и соврать, это ему приходилось делать и получалось ничего, вполне сносно. Однако чтобы врать, нужна была веская причина, а так, с ходу, серьезную причину найти не удалось.

— Да нет... Я вот... Я иду поработать в кабинете информатики. Туда шефы передали новую ЭВМ. Может, слыхала?

— Да-а?! А чего туда идти и время попусту тратить? Тебе что, на уроках мало задают или ты в математики собрался?

— Я еще не знаю, куда я собрался,— Петя действительно не знал,— а задают мне, я думаю, как всем. Но многие же ходят на плавание или там на занятия кружка филателистов, а я вот — на программирование. Ведь сейчас программировать должен уметь каждый человек.

Тут Тоня возмутилась.

— Вот загнул, Бейсиков! Если я не учусь в физматклассе, то я... то я не человек что ли?!

Такой бурной реакции Петя не ожидал, но спокойствия не потерял и стал объяснять:

— Ну, что ты! Ты тоже можешь научиться. Это несложно, ведь ты по математике хорошо соображаешь.

Петя Бейсиков это знал, потому что приходилось с Тоней встречаться на школьных математических олимпиадах и — что тут скрывать — она, как правило, выступала лучше его, хотя учителям и приходилось минут по десять тратить на успокоение Тони, которая то заявляла, что задача неправильная, то искала ручку по всему классу, то затевала возню со своим соседом по парте и т. п.

Чуть подумав, Петя добавил:

— Правда, мне кажется, что математику можно и не знать очень хорошо, чтобы научиться программировать.

— Э-э! — воскликнула Тоня, — знаю я это программирование. У соседки по лестничной клетке такие занятия уже были. Сначала все они ждали-гадали, что это такое, а потом

оказалось — такая скучища, что-то там пишут, что-то придумывают и все время эти разговоры о том, как это великолепно, как это грандиозно, как это потрясающе. . . Было бы минут десять поработать на настоящей ЭВМ. Ты тоже так занимаешься? — съехидничала Тоня.

— Совсем не так! Извини, но я работаю по-настоящему — составляю программу, которая действительно будет нужна. Конечно, если что-то путное получится.

— Ну да! Так уж я и поверила! Станут тебе доверять что-нибудь действительно нужное.

— Почему бы и нет?! Я же два года занимался в кружке юных программистов в Вычислительном центре и не одну программу составил. Были, конечно, и учебные, но были и совсем настоящие, которые вполне можно использовать. А если честно, то это занятие мне нравится и я просто не могу дожидаться возможности посидеть за дисплеем, так что очень обрадовался, когда руководитель этого кружка стал заведовать кабинетом информатики нашей школы. . .

Здесь Петя немножко запнулся, ибо из своего опыта уже знал, что в разговорах с девушками нельзя распространяться о том, что нравится, в особенности если это серьезно. Можно было нарваться на насмешки. Но на этот раз в голосе Тони был слышен явный интерес.

— Ого! Можно сказать, что ты хорошо устроился. На настоящей ЭВМ поработать — это, пожалуй, неплохо. . . Но нам, простым смертным, на это рассчитывать не приходится.

— Нет ничего проще! — воодушевился Петя. — Можешь пойти со мной. Поиграешь с машиной в игры, посмотришь, как я работаю, а если захочешь, я и тебя научу.

— Ха-ха-ха! Петя Бейсиков в роли педагога-экспериментатора. Тут любой компьютер со смеху умрет. Но посмотреть-то можно. Поехали!

Вот так Тоня Соображалкина оказалась в кабинете информатики задолго до официального начала изучения основ информатики и вычислительной техники.

2. ПЕРВОЕ ЗНАКОМСТВО

— Ну вот, смотри! Это и есть электронная вычислительная машина, или компьютер, называют ее СМ-4. Это такое же название, как для автомобилей «Москвич» или «Жигули». К сожалению, разных типов компьютеров куда больше, чем типов автомобилей, и притом если человек освоил один тип, то это не означает, что он сможет работать на другом.

КОМПЬЮТЕРЫ БЫВАЮТ ВСЯКИЕ

— Тогда какой же смысл браться за изучение, если не знаешь, на каком компьютере придется потом работать?

— Да нет! Это не так страшно. По существу, они все очень близки, и если освоен какой-то один компьютер, то другой освоить будет намного проще. Это почти как с автомобилем — если умеешь водить «Жигули», самосвал освоить намного легче: ведь правила уличного движения остаются теми же. А на новой ЭВМ придется освоить десяток-другой новых слов или пару новых понятий.

— Тогда ничего:

— Могу тебе рассказать и о других отличиях. Например, о возможностях компьютера. Один компьютер может сделать больше, другой — меньше.

— Это-то ясно. Чем машина больше, тем она лучше. Эта машина же намного меньше других. Я по телевизору видела совсем большие — целый зал занимают.

— Ты неверно рассуждаешь. Во-первых, бывают компьютеры еще меньше. Я имею в виду персональные компьютеры. Во-вторых, подумай вот о чем: у больших машин большая память — намного больше, чем тебе надо. И что тебе от этого? Быстродействие тоже у разных машин разное. Большие машины выполняют больше арифметических операций в секунду, например несколько миллионов. А если тебе надо выполнить всего сто тысяч операций? На большой машине затратишь одну десятую секунды, на персональном компьютере — всю секунду. Какая тебе разница, ждать секунду или десятую секунды?

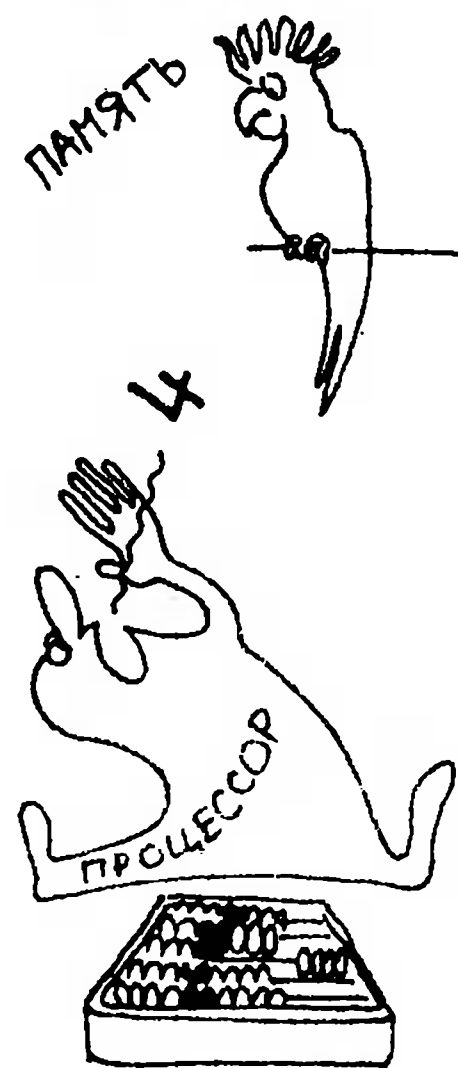
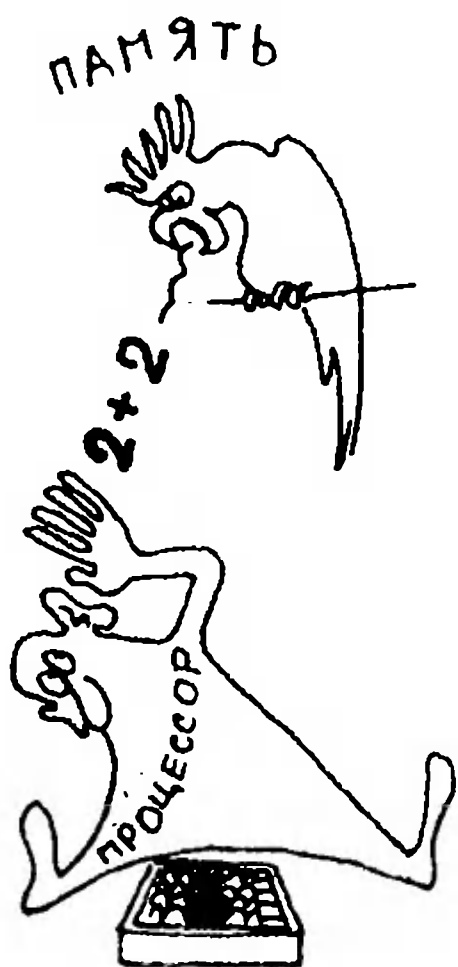
— Сто тысяч операций в секунду... Жуткое количество! Кому может понадобиться выполнить столько операций?

— Это еще мелочь. Я видел, что люди в Вычислительном центре считают часами. Правда, нужно еще учесть, что выполняются не только арифметические операции, но и другие. Например, пересылка данных с одного места на другое.

— Ну, про это потом. Скажи лучше, что где находится в этом компьютере?

ИЗ ЧЕГО СОСТОИТ КОМПЬЮТЕР!

— Ну... у этой машины, как, по-моему, и у всех остальных, основные составные части — это память и процессор. В памяти хранятся исходные данные, с которыми машина будет работать, и те результаты работы, которые еще могут понадобиться. А процессор — это устройство, которое с этими дан-



ными работает и эти результаты получает. Работать с данными — это означает выполнять арифметические или другие операции, например, как я уже говорил, пересылку данных. Данные приходится пересылать весьма часто. Скажем, при сложении двух чисел данными будут слагаемые, и их нужно будет переслать из памяти в процессор и, когда сумма получена, отправить ее в память для хранения. И еще! Сама программа тоже хранится в памяти и для выполнения должна быть переслана в процессор.

— И все это происходит в этих шкафах?

— Да. И посмотри, пожалуйста, сюда. Это тоже важное устройство. Здесь информация записывается на магнитные диски и считывается с них. И вот почему это так важно. Память машины — ну, та самая, из которой процессор получает данные и куда отправляет результаты, — называется оперативной памятью. Оперативная память всем пользователям машины нужна, и когда ты уйдешь домой, то твою программу очень скоро из памяти машины выбросят и она безвозвратно пропадет. Кроме того, когда вечером машину выключают, все, что хранилось в оперативной памяти, навсегда исчезает. И это, как ты понимаешь, очень плохо. Поэтому и изобрели память на магнитных дисках. Записанное на этих дисках сохранится и потом, когда ты их снимешь с машины и положишь на полку. Но зато запись данных на диск и считывание с него идут значительно медленнее, чем при работе с оперативной памятью.

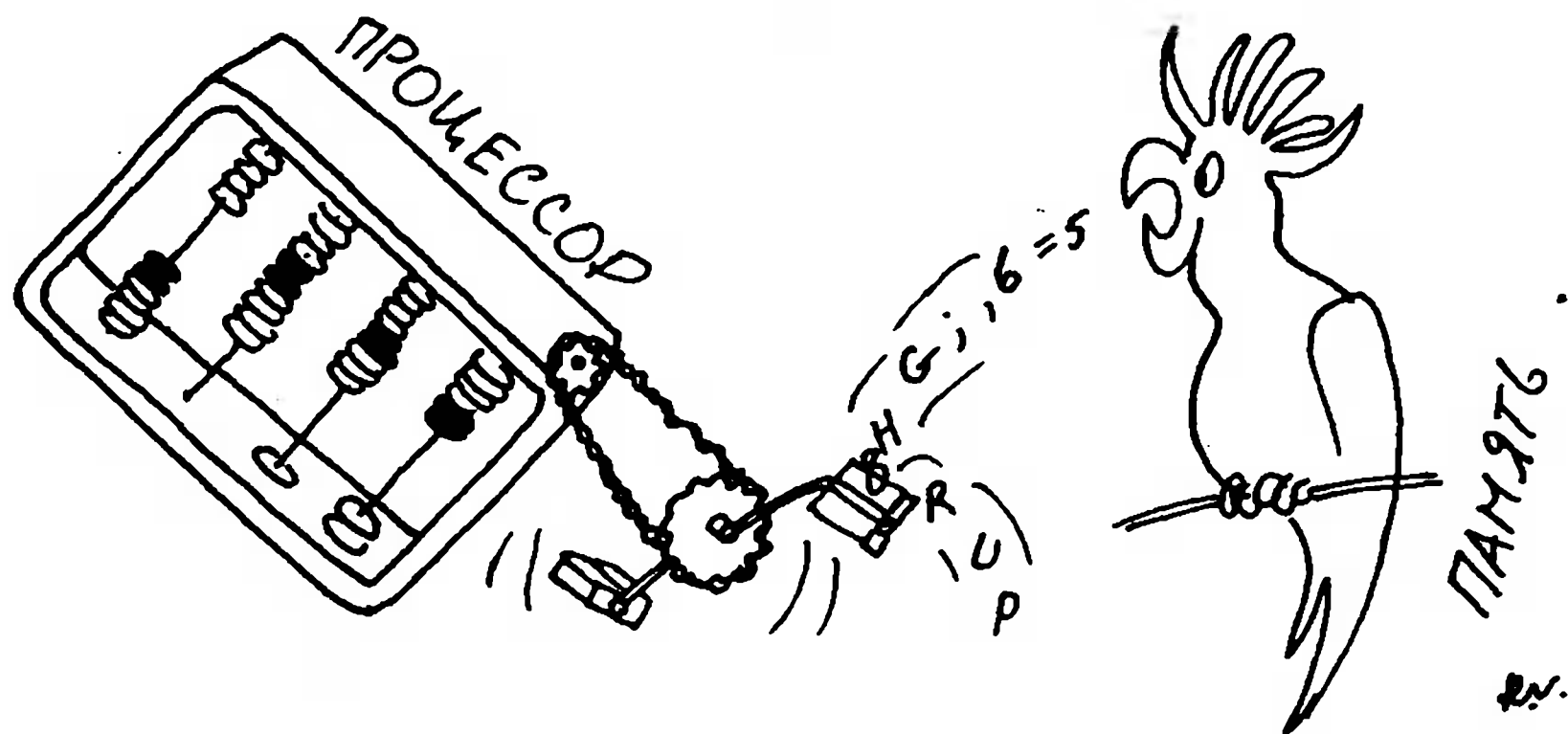
— Правильно ли будет считать, что запись на эти диски происходит таким же способом, как на магнитофонную ленту?

— Да, принцип тот же, только магнитный диск — это похожая на грампластинку твердая пластина, все время вращающаяся с бешеной скоростью. Все время потому, что нельзя предугадать, когда это ты вздумаешь записывать что-нибудь на диск. Если начать раскручивать только тогда, когда возникает необходимость что-либо записать, то можно поте-

Оперативная память



Магнитные диски



рять целую десятую долю секунды! А с бешеной скоростью диски тоже вращаются для того, чтобы можно было быстрее записывать и считывать информацию. На других машинах я видел и такие устройства, которые начинают вращать диски, когда необходимо на них записывать или наоборот с них что-то считывать. Это устройства на гибких дисках. Просто диски там значительно меньше и действительно гибки, хотя гнуть их не разрешается. Запись на этих дисках происходит намного медленнее, чем на наших.

— Здесь, наверное, опять можно повторить твоё рассуждение о скорости работы процессора — можешь ли ты ждать секунду или хочешь сделать работу за десятую долю секунды?

— Конечно, это все мелочи. А вот по-настоящему медленно происходит запись на магнитных лентах, как и считывание с них, что компьютеры тоже могут делать.

— Почему же нельзя обойтись одними дисками?

— Почему? Может быть, потому, что магнитная лента дешевле магнитного диска.

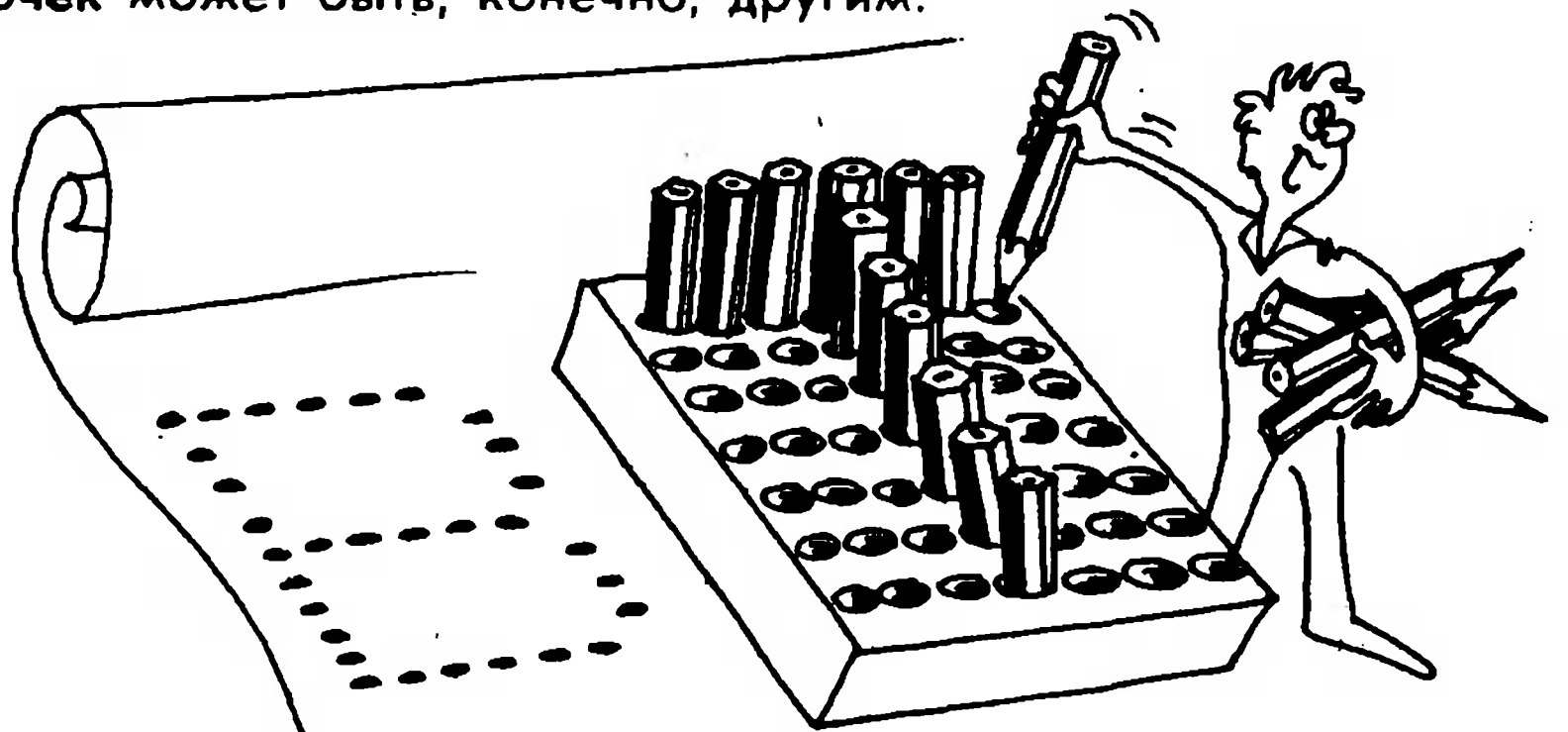
— Хорошо, оставим ленты в покое. Пока я ничего на них не собираюсь писать. А это что за устройство?

— Это печать. На ней можно печатать тексты программ и результаты их работы. Видишь, здесь что-то напечатано.

АБВГДЕЖЗИЙКЛМНОПРСТУФХЦШЩЬЪЭЮЯ1234567890

— Выглядит совсем неплохо. И буквы, и цифры составлены из симпатичненьких точек.

— Поэтому-то такие устройства печати называют мозаичными устройствами. Точки, из которых составлены буквы, цифры и специальные знаки, одним словом символы, на этой машине выбираются из 35 точек, расставленных в виде прямоугольника 7×5 . На других машинах количество точек может быть, конечно, другим.



Печатаем так!

ДИСПЛЕЙ — КОМПЬЮТЕРНОЕ ОКНО В МИР

— А уж этот телевизор наверняка самый главный?

— Да, эта штука очень важна. А называется она — дисплей. Впереди дисплея ты видишь клавиатуру ввода. Нажмешь клавишу клавиатуры, и на экране появится символ, тоже составленный из точек. Нажмешь другую — появится другой. Прямо как на пишущей машинке, только буквы печатаются не на бумаге, а появляются на экране. А самая главная клавиша — это клавиша ввода, видишь, здесь справа — на ней написано ВВОД. При нажатии этой клавиши все, что было на последней строке дисплея набрано, поступает в оперативную память машины, и машина начинает эту информацию изучать. Короче говоря, происходит ввод строки с дисплея. А до нажатия клавиши ввода машине все равно, что ты там набираешь.

— Непонятно. Машина же вычислительная, значит, работает с числами, а тут на клавиатуре буквы, и ты еще говоришь, что их можно вводить в машину. Как она может буквы понять?

— Ты, в общем, права. При нажатии клавиши буква появляется только на экране, а в память машины отправляется число, как у нас говорят — код буквы. Ясно, что одинаковым буквам соответствуют одинаковые числа-коды, а разным, конечно, разные.

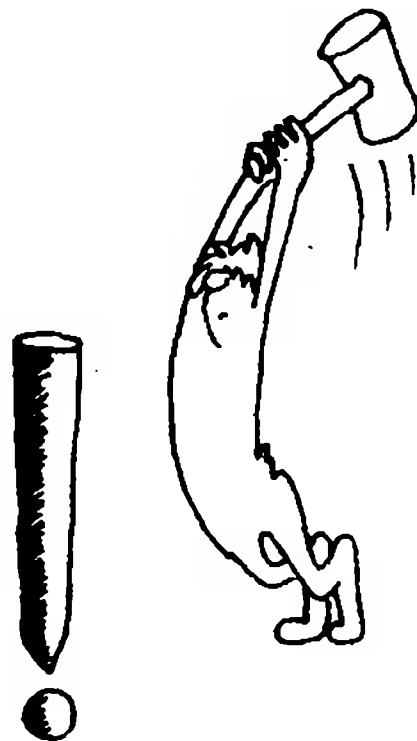
Ввод — это только половина той работы, которую на дисплее можно делать. Еще и машина сама может вывести на экран результаты расчета или какие-нибудь сообщения программисту. Хотя подожди, когда меня учили, то сильно ругали за такие слова — «машине все равно», «машина начинает изучать информацию», «машина сама может вывести» или другие выражения, приписывающие машине человеческие способности. Спору нет, правильнее было бы говорить: можно запрограммировать вывод информации на экран.

— Ну вот! Это же то же самое, и нечего придираться.

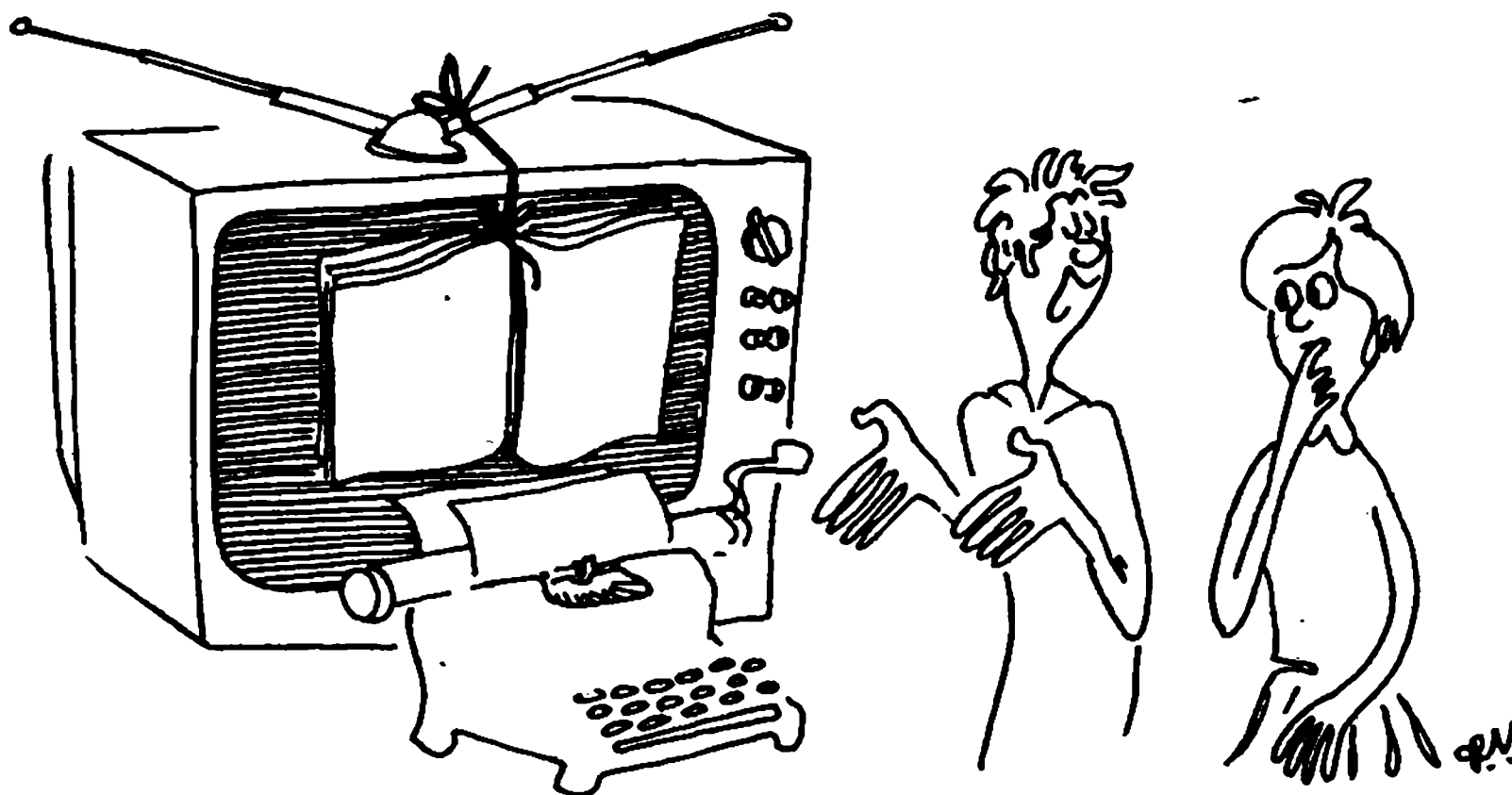
УМЕЕТ ЛИ ДУМАТЬ КОМПЬЮТЕР

— Важно понять, что компьютер сам ничего не решает. Он делает только то, что программист запрограммировал. Конечно, действия компьютера иногда выглядят вполне разумно, но сам — я говорю сам! — он может не больше, чем трактор или... или стиральная машина.

— Но ведь говорят же: трактор пашет, стиральная машина стирает или иногда не стирает.



Таким видится компьютер начинающему



— Да... Пожалуй, так действительно говорят, и действительно никто трактор не толкает; он едет сам. Но ты же не будешь отрицать, что всеми этими машинами управляет человек, на самом-то деле пашет тракторист — ему же дают премию за хорошую работу, а не трактору. То же самое и со стиральной машиной, и с компьютером. Как бы ни казалось, что компьютер что-то делает сам, фактически это делает программист, который когда-то составил программу. Мы сейчас эту программу пускаем, восторгаемся, как она здорово работает, а кто ее составил, мы и не знаем, и сам этот программист не знает про нас ничего. Но именно он определил, как будет сейчас работать машина. Ну, может быть, через сто лет машины и достигнут какой-нибудь самостоятельности...

— Ладно, я смотрю, ты в философию ударился. Давай ближе к делу.

— Садись тогда перед дисплеем и посмотри на клавиатуру. Клавиши с буквами здесь расположены совершенно так же, как на любой пишущей машинке. Надеюсь, ты в своей жизни видела пишущую машинку?

Клавиатура

AP2	; +	! "	2 "	3 #	4 \$	5 %	6 &	7 '	8 (	9)	0	=	—	ВОЗВ ШАГ	ПРРВ
ПЧТ	→	Й J	Ц U	У H	Е E	Н N	Г G	Ш [	Щ]	З Z	Х H	:	*	ПЕРЕВ СТРОКИ	ИСНЛ
СЛУЖ Н Д	↓	Ф F	Ы Y	В W	А A	П P	Р R	О O	Л L	Д D	Ж V	Э \	>	ПРДЧ	ОЗВР
ПЕРЕВ	↑	Я Q	Ч T	С S	М M	И I	Т T	Ь X	Б B	Ю @	<	/ ?	↑	ПОВТР	

— Машинку я, конечно, видела, но, как там буквы расположены, не помню. Я, откровенно говоря, как-то и не знала, что у всех пишущих машинок буквы расположены одинаково.

— Это уж точно! Но клавиатура дисплея все-таки посложнее клавиатуры пишущей машинки. Вот смотри, на каждой клавише нарисована и русская, и близкая по звучанию латинская буква, хотя попадаются и совсем разные. Видишь, русская А и латинская A, русская Б и латинская B, а вот здесь русская Ж и латинская V. Так получилось потому, что в латинском алфавите почему-то две буквы «вэ» — W и V. Буква W находится вместе с русской В. Можешь понажимать на клавиши, только учти, что русские буквы будут печататься только тогда, когда нажата еще и эта большая клавиша со стрелкой. Любая из двух — правая или левая. И еще: если напечатала неправильно, то нажми клавишу с буквами ИСКЛ здесь справа — и последняя напечатанная буква исчезнет. Еще раз нажмешь, еще одна буква исчезнет. И так, пока букв в строке не останется.

ТОНЯ УЖЕ НАЧИНАЕТ ДЕЙСТВОВАТЬ

— Да, клавиши нажимаются легко и приятно. Дай-ка наберу я какое-нибудь слово.

>ДУРАК_

— Странно, почему это, когда люди печатают «какое-нибудь слово», выходит именно «дурак».

— Да ну тебя! Не нравится, могу набрать вот такое:

>ДУРАКИНТЕЛЛИГЕНТ_

— Намного лучше!

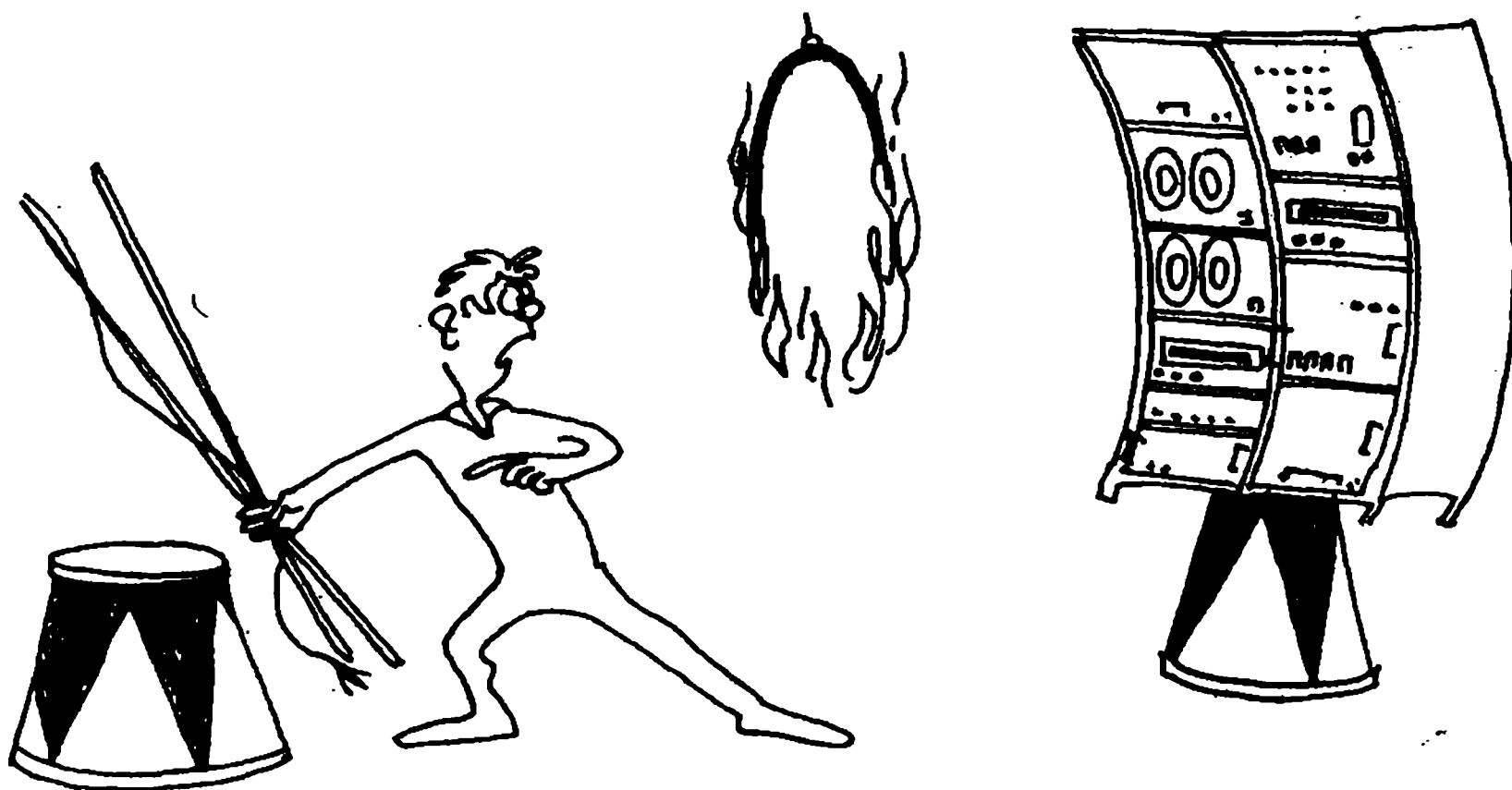
— Послушай, а я хотела, чтобы «интеллигент» печатался на следующей строке.

			↑
7	8	9	←
4	5	6	→
1	2	3	↓
ø	.	ВВОД	

— Одного желания мало, нужно еще и все правильно сделать. Посмотри на конец строки. Видишь там маленькую горизонтальную черточку чуть ниже строки? Это курсор. Там, где находится курсор, будет печататься символ при нажатии клавиши. Когда ты напечатала первое слово, курсор остался в конце слова, поэтому нечего удивляться, что второе слово напечаталось сразу после первого. Если хочешь попасть на следующую строку, можешь нажать клавишу ввода.

Курсор

Программирование, или дрессировка компьютеров, — дело безопасное



— Ой-ой! Я, наверное, что-то сломала!

MCR -- TASK NOT IN SYSTEM

— Не пугайся. Ты, нажимая эти клавиши, могла бы в худшем случае испортить информацию на диске, если умела бы записывать что-нибудь на диск. А саму машину таким путем сломать нельзя.

— А что же произошло?

— Видишь перед курсором знак «больше» — «>»? Появление этого знака в начале строки означает: машина ждет команды, что ей делать. Таких команд можно было бы сейчас дать несколько, но все они строго определены, и если хотя бы один символ неправилен, то и вся команда неправильна, и машина ее, так сказать, не понимает. Такой же знак был в начале той строки, которую ты ввела в машину.

— Ой, верно! Я его и не заметила!

— Так вот. Тот «дуракинтеллигент», что ты ввела в машину, не является, как легко догадаться, командой для машины. И машина просто вывела сообщение, что ввод неправильный.

ЧТО ЕЩЕ ВИДНО НА КЛАВИАТУРЕ

— Давай осваиваться с клавиатурой дальше. Запомни, где находятся точка и запятая, они на пишущих машинках расположены иначе. А вот специальные знаки чуть не забыл! Их на пишущих машинках нет вообще. Кроме знака процента «%», который все знают.

— А этот, рядом с ним, такой рогатый?

— Этот знак « X » называется знаком денежной единицы, но программисты его называют солнышком. Он им часто бывает нужен. А этот знак « H » — знак номера — в программировании нужен меньше.

— Так, так. Вроде все знаки видела, кроме вот этого « $\&$ ».

— Это, это... Я, кажется, забыл. Но мне такой знак еще не был нужен. Еще, наверно, ты не видела и другие знаки, например « $\backslash$ », который называется «обратная косая черта», так как есть еще просто «косая черта» — « $/$ » для обозначения деления. А для обозначения умножения используется «звездочка» — « $*$ ». Но вряд ли все эти мелочи тебя интересуют.

ТОНЕ ЗАХОТЕЛОСЬ ПРОГРАММИРОВАТЬ

— Ничего, ничего. Рассказывай. Сдается мне, что можно было бы это программирование и немножко поизучать. Совсем напрасным такой труд наверняка бы не был. Но только можно ли будет попасть на машину?

— Думаю, что можно. Но об этом надо поговорить с Алгоритмычем.

— С кем, с кем?

— С руководителем нашего кружка программистов, а также и кабинета информатики. Теперь, когда мы что-то о программировании знаем, он дает нам только руководящие указания, а работаем мы самостоятельно. А Алгоритмычем мы его прозвали потому, что он про алгоритмы только и говорит: «Главное — это алгоритм, придумайте алгоритм, не вижу алгоритма, хороший алгоритм, плохой алгоритм».

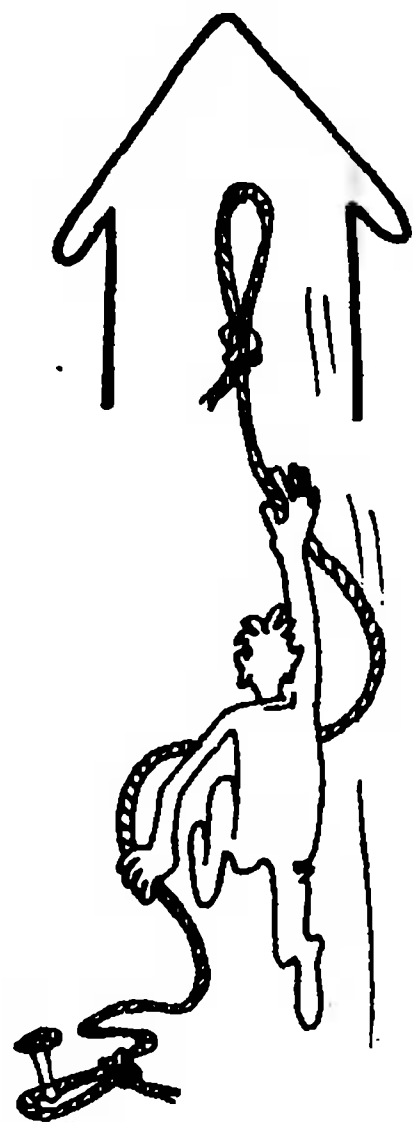
— Ну и поговорим с этим Алгоритмычем. Найдем его сегодня же, как только кончим. А сейчас продолжай.

— На сегодня, пожалуй, хватит. О клавиатуре уже почти все. Нужно только сказать о цифрах. Смотри, они расположены на верхнем ряду основной клавиатуры, но для того, чтобы их вводить, надо переключаться на русские буквы. Так как в основном работают с латинскими буквами и цифрами, то для того, чтобы не надо было все время переключаться туда-сюда, имеется еще одна маленькая клавиатура для цифр. Вот здесь, справа. Обрати внимание, что ноль выводится перечеркнутым — « $\emptyset$ ». Чтобы не путать с буквой «О». Ну вот и все.

— Э-э! Подожди! Поговорил и хватит! Да?! Надо же что-нибудь с машиной и поделать.

— Можешь поиграть с машиной или, правильнее сказать, с программой, в игры. Ну, например, погонять удава. Тебе как начинающей это может быть интересно, мне-то эти игры уже надоели.

— Не понимаю, как это игры могут надоесть. Вперед!



3. ТОНЯ СООБРАЖАЛКИНА УЗНАЕТ, ЧТО ТАКОЕ АЛГОРИТМ

В следующий раз Тоня и Петя встретились дома у Пети, так как он хотел рассказать, что такое программа и программирование и что такое язык программирования. А об этих вещах можно говорить и без самой вычислительной машины.

Переговоры с Алгоритмычем действительно были успешными. Для обучения Тони выделили дисплейное время — когда можно работать на ЭВМ, а учителем назначили Петю. При этом Алгоритмыч добавил, что речь может идти только о серьезном изучении, а не о каком-нибудь там баловстве, как некоторые себе представляют освоение программирования. Тоне это замечание показалось занудливо педагогическим, но при Алгоритмыче она промолчала.

Став учителем, Петя к разговору тщательно подготовился, все, что знал, обдумал, а знал он не так уж мало.

Начал он, правда, слишком заумно.

В ОСНОВЕ ПРОГРАММ ЛЕЖАТ АЛГОРИТМЫ

— Прежде чем дискутировать о вторичных понятиях программы и программирования, необходимо проанализировать фундаментальное понятие алгоритма.

— Чего, чего?

— Я говорю, вначале надо понять, что такое алгоритм.

— Извольте, господин учитель. Интересно будет узнать.

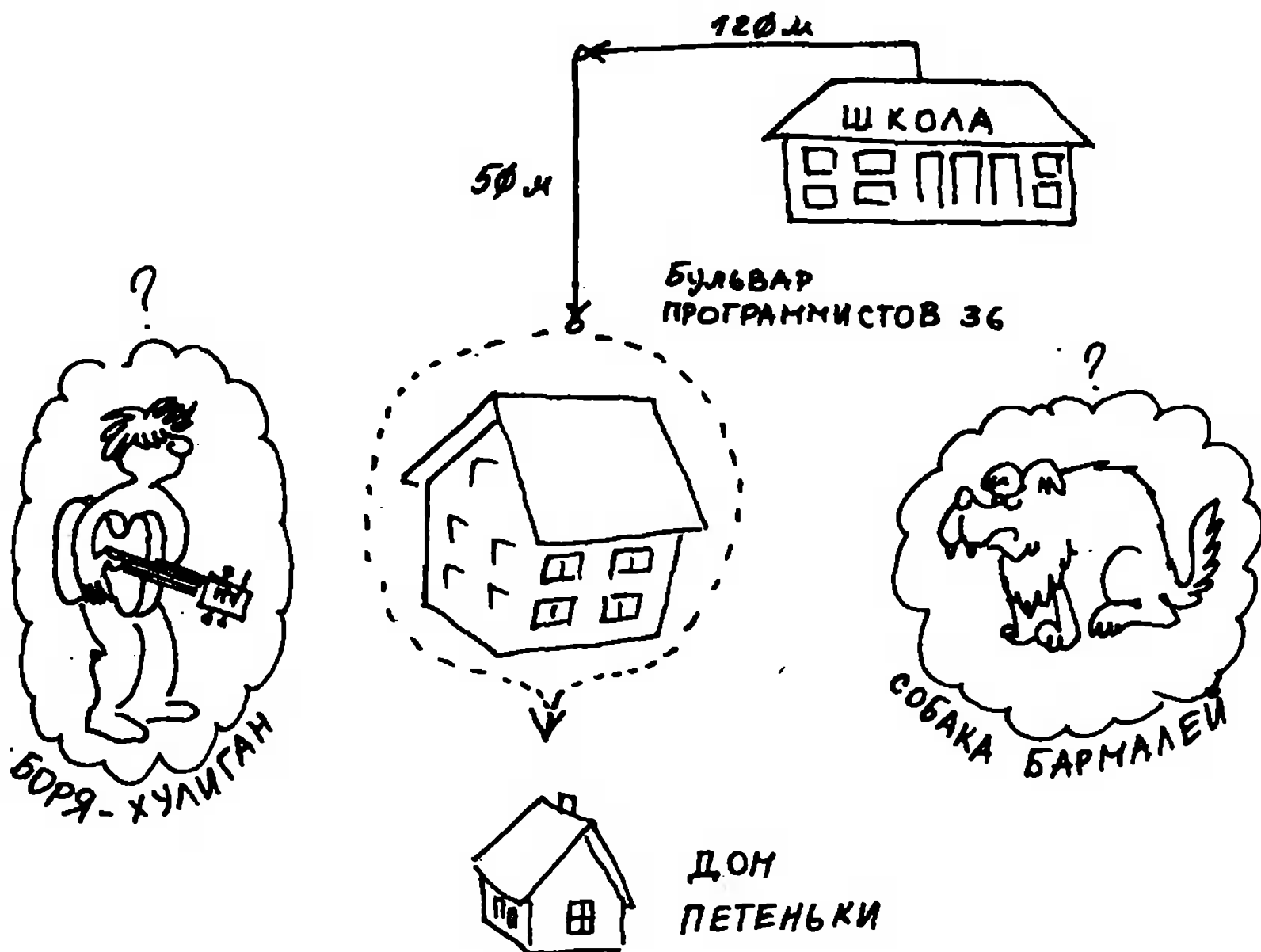
— Так, как на уроке требуют, я это понятие объяснить не могу. А если просто, то это набор правил, которым надо следовать, чтобы что-то сделать. Или, может быть, так... Алгоритм — это изложение того, что нужно делать, чтобы что-то заранее известное сделать. Но изложение это должно быть очень строгим и четким. Каждый раз, когда что-то делается, нужно предусмотреть все, что может получиться и что делать дальше в зависимости от того, что получилось. Не очень понятно?

— Почему же? Иди туда, не знаю куда, сделай то, не знаю что. Очень даже понятно получилось.

— Хм-м-м. Разберем, может быть, пример алгоритма — алгоритма возвращения первоклассника Петеньки из школы домой?

— Тебя, что ли?

— Неважно. Этот алгоритм получился бы очень сложным,

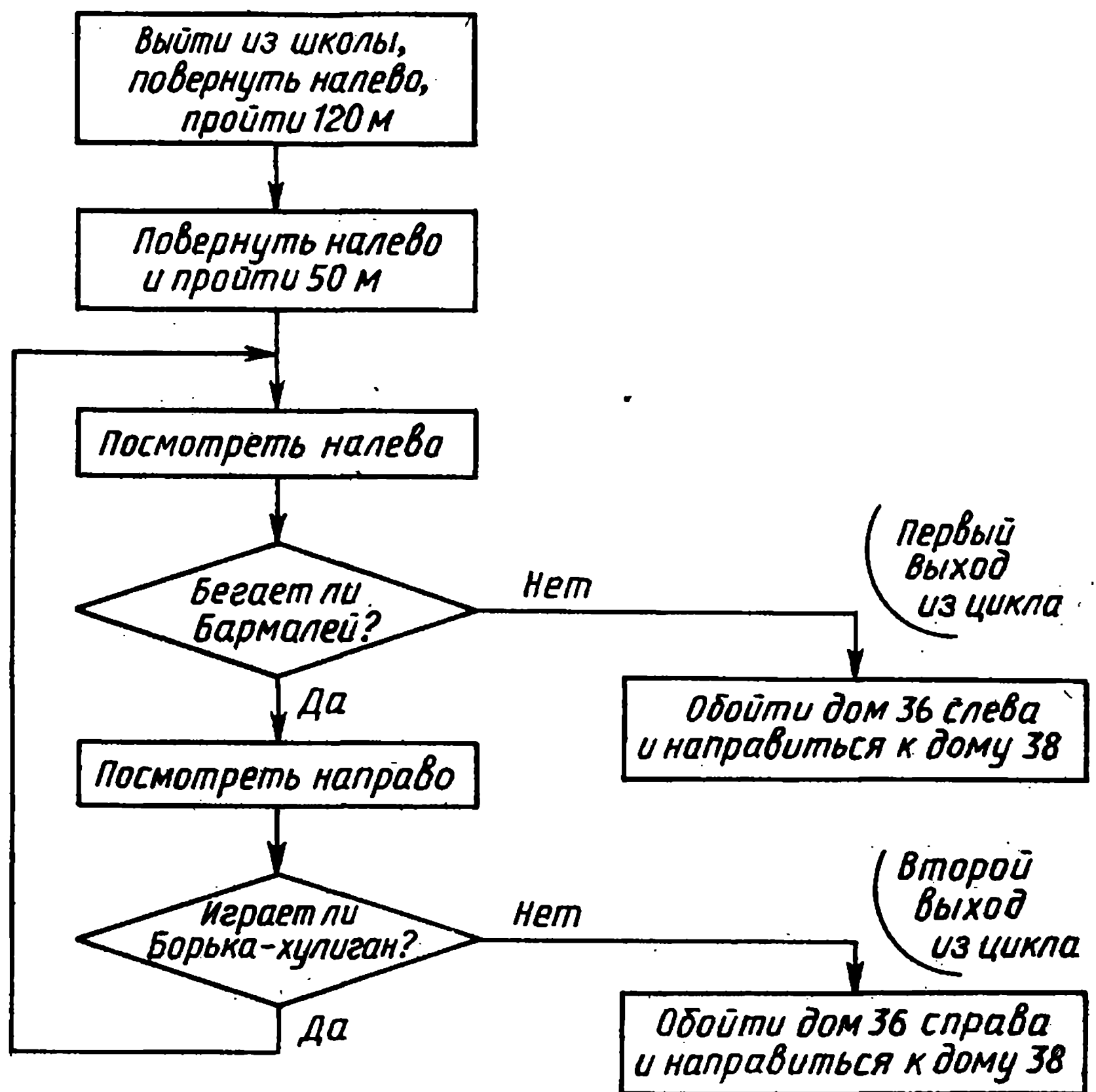


если его рассматривать подробно, а если так, в общих чертах, для понимания, то изложить можно легко. Кратчайший путь Петеньки домой по адресу бульвар Программистов, 38 изображен на этом рисунке. Вся сложность в том, что дом номер 36 нельзя обходить слева, если там бегают пес Бармалей, которого хозяева часто оставляют без присмотра. Бармалей по причинам, которые обсуждать нет необходимости, имеет зуб на Петеньку и попросту его не пропустит. Итак, если Бармалей бегают, то дом 36 нужно обходить справа. Но тут тоже могут возникнуть сложности: у правого подъезда часто играет задира и мучитель Борька-хулиган. Это еще хуже Бармалея.

АЛГОРИТМЫ ИЗОБРАЖАЮТ В ВИДЕ СХЕМ

— То, что я рассказал словами, можно изобразить схематически. Это будет называться схемой алгоритма. Посмотри на этот рисунок. На схеме в прямоугольных клеточках-блоках описаны действия Петеньки, не требующие никаких рассуждений, никакого решения, никакого выбора, а в ромбических клеточках-блоках отражены вопросы, на которые Петенька должен ответить «Да» или «Нет», исходя из информации, получаемой по ходу действия. Как видишь, имеются два таких ромбика с вопросами «Бегают ли Бармалей?» и «Играет ли Борька-хулиган?».

Схема алгоритма



— Да-а, схема интересная, в особенности Борька-хулиган меня впечатляет. Но скажи, правильно ли я понимаю, что если и Бармалей бегают, и Борька-хулиган играет, то Петенька так и будет крутить головой то вправо, то влево.

ВАЖНОЕ ПОНЯТИЕ — «ЦИКЛ»

— Именно так. Он будет крутить головой до тех пор, пока на один из этих вопросов можно будет ответить «Нет» либо пока не упадет от изнеможения. А по-научному можно сказать, что в алгоритме имеется цикл. Судя по вопросу, ты его уже заметила. Цикл получается потому, что если Бармалей бегают и Борька-хулиган играет, то стрелка от последнего ромбика снизу возвращается наверх и попадает в блок «Посмотри налево». Цикл имеет два выхода по условию. Вот здесь на схеме я их отметил. Других выходов из цикла нет,

Цикл

следовательно, другим образом цикл прекратиться не может. Вот и все.

— Хорошо. Излагаешь ты не бог весть какие премудрости, но привыкнуть надо. И еще вот что. Когда мне удастся решить задачку по математике, я иногда ее «проразмысливаю»: а что будет, если это условие изменить, а что, если этот угол не прямой, как в задаче, а острый, а что если... и т. д. Давай «проразмыслим» и твой алгоритм. Подумаем, так сказать, диалектически.

— Ну?

— Например, если Петеньке пришлось бы петлять — делать всякие повороты, то твой алгоритм существенно не изменился бы: ты бы только добавил сколько надо прямоугольных клеток, и все?

— Да, конечно. В каждой клетке было бы написано, куда повернуть.

— А если длина пути меняется?

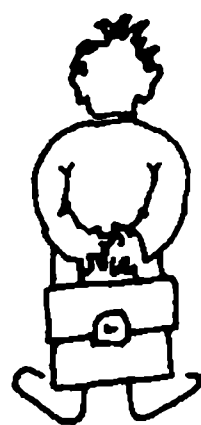
— Тогда это совсем просто! Мы должны только поменять цифры в клеточках схемы алгоритма.

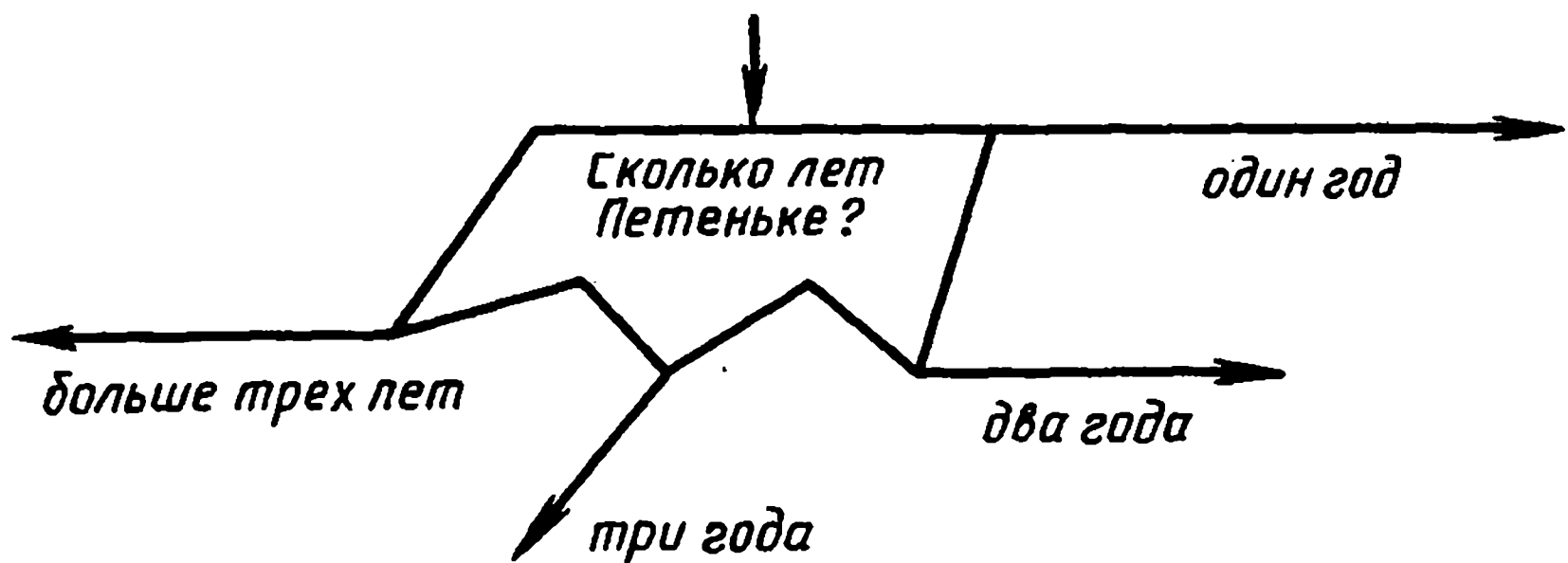
— Дальше. Если бы у Петеньки была бы возможность пойти не только направо или налево, но и прямо и там он мог бы встретиться, скажем, с дворничихой, которая опять же по некоторым причинам могла бы Петеньку начать перевоспитывать, чего он просто не выносит, то что делать в этом случае?

— В этом случае добавились бы два блока: прямоугольный — «Посмотри прямо» и ромб с проверкой условия — «Работает ли дворничиха?». Вот здесь внизу, сразу после блока с Борькой-хулиганом.

— Я так и думала. Но скажи мне вот что. Обязательно ли, чтобы из блоков с проверкой условия, как ты их называешь, выходили бы именно две стрелки: одна для ответа «Да», другая для ответа «Нет»? А что будет, если задавать вопрос, на который возможно дать больше чем два ответа, и совсем не такие простые, как «Да» или «Нет»? Такой вопрос может быть в алгоритме или нет? Ну, скажем, если я спрашиваю «Сколько тебе лет?» или «Куда направился, Петенька?». Понимаешь, мне кажется, что в жизни намного чаще встречаются именно такие вопросы, а не такие простые на которые можно ответить «Да» или «Нет».

— Так... Минуточку... Глубоко копаешь... Надо чуть подумать... Чтобы получился алгоритм, очень важна ясность и точность. Это я знаю. Конечно, можно себе представить такое действие, как «Узнать, сколько лет гражданину Петеньке». Однако это не будет проверкой условия и в схеме алгоритма может быть отражено обычным прямоугольным блоком. А если придумать звездообразные блоки вместо ромбических? Вот такие:

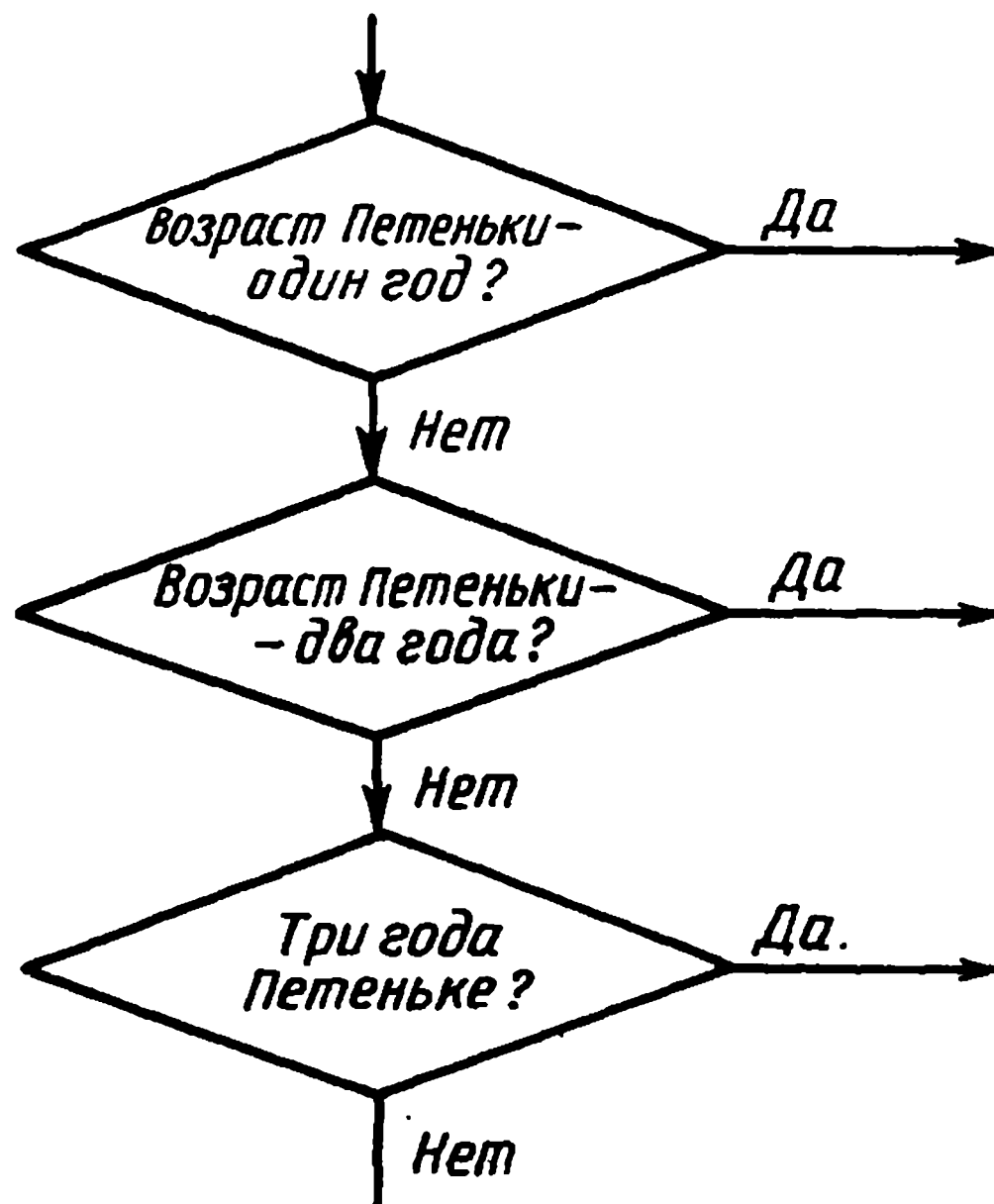


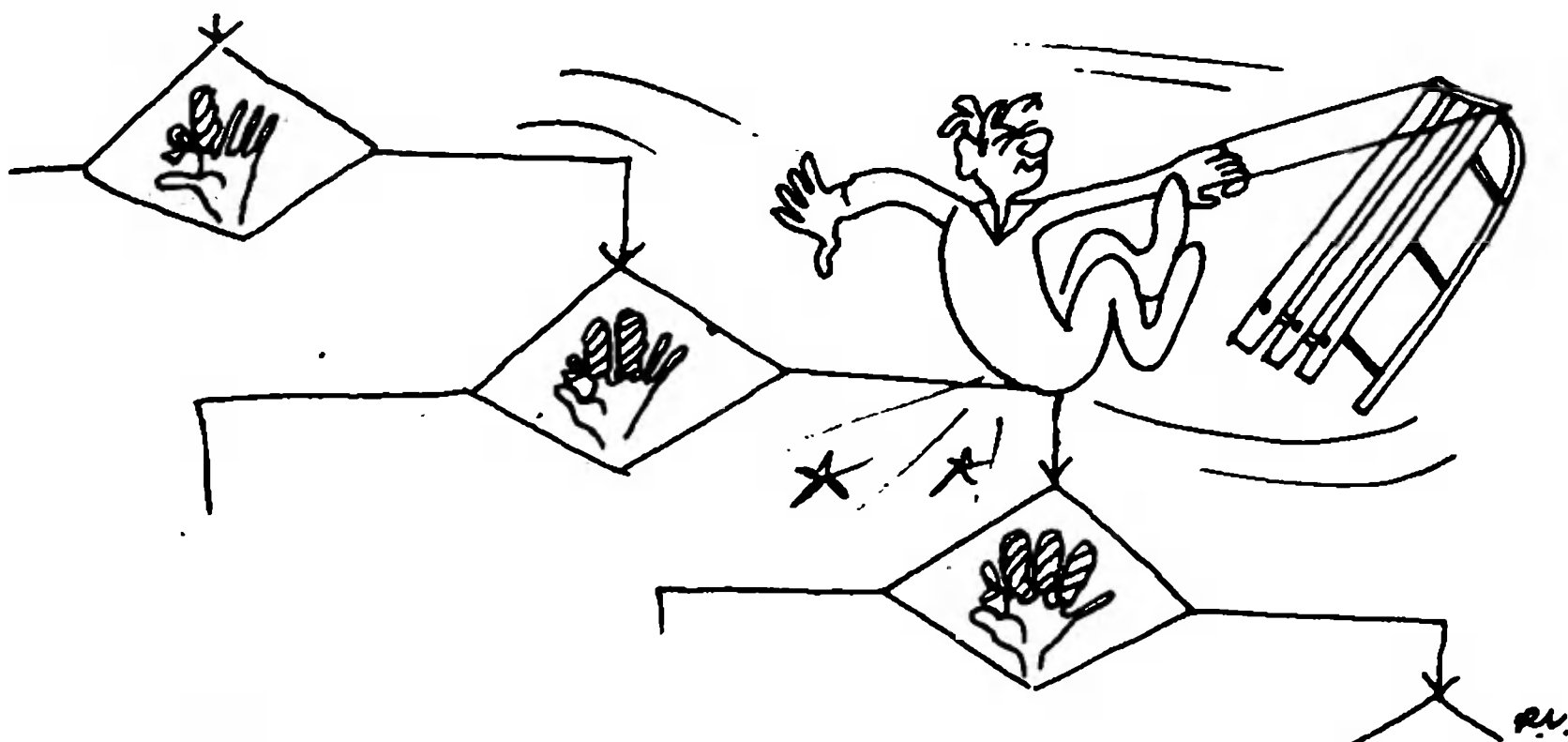


Такой блок изображал бы проверку количества лет и посылал бы на разные места схемы в зависимости от результатов проверки. . .

— Вот, вот! Я как раз о чем-то таком думала.

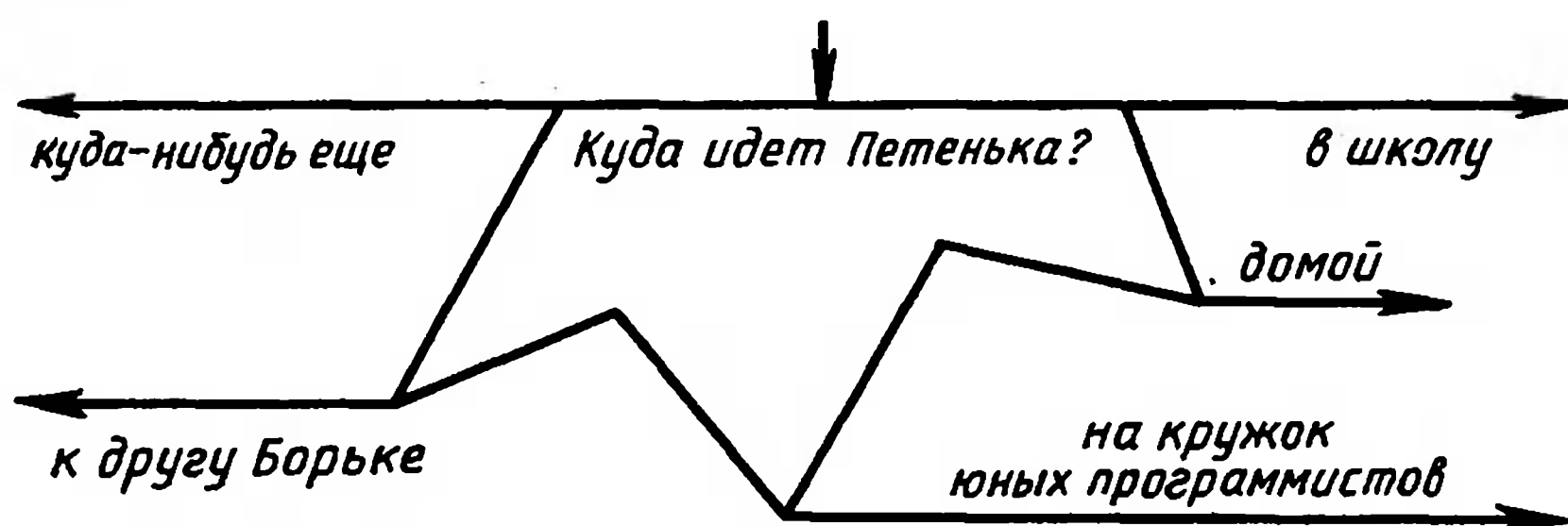
— Мы, конечно, такие звезды рисовать можем, никто нам это не запретит, классного руководителя здесь нет. Важно лишь, чтобы все было точно и ясно. Поэтому я, например, написал у крайней стрелочки слева «Больше трех лет», а не просто «Четыре года». Иначе не было бы ясно, что делать, если при проверке выясняется, что гражданину Петеньке 5 или даже 7 лет. Ха! Мне все ясно! То же самое можно изобразить несколькими ромбическими клетками. Смотри! Разве так сложнее?





— Может быть и нет, но зато больше рисовать.

— И твой второй вопрос «Куда идешь, Петенька?», рассмотренный отдельно, не имеет должной четкости. В алгоритме такой вопрос можно задавать только тогда, если хорошо известно, куда вообще Петенька может направляться. Тогда мог бы получиться, например, такой звездообразный блок:



И в этом случае то же самое можно изобразить последовательностью ромбических клеток! И в этих клетках проверялись бы условия типа «Идет ли Петенька домой?», «В школу?», «На кружок?» и т. д., перечисляя все возможные варианты. На все эти вопросы уже можно ответить «Да» или «Нет». Получается не так уж плохо: даже довольно сложные действия можно описать вопросами, на которые можно ответить «Да» или «Нет». В алгоритмах, как это ты скоро поймешь, могут быть только такие вопросы.

— Только такие?! А ведь тогда придется мучиться и придумывать эти новые вопросы.

— Я был не совсем точен. Сложные вопросы со множеством возможных ответов не должны быть в тех алгоритмах, о которых я тебе буду рассказывать и которые ты будешь программировать, когда дорастешь. А вот вырастешь совсем большая, кто-нибудь тебе расскажет, как работать с такими,

более сложными алгоритмами, в которых нужно выбрать одну из нескольких возможностей.

— Понятно. Теперь скажи, почему ты в самом начале говорил, что настоящий алгоритм движения Петеньки домой намного сложнее?

ВВЕДЕНИЕ В РОБОТОТЕХНИКУ

— А как же! Представь себе робота, который вместо Петеньки ходил бы в школу, ну и, конечно, из школы домой. . .

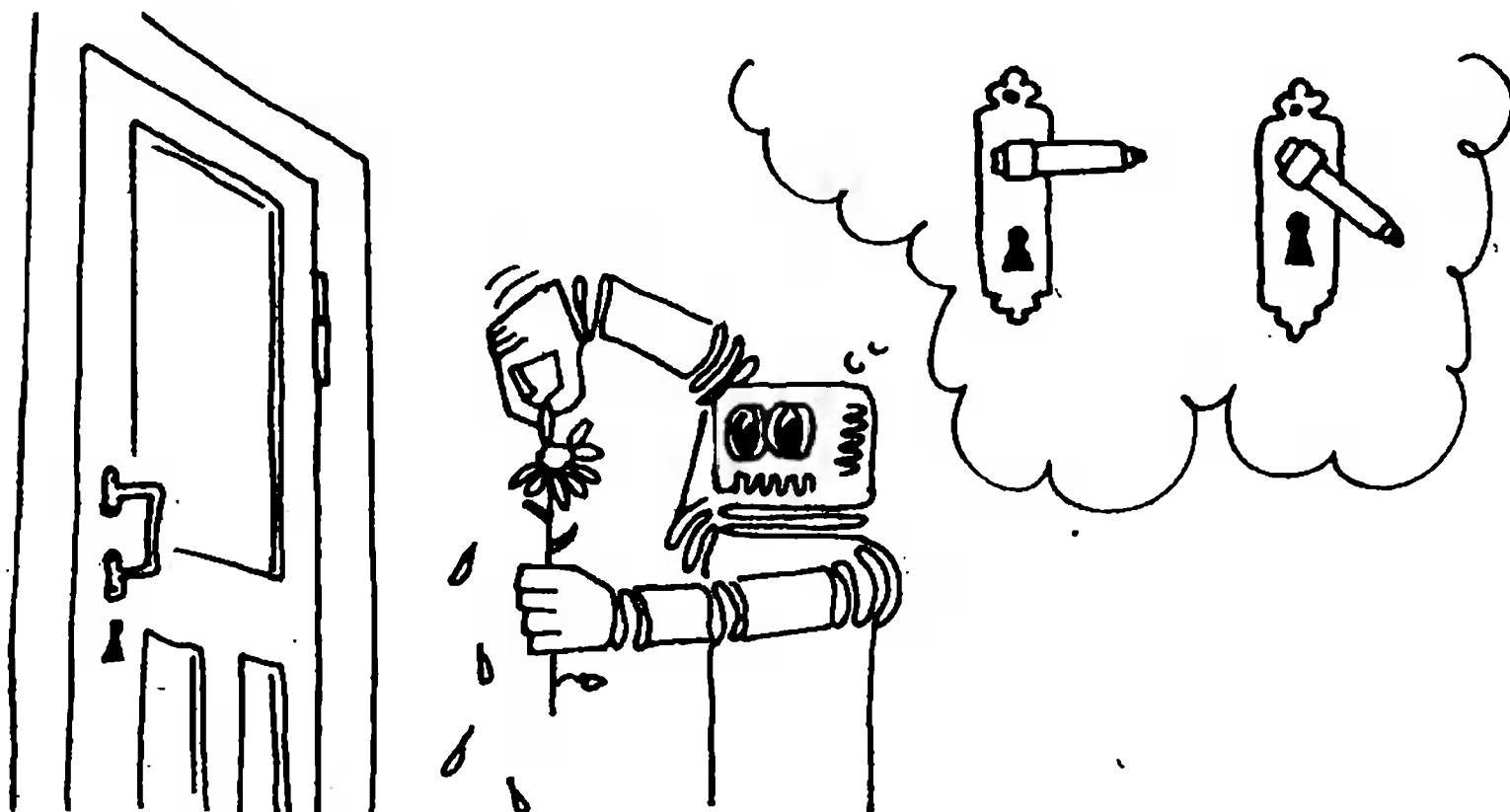
— О-хо-хо! Вот такого-то робота я бы хотела!

— И был бы этот робот совсем похож на Петеньку, даже в его сложных отношениях с Бармалеем и Борькой-хулиганом. Что он должен был бы уметь? Если говорить о приходе из школы домой, то перво-наперво он должен был бы уметь выходить из школы. А что означает выходить из школы? Самое первое — открыть дверь, т. е. найти дверную ручку, нажать ее, потом дернуть дверь с нужной силой, раскрыть ее, насколько нужно, потом протиснуться в образовавшуюся щель и закрыть дверь, притом достаточно тихо, чтобы не вступить в конфликт с представителями коллектива педагогов.

Все это надо делать, постоянно проверяя свои действия: достаточно ли нажал, не слишком ли сильно или, наоборот, слабо дергаешь, достаточно ли раскрыл. При этом робот должен уметь реагировать на различные аварийные случаи.

— Какие же здесь могут быть аварии?

— Просто так называются отклонения от заранее предусмотренного хода событий. Например, эти двери оказались заперты и всех детей и роботов выпускают через боковые



Нажать?.. Не на-
жать?.. Нажать?..

двери, или мальчишки отвинтили дверную ручку, или такая неожиданная ситуация, когда двери уже открыты или приоткрыты. Все эти ситуации в программе для робота должны были бы быть объяснены четко и недвусмысленно, с ясными указаниями, что делать в каждом конкретном случае. Мы поговорили только об открытии дверей, а это всего лишь начало пути, полного неожиданностей и опасностей. И наконец, каждое действие, о котором я говорил, тоже должно быть разделено на более мелкие действия, которые опять могут быть совсем непростыми.

— Да-а... Становится ясно, почему до сих пор нет роботов, ходящих в школу вместо ребят. А жаль...

— Сейчас я предложу тебе несколько упражнений...

— Чего? Упражнений еще только не хватало!

— Э-э... Нет, не упражнений, а тем для размышления.

— Тогда ладно, тогда предлагай.

— Можешь ли ты придумать следующие алгоритмы:

1. Алгоритм действий ученика, которого вызвали к доске за несколько минут до звонка, и, так как он урока не знает, ему надо протянуть время до звонка пустыми разговорами. По-видимому, он может ограничиться только проверкой условия, прозвенел ли звонок, а что говорит учитель, проверять необязательно.
2. Алгоритм действий ученика утром, начиная с момента входа в дверь школы и кончая входом в дверь класса.
3. Алгоритм действий ученика утром, начиная с момента входа в класс и кончая звонком на уроки. Рассмотреть также и случай, когда ученик опоздал и ему приходится входить в класс позже учителя.
4. Алгоритм определения отличника, неуспевающего, «хорошиста».
5. Алгоритм действий учителя, обнаружившего, что его воспитанники разбили окно.

4. ЕЩЕ ОДИН АЛГОРИТМ

Когда на следующий день Тоня и Петя опять встретились, чтобы продолжить занятия по программированию, Петя предложил:

— Сегодня рассмотрим еще один алгоритм.

— Слушай, может быть, хватит алгоритмов и общих рассуждений, давай поближе к ЭВМ и как на них работать.

— Мм-м... Я все-таки думаю, что этот алгоритм нужно посмотреть. Там имеется один важный, на мой взгляд, нюанс. Я быстро.

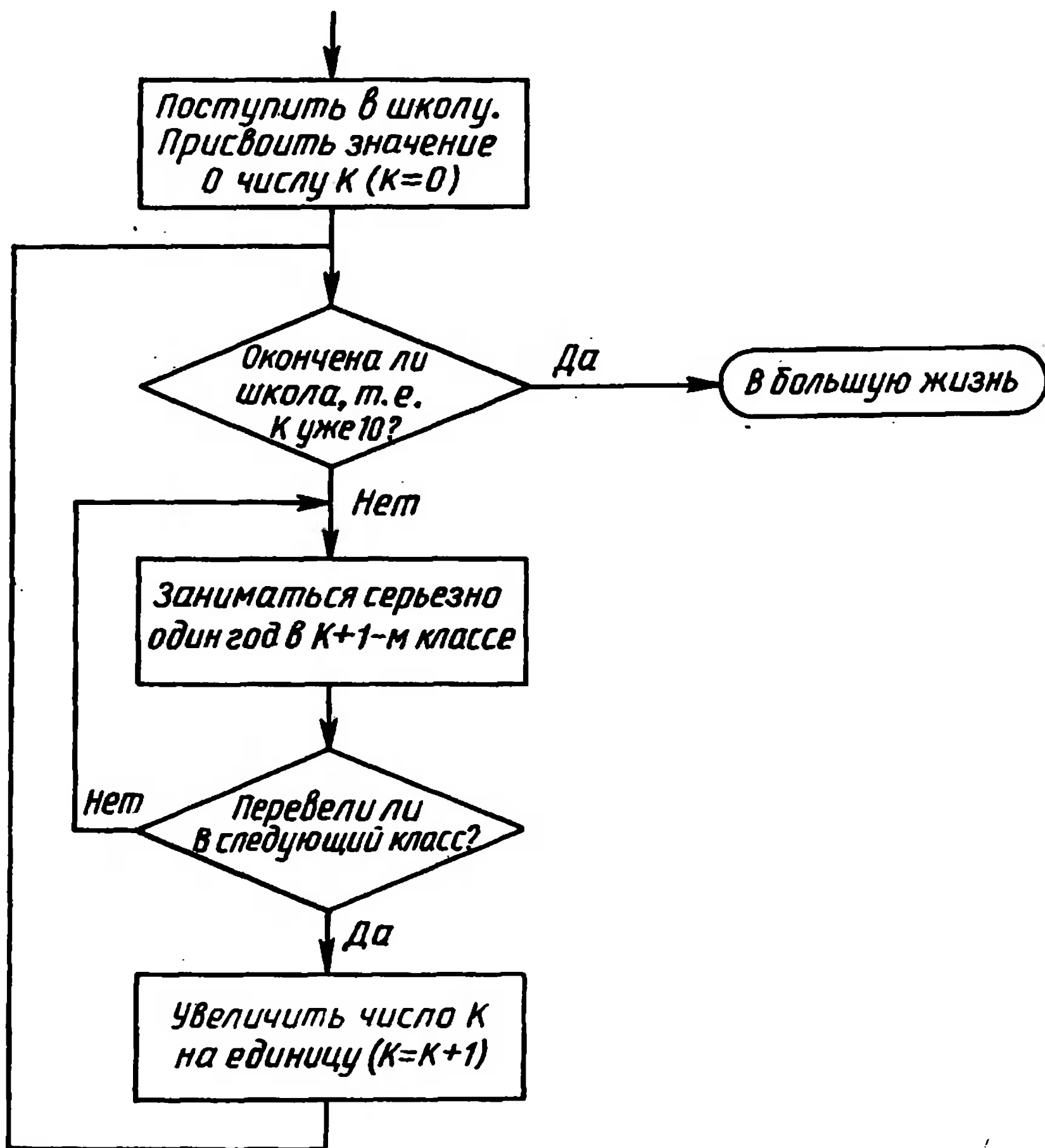
— Ну ладно...

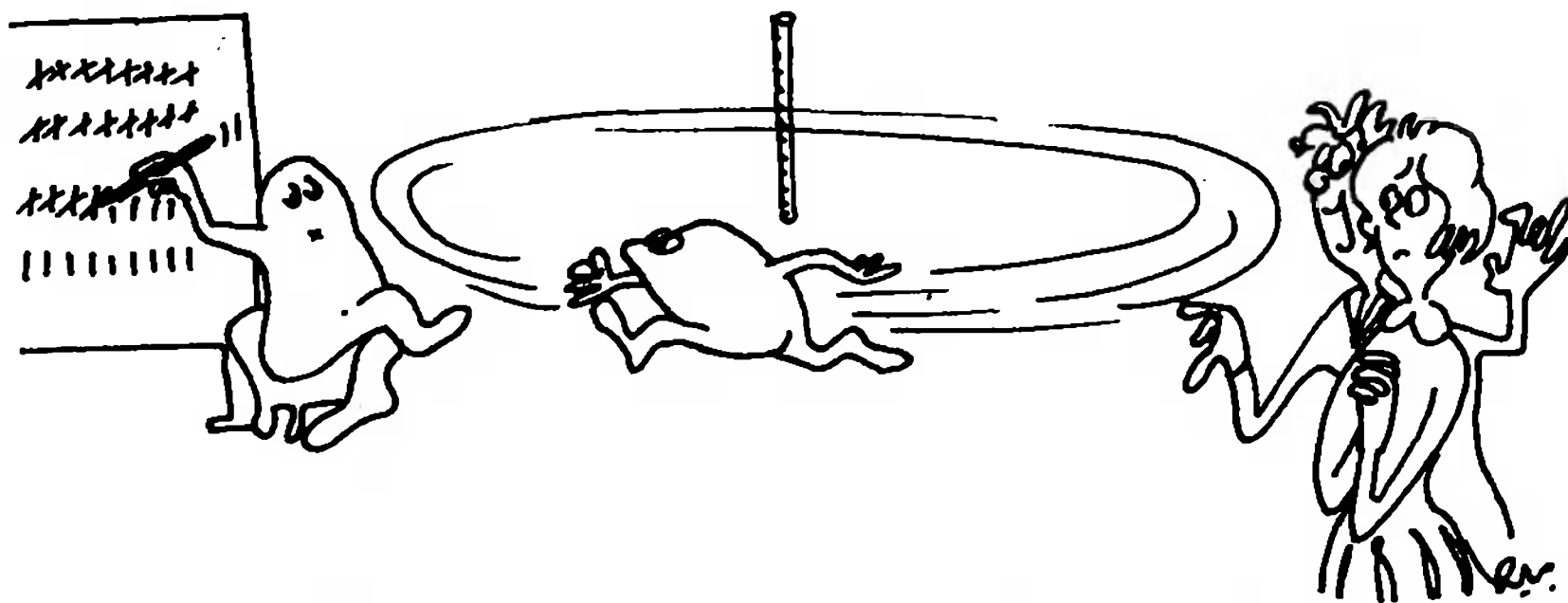
АЛГОРИТМ ДЛЯ НАЧИНАЮЩИХ УЧИТЬСЯ

— Это алгоритм обучения в средней школе. Конечно, тоже сильно упрощенный. Ясно, что десять лет занятий, пусть и не очень прилежных, полностью на одном листе бумаги не изобразишь.

— Ну, ну!

— Схема алгоритма выглядит так:





... а счетчик каждый раз отмечает

Счетчик

Важный нюанс этого алгоритма — наличие счетчика классов K . Точнее сказать, счетчика законченных классов, потому что в начале обучения в школе значение счетчика равно 0, а после окончания каждого класса оно увеличивается на 1.

— Но очень странно, что сразу после того, как этот счетчик стал равным нулю, проверяется, не равен ли он десяти. Чего же там проверять?

— В самый первый раз это действительно излишне, хотя никакой ошибки и нет. Но если ты изучишь эту схему до конца, то заметишь, что после окончания каждого класса нужно в этот блок возвращаться и проверять, не стал ли счетчик K равным десяти, т. е. не окончила ли ты школу. Зачем же зря стараться и еще ходить в школу, если она уже закончена?!

РАБОТА СО СЧЕТЧИКОМ КЛАССОВ

— Но скажи, не ошибся ли ты, написав такое странное равенство: $K = K + 1$? Ведь такое никогда невозможно!

— Так-то оно так, но $K = K + 1$ здесь не означает, что число K равно себе плюс единица. Это означает следующее: к теперешнему значению величины K прибавить единицу и полученный результат объявить новым значением K . Еще программисты говорят: присвоить величине K ее значение, увеличенное на единицу, или совсем коротко: увеличить значение K на единицу. На самом деле здесь знак « $=$ » означает не «равно», а «присвоить значение». В программировании такие отличия от привычного для школьника встречаются довольно часто.

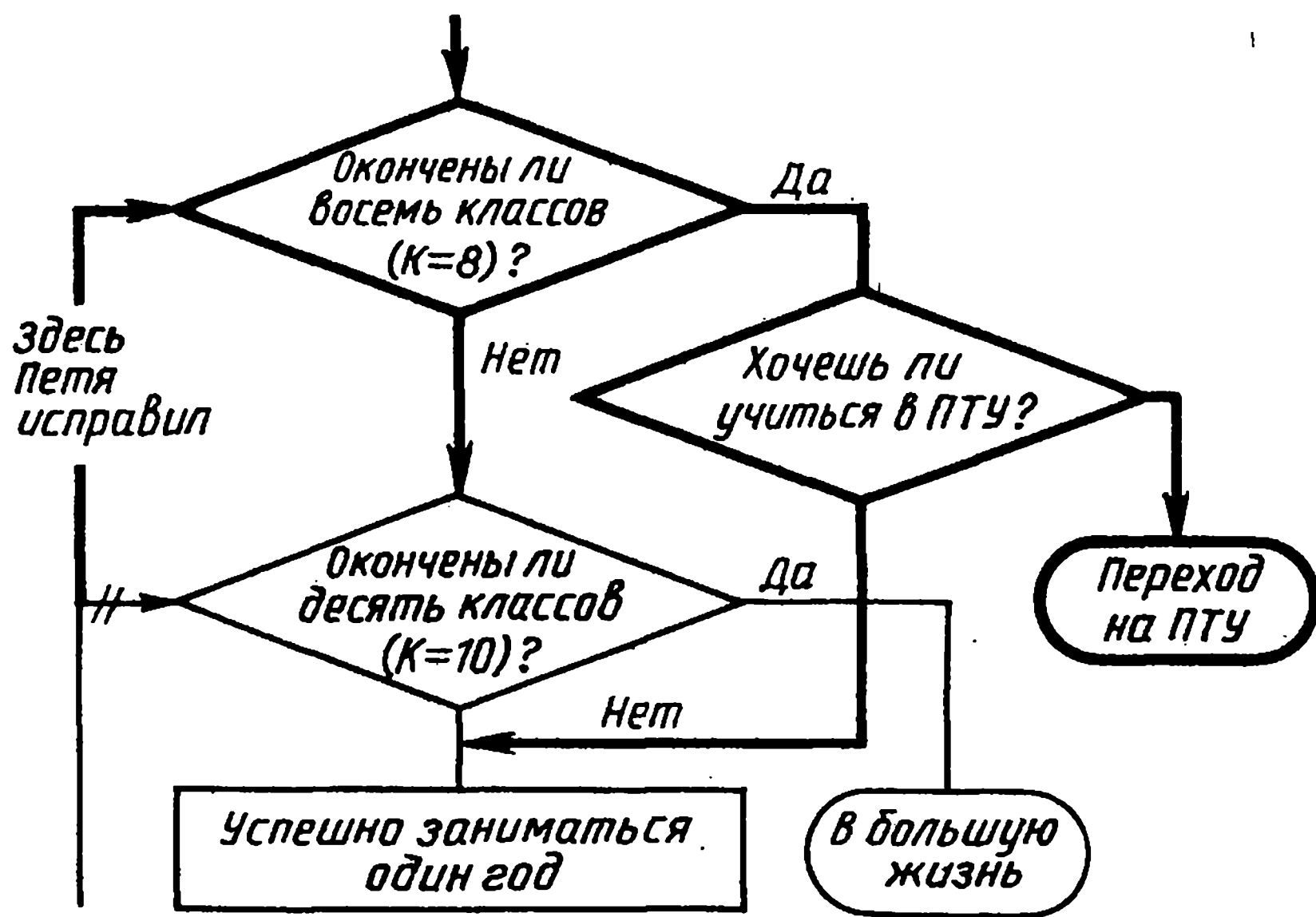
— Ничего, привыкнем и к программированию. Но как мы могли бы «проразмыслить» твой новый алгоритм? Конечно, он неполный, как ты и признавал. Но особенно мне не нравится, что ты не предусмотрел возможность уйти из школы после восьмого класса. Дай-ка я добавлю такую возможность, посмотрим, что из этого получится.

ТОНЯ УЛУЧШАЕТ АЛГОРИТМ

— Думаю, что проверку на выполнение условия «Равно ли K восьми?» нужно поставить перед проверкой на равенство десяти. Тогда, если восемь классов не окончены, ничего особенного не происходит, идем дальше проверять условие «Равно ли K десяти?». А если восемь классов окончены?.. Тогда проводим проверку условия «Хочешь ли уходить в ПТУ?». Если хочешь, то, конечно, уходи, а если нет. . . а если нет. . . то вот как! Иди прямо на прямоугольный блок «Учиться серьезно один год в $K+1$ -м классе», так как нет никакой необходимости возвращаться и проверять, равно ли K десяти, когда только что это K исследовали и нашли, что оно равно восьми и, значит, никак не может быть равно десяти.

— Кстати, можно было бы и проверить, никакой ошибки не возникло бы, просто лишняя работа. На мой взгляд, лучше было бы проводить проверку насчет ПТУ не только, когда окончено ровно 8 классов, а когда окончено и больше. Ведь человек может захотеть уйти в ПТУ и после девятого класса. Но, конечно, может быть и такой алгоритм, как ты сочинила. Сейчас изобрази свои выдумки графически.

— Я нарисую только небольшой кусок схемы алгоритма.



Жирными линиями я нарисовала добавки, а светлыми — то, что было раньше.

— Нарисовано великолепно. Особенно красивы эти твои

жирные линии. Но, увы, в твоём алгоритме имеется грубейшая ошибка, которая делает весь алгоритм совершенно непригодным, несмотря на изумительное оформление.

ОШИБКА!

— Да ну! И нечего издеваться, даже если и имеются ошибки... Хм... я ничего такого ошибочного не вижу.

— А ты посмотри сюда: стрелка приходит снизу в блок с проверкой условия «Равно ли К десяти?» сразу после увеличения значения К. Именно эта стрелка фактически создает цикл, и сейчас получилось...

— А-а-а! ... Получилось, что проверка условия «Равно ли К восьми?» осталась снаружи цикла и проводится только один раз в жизни, когда К ещё равно нулю и человек только что начал учёбу. Получилось, что добавление нового условия ничего не изменило. Да-а-а...

— Ничего страшного. Эту ошибку легко исправить: надо стрелку, входящую в блок с условием «Равно ли К десяти?», поднять, чтобы она входила в блок с условием «Равно ли К восьми?». Вот так, как я сейчас исправил.

— Ясно. Сейчас все правильно.

— Если сейчас, как ты выражаешься, «проразмыслить» этот алгоритм, то надо сказать, что при описании алгоритмов не советуют задавать постоянные в явном виде — в виде чисел.

— Где ты здесь видишь хоть одну постоянную?

— Я вижу целых две. Первая — 8, а вторая — 10. Так вот, их можно было бы как-то обозначить, например N1 и N2, и всюду, где они нужны, писать эти обозначения. А в самом начале присвоить им нужные значения: $N1=8$ и $N2=10$.

— Почему так сложно?

— В алгоритме, который мы здесь нарисовали, это действительно не очень нужно, так как и первая и вторая постоянная используется только один раз, потому что мы разработали алгоритм, весьма сильно отличающийся от действительного. А подумай, что могло бы случиться, если бы мы создали большой, большой алгоритм. И вот пришли и сказали, что все поменялось и сейчас дети должны учиться не 10, а 11 лет. И вот мы должны просмотреть всю схему алгоритма и всюду 10 заменить на 11. Если схема алгоритма большая и такие значения встречаются сплошь и рядом, то очень легко что-нибудь не заметить и не заменить. Ещё хуже получится, если заменишь какую-то десятку, которую вообще не надо было менять, потому что она была совсем не количество классов, а, скажем, количество уроков математики в неделе. Помнишь,

ты захотела только чуть-чуть изменить алгоритм и сразу допустила ошибку?

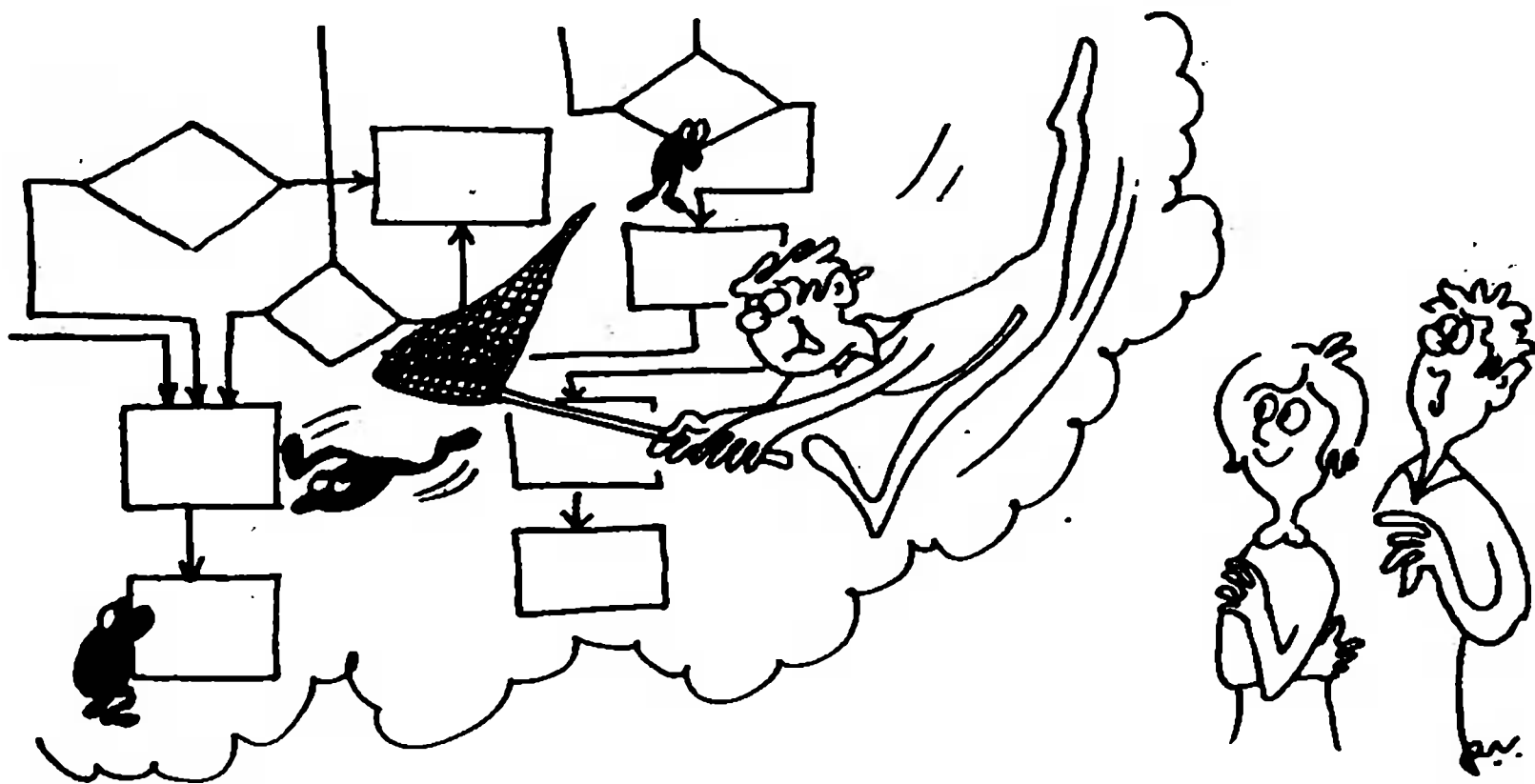
— Нечего вспоминать! Больше таких глупостей я не сделаю.

— Не зарекайся. Ошибки случаются всегда, и глупые тоже. Но могу тебя обрадовать: программисты это знают очень хорошо и никого за ошибки не ругают. Программировать — это совсем не то же самое, что контрольную писать. Программа считается готовой, и ее можно сдавать только тогда, когда все ошибки, как говорят программисты, выловлены, и притом самим программистом. Правда, говорят, что есть такие скептики, кто не верит, что в большой программе можно выловить все-все ошибки. Как бы то ни было, но программист должен уметь искать ошибки, готовиться их искать и учиться искать, насколько уж можно научиться. Главное, не надо воображать, а надо серьезно работать и стараться, а не ...

— Не кажется ли тебе, что ты здорово увлекся моим воспитанием? А я должна заметить, что имею полный набор воспитателей, и не каких-нибудь там бездельников и халтурщиков, а честных тружеников, работающих в поте лица, и притом весьма успешно.

Пете так не казалось, но спорить он не стал, а предложил еще раз просмотреть всю схему алгоритма целиком, а потом подумать, как можно было бы решить следующие задачи:

1. Составить алгоритм для учителя, решающего, кого из учеников сегодня вызвать отвечать урок с учетом наличия оценок, давности последнего ответа, необходимости исправить двойки и т. д. Необходимо организовать цикл по всем ученикам.
2. Составить алгоритм для ученика, решающего, какой из завтрашних уроков нужно наиболее тщательно подгото-



Выловить ошибки
в программе не-
легко

- вить, исходя из того, вызовут его или нет. Необходимо организовать цикл по всем урокам.
3. Составить алгоритм для звена пионеров — «заготовителей» макулатуры, обходящих квартиры. Организовать маленький цикл по квартирам и другой большой цикл по домам. Цикл по домам будет содержать в себе цикл по квартирам.
 4. Составить алгоритм для умножения столбиком. Тоже получатся два вложенных цикла. Один по умножению отдельных цифр первого числа на отдельную цифру второго числа, а второй цикл по всем цифрам второго числа.

5. ЯЗЫКИ, ЯЗЫКИ, ЯЗЫКИ...

В следующий раз оба юных программиста пошли в школу работать на компьютере, поскольку Тоня заявила, что разговоров вполне достаточно, и хотя, может быть, и занятно выдумывать всякие алгоритмы для неизвестного Петеньки и учеников, начинающих свой путь в науках, но ведь не станет же Бейсиков отрицать, что такие алгоритмы никому не нужны и пора начинать что-то такое, что может принести пользу, пусть и совсем малюсенькую.

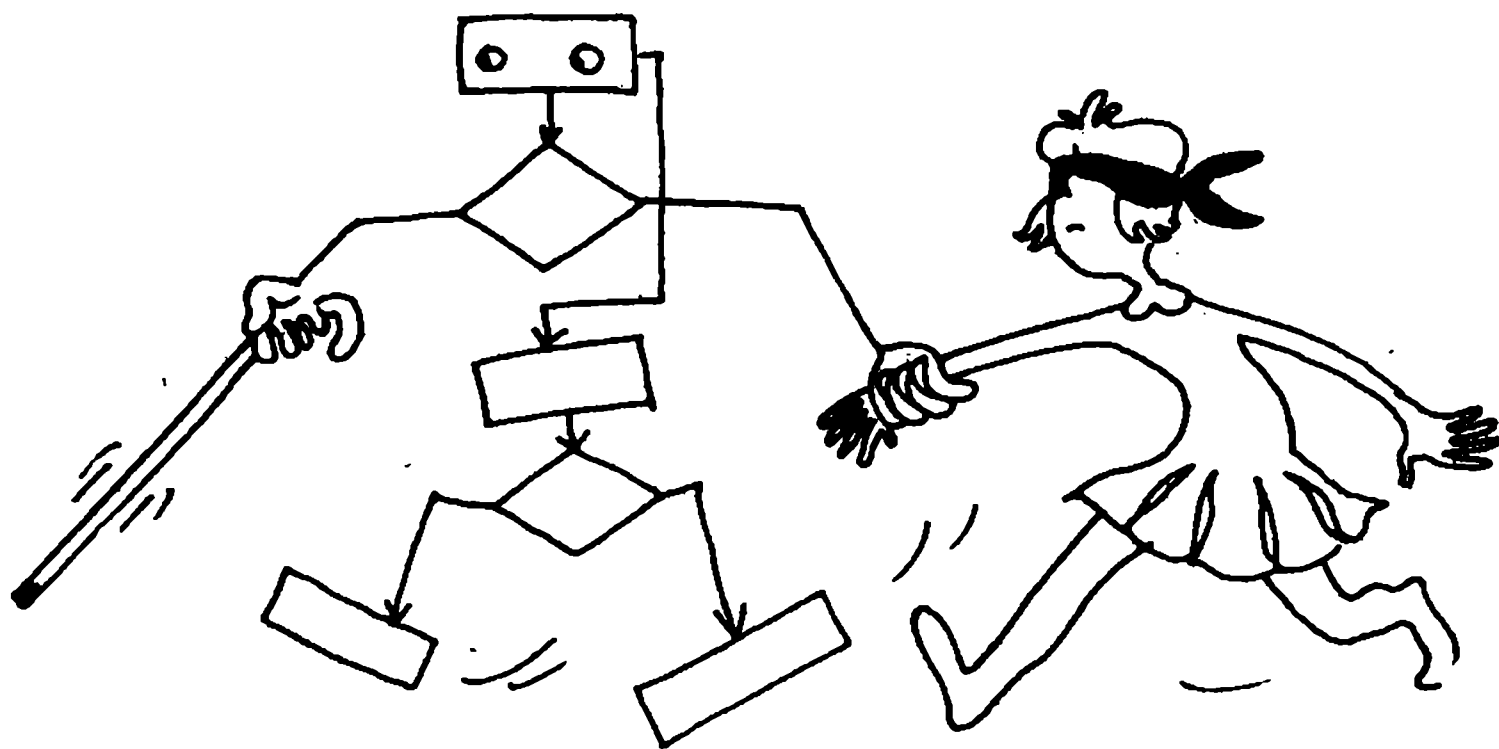
Однако по дороге Петя все еще продолжал говорить об алгоритмах.

МИР СОСТОИТ ИЗ АТОМОВ И АЛГОРИТМОВ

— Знаешь, Тоня, сами эти алгоритмы ты, конечно, можешь забыть хоть сейчас. Но чтобы работать с ЭВМ, на мой взгляд, надо проникнуться духом алгоритмизации. Понять, что всюду вокруг нас одни только алгоритмы, алгоритмы и ничего, кроме алгоритмов. Сложные и простые, жутко глупые и восхитительно остроумные. Что каждое утро, проснувшись, ты начинаешь выполнять алгоритмы — алгоритм одевания, алгоритм уборки постели, алгоритмы умывания, причесывания, алгоритм подогрева воды для чая, алгоритм хождения и т. д. и т. п. В общем всегда, когда мы что-то делаем, мы выполняем какой-то алгоритм.

— Так уж и всегда? Может быть, ты слышал про такое явление природы, как любовь? У тебя имеется уже алгоритм на тот случай, если вдруг совершенно случайно тебе придется решать подобные проблемы?

Нас ведут алгоритмы



— Ну, насчет всегда я преувеличил малость. Это должно было быть ясно. Кроме твоего, можно привести и другие примеры, когда алгоритма нет. Например, вряд ли можно было бы предложить алгоритм мышления.

— Да? Но говорят же про мыслящие машины, про искусственный интеллект. Разве ты не читал?

— Ой, знаешь, мне кажется, что это — сплошное запудривание мозгов! Правда, я пока мало что знаю по этим вопросам, но одно мне совершенно ясно: мышления ни в одной машине пока нет и интеллекта тоже. Дело в том, что люди, говорящие про искусственный интеллект, придают этим словам совсем другой смысл, чем мы, и тогда их бывает трудно понять. Хотя бывают и такие, кто этими словами пользуется для обмана простаков.

Чуть подумав, Петя добавил:

— Если ЭВМ иногда и в состоянии делать что-то достойное разумного существа, то проявляется при этом совсем не какой-то там мистический искусственный интеллект, а самый что ни на есть естественный интеллект программиста. И первое, где интеллект программиста должен проявиться, — это создание алгоритма. Нужно четко описать все, что надо делать, как я уже тебе говорил.

— Что-то не верю я в эти твои алгоритмы. Вот мучился ты, мучился и создал великий алгоритм прихода Петеньки из школы домой. А что толку? Какая польза от этого алгоритма?

ЕЩЕ ОБ АЛГОРИТМАХ

— Это-то верно, любой школьник доберется до дома и без точного алгоритма. А представь себе, что ему нужно пересекать улицу. Тут он должен следовать алгоритму или, други-

ми словами, достаточно четким правилам уличного движения. Правила, конечно, не такие уж сверхсложные. Красный свет — прохода нет, зеленый свет — вперед, Петь! Но ведь никто не рождается со знанием этих правил или, по-нашему, по-программистски, алгоритмов, их надо выучить, а всегда легче выучить то, что ясно и точно сформулировано. Разве не так?

— Конечно, так, но я не понимаю, к чему ты все это говоришь.

— Хочу сказать, что, во-первых, есть смысл некоторые задачи ясно и точно сформулировать или, другими словами, алгоритмизировать даже тогда, когда никто не собирается их решать с помощью компьютеров. А во-вторых, в мире имеется жутко много похожих алгоритмов, и, кстати, именно поэтому ЭВМ бывают полезными.

Вспомним, хотя бы тот самый алгоритм Петеньки. Пусть он прост и никому не нужен, но тем не менее его можно рассматривать как первый шаг в создании автоматической системы управления уличным движением. А такая система уже была бы бесспорно полезной и даже вполне реальной. Конечно, для такой системы нужно в тысячу раз больше алгоритмов, и притом намного сложнее. В частности, на мой взгляд, было бы важно знать точный алгоритм пересечения автомобилями нерегулируемого перекрестка улиц. При приближении к этому перекрестку водитель должен посмотреть направо и налево, совсем как Петенька, но смотреть он должен не на Бармалея или Борьку-хулигана, а на машины, подъезжающие слева и в особенности справа, и должен предвидеть дальнейшее их поведение.

— Хэ-хэ! «Машины подъезжают», «дальнейшее поведение машин» — и это говорит человек, который совсем недавно выступил с пламенной речью о том, что машины сами ничего не могут делать.



Точность! Точность
превыше всего!

— Да... действительно попался. Могу только добавить, что для составления алгоритма поведения водителя совершенно неважно, подъезжают ли другие автомобили к перекрестку сами или ими управляют люди. Важно то, что с какой-то скоростью что-то продвигается в сторону перекрестка. А если же составлять алгоритм для автоматической системы управления уличным движением, то все будет иначе: нужно будет подсчитывать, сколько машин собралось перед красным светом светофора и сколько времени они там стоят. И в зависимости от этой информации нужно переключать огни светофора.

— Кажется, совсем просто.

— Именно, что кажется, а когда начнешь делать, то соберется столько мелочей, которые надо учесть, что немудрено запутаться.

— Это кому как. Мне иногда труднее всего начать работу, а как начнешь!.. Сразу все становится легко... Странно получается, что я всю жизнь выполняла всякие алгоритмы и даже не подозревала об этом. Как-то и не верится. Единственное, что меня немножко в этом убеждает, так это твои примеры с утренними алгоритмами школьницы, готовящей чай...

КУЛИНАРИЯ, АЛГОРИТМЫ И КОМПЬЮТЕРЫ

— На занятиях по программированию самый обычный пример алгоритма — это кулинарный рецепт, например варка супа или выпечка блинов. В этих рецептах четко и ясно описано, что и сколько надо брать и что делать...

— Не так уж и ясно! Какая там ясность, если написать, например: «Варить до готовности».

— Хорошо, пусть не каждый, а опытный и знающий человек по поваренной книге приготовит, что захочет. С другой стороны, ты права, потому что ясности кулинарного рецепта было бы совершенно недостаточно для такого компьютера, который приспособили для управления суповарочным котлом. В алгоритме работы такого автомата совершенно четко должно быть указано, что такое «варить», что такое «готовность» и что такое «варить до готовности».

— А такие автоматы, может быть, даже существуют!

— Я тоже так думаю. Существуют же такие измерители температуры, которые можно подключить к ЭВМ, электронные часы в компьютерах есть почти всегда, выключатель электронагревательного прибора тоже можно было бы подключить. Что там еще нужно было бы? Похоже, что почти ничего, насколько я знаю кулинарию.

— Ты опять не о том говоришь. Как же все-таки работает компьютер, как он понимает, что именно надо делать?

— Ничего он не понимает! Я же сказал, что на самом деле он понимает не больше, чем... чем телевизор, показывающий очень умную передачу!

— Но все-таки, как это получается, что выглядит, будто он понимает?

— Если честно, то до конца я этого не постиг. Меня этому не учили, сказали, что тогда надо осваивать машинные команды, двоичную арифметику, математическую логику и бог знает что еще. Все это можно и освоить, науки хоть и непрос-тые, но понятные. Однако совсем необязательно знать, как работает двигатель, чтобы ездить на автомобиле без аварий. Я тебе могу рассказать только то, чему нас научил Алгорит-мыч. Он утверждал, что этого достаточно, чтобы составлять программы для ЭВМ, имеющие смысл и пользу. И похоже, что не обманул, действительно достаточно, но иногда и меня мучает мысль: что же там внутри машины происходит?

— Что же делать, раз ты такой недообразованный. Рас-скажи тогда то, что ты так блестяще усвоил.

ЯЗЫКИ ДЛЯ ОБЩЕНИЯ С КОМПЬЮТЕРАМИ

— Речь идет об алгоритмических языках, или языках про-граммирования. Точнее, об одном из них — Бейсике, который я знаю и на котором составляю программы, какие уж умею. Об этом языке я тебе скоро расскажу.

— Ясно. Но чтобы все выглядело систематично и научно, ты, конечно, не станешь сразу говорить об этом Бейсике, который знаешь, а начнешь рассуждения о языках вообще — о языках африканцев и индейцев, о языке птиц, пчел, собак и кошек, о языке цветов, короче, о вещах, в которых ты ни-чего не понимаешь.



Язык программирования — это что-то вроде заклинаний

— Зачем ты так. Я же честно говорю, что я знаю и чего не знаю.

— Это для профилактики, Бейсиков. Чтоб не зазнавался.

— Но немножко о языках программирования вообще придется поговорить. А то ты освоишь один только Бейсик и покажется тебе, что ты стала такой умной, такой умной, ну прямо как директор школы.

— Ладно, ладно. Можешь продолжать, счет 1:1.

— Так вот, ЭВМ обрабатывают информацию, только и всего. Это означает, что в них вводят какую-то информацию и от них получают какую-то информацию. Более просто: в них вводят какие-то символы — цифры или буквы и получают в ответ тоже какие-то символы. Разумеется, крайне желательно, чтобы полученные символы имели какой-то смысл.

— Что-то тебя все тянет к философствованию. Объясни глупой девчонке как-нибудь попроще. На примерах, на простых примерах.

— На примерах? Знаешь, ты получишь целую кучу разнообразных примеров, когда доберемся до работы на машине. А пока такой пример. Представляешь машину, играющую в шахматы?

— Ну, представляю.

ВХОДНАЯ И ВЫХОДНАЯ ИНФОРМАЦИЯ

Информация ввода

Информация вывода

— Даже неважно, в шахматы или в какую-нибудь другую игру. Сначала в машину надо ввести исходную позицию — это будет для нее входная информация, или входные данные. После этого, чтобы получилась игра, нужно ввести в машину свой ход. Это тоже входная информация. А машина, получив ход играющего с ней, должна эту информацию обработать, конечно, учитывая текущую позицию. Результатом такой обработки информации будет ответный ход, который она должна вывести, иначе никто его не узнает. Этот ход и будет выходной информацией, или выходными данными. Вот и все про входную и выходную информацию. Самое важное, как ты понимаешь, происходит между вводом и выводом информации, когда полученная информация обрабатывается по определенному алгоритму. В действительности этот алгоритм определяет не только то, что произойдет с информацией в машине, но и то, какую еще информацию нужно ввести и какая будет выведена. И все эти действия нужно тщательно описать в алгоритме, и притом в форме, приспособленной для работы на машине, или, другими словами, написать программу.

— Наконец-то мы и добрались до понятия, из-за которого весь сыр-бор и разгорелся.

ПРОГРАММЫ КТО-ТО ДОЛЖЕН СОСТАВЛЯТЬ

— Что и говорить, программы нужны. Без программы компьютер будет так же бесполезен, как испорченный телевизор. Многие считают, например, что правильнее говорить не «компьютер играет в шахматы», а «программа играет в шахматы» или же «программа обрабатывает информацию». С ними можно было бы согласиться, так как именно программа претворяет в жизнь алгоритм игры в шахматы. Можно было бы согласиться, если не знать, что без самого компьютера программа будет только бессмысленными каракулями. Далее, ты должна понять, что программы пишут на языках программирования. Так как программа — это описание алгоритма, то можно эти языки называть и алгоритмическими языками. Еще можно сказать, что программа — это алгоритм, записанный в форме, понятной для человека и ЭВМ.

Языки программирования

Программа

— Ха-ха! Опять попался! Ты же говорил, что машина ничего не понимает и тот, кто говорит, что понимает, — круглый дурак.

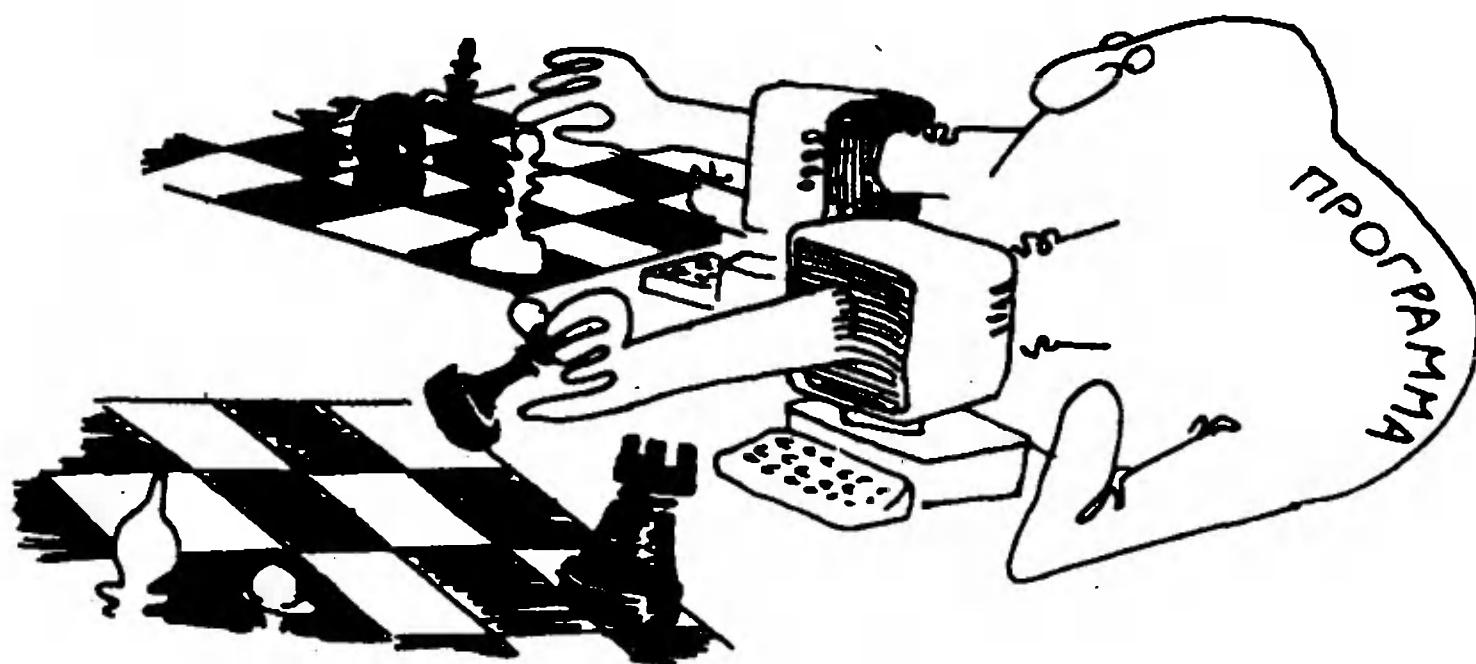
— Я, допустим, не совсем так говорил, но смысл ты уловила великолепно. А оговорился я, считай, в педагогических целях, чтобы проверить, заметишь ты или нет. Сейчас скажи эту фразу точно.

— Программа — это алгоритм, записанный в форме, понятной человеку и... и...

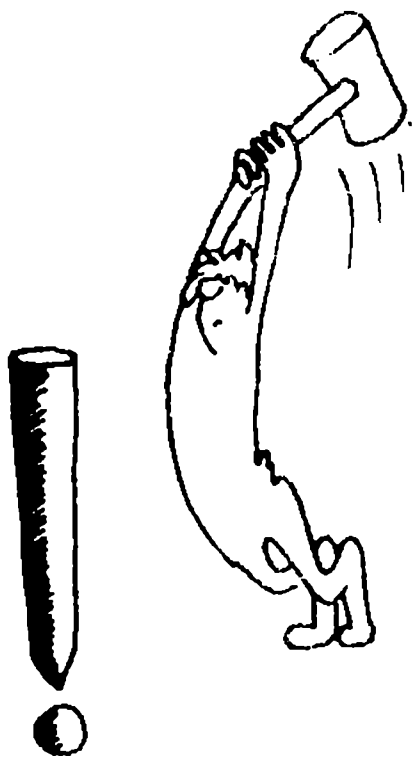
— ...Приспособленной для выполнения на ЭВМ. Так можно было бы сказать. Но это не определение, можешь наизусть не учить, никто тебя не спросит.

— Неужели! А я собиралась вызубрить все твои разглагольствования.

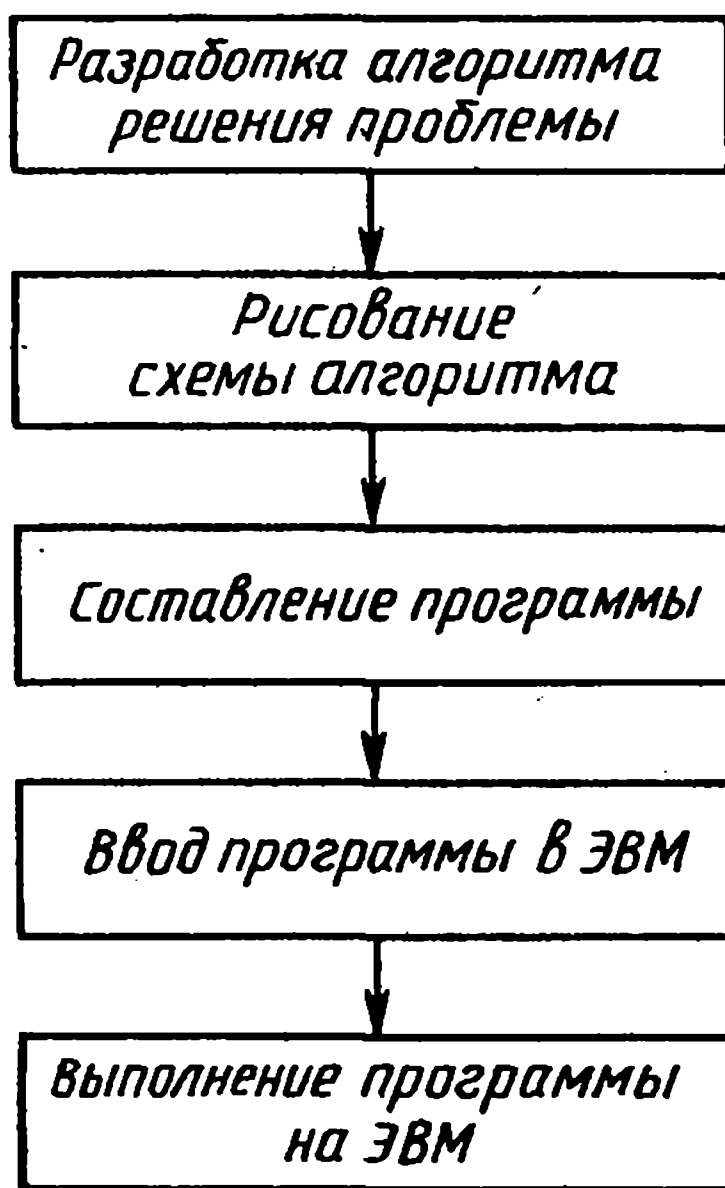
— Тем не менее запомни: для решения какой-нибудь задачи на ЭВМ мы сперва должны придумать алгоритм. Потом для ясности должны нарисовать схему этого алгоритма, как мы уже делали, потом написать программу, ввести эту про-



В шахматы играет программа, а не машина!



грамму в машину и, наконец, выполнить эту программу на машине. Считай, что я тебе изложил алгоритм решения задач на ЭВМ по заданному алгоритму, так сказать супералгоритму. Этот алгоритм тоже можно изобразить в виде схемы, правда совсем простой:



кон. 35

Ясно?

— Ну... Ясно.

ЯЗЫКИ — СРЕДСТВО ОБЩЕНИЯ

— Языков программирования очень много. Для одного типа задач лучше подходят одни, для другого типа — другие. Это прямо как в жизни. Мы с тобой говорим на языке слов, а, скажем, музыканты общаются между собой на языке нот, конечно, если дело касается исполнения музыкальных произведений. Смело можешь считать, что композитор записывает алгоритм исполнения своего сочинения на алгоритмическом языке нот. В свою очередь, и сама музыка — я имею в виду не записанную нотами, а уже звучащую — является языком — языком музыкальных звуков. Тем, кто ее понимает, она же что-то выражает.

— Только музыка выражает не словами, а образными средствами.

— А чувствуешь ли ты разницу между алгоритмическим языком нот, как я его назвал, и языком музыкальных звуков?

— Чувствую ли? Ты, наверно, имеешь в виду, что в алгоритмическом языке содержатся только указания на то, что надо делать, ну, он как бы содержит одни только приказы, а язык музыки может выражать все что угодно.

— В том числе и приказы. Скажем, в старой армии были сигналы горна «Слушайте все!», «Тревога!» или «Смирно!». А что касается языка нот, то, имея мы программируемый синтезатор, могли бы переписать любую нотную запись один к одному в командах встроенного в синтезатор компьютера. Даже думать особенно не надо было бы.

— Не дразни музыкантов! Музыканты — хорошие люди.

— А вот другой случай. Автоинспекция программирует уличное движение на языке дорожных знаков и сигналов светофора. Вот тебе еще один алгоритмический язык, ибо как дорожные знаки, так и сигналы светофора определяют алгоритмы действий водителей в конкретной ситуации. Если же немножко подумать, то становится ясным, что в принципе и композитор, и автоинспекция могли бы все необходимое сообщить словами русского языка. Можно было бы написать: «Стукни что есть силы по 27-й справа белой клавише рояля», а вместо «кирпича» повесить надпись «Проезд запрещен». Понимаешь, какие это создало бы трудности?

— Чего уж тут не понять? Особенно трудно было бы тем, кто не понимает русского языка.

— Ситуация же с языками программирования такова. Каждый язык или по крайней мере большинство их можно считать универсальными, и как-то можно запрограммировать почти все задачи на каждом из них. Однако хорошо получится только для некоторого класса задач. Лично я знаю только один язык программирования — Бейсик. Этот язык не счи-



Бейсик понимают все!

тается самым лучшим, может быть, потому, что проще других. Не берусь судить, насколько серьезен этот недостаток, но знакомый студент Костя Фортранкин, который программирует на языке Фортран, смотрит свысока на мои занятия. Правда, о нем самом нелестно отзывается его воображалистый однокурсник Витя Пээлло, который работает на языке PL/1, а сотрудница Алгоритмыча Ира Паскалых, программирующая на Паскале, считает, что мы все не в состоянии ничего запрограммировать без ошибок.

Как нам по секрету сказал Алгоритмыч, среди программистов широко распространена «языковая рознь» — каждый программист считает лучше всех тот язык программирования, который ему удалось освоить. Ладно, если бы про себя тихо-смирно считал, а то бывает и публично начинает поносить другие языки.

— А все-таки какой язык самый лучший?

— Берем газонокосилку и электробритву. Какая из этих очень похожих машин самая лучшая?

— Я поняла.

6. НАЧИНАЕТСЯ РАБОТА

И вот Тоня и Петя опять у ЭВМ. Тоня сразу устроилась за дисплеем и заявила, что работать будет она, а Петя пусть наблюдает и дает ценные указания.

— Хорошо. Пусть будет так. Но должен сказать, что пока ничего на машине произойти не может. Сначала нужно вызвать интерпретатор Бейсика.

ВСЕ ДЕЛАЕТ ИНТЕРПРЕТАТОР

— Что это такое?! Про это ты ничего не говорил!

— Это программа, которая уже составлена и хранится на магнитном диске. А предназначена она для обработки нашей программы, написанной на Бейсике, — расшифровки, какие действия наша программа поручает выполнять машине и, более того, эти действия исполнить. Интерпретатор как бы читает строку программы и прочитанное сразу выполняет, не откладывая ни на секунду. Одним словом, интерпретирует программу Бейсика.

— Так что? Получается, что одна программа обрабатывает другую?

— Да. Так и получается, что программа на Бейсике является входными данными для интерпретатора.

— Ну и как же вызвать этот ин-тер-пре-та-тор? Ты, кажется, так сказал?

— Да. Вызвать — это означает списать программу с диска в оперативную память и запустить ее. А делает это еще одна программа...

— Да ты что! Сколько же здесь этих программ?

— Спокойно. Пока были только две. Одна — интерпретатор, а другая — та, которая может вызвать интерпретатор. Всего таких обслуживающих программ, наверно, несколько десятков — так когда-то говорил Алгоритмыч. Все обслуживающие программы вместе образуют так называемую операционную систему. Алгоритмыч о ней нам рассказывал очень мало, все шутил: чтобы понять, как работает операционная система, надо кончать физмат. Надеюсь, ты догадываешься, что программы операционной системы нельзя было написать на Бейсике, потому что без них некому вызвать интерпретатор. И сам интерпретатор, очевидно, тоже написан на другом языке программирования.

— Но что же в конце концов нужно делать с этим твоим интерпретатором?



ИНТЕРПРЕТАТОР



ВЫЗОВ ИНТЕРПРЕТАТОРА

— Сначала обрати внимание на последнюю строчку экрана. Там ты видишь знак «больше» и рядом с ним курсор. Это означает, как я тебе уже говорил, что сейчас операционная система ждет команду. Раз ты хочешь вызвать интерпретатор Бейсика, то, разумеется, нужно ввести команду вызова. Она очень проста — BAS. Три буквы латинского алфавита. Ну, можешь вводить.

BAS_

Команда

Все в порядке. Найди сейчас клавишу ввода и нажми. И Тоня нажала. В одно мгновение все строки экрана переместились вверх, в том числе и введенная Тоней строчка, а снизу показался текст.

IAS/RSX BASIC V02-01M

READY

— «Рэди», — прочла Тоня, — «готово».

— И верно. Интерпретатор готов к работе. А строка выше означает, что работает именно интерпретатор Бейсика.

A black and white cartoon illustration. In the center, a man in a suit is running towards a computer monitor that displays the letters 'BAS'. A woman stands behind him, looking on with a worried expression. To the left, a large, stylized figure labeled 'ИНТЕРПРЕКАТОР' (Interpreter) is being crushed by a large, dark, circular object, with lightning bolts above it.

— Слушай, а что машина и впредь только по-английски будет изъясняться? Мои предки, честь им и хвала, старались изо всех сил и водили меня на растерзание к одной «англичанке», но я не могу сказать, что наши с нею мучения дали какие-то плоды.

— Тебе повезло. А что делать тем, кто в школе изучает немецкий язык и притом не очень преуспевает? Но не расстраивайся, английского языка при изучении Бейсика надо не очень много. Твоих знаний вполне хватит, раз уж ты знаешь, что «рэди» — это «готово».

— Хватит болтать, ведь интерпретатор уже готов к работе. Что сейчас можно сделать?

— Сейчас можно работать с интерпретатором в непосредственном режиме, когда все, что прикажешь, будет сразу выполнено. При этом получается, что наша громадная ЭВМ используется почти как микрокалькулятор.

— Например, можно образовать переменную А и присвоить ей значение 12. Просто введи $A=12$.

READY

— Что ты! Произошло очень много чего. Образовалась переменная A и значение этой переменной 12. Сейчас введи $B=20$.

READY?

— Опять ничего не произошло! Во всяком случае ничего не видно.

— Произошло то же самое. Образовалась еще одна переменная — В со значением 20. Сейчас со значениями этих двух переменных, т. е. с числами 12 и 20, ты, например, можешь совершить любое арифметическое действие. И какое именно ты бы хотела?

— Я хочу сделать все — и сложить, и вычесть, и умножить, и поделить тоже, раз уж такая прекрасная возможность появилась.

— Тогда поступай так. Введи $C = A + B$. Это означает, что интерпретатор должен образовать еще одну переменную — С с численным значением, равным сумме значений переменных А и В.

$C = A + B$

READY

— Сделано. Может быть, там внутри машинной памяти что-нибудь и образовалось, но мне от этого мало радости. Чему равно 12 плюс 20, я так и не знаю.

$$\begin{array}{r} 12 \\ + 20 \\ \hline 32 \end{array}$$

$\begin{array}{r} A \\ + B \\ \hline C \end{array}$



РЕЗУЛЬТАТЫ ВСЕГДА НУЖНО ВЫВЕСТИ

— Чтобы узнать результат, его надо вывести на экран. Для этого существует оператор вывода PRINT. Напиши это слово и после него список тех переменных, значения которых ты хочешь вывести. В списке переменные надо отделять запятыми. Ах, да! «Принт», если ты не знаешь, означает «печатать». Ну, выведи значения переменных А, В и С.

— Хорошо.

Теперь Тоня на экране видела

PRINT A,B,C

12

20

32

— Да, все-таки получилось!

— Очень удобно, что после PRINT совсем необязательно писать одни только переменные, можно писать сразу их сумму $A + B$ или произведение $A * B$, знак «*» здесь означает умножение, или частное A / B , где дробной чертой «/» обозначают деление. Хорошо запомни обозначения арифметических действий. По-другому писать нельзя.

— А ты ничего не сказал про вычитание.

— Вычитание обозначается, как всегда, знаком «минус».

Итак, в операторе вывода можно писать любое арифметическое действие и, более того, любое арифметическое выражение, состоящее из большого количества арифметических действий, которые могут быть записаны с использованием скобок, например $(A + B + C) / C + 20$. Важно, чтобы длина этого выражения не превышала длину одной строки дисплея, т. е. содержала бы не более 80 символов. Выведи, пожалуйста, результат расчета какого-нибудь арифметического выражения!

— Ладно. Введу я вот что.

```
PRINT 24/A*2
```

4

Фэ! Машина совсем не умеет считать. Результат должен быть 1, а не 4!

— Не может быть! Ясно. Я тебе не сказал, как в Бейсике использовать скобки. Если в знаменателе имеется произведение, оно обязательно должно быть в скобках. Если скобок не будет, то интерпретатор разделит только на первый сомножитель, а на последующие сомножители умножит. Если ты, например, напишешь $6/2 * 3$, то 6 будет разделено на 2, получится 3, после умножения еще на 3 результат будет 9. А расчет выражения $6/(2 * 3)$ или $6/2/3$ даст единицу. Повтори свой пример.

```
PRINT 24/(A*2)
```

1

— Теперь правильно.

— Правильно было и в предыдущий раз: машина же честно выполнила то, что ты ей задала. А если ты задала неправильно, вини себя... Еще могу сказать, что в Бейсике имеется специальный знак для возведения в степень — стрелка вверх «↑». На клавиатуре он находится здесь слева, на той же клавише, что и буква «Ч».

— Это надо попробовать.

```
PRINT 2^2
```

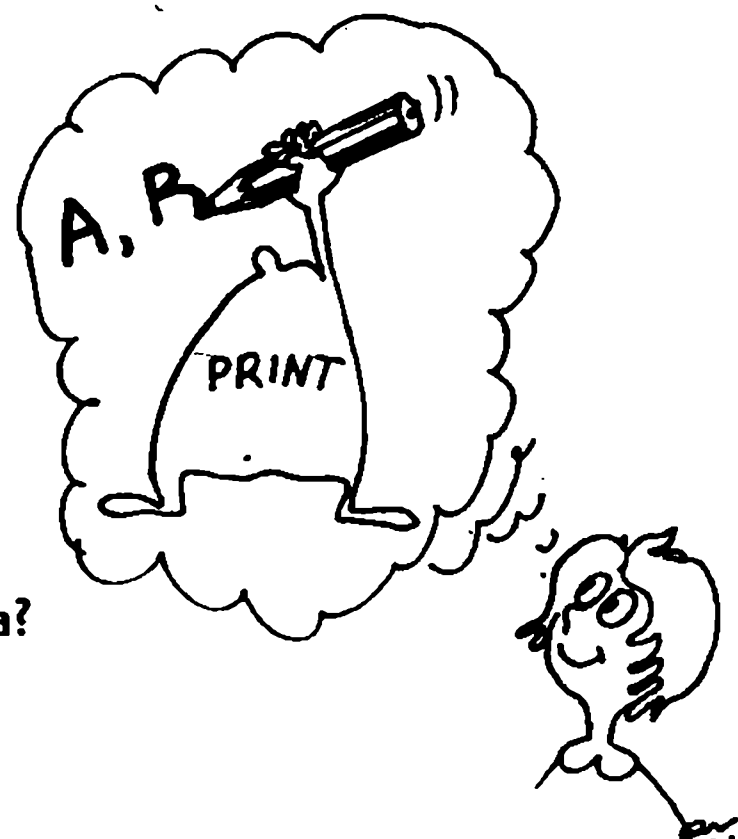
4

Интересно, а так получится?

```
PRINT A^A
```

8.91610E+12

Бейсики! А это что за чепуха?



ДВЕ ФОРМЫ ЧИСЛА В БЕЙСИКЕ

— Что такое? А-а! Результат выведен в экспоненциальной форме.

— Опять новое понятие!

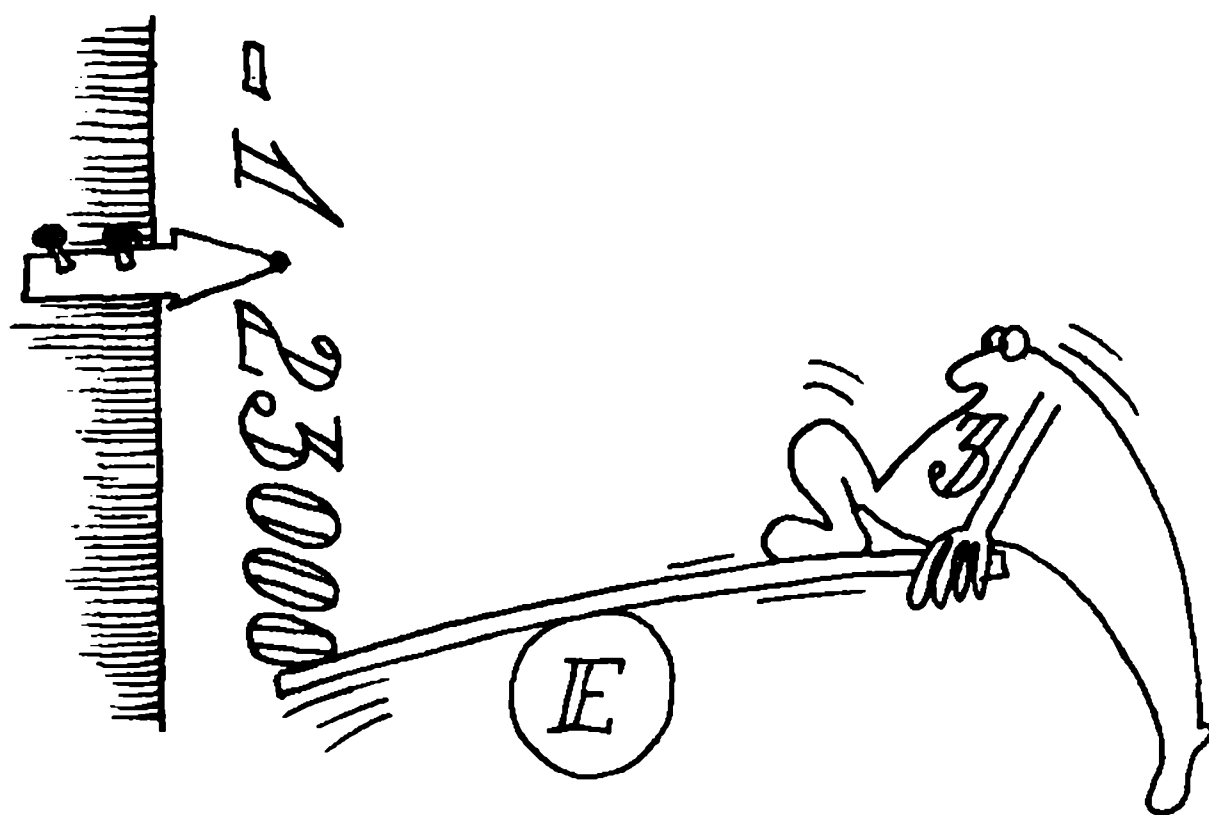
— А что делать? Ты лучше внимательно слушай и наматывай на ус. Кстати, ничего сложного здесь нет. Алгоритмыч в таких случаях говорит: это просто, как программировать на Бейсике. Чтобы понять, какое число выведено, используй такой алгоритм. Например, машина вывела в экспоненциальной форме такое число: $-1.23456E+07$. Число, стоящее слева от буквы E — 1.23456 называется мантиссой исходного числа. Символы справа от буквы «E», т. е. $+07$, в свою очередь, дадут так называемый порядок числа. Сейчас мантиссу числа нужно умножить на 10 в степени, равной порядку числа: $-1.23456 \cdot 10^7$, и ты получишь число в обычной записи — 12345600. Еще могу добавить, что знак порядка числа выводится всегда, а знак самого числа, или, что то же самое, знак мантиссы, интерпретатор выводит только для отрицательных чисел.

Экспоненциальная форма

— Бейсиков! Из тебя порядочный педагог никогда не получится. Как можно так долго и нудно говорить о таких простых вещах!

— А ты эти простые вещи попробуй изложить алгоритмически. В качестве домашней работы нарисуй схему алгоритма для перевода числа из экспоненциальной формы в обычную, тогда будешь знать, получится из меня педагог или нет.

— Ну и мстительный же ты тип! В школе домашние работы, дома домашние работы, так и концы отдашь, не узнав, что же такое программирование.



А сейчас передвигем точку...

$$12^{12} = 89161000000000$$

$$\begin{aligned} 12.3E + 04 &= \\ = 12.3E + 4 &= \\ = 1.23E + 5 &= \\ = 12.30E4 & \end{aligned}$$

СКОЛЬКО БУДЕТ ДВЕНАДЦАТЬ В СТЕПЕНИ ДВЕНАДЦАТЬ?

— Ничего, ничего. Домашние работы крайне полезное дело. А теперь напиши, какое число в последний раз выводилось на экран.

— Пожалуйста! 89161000000000.

— Правильно. Я надеюсь, ты не думаешь, что это число в точности равно двенадцати в степени двенадцать. Машина арифметические действия выполняет с точностью в шесть знаков и выводит те же шесть знаков. Нули в конце числа на самом деле не настоящие нули, а неизвестные цифры числа.

— Хоть ты и очень умный, Бейсиков, но все-таки попался. Первый нуль слева совсем настоящий нуль, потому что он был в выведенном числе. Не настоящие нули — это те семь, которые я приписала в конце числа.

— Твоя правда. Тот нуль действительно намного более настоящий, чем другие. Но полностью за него ручаться нельзя, так как при проведении вычислений с шестью знаками шестой получается неточным. Еще запомни, что числа можно также и вводить в экспоненциальной форме. При вводе между точкой и буквой «Е» необязательно должно быть ровно пять цифр, а перед точкой необязательно писать ровно одну цифру. Например, ты можешь писать так:

$$A = -12.3E + 04$$

или так:

$$A = 8612E - 12$$

$$A = 12.8E + 5$$

— А писать запятую вместо точки можно?

— Нет, нельзя. В Бейсике при записывании чисел с дробной частью вместо привычной запятой пишется точка. Запятая используется, чтобы отделить одно число от другого, когда надо. Можем нарисовать небольшой бейсико-арифметический словарь:

На Бейсике	В арифметике
—1.23000	—1,23
—1,23000	—1 и 23 000

А сейчас можешь побаловаться с машиной — проделать какие-нибудь расчеты в непосредственном режиме. Например, рассчитай подряд степени двойки и посмотри, когда машина начнет выводить результаты в экспоненциальной форме.

— Я это и так знаю. Когда в результате получится больше шести знаков.

А ЕСЛИ ОШИБЕШЬСЯ!

— Тогда попробуй что-нибудь другое. Например, подели значение переменной A на нуль.

— Разве я не понимаю? На нуль делить нельзя!

— А ты все-таки попробуй.

```
C=A/0
```

```
?DIVISION BY ZERO
```

```
READY
```

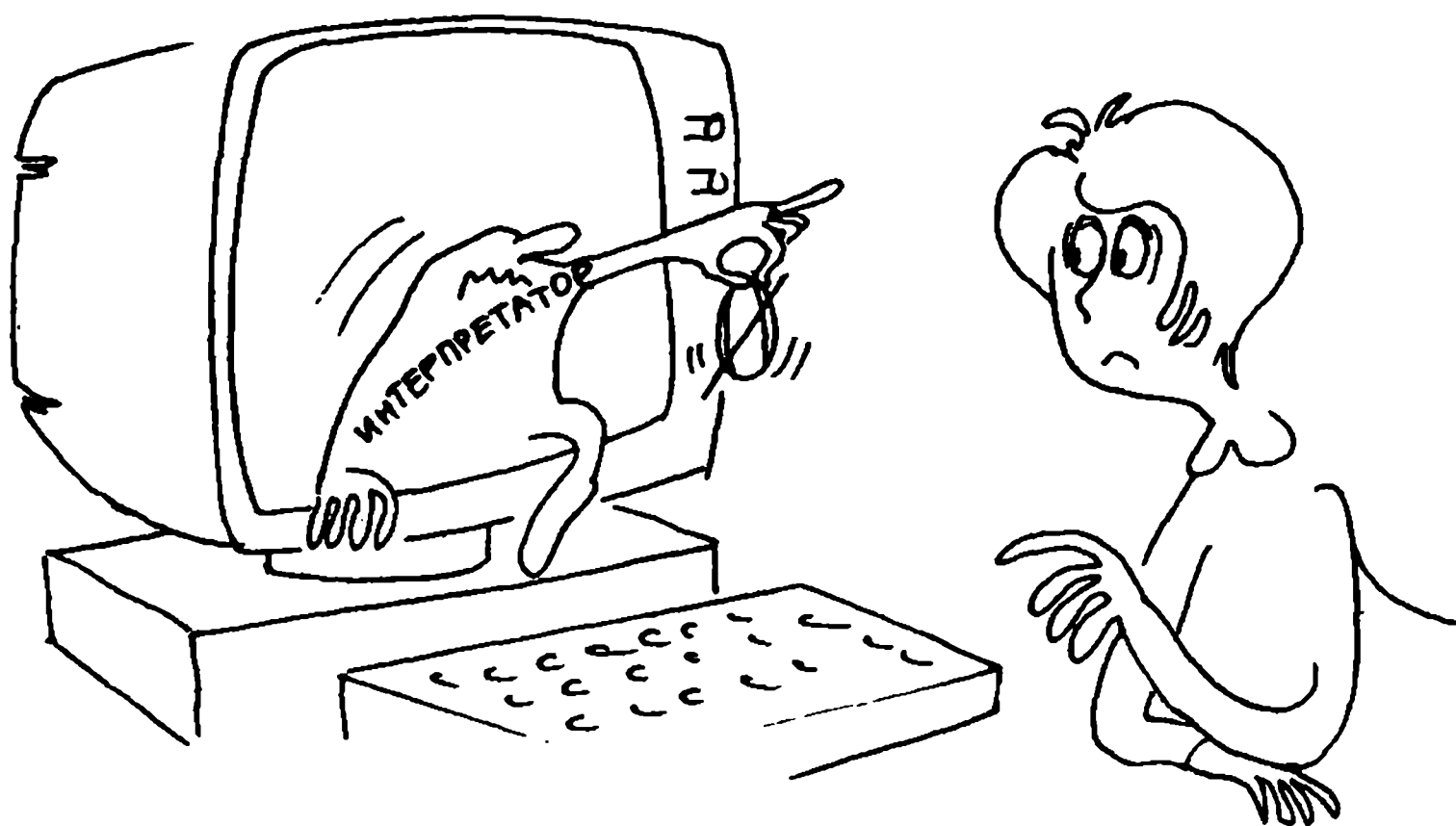
— Вот видишь! На нуль делить действительно нельзя, и интерпретатор тебе об этом сообщает. Принцип здесь такой: если интерпретатор может определить, что человек ошибся, он должен это делать. И в этом простом случае, и в более сложных, о которых речь еще впереди. Я тебя заставлял делить на нуль, чтобы ты увидела, как выглядят такие сообщения об ошибках. Их отличительная особенность — вопросительный знак в начале строки. Так что, когда получаешь такую строку и чувствуешь, что ничего плохого не делала, беги, ищи знатока, потому что сама вряд ли поймешь английский текст сообщения. Там всегда такие слова, которые в школе не учат.

— Вот тебе на! Единственный выход — не ошибаться.

— Или, наоборот, быстро-быстро сделать все возможные ошибки и узнать, как на них реагирует интерпретатор. Сейчас я сделаю вот такую ошибку: возведу значение переменной A в степень A и результат еще раз в степень A.

— Результат будет очень большое число.

— Посмотрим.



C=A^A^A
?FLOATING OVERFLOW

READY

Действительно, получилось такое большое число, что его нельзя, как говорят, представить в машине. Порядок самого большого числа, которое еще можно представить в машине, равен $+37$, а порядок самого маленького положительного числа -37 , все отрицательные числа, конечно, еще меньше. Так что самое большое число, представимое в машине, будет $9.99999E + 37$.

Сегодня я тебе больше ничего рассказывать не буду, так как наше время скоро кончается и на машину придут работать другие.

7. ПЕРВАЯ ПРОГРАММА ТОНИ

В следующий раз Тоня и Петя пришли на машину с твердым намерением программировать. Тоня потребовала прекратить бесполезные разговоры и приступить к делу.

— Разговоры не так уж бесполезны, как тебе кажется. Просто ты, как и все маленькие дети, плохо понимаешь, что для тебя хорошо, а что плохо.

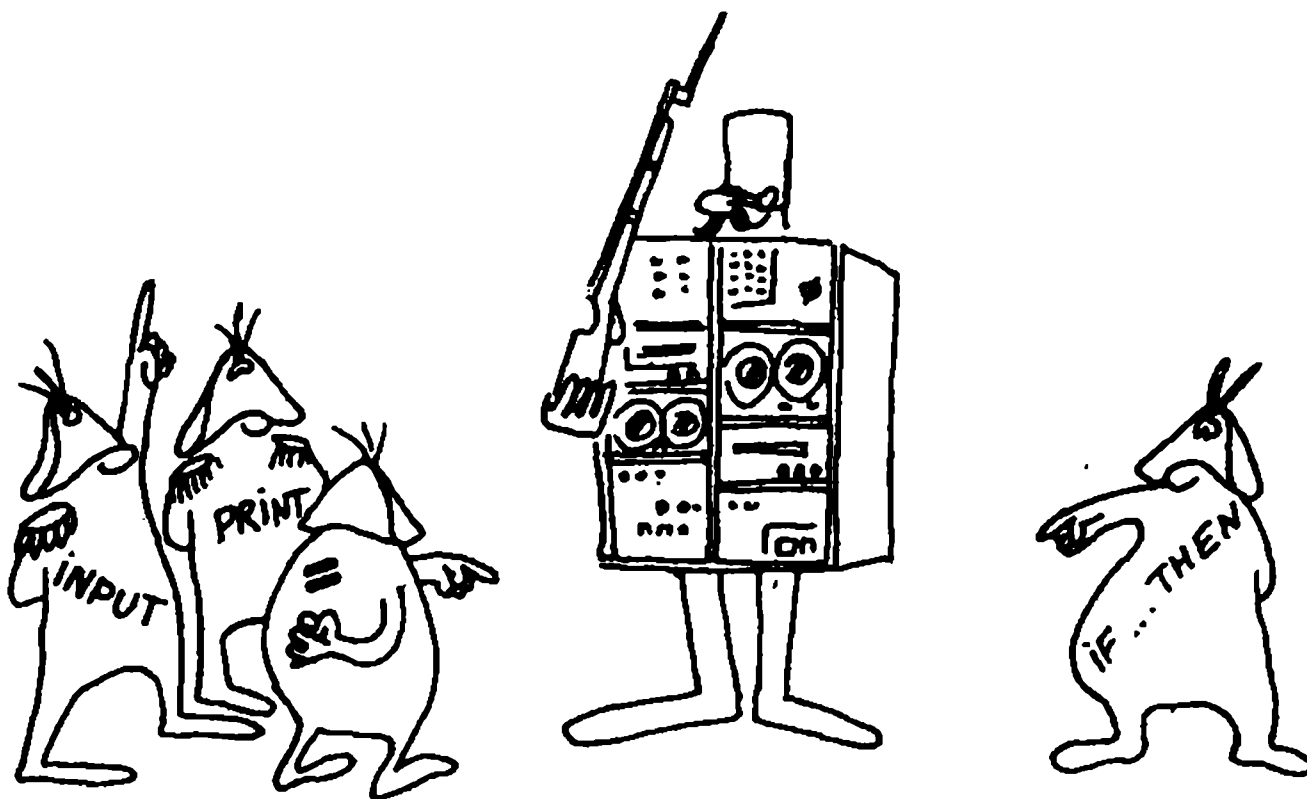
— Кончай болтать и говори, что еще нужно, чтобы начать программировать.

КИРПИЧИКИ ПРОГРАММ — ОПЕРАТОРЫ

— Нужно совсем немного. Сперва запомни, что языки программирования состоят из операторов и каждый оператор — это просто приказ машине что-то сделать. Операторы являются составными частями языков программирования, примерно так же, как слова в русском языке. Только операторов намного меньше, чем слов. Основу программирования составляют четыре оператора. В прошлый раз ты уже познакомилась с двумя из них и мы уже употребляли слово «оператор». Так что осталось всего два.

— Я в прошлый раз те два оператора как-то не заметила. Один — это PRINT. Да?

Оператор



Команды операторов для машины — закон

— Да. Один из них — оператор вывода PRINT. А другой — это оператор присваивания. Когда ты писала $A=12$, ты на самом деле вводила оператор присваивания — давала машине команду присвоить переменной A значение 12. Можно сказать, что оператор присваивания обозначается знаком равенства «=», слева от которого нужно писать ту переменную, которой присваиваем значение, а справа — само это значение в виде числа или арифметического выражения, которое надо вычислить. Помнишь, мы говорили об арифметических выражениях в операторе вывода PRINT? Полностью то же самое можно сказать и о выражениях, которые разрешается писать справа от знака равенства.

Присвоение

ВСЕ О ПЕРЕМЕННЫХ

Наверное, следовало бы сказать, что такое переменная, но я затрудняюсь это сделать. Ну... просто это — нечто, имеющее значение и участвующее в каких-нибудь операциях или действиях, в которых это значение может измениться, а может и не измениться. И еще переменная имеет обозначение, или, как часто говорят, имя. Например, в последнем примере обозначение переменной — буква « A », а значение — число 12.

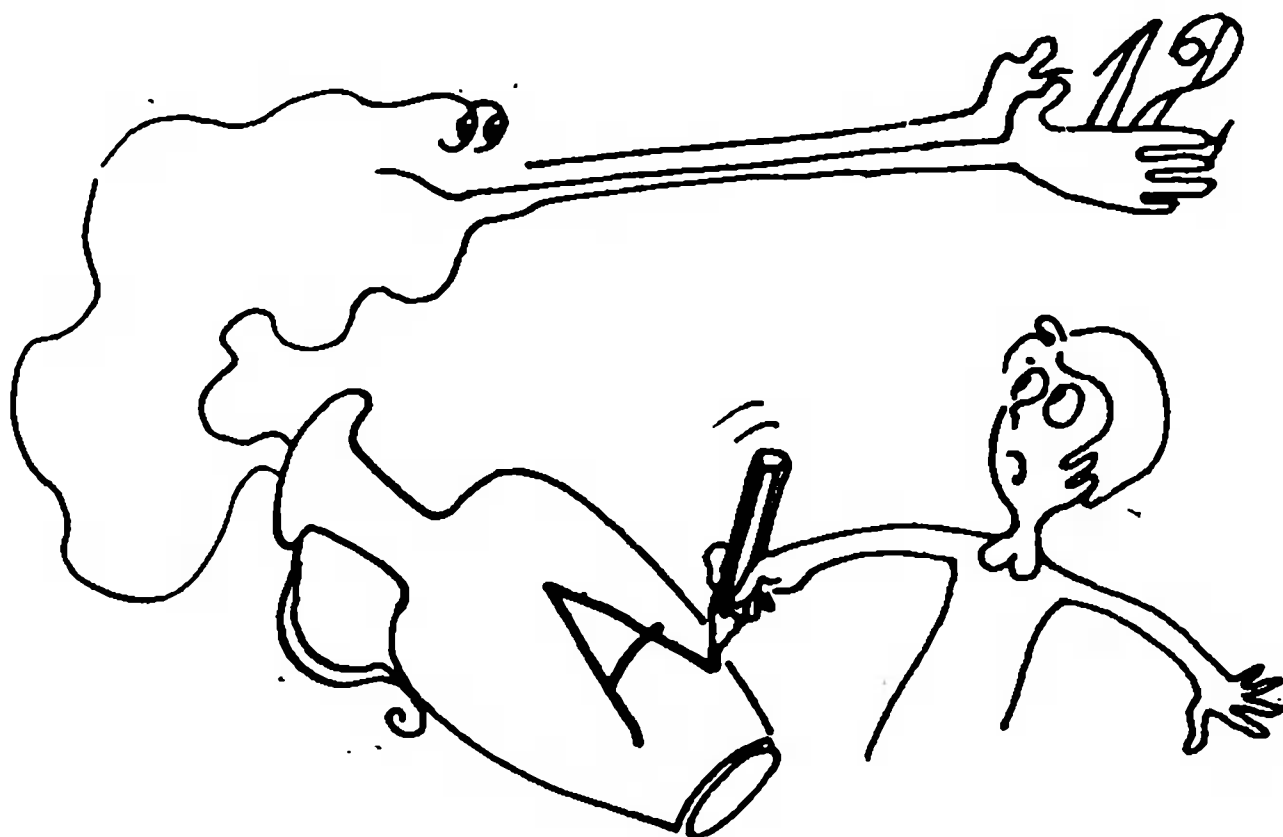
Переменная

— Этого я не понимаю. У тебя получается нечто такое: переменная A имеет обозначение « A ». Это же какая-то бессмыслица.

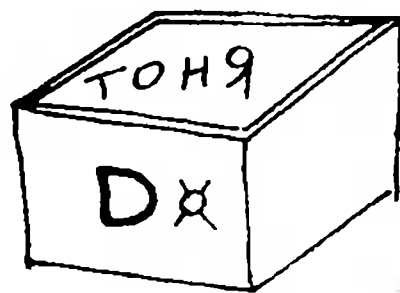
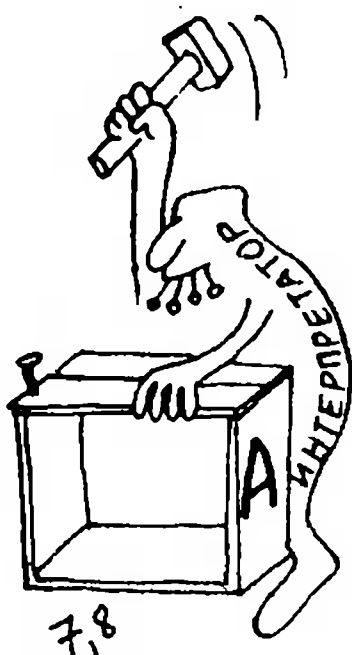
— А это ты понимаешь: «Ученица Соображалкина имеет фамилию Соображалкина»?

— Это-то я понимаю.

— Ну, вот. Понятно ведь, что ученица и ее фамилия — вещи совершенно разные. И совершенно так же, как ученица имеет фамилию, переменная имеет обозначение, или имя.



$A = 7,8$



Различие между ученицей и переменной, конечно, имеется — хотя бы в том, что Соображалкина существует независимо от того, как ее называют и называют ли вообще, а переменная в памяти машины начинает существовать только в тот момент, когда ее как-то называют, другими словами, обозначают, когда она в первый раз появляется слева от знака равенства в операторе присваивания.

— Ой, надо как-то попроще!

— Ну, тогда вообрази, что интерпретатор для каждой определяемой переменной в памяти отводит место, где хранит ее значение, или, еще проще, что каждая переменная — это ящичек, на котором написано обозначение, а внутри хранится значение. И когда значение переменной присваивается в первый раз, то такой ящичек с надписью и со значением возникает как будто по мановению волшебной палочки.

Ничего страшного, правда, не случается, если программист ошибется и в каком-либо арифметическом выражении использует переменную, которой еще не присваивалось значение. Интерпретатор принимает, что оно равно нулю.

БЕЙСИК МОЖНО ПЕРЕВОДИТЬ НА РУССКИЙ

Сейчас наш знаменитый оператор присваивания $A = 12$ переведем с Бейсика на русский язык. Получится что-то вроде

A	$=$	12
Переменной с обозначением «A»	присвоить значение	двенадцать

На Бейсике

По-русски

Правда, обычно программисты говорят не «переменная с обозначением «А», а просто «переменная А».

Что еще? Могу рассказать, что переменные могут обозначаться любой буквой латинского алфавита и цифрой после нее — двумя знаками либо только буквой. Таковы правила игры нашего Бейсика.

Скажи, пожалуйста, которые из этих обозначений в Бейсике непригодны:

А, В, С, E1, E2, E9, E22, K0, АВ.

— Так, E22, потому что содержит три знака... и АВ, потому что содержит больше одной буквы... а K0, может, и пригоден, если считать нуль цифрой...

— А как же иначе! Если подсчитать все возможности по образованию имен переменных, то получается, что всего различных имен может быть 286.

— Не мало ли?

— Тебе хватит.

— Расскажу, как я это все понимаю. Когда я ввожу в интерпретатор строку $A=12$, он работает примерно так. Сначала проверяет, что у него находится слева от знака равенства...

— Нет, скорее он смотрит, есть ли у него вообще знак равенства во введенной строке.

— Не мешай думать! Если слева от знака равенства находится буква или буква с цифрой, то ясно, что хотят что-то вычислять... а что именно, можно узнать из выражения справа от знака равенства. Ага! Когда вычислили нужное выражение, надо определить, новая эта переменная или уже встречалась раньше, нужно ли создавать для ее значения ящик или такой уже есть.

— Ты очень мудро рассуждаешь. Я могу еще рассказать, что произойдет, если интерпретатор обнаружит ошибку. Хотя нечего об ошибках рассказывать, лучше их сделать. Сначала вызовем интерпретатор. А сейчас сделаем такую ошибку.

```
A12=12
```

```
?SYNTAX ERROR
```

```
READY
```

— Синтаксическая ошибка.

— Да. Это означает, что ты ввела строку, которую интерпретатор при всем желании не может признать за строку в языке Бейсик. Ты бы на месте интерпретатора сказала: «Чушь собачья!»

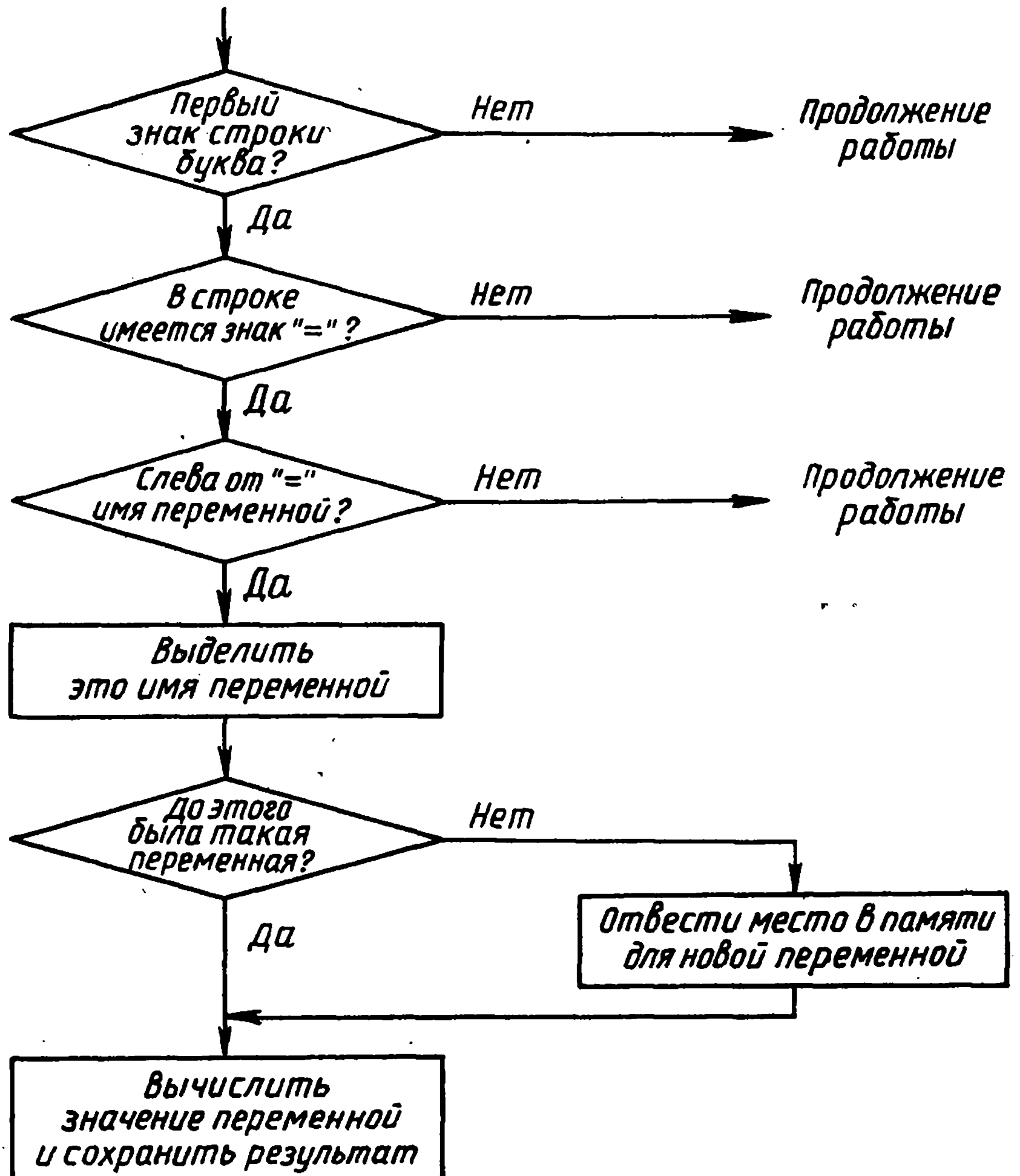
— Слушай! Ты что не знаешь, что учитель не имеет права оскорблять человеческое достоинство ученика?

— Извини. Посмотри, что будет, если ошибиться и написать неправильное выражение.

$A = ((A+B)+C)$
SYNTAX ERROR

READY

Опять получили сообщение об ошибке. Теперь я предлагаю нарисовать схему алгоритма, по которому интерпретатор обрабатывает оператор присваивания. Если рисовать точно, она получилась бы очень большой, но приблизительно она будет такой.



— Приблизительность в том, что ты написал «Продолжение работы» там, где надо было писать «Вывод сообщения об ошибке»?

— Не совсем. Мы просто еще ничего не знаем о Бейсике. Например, если первый знак введенной строки оказался цифрой, это еще не означает, что строка введена неверно, хотя и имя переменной не может начинаться с цифры. Это ты скоро узнаешь, сразу после того, как поговорим об операторе вывода. Переведем его на русский язык.

PRINT	A, B, C
Вывести	значения переменных A, B, C

На Бейсике

По-русски

— Этот перевод совсем простой.

— Какой уж есть. Про оператор вывода еще важно знать, что этим оператором можно выводить не только числа, но и другие знаки, например буквы. Можно выводить любой текст. Надо делать так: после слова PRINT написать нужный текст в кавычках.

— Что бы такое вывести на экран? .

```
PRINT "ПРИВЕТ"  
ПРИВЕТ
```

— Выводимый текст можно заключать еще и в апострофах. Эта вторая возможность придумана для того, чтобы можно было бы выводить текст, содержащий кавычки. Например, такой:

```
PRINT ' "ПРИВЕТ" '  
"ПРИВЕТ"
```

Ведь понимаешь же, что кавычки в операторе вывода указывают границы выводимого текста, а сами не выводятся?

— Понимаю, понимаю.

— А это понимаешь, что вот так, в непосредственном режиме, нет никакого смысла выводить какой-нибудь текст на экран при помощи оператора вывода? Ты же могла бы просто на экране написать слово ПРИВЕТ, и никакой возни с Бейсиком. Обычно текст на экран выводят во время работы программы, чтобы сообщить что-то о результатах ее работы.

— Интересно! Как я это могу понять, если ты еще ни слова не сказал о программах!

КАК ПОЛУЧИТЬ ПРОГРАММУ

— Уже говорю. Программа получается, если писать операторы с номерами в начале строки. Их так и называют — номера строк. Понимаешь, если ты пишешь оператор без номера, то он

выполняется сразу — это непосредственный режим, с которым ты знакома. Но если оператор ввести с номером в начале строки, то этот оператор запоминается и вместе с другими, подобным образом пронумерованными операторами, создает программу в оперативной памяти. Запомни крепко-накрепко — в памяти! Начинаящий плохо представляет, где у него программа — в памяти машины или на диске.

— Поняла! Поняла, почему ты при обсуждении алгоритма обработки оператора присвоения что-то бессвязно бормотал о строках, первый знак которых цифра.

— Верно. Если первый знак введенной строки — цифра, интерпретатор эту строку воспринимает как строку программы, а если нет, то как строку, выполняемую немедленно.

— Надо бы попробовать с этими номерами. . . Скажи, пожалуйста, любое ли число может быть номером?

— Любое целое число не больше $2^{15} - 1 = 32767$. И перед тем как начинать вводить программу, нужно ее как-то назвать и название сообщить интерпретатору — пишешь слово NEW и название программы.

— «Нью» по-русски «новый», я полагаю.

— Разумеется. Вводи.

NEW SUPER_

— Так будет хорошо?

— Умопомрачительно! Да, название программы может содержать до девяти букв и цифр, но первый знак должен быть буквой.

— Послушай, а NEW — это новый оператор?

— Не совсем. Это не оператор, а директива языка Бейсик. Отличие от оператора в том, что директивы что-то делают с самими программами и выполняются только в непосредственном режиме. А операторы работают в программах с входными данными и значениями переменных.

Хорошо. Программа названа, можешь начинать ее вводить. Сначала напиши номер строки. . .

1 N_

А почему начинаешь с единицы?!

— Как же иначе? Раз, два, три. . . Так и буду нумеровать.

КАК НУМЕРОВАТЬ СТРОКИ

— Видишь ли, так, конечно, можно, но так не делают. Программисты считаются с тем, что программу потом придется переделывать, и притом не один раз: что-нибудь будет непра-

вильно, что-нибудь придумаешь нового, и может оказаться, что нужно вставлять еще одну строку между, скажем, строками с номерами 2 и 3. А ты не сможешь, потому что дробных номеров не существует. Поэтому строки обычно нумеруют не подряд, а с каким-то шагом, например с шагом десять, т. е. так: 10, 20, 30 и т. д. При выполнении программы закон такой: операторы выполняются не в том порядке, как ты их вводила, а в порядке возрастания номеров строк, кроме случаев, когда специально указывается, какую строку выполнять следующей. Если имеются строки с номерами 10 и 20, а нет строк с промежуточными номерами, то никакого шума интерпретатор не поднимает, и если потом появляется необходимость вставлять строку с промежуточным номером, то это спокойно можно сделать, просто вводя строку с желаемым номером. Совершенно все равно, в каком порядке вводятся строки программы.

— Хорошо, могу и так вводить. А что делать с этой недо введенной строкой?

— Эту строку можно погасить, но лучше это вместо тебя сделаю я.

— Начну тогда с номера 10.

```
10 A=12
20 B=20
```

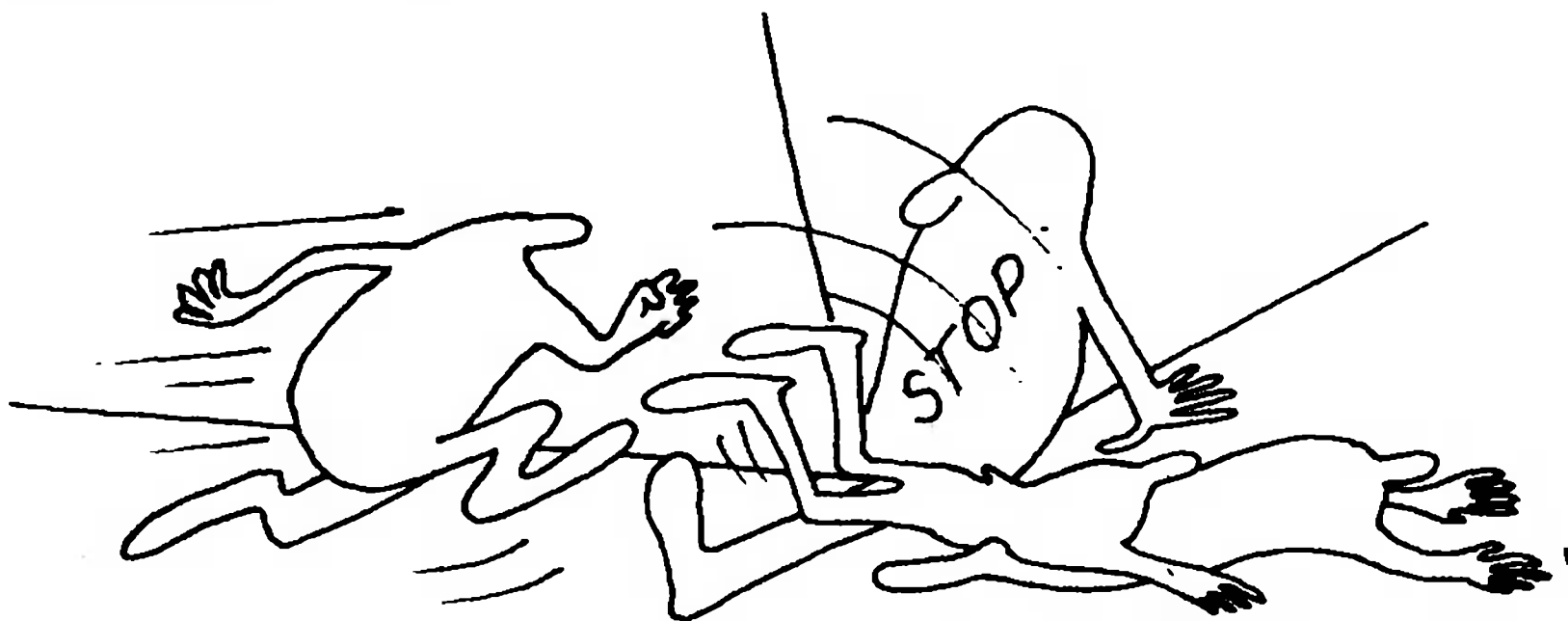
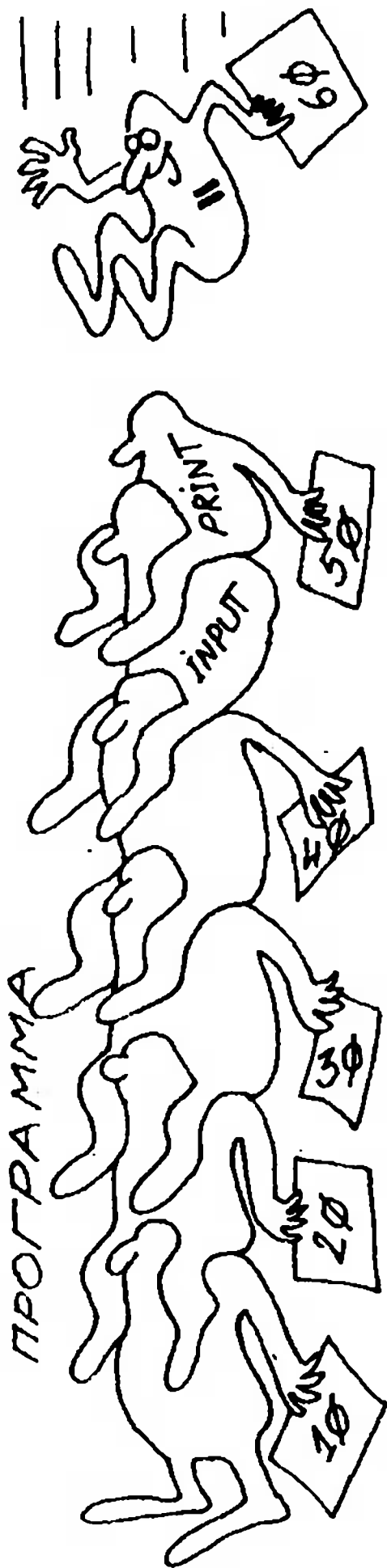
Ой, после номера строки 20 я забыла оставить пробел! Надо опять погасить строку.

— Собственно говоря, не было речи о том, что после номера нужен пробел.

— Мне кажется, так было бы естественнее.

— Для интерпретатора естественно то, что правильно. Но можешь не волноваться: пробел после номера интерпретатор поставит сам. Более того, ты можешь в любом месте вставлять пробелы и можешь не вставлять, все равно лишние интерпретатор выбросит и нужные добавит. Так что продолжай.

```
30 C=A+B
40 PRINT A,B,C
```



Программу останавливает оператор STOP

- Вот и все.
- Еще желательно добавить оператор END. Что это означает по-русски?
- «Энд» означает «конец» или «кончать».
- Да, END означает, что программа кончается, что больше в ней операторов нет, или, как говорят программисты, указывается физический конец программы. Интерпретатор, встретив этот оператор, прекращает работу программы. У программы еще может быть логический конец — когда ее работу останавливает оператор STOP, который может находиться и в середине программы. Вот тебе еще два оператора, но так как они не являются основными, то ты еще не закончила обучение программированию.

50 END

- Оператор END так надо вводить?
- Именно так. В этом простейшем случае логический конец и физический конец программы совпадают.

ПРОВЕРКА ВВОДА...

- Итак, программа введена. Хотя ты и видишь на экране все введенные строки, но лишняя проверка правильности ввода не повредит. Имеется возможность посмотреть всю программу, которая находится в памяти, при помощи директивы LIST. «Лист» означает «составлять список», но это по словарю, у нас же это означает «показать программу». Это будет у тебя вторая директива языка. Директива потому, что работает с самой программой.

LIST

SUPER 21-NOV-90 14:35:20

```
10 A=12
20 B=20
30 C=A+B
40 PRINT A,B,C
50 END
```

- Да, ты был прав. Сейчас после номера 20 пробел уже есть.

- Никто не сомневался. Еще, как ты видишь, перед текстом программы выведены название программы, сегодняшнее число и текущее время. Чтобы эта строка не показывалась, нужно использовать директиву LISTNH.

...И ЗАПУСК ПРОГРАММ

— Раз все хорошо, запускаем программу. А делается это директивой RUN.

— Ха! «Ран» — это же «бежать».

— Не только бежать, но и работать — так говорят о машинах и программах. После ввода этой директивы и начинается работа интерпретатора — просмотр строк программы и выполнение операторов. Вводи.

RUN

SUPER 21-NOV-90 14:46:30

12

20

32

READY

Как видишь, все в порядке. Сначала интерпретатор выводит название программы, а потом результаты работы самой программы. Если выводить название программы тебе не надо, то можешь использовать вариант директивы RUNNH. Когда программа свою работу кончает, выводится сообщение интерпретатора READY. Значения переменных находятся в своих ящиках и после окончания работы программы, так что до них еще можно добраться. Их можно проверить и даже изменить, используя возможности непосредственного режима.

— Интересно, какие это возможности я бы могла использовать?

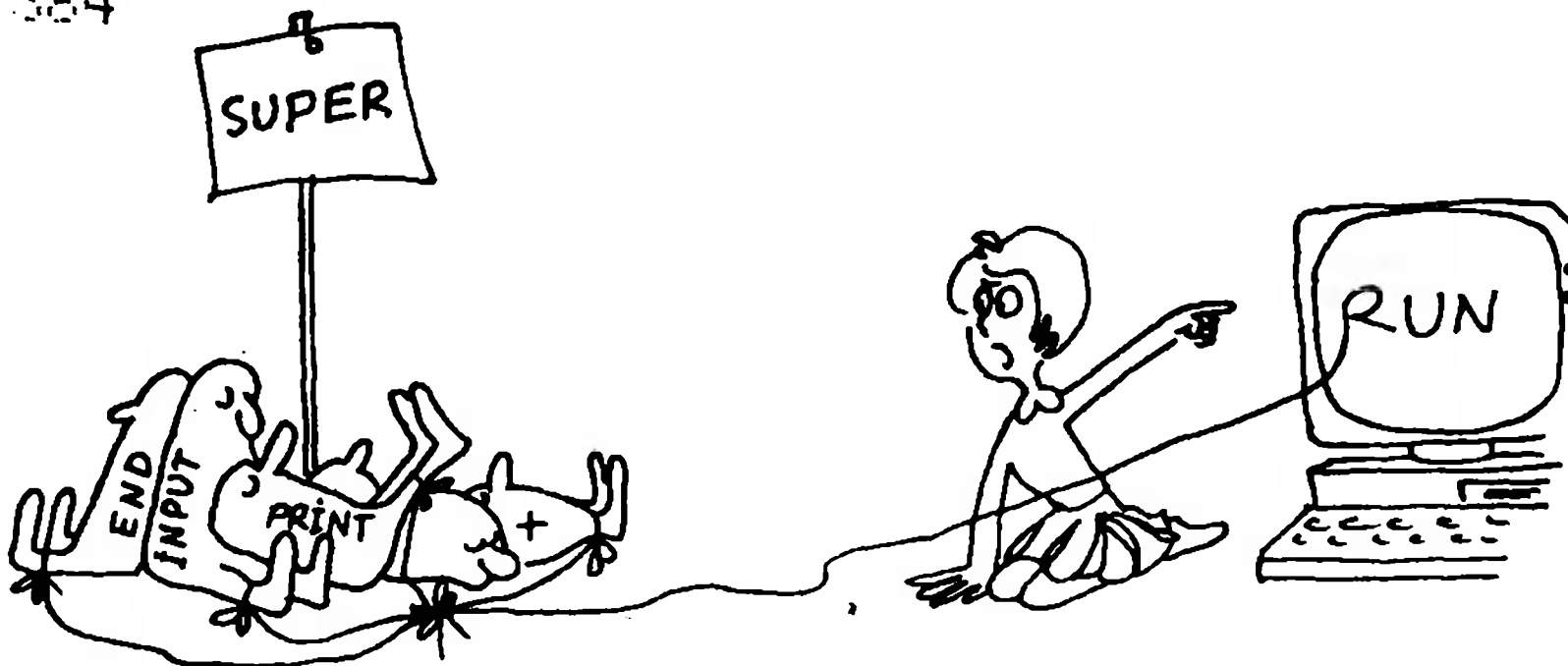
— Например, вывести произведение A и C.

— Оператором PRINT?

— Разумеется.

PRINT A*C

384



Все за работу!

- Так, дай подумать. Результат правильный.
- Еще бы!
- Поймай! У меня есть идея!

A=10

```
READY
PRINT A,C
10
```

32

Я подумала, что, возможно, при изменении значения A будет меняться и значение C. Ведь C — это же сумма A и B.

— Ну, нет! Для того чтобы получить новую сумму, нужно заново пропустить программу, и притом с новым значением A, а так как значение этой переменной присваивается в строке 10, то эту строку нужно поменять. Менять строку крайне просто: нужно с тем же номером ввести новую, нужную строку.

```
10 A=10
```

А сейчас можно запускать. Запускай, используя RUNNH.

```
RUNNH
10
```

20

30

```
READY
```

— Не очень-то удобно делать такие замены строк, когда необходимо просчитать с другими числами.

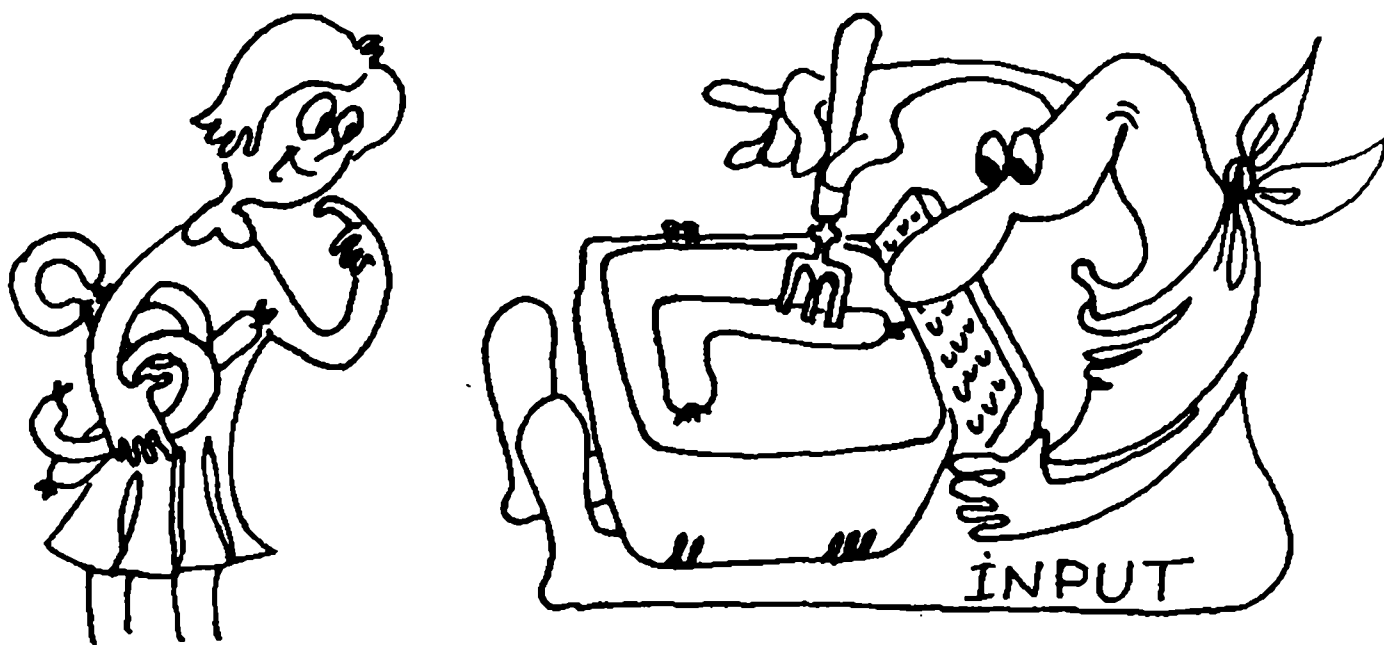
ВВОД ВО ВРЕМЯ РАБОТЫ ПРОГРАММЫ

Оператор ввода

— Разумеется. Если нужно пропускать программу на Бейсике многократно, всякий раз с другими числами или при составлении программы еще неизвестно, с какими числами нужно будет работать, то пользуются оператором ввода INPUT. Это будет новый — третий основной оператор.

— «Инпут», «инпут»... что же это значит?

— Это значит «ввод» или «вводить» еще и «положить». Каждый раз, когда интерпретатор в программе встречает оператор ввода, он заставляет машину ждать ввод с дисплея и, чтобы ты знала, что машина чего-то ждет, на экран выводится вопросительный знак. А что именно она должна ждать, значение какой переменной, это программист должен написать сразу после слова INPUT. Если ты собираешься в этом месте ждать ввод нескольких переменных, то их имена ты должна



Данные с экрана
забирает INPUT

записать через запятую. Конечно, и строку с оператором ввода можно перевести на русский.

INPUT	A, B
Дождаться ввода с дис- плея	значений переменных A и B

На Бейсике

По-русски

— Ну что, попробую.

INPUT A

?ILLEGAL IN IMMEDIATE MODE

READY

Что-то совершенно непонятное.

— Ты поспешила. Оператор ввода в непосредственном режиме использовать нельзя. Именно это тебе и сообщает интерпретатор. Оператор INPUT можно использовать только в программе, т. е. впереди него должен быть номер строки.

— В моей программе этот оператор ввода должен быть, я думаю, в десятой строке. Я будто бы вместо присваивания значения переменной A вставляю оператор ввода... Так?

— Присваивание значения переменной A у тебя происходит в первой, а не в десятой строке. В твоей программе вообще нет десяти строк.

— Ладно. В строке с номером десять. Ты хочешь, чтобы я так говорила?

— Да. Думаю, что так будет точнее; еще можно говорить «в строке 10». Но вообще ты права, действительно вместо этой строки можно вставить оператор ввода.

10 INPUT A

Сейчас запусти программу.

RUNNN

?_

Так. Появился вопросительный знак — машина стоит и ждет значения переменной А. Можешь вводить нужное значение.

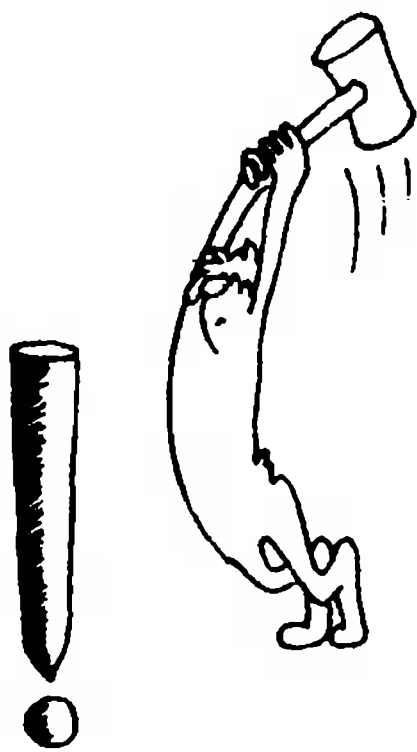
7500

500

20

READY

— Ясно. Теперь, конечно, эту программу можно пускать много, много раз, но я должна признать, что ее интеллектуальный уровень оставляет желать лучшего. Вряд ли она будет нужна народному хозяйству.



Файл

ЗАПИСЬ ПРОГРАММЫ ИЗ ПАМЯТИ НА ДИСК

— Ничего, ничего. У тебя еще все впереди. А теперь, пока наше время не кончилось, я хочу тебе рассказать, как сохранить программу на диске.

— Вот это надо! Эта историческая программа должна храниться вечно.

— Это не так быстро делается. Сначала нужно иметь свободное место на диске для записи программы. А сейчас запомни новое слово: место на диске, имеющее название и отведенное для хранения чего-то, хотя бы программы, называется файлом. Откровенно говоря, я не сразу понял, что такое файл, да, может быть, и ты не сразу оскоишь это понятие, но истина такова: хочешь программировать — знай, что такое файл. А программы на диске сохраняются директивой SAVE.

— «Сейв», по-моему, означает «спасать». Кого тут спасать?



Программу на диске сохраняет SAVE

— «Спасать» у нас «сохранять», но и «спасать» не так уж и плохо — спасать от исчезновения. После SAVE идет описание файла, т. е. сведения о нем: как ты обозначаешь файл, в котором собираешься писать программу, и где он должен быть образован. С обозначением просто, файл для сохранения программы может иметь такое же название, как и сама программа. С местом, где хранить файл, потруднее, здесь надо знать, как эти места принято описывать в операционной системе. Сегодня, чтобы сохранить свою программу, тебе хватит самого простого описания, которое я сейчас и покажу:

Двоеточие
DK1: SUPER

Код устройства Номер устройства Обозначение файла

Описание файла

Первым указывается код устройства, на котором создается файл. Этот код всегда состоит из двух букв, а если файл создается на диске, то эти буквы будут такими: DK. Далее следует номер устройства, ибо на машине может быть несколько совершенно одинаковых устройств и их нужно пронумеровать, чтобы отличить друг от друга. После номера ставится двоеточие. Оно фактически означает конец кода устройства. И наконец, после двоеточия следует обозначение файла, которое может содержать до девяти знаков — букв и цифр.

— Ух, как сложно!

— Совсем не сложно. Это просто, как программировать на Бейсике! Покажу-ка я тебе несколько примерчиков.

Будут ли правильными следующие описания файлов?

DK0:C

— Вроде правильно.

— В самый раз! Учти, что нулевой номер устройства можно и не писать.

DK3:PARA

— Правильно.

— Если машина имеет четыре дисковых устройства: нулевое, первое, второе и третье.

DK1:TONJA 12345

— Нет, не подойдет, потому что слишком много знаков.

LP:

— Тут я молчу. Что это значит?

— Немудрено, что не знаешь. Ведь я тебе про такой код устройства ничего не говорил. Но о том, что это именно код устройства, ты могла бы и догадаться по двум буквам и двоеточию. Это код печатающего устройства.

ЗАПИСЬ ПРОГРАММЫ ИЗ ПАМЯТИ НА БУМАГУ

Этот код нужно задавать в директиве SAVE, если хочешь сохранить свою программу на бумаге, т. е. напечатать ее. Нужно указать только код печатающего устройства и больше ничего. Попробуй напечатать свою программу.

```
SAVE LP: _
```

— Это, очевидно, все. Я поняла, что больше ничего писать не надо.

— Конечно, нет! При записи на диск нужно было бы указывать еще и обозначение файла, чтобы операционная система в другой раз смогла найти его. А на печатающем устройстве ничего такого делать не придется, да и сделать нельзя, ибо напечатанную программу ты унесешь домой. На мой взгляд, директива SAVE очень похожа на директиву LIST, только первая программу записывает на бумаге, а вторая выводит на экран.

— Все! Нажимаю клавишу ввода.

```
10 INPUT A
20 B=20
30 C=A+B
40 PRINT A,B,C
50 END.
```

Ой, как красиво напечатала!

— Сохрани это на память. Внукам покажешь, на каких машинах в молодости работала. А теперь живо запиши программу на диск, и бежим домой, уже поздно. Вместо LP: укажи устройство и обозначение файла.

— А номер диска? Все равно какой указывать?

— Нет, нет! Можно указывать только тот, на котором имеется свободное место и тебе разрешают писать. На сей раз никаких проблем нет, так как на дисковом устройстве номер 1 стоит выделенный для меня диск, на котором свободного места много. Так что пиши на мой диск.

```
SAVE DK1: _
```

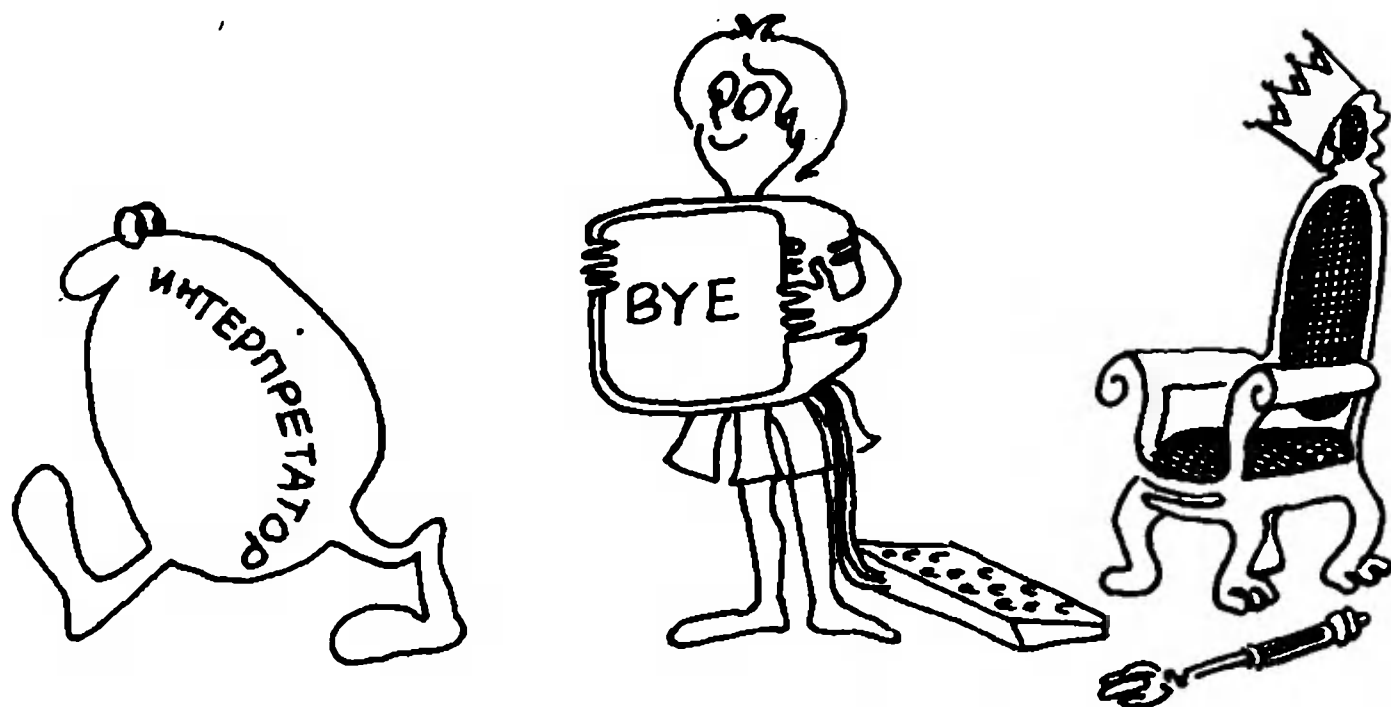
— Какое же обозначение файла я должна сейчас написать?

— Как какое? Какое хочешь, такое и пиши! Только постарайся его не забыть, чтобы в следующий раз, когда захочешь, смогла бы найти свою программу.

— И как я ее найду?

— Об этом узнаешь, когда нужно будет искать. Придумывай же обозначение файла!

```
SAVE DK1:PERUAJA_
```



Отдохни
немнож-
ко!

— Файл с моей первой программой логично назвать PERVAJA.

— Нормально. Нажми клавишу ввода, и твоя программа запишется на диск. И под занавес запомни еще одну директиву: BYE — «бай», что я перевожу «Пока!». Работу всегда нужно заканчивать этой директивой, чтобы интерпретатор Бейсика кончил работу и, как говорят профессионалы, «передал управление обратно операционной системе».

Теперь можем спокойно повозиться с упражнениями.

1. Записать на Бейсике следующие выражения:

$$\frac{a-x}{a+x}, (a-e)(a+e), \frac{(a+x)}{a(k+x)}.$$

2. В каждой из следующих строк имеется ошибка. Какая?

```
10 12=A
50 A+B=C
20 A1=12,3
42 PRINT A=12
45 H=X(A+B)/6
300 M=((A+X)+H/C
```

3. Разобраться в этой программе:

```
10 A=25
20 B=13
30 C=14
40 A=(A+B+C)/3
50 PRINT A
60 END
```

4. Написать программу, которая выводила бы на экран какое-нибудь слово гигантскими буквами. Например, такое:

```
TTTTT   OOO   H   H   яяя
  T   O   O H   H   я   я
  T   O   O H   H   я   я
  T   O   O H   H   яяяя
  T   O   O HHHHH   я   я
  T   O   O H   H   я   я
  T   OOO   H   H   я   я
```

8. НЕБОЛЬШАЯ ПОЛЬЗА ОТ ПРОГРАММИРОВАНИЯ

— Ну что, Тоня? Ты знаешь только несколько операторов, но их достаточно, чтобы написать маленькую полезную программу. Конечно, польза будет совсем небольшая, но начинать надо с небольшого. Ты Джека Лондона читала?

— Я? Да... А зачем тебе это?

— Я сейчас его читаю, рассказы о приключениях на Аляске. Между прочим, температуру воздуха он всегда дает в градусах Фаренгейта. Я знаю, что эти градусы превратить в привычные градусы Цельсия совсем несложно, но запомнить нужные числовые константы никак не могу. Напиши небольшую программу для таких, как я. Алгоритмыч как-то смеялся, что все программисты начинают с этой программы и в каждой книжке по программированию она приводится.

— С удовольствием; но что программа должна делать?

ГРАДУСЫ ФАРЕНГЕЙТА И ЦЕЛЬСИЯ

— Эта программа должна, получив при помощи оператора ввода температуру в градусах Фаренгейта, вывести на экран температуру в градусах Цельсия. Формулу перевода из градусов Фаренгейта F в градусы Цельсия C я для тебя выписал из

книги:

$$C = \frac{5}{9}(F - 32).$$

— Немножко подумаю... Мне кажется, что это очень простая программа. Я ее мигом введу.

— Не спеши. Сначала придумай название и сообщи интерпретатору о своем желании вводить новую программу директивой NEW.

— Я назову эту программу CELS.

NEW CELS

```
READY
10 INPUT F
20 C=(5/9)*(F-32)
30 PRINT C
40 END
Вот и все. Можно запускать?
```

— Пускай, пускай.

RUNNN

?

Введи —50. Знаешь, почему меня интересует это значение? Джек Лондон пишет, будто бы при этой температуре плевков превращается в лед еще до того, как достигнет поверхности снега.

— Бр-р-р. Ну и интересы у тебя!

? -50

-45.5556

READY

— Ясно. Я думал, температура будет ниже. А что касается твоей программы, то ее даже можно назвать первоклассной в том смысле, что такие делают первоклассники.

— Ты что! Чего тебе еще надо?

— Спокойно. Если посмотреть на текст программы, то совершенно непонятно, что она делает. . .

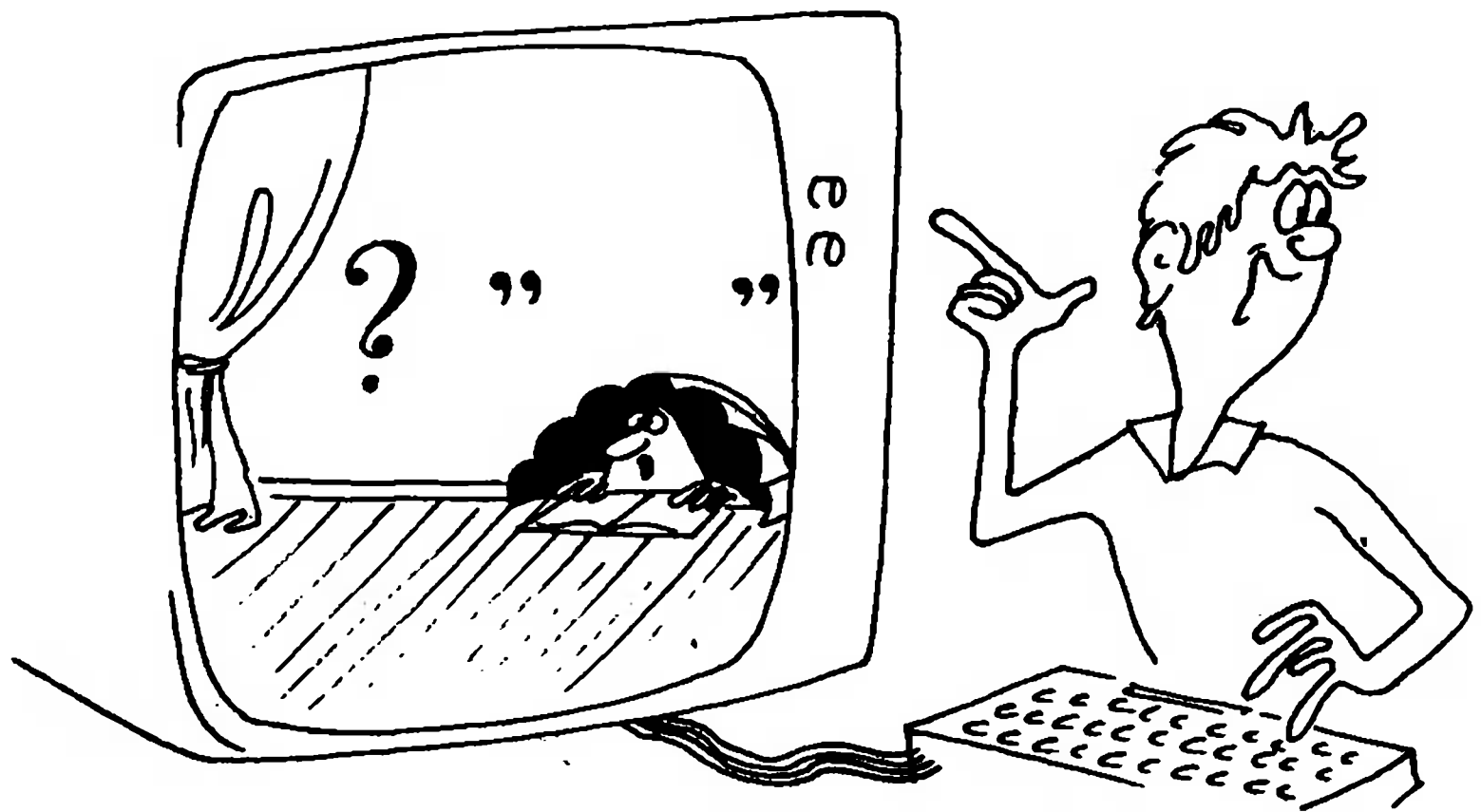
— А ты что, не знаешь?

ПРОГРАММА ДОЛЖНА БЫТЬ ПОНЯТНОЙ ДЛЯ ВСЕХ!

— Я-то знаю, но пойми, программы почти всегда делаются для других, а если кто-то другой посмотрит на эту программу, то ничего не поймет. Конечно, ты можешь здесь целыми днями сидеть и всем объяснять, что к чему, но это же не выход, долго ты не выдержишь. Если же ты и сделала программу только для себя, то все равно через пару месяцев напрочь забудешь, что же она делает. Таков горький опыт всей истории программирования. . . А во время работы программы? Показывается вопросительный знак, нужно что-то ввести, а что именно, догадайся сам! Ладно еще, что в твоей программке предусмотрен ввод только одного числа. А если нужно будет вводить разные числа в разное время? Ты со своими вопросительными знаками запутаешь любого человека, который захочет пользоваться твоей программой. В общем привыкай с юных лет создавать удобства пользователю.

— Совершенно не понимаю, чего ты хочешь! Ты же сам говорил, что без вопросительных знаков не будет ясно, нужно ли уже вводить или еще нет, а сейчас издеваешься над ними и надо мной тоже. Ничего же другого сделать нельзя. По крайней мере ты ничего другого мне не рассказывал.

— Если ты умеешь выводить текст оператором PRINT, то ничего другого и не надо. Сначала выведи на экран что-нибудь такое: «Работает программа, переводящая градусы Фаренгейта в градусы Цельсия». Потом подсказку пользователю: мол, введите градусы Фаренгейта, а результат выводить будешь —



Подсказывать обязательно!

не одно голое число выводи, а добавь текст, что же это за число, а именно что градусы Цельсия.

— Да-а-а?.. Так это в два раза больше работы будет!

— Возможно, что и во все четыре. Но позволь спросить, зачем тебе программирование, если легкой жизни хочется?

— Ух, язва! Намучаюсь же я с тобой!

Роптала Тоня, роптала, но в конце концов сделала, как ей Петя говорил: Получилось не очень быстро: и подумать пришлось, и кое-какие ошибки исправлять, потому что нет-нет да и нажмет она не ту клавишу. Но когда все было готово, ввела она директиву LISTNH и на экране появилась ее программа.

```
5 PRINT "ПРОГРАММА ПРЕОБРАЗУЕТ ГРАДУСЫ"
6 PRINT " ФАРЕНГЕЙТА В ГРАДУСЫ ЦЕЛЬСИЯ"
7 PRINT " ВВЕДИТЕ ТЕМПЕРАТУРУ"
8 PRINT " В ГРАДУСАХ ФАРЕНГЕЙТА"
10 INPUT F
20 C=(5/9)*(F-32)
25 PRINT "ЭТО БУДЕТ"
30 PRINT C
35 PRINT "ГРАДУСА ЦЕЛЬСИЯ"
40 END
```

— Ну, посмотри, мучитель, на этот шедевр программирования!

— Колоссально! Но надо посмотреть, как она работает. Запусти ее и задай перевести те же самые —50 градусов Фаренгейта.

RUNNN

ПРОГРАММА ПРЕОБРАЗУЕТ ГРАДУСЫ

```
ФАРЕНГЕЙТА В ГРАДУСЫ ЦЕЛЬСИЯ  
ВВЕДИТЕ ТЕМПЕРАТУРУ  
В ГРАДУСАХ ФАРЕНГЕЙТА  
? -50  
ЭТО БУДЕТ  
-45.5556  
ГРАДУСА ЦЕЛЬСИЯ
```

READY

Результат тот же самый, следовательно, ты по крайней мере ничего не испортила. Ну и как, красивей сейчас выводятся результаты?

ТОНЯ ЕЩЕ УЛУЧШАЕТ ВЫВОД

— Красивей-то красивей, но можно было бы еще краше. Не нравится мне, что основное сообщение о работе программы и просьба ввести температуру идут сплошным текстом. Я бы хотела, чтобы эти сообщения разделяла пустая строка, это же совершенно разные сообщения. Но так, наверно, сделать нельзя.

— Почему же. Ты можешь вставить просто PRINT без ничего, и выведется пустая строка.

— Ясненько. . . Но подожди! Я там ничего не могу втиснуть между строками с номерами 6 и 7. Так что же делать? Вводить всю программу заново с другими номерами?

— Разумеется, нет. В таких случаях нужно применить директиву RESEQ. Я это слово произношу «рисек», и, по-моему, оно является сокращением от английского слова resequence, что должно означать «заново упорядочить последовательность». В словаре я этого слова не нашел, хотя искал. Так вот, эта директива вызывает перенумерацию всех строк программы, начиная с первой, которая получает номер 10, с шагом десять — 10, 20, 30 и до конца.

Перенумерация

— Интересно было бы попробовать.

```
RESEQ
```

READY

```
LISTNN
```

```
10 PRINT "ПРОГРАММА ПРЕОБРАЗУЕТ ГРАДУСЫ"  
20 PRINT " ФАРЕНГЕЙТА В ГРАДУСЫ ЦЕЛЬСИЯ"  
30 PRINT " ВВЕДИТЕ ТЕМПЕРАТУРУ"  
40 PRINT " В ГРАДУСАХ ФАРЕНГЕЙТА"  
50 INPUT F
```

```

60 C=(5/9)*(F-32)
70 PRINT "ЭТО БУДЕТ"
80 PRINT C
90 PRINT "ГРАДУСА ЦЕЛЬСИЯ"
100 END

```

ЗОНА
ВЫВОДА

Да, кое-что из твоих обещаний выполняется. А скажи мне вот что. Я хотела, чтобы у меня получилось красиво в одну строку: ЭТО БУДЕТ — 45.5556 ГРАДУСА ЦЕЛЬСИЯ, а все напечаталось в три строки. Можно что-нибудь сделать?

— Без сомнений. Во-первых, одним и тем же оператором ты можешь выводить и строки символов, и числа, так что смело могла писать: PRINT "ЭТО БУДЕТ" C.

— Понятно. И через запятую — "ГРАДУСА ЦЕЛЬСИЯ".

— Опять спешишь. Дело вот в чем. Единицы вывода в списке оператора вывода можно отделять друг от друга двумя способами: либо запятой, либо точкой с запятой. Если они, как в этом примере, записаны через запятую, то происходит вот что: строка экрана разделяется на зоны длиной 14 знаков. И в каждой зоне появляется значение одной единицы вывода. Или в нескольких, если одной не хватает. Надеюсь, ты понимаешь, что значения числовых переменных всегда смогут разместиться в одной зоне и только сообщения программы...

— Надейся, надейся.

— Вывод размещается в левой части зоны, и если он короче 14 знаков, то правая часть зоны остается пустой. Помнишь, в первой твоей программе при выводе значений переменных A, B, C между ними получилось довольно большое расстояние?

$\overbrace{\quad\quad\quad}^{14 \text{ знаков}}$ $\quad\quad\quad 12$	$\overbrace{\quad\quad\quad}^{14 \text{ знаков}}$ $\quad\quad\quad 20$	$\overbrace{\quad\quad\quad}^{14 \text{ знаков}}$ $\quad\quad\quad 32$
1-я зона	2-я зона	3-я зона

— Да! Я еще удивилась, но спрашивать не стала.

— Если отделять запятыми, то у тебя получится много пустого места между числом — 45.5556 и текстом ГРАДУСА ЦЕЛЬСИЯ. Если же единицы вывода отделять точкой с запятой, то их значения будут выводиться сплошной стеной, одно за другим.

— Ой, ой, как бессвязно ты говоришь!

— Можно еще бессвязней, если сказать, что было не совсем правильно говорить об отделении единиц вывода одной от другой. Правильнее было бы сказать, что запятая или точка с запятой, написанная после названия переменной, указывает интерпретатору, где проводить следующий вывод — в следующей ли зоне или в следующей ли позиции.

Спешу дать пример. Вот я напишу:

```

70 PRINT "ЭТО БУДЕТ";

```

Зона вывода

и значение переменной С, вывод которой у тебя запланирован в строке 80, выпечатается сразу после слов ЭТО БУДЕТ. А если запишем точку с запятой в конце строки 80, то слова ГРАДУСА ЦЕЛЬСИЯ выведутся сразу после значения переменной С в той же самой строке.

— Так я делать не буду. Я введу такой оператор, как и хотела, но только эти, как ты их называешь, единицы вывода отделию не запятыми, а точками с запятыми.

— Только учти, что никаких пробелов между единицами вывода машина сама вставлять не будет, это ты должна нужные пробелы спланировать и запрограммировать. На сей раз ты должна предусмотреть пробел после ЭТО БУДЕТ, так же как и перед ГРАДУСА ЦЕЛЬСИЯ.

— Очень уж хлопотно, но я попробую.

```
70 PRINT "ЭТО БУДЕТ ";C;" ГРАДУСЫ ЦЕЛЬСИЯ"
```

Можно пускать?

— Нет, нельзя. Хотя... пускай, посмотрим, что получится.

```
RUNNN
```

```
ПРОГРАММА ПРЕОБРАЗУЕТ ГРАДУСЫ
```

```
ФАРЕНГЕЙТА В ГРАДУСЫ ЦЕЛЬСИЯ
```

```
ВВЕДИТЕ ТЕМПЕРАТУРУ
```

```
В ГРАДУСАХ ФАРЕНГЕЙТА
```

```
? -50
```

```
ЭТО БУДЕТ -45.5556 ГРАДУСА ЦЕЛЬСИЯ
```

```
-45.5556
```

```
ГРАДУСА ЦЕЛЬСИЯ -
```

— Ерунда какая-то... Лишнее что-то вывела...

ОПЯТЬ ОШИБКА

— Все получилось так, как предусмотрено в программе. Ты перенесла весь вывод результата на строку 70, а строки 80 и 90 со старым выводом оставила нетронутыми, и они, конечно, были выполнены. Можешь на них посмотреть при помощи директивы LIST или LISTNN. Кстати, в этих директивах предусмотрена такая возможность: если после директивы идет номер строки, то покажется только эта строка; если два номера, разделенные тире, то покажутся строки с номерами в этих границах. Так, например, LIST 70 покажет строку 70, а LIST 70—90—строки с номерами между 70 и 90 и, конечно, строки с номерами 70 и 90, если такие найдутся. Именно такую директиву ты сейчас должна ввести.

— Пожалуйста!

```
LISTING 70-90
```

```
70 PRINT "ЭТО БУДЕТ ":" ГРАДУСА ЦЕЛЬСИЯ"
```

```
80 PRINT C
```

```
90 PRINT "ГРАДУСА ЦЕЛЬСИЯ"
```

— Посмотри внимательно. Оператор вывода в строке 70 выведет тебе все, что нужно. А операторы в строках 80 и 90? Добавят еще вывод. Эти строки надо убрать. Ах да! Я же еще не сказал, как это делают. Просто набираешь номер строки и нажимаешь клавишу ввода. Строка стирается.

```
80
```

```
90
```

— Все в порядке. Я запускаю.

ПРОГРАММИСТЫ И ИХ ОШИБКИ

— Никогда не спеши пускать программу. Посиди, подумай, может быть, сможешь что-то улучшить или найдешь ошибку. Какой смысл пускать программу с ошибками?! Речь здесь не о том, что за ошибки тебя кто-то будет страшно ругать. Совсем нет. Программисты считают ошибки естественным явлением и воспринимают их намного хладнокровнее учителей. Здесь тебя не спросят, сколько в программе было ошибок. Спросят, идет ли программа, и если идет, то все в порядке.

— Хочу возразить по существу рассматриваемого вопроса. Почему программу не запустить? Будет работать правильно — хорошо, а если неправильно, тогда и начну думать, что и как. Неправильная программа же машину не испортит.

— Разумеется, нет. Но речь не о том. Важно, сколько времени ты потратишь — и своего, и машинного. Могу открыть секрет: машинное время тоже деньги. А если деньги тебя не интересуют, то остается спортивная сторона: лучше тот программист, кто быстрее работает — быстрее исправляет свои ошибки. А быстрее получится, если думать, а не пускать программы на авось.

— Хватит разглагольствовать, я пускаю. А там посмотрим.

```
RUNNN
```

```
ПРОГРАММА ПРЕОБРАЗУЕТ ГРАДУСЫ
```

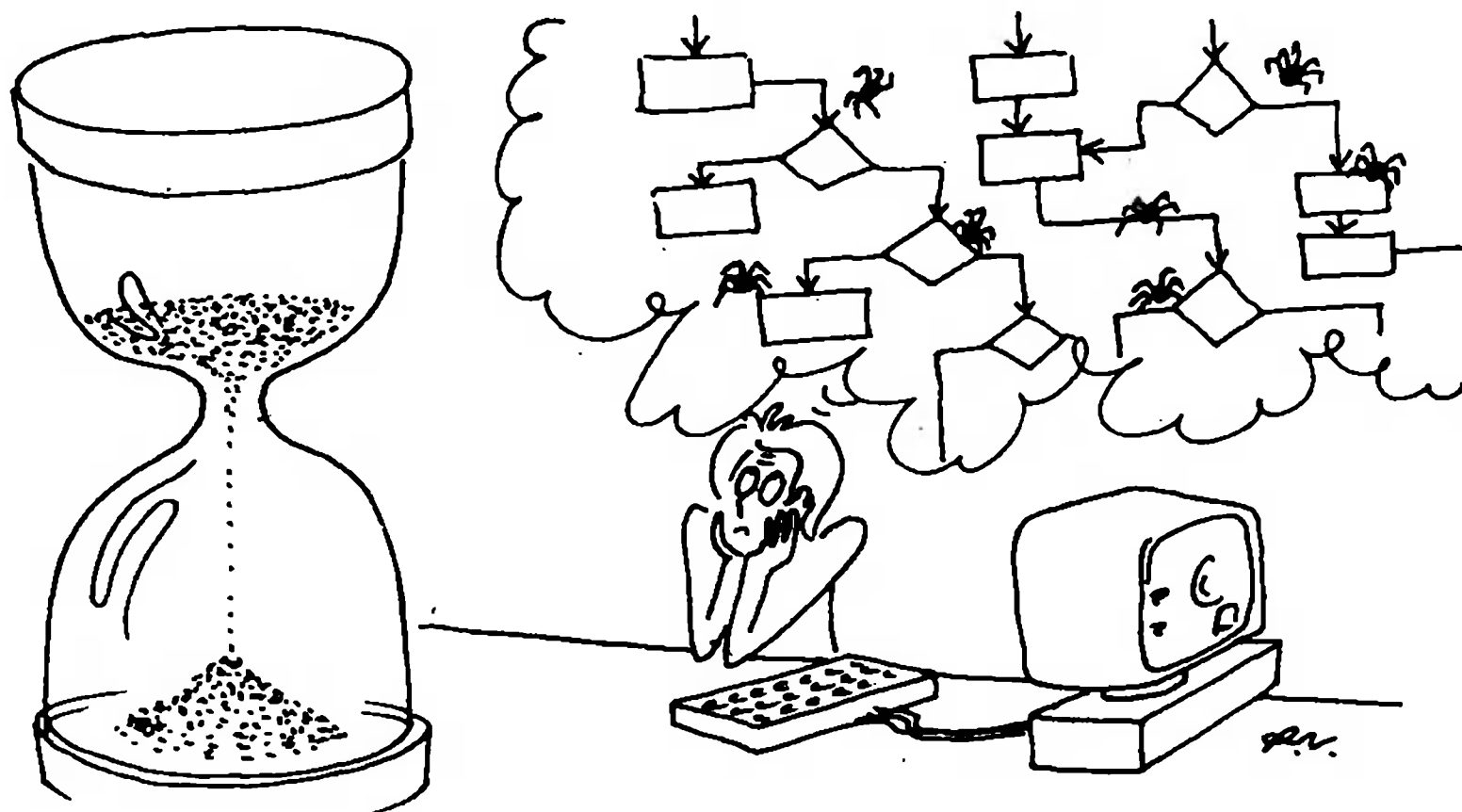
```
ФАРЕНГЕЙТА В ГРАДУСЫ ЦЕЛЬСИЯ
```

```
ВВЕДИТЕ ТЕМПЕРАТУРУ
```

```
В ГРАДУСАХ ФАРЕНГЕЙТА
```

```
? -50
```

```
ЭТО БУДЕТ -45.5556 ГРАДУСА ЦЕЛЬСИЯ
```



Размышление —
наилучшее средство
против ошибок в
программах

- Тебе повезло. Все получилось, как ты хотела.
- Это совсем не везение, а высокое мастерство.
- Как же! А сейчас мы должны записать твою программу на диск, поскольку наше время истекает.
- Ой, прости! Я совершенно забыла, как это делают.
- Не переживай! Вначале все забывается. Позже, когда привыкнешь, прекрасно будешь помнить и Бейсик, и операционную систему, и особенности других компьютеров, если придется с ними познакомиться. Но для сохранения программы нужно ввести директиву **SAVE** с описанием файла — кодом устройства и названием файла.
- Мне лень выдумывать новое название для файла. Пусть будет такое же, как у программы.

SAVE DK1:CELS_

- Все правильно. Нажми клавишу ввода, и программа запишется на первый диск.

9. ВЕТОК МНОГО-МНОГО, НО НЕ ДЕРЕВО

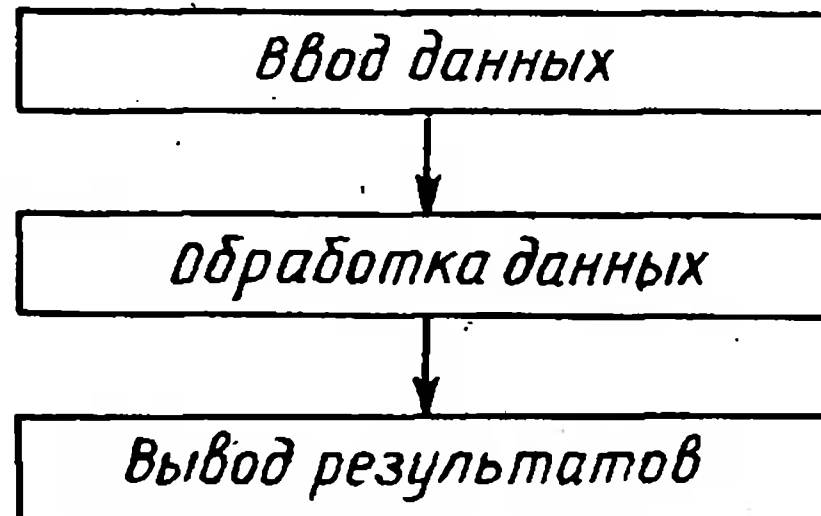
В начале следующего занятия Петя был настроен довольно высокомерно.

- Ну что тебе, Соображалкина, сказать? Программа, которую ты в прошлый раз с такими муками сделала, конечно, жутко тривиальна, но все-таки немножко похожа на настоящую програму. Она имеет кусок. . . ой, что я говорю. . . кусочек! . . ма-аленький кусочек. . . который вводит данные, такой же ма-

ленький, маленький кусочек вывода результатов и еще меньше кусочек, который перерабатывает данные в результаты.

— Я возмущена до глубины души! Что ты вообще от меня хочешь?

— Тихо, тихо. Это я для того, чтобы ты не подумала, что уже все постигла. И еще я хотел отметить, что всякая программа эти три части будет иметь, и можно нарисовать такую вот схему работы всех программ:



— Ну, знаешь...

— Знаю, знаю, но давай быстрее вызовем твою программу и сделаем в ней еще одно улучшение, чтобы она перестала быть такой прямой, как палка, и заимела хотя бы какую-то логику.

— Опять! Я же не умею вызывать программы с диска.

— Сейчас, сейчас. Вызывают их директивой OLD. «Оулд» по-русски что?

— Старый.

— А после OLD описываешь файл, в котором программа хранится. В точности, как в прошлый раз, когда программу записывала на диск директивой SAVE.

OLD DK1:CELS

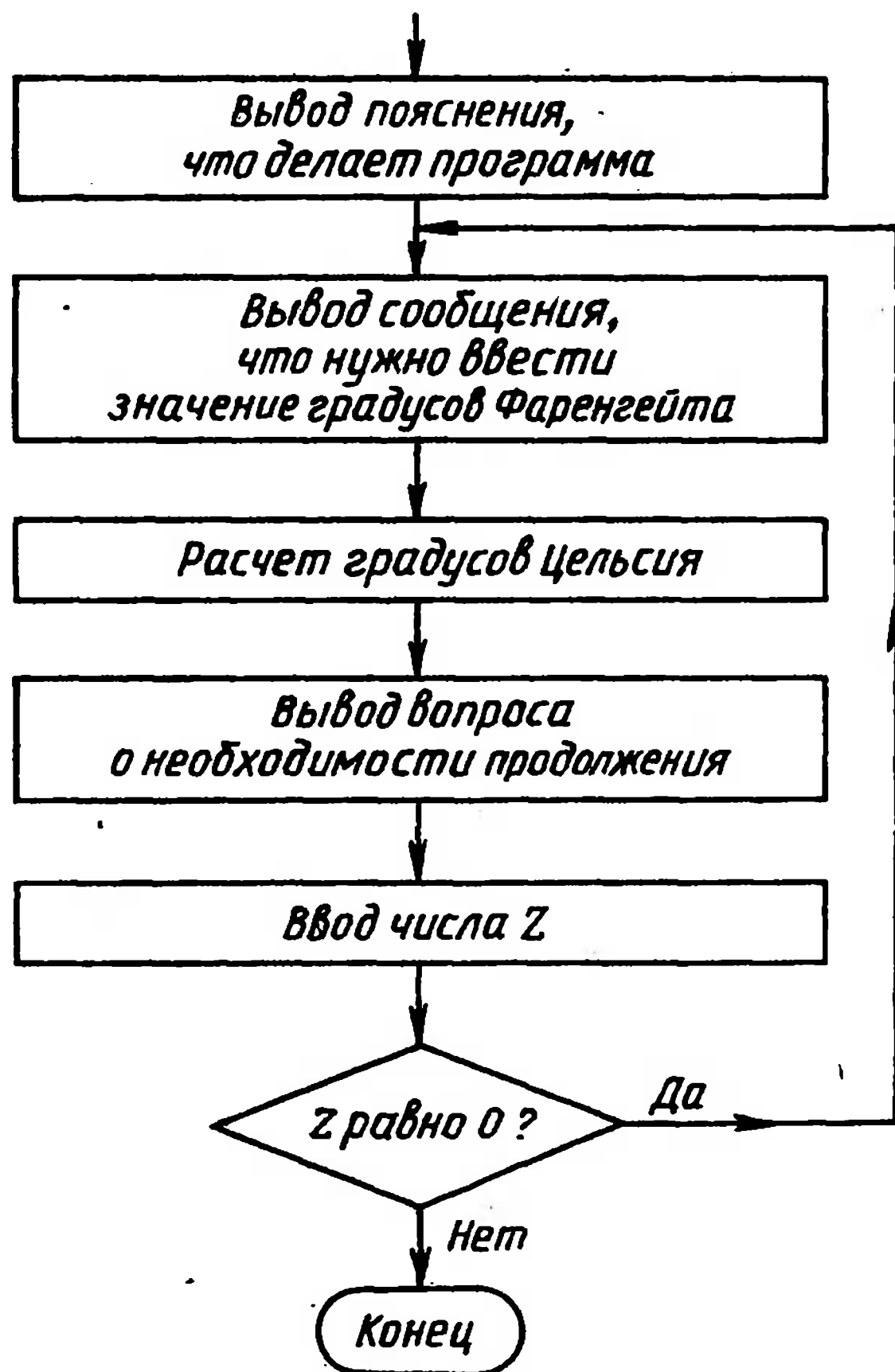
READY

READY показался, значит, все в порядке. Сейчас переделаем программу, чтобы ее можно было использовать для превращения нескольких значений градусов Фаренгейта подряд.

Вдруг зачем-то понадобится.

— Как это сделать?

— Можно по-всякому. Я предлагаю такой вариант. После того как одно значение переведено, программа выдает запрос: «Желаете ли продолжить работу?» — и ждет ввод числа. Если пользователь вводит нуль, то работа прекращается, а если другое число, то продолжается. Разумеется, пользователю нужно объяснить, что от него требуется. Схема нового алгоритма получится такой:



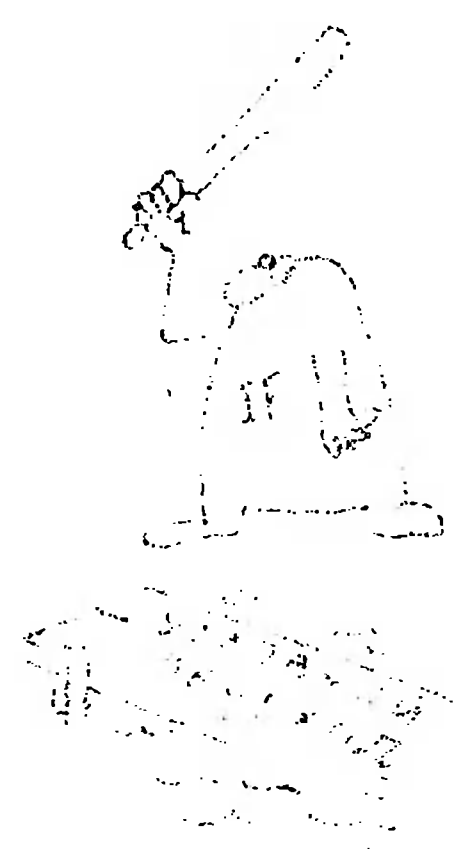
Как вывести запрос о продолжении работы, ты должна понимать: оператором PRINT. Так же ясно, как вводить число: оператором INPUT. Это число придется как-то обозначать, допустим буквой Z.

СЕЙЧАС ОБ ОПЕРАТОРЕ ВЕТВЛЕНИЯ

— А вот, чтобы узнать, является ли нулем какое-то число или нет, известных тебе операторов недостаточно. Для этого нужен новый оператор — оператор условного перехода, или оператор ветвления IF ... THEN. Ты понимаешь, что это означает?

— «Иф» значит «если», а «зен»... может быть, «тогда»?

— Да, так эти слова переводятся. А сам оператор условного перехода, или, кратко, оператор IF, будет труднее понять,

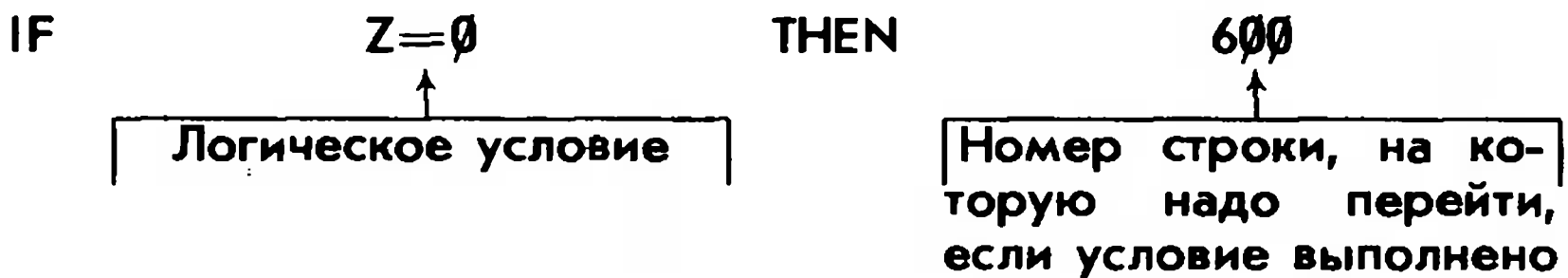


чем предыдущие, так что задержи дыхание, пока я не кончу.

Сначала пишется IF, потом логическое условие, в нашем случае это будет выражение $Z=0$, затем пишется THEN, а после того — номер строки, с которой продолжать работу, если... ты заметила, я сказал «если»... если выполняется указанное условие, т. е. значение Z действительно равно нулю. А если не выполняется, то работа продолжается со следующей строки программы.

— Фу-у-у... Слушай, я чуть не задохнулась, но поняла крайне мало. Объясни поподробнее.

— Я нарисую тебе такую схему:



Я сам тоже не сразу освоился с логическим условием, потому что иногда оно мне казалось вроде оператора присваивания. Отличие тут такое: при выполнении оператора ветвления значение переменной не станет равным нулю, а останется таким, как было, — два, десять и, в конце концов, оно могло бы и быть нулем.

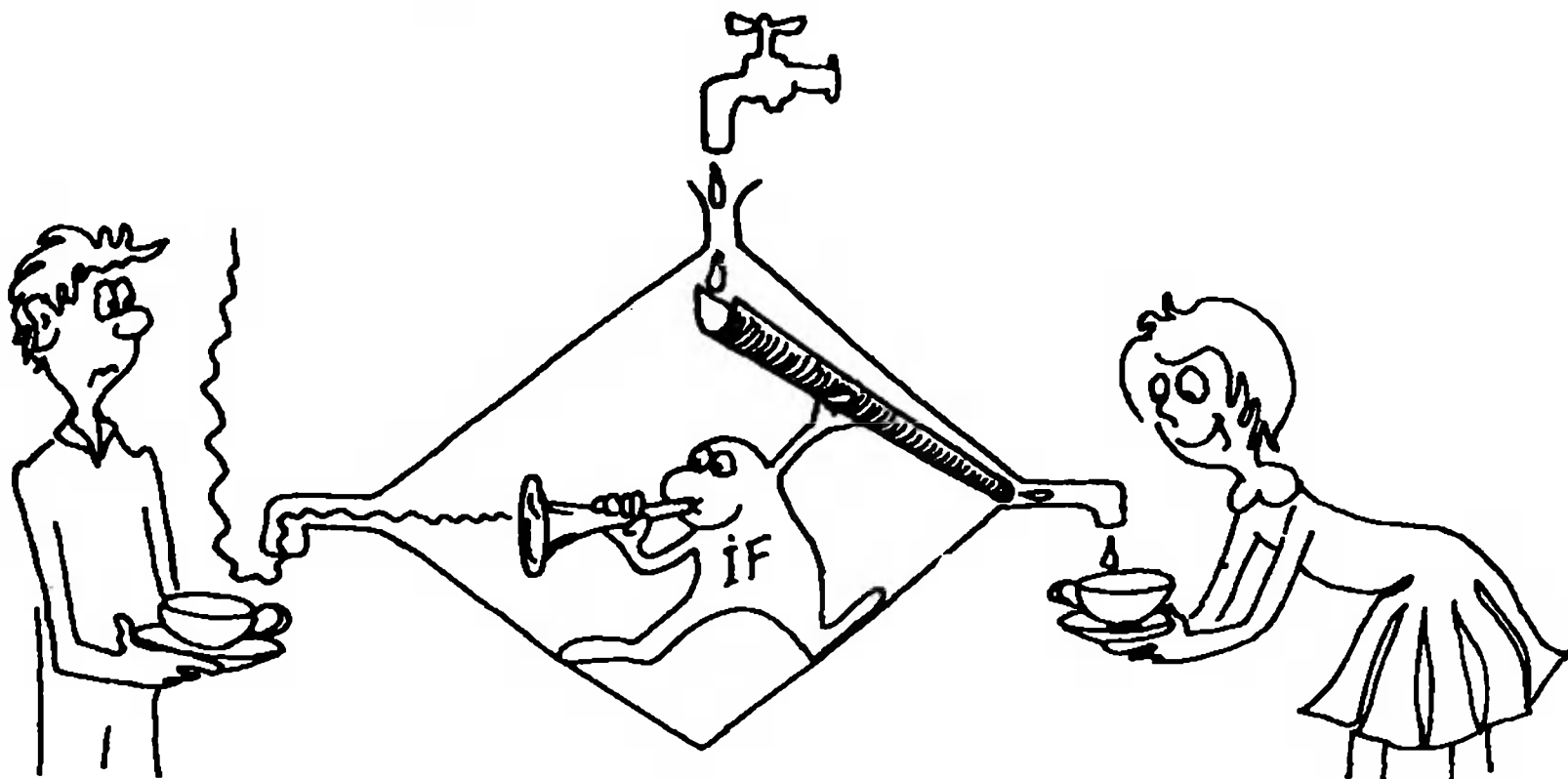
А на русский язык этот оператор можно перевести так:

IF	$Z=0$	THEN	600
Если	значение переменной равно нулю,	то	перейти на строку с номером 600

На Бейсике

По-русски

Вот что получается: знак « $=$ » в операторе присваивания мы читали «присваивается значение», а в логическом условии оператора IF читаем «равно», как и полагается.



Имеются, конечно, и другие возможности сравнения двух чисел в операторе IF. Можно составить небольшой Бейсикорусский словарь логических условий, которые допускаются в этом операторе. Никаких других писать нельзя.

На Бейсике	По-русски
=	равно
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно
<>	не равно

Этот оператор еще может заморочить голову и своими многими названиями. Вот еще одно — его можно называть и оператором сравнения. Сейчас надо посмотреть какие-нибудь примерчики. Пусть дана такая строка:

```
IF A < B THEN 600
```

Это означает «Если значение A меньше значения B, то иди на строку 600». А как ты поймешь такую строку?

```
IF A + B <= (C + B) / 2 THEN 700
```

— Охо! В этом операторе можно писать и арифметические выражения!

— Любой сложности.

— А этот оператор заставит выполнять следующей строку 700, если сумма A и B меньше или равна полусумме C и B.

— А если первая сумма больше второй, то работа продолжится со следующей строки, как будто такого оператора не было вовсе. Еще тебе могу рассказать, что в случае если оператор IF посылает на строку с таким номером, какого в твоей программе вообще нет, то интерпретатор переходит на ту, номер которой идет сразу за указанным; например, если строки 700 нет, то выполнится первая за нею строка.

У ТОНИ ПОЛУЧАЕТСЯ ДИАЛОГ С МАШИНОЙ

— Пора начинать действовать, дело покажет, что и как. Этот твой нововведенный оператор придется вставлять. . . там, где была уничтоженная строка 80. Интересно, сколько новых строк понадобится? . . . Много не должно быть — буду их нумеровать через 5. Сначала запрос о продолжении работы:

```
80 PRINT "ЕСЛИ КОНЧАЕТЕ, ВВЕДИТЕ 0"
```

Поймет это твой пользователь?



СРАВНЕНИЕ

— Сейчас предусмотрим ввод числа Z . Это просто.

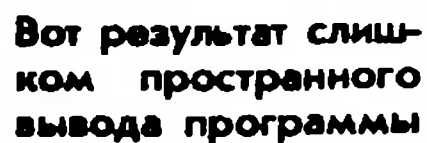
Теперь пресловутый оператор ветвления. Так, так. Записать «если $Z=0$, то уходи...» Неизвестно, куда надо уходить... Да это и неважно, потому что так писать нехорошо, Переходить надо тогда, когда $Z \neq 0$! А это записывают так:

Ну вот, это сделано. Попросим еще вывести пустую строку, что достигается вводом оператора PRINT без ничего, если выразаться странным русским языком Бейсикова.

— Что ж. Выглядит вполне прилично. Поставь еще только оператор STOP после оператора ветвления. Нужен, конечно, он исключительно ради порядка, так как интерпретатор, достигнув последней строки программы, все равно кончит работать.

— Сейчас я перенумерую строки программы.

19404



Потом ее посмотрю. Но всю смотреть нет необходимости, достаточно посмотреть конец с изменениями. Я начала вводить изменения с восьмидесятой строки, прости, со строки с номером 80, но так как я еще добавила строку 25, то после перенумерации строка 80 стала строкой 90.

```
LISTNN 90-200
90 PRINT "ЕСЛИ КОНЧАЕТЕ, ВВЕДИТЕ 0"
100 INPUT Z
110 IF Z<>0 THEN 40
120 STOP
130 END
```

ОСОБЕННОСТИ ПЕРЕНУМЕРАЦИИ

— Смотри сюда! Какой ужас! Строка 110! Когда я ее вводила, то после THEN писала 30, и это еще на экране видно. А после перенумерации машина 30 превратила в 40. Что же делать, если она все портит?!

— Несмысленыш ты несчастный! Сама же рассуждала, что, вставив строку 25 и перенумеровав строки, ты все номера сдвинула. Значит, и номер 30, который стал 40. По этой причине и интерпретатор после THEN поставил 40.

— Потрясающе мудро сделано.

— А как же иначе! Ты просто не представляешь, как трудно было бы самому в большой программе найти и исправить все операторы ветвления, не допустив ни одной ошибки.

— Других претензий у меня нет. Я запускаю.

RUNNN

ПРОГРАММА ПРЕОБРАЗУЕТ ГРАДУСЫ
ФАРЕНГЕЙТА В ГРАДУСЫ ЦЕЛЬСИЯ

ВВЕДИТЕ ТЕМПЕРАТУРУ

В ГРАДУСАХ ФАРЕНГЕЙТА

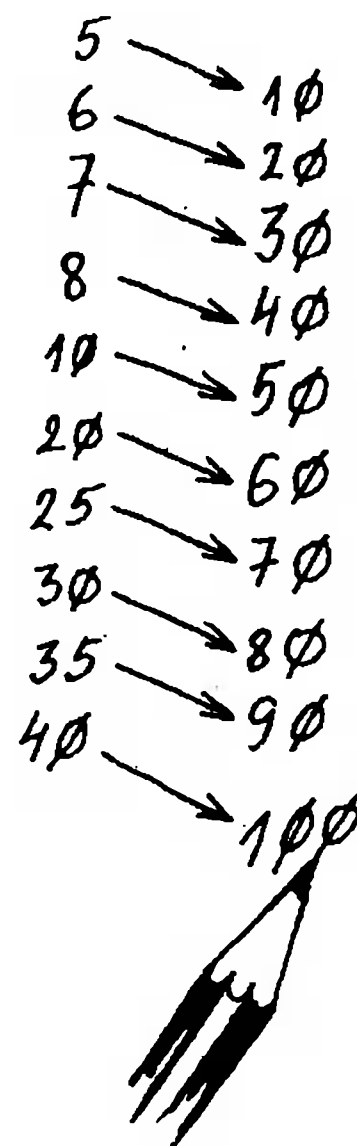
? -50

ЭТО БУДЕТ -45.5556 ГРАДУСА ЦЕЛЬСИЯ

ЕСЛИ КОНЧАЕТЕ, ВВЕДИТЕ 0

? 1

Со значением — 50 получилось отлично. Попробуем с другим значением.



ВВЕДИТЕ ТЕМПЕРАТУРУ
В ГРАДУСАХ ФАРЕНГЕЙТА
? 100
ЭТО БУДЕТ 37.7778 ГРАДУСА ЦЕЛЬСИЯ
ЕСЛИ КОНЧАЕТЕ, ВВЕДИТЕ 0
? 0_

Теперь закончим работу.

STOP AT LINE 120

READY

— Посмотри, если работа прекращается по оператору STOP, то интерпретатор об этом сообщает и еще показывает, на какой строке это произошло. «Эт лайн» означает по-английски «на строке». Делается это для того случая, когда у тебя в программе имеется много операторов останова и ты можешь запутаться, не зная какой из них сработал. Ну, с этой программой покончено, можешь ее записать на диск.

SAVE DK1:CELS_

— Интересно бы узнать, можно ли с таким же названием файла записывать программу на диск второй раз.

— Можно, можно.

— А та, ранее записанная программа забудется?

— Ты хотела сказать, стирается ли старый файл. Так правильно говорить. Нет, на нашей машине он не стирается. Работу с файлами проводит операционная система, и там сделано так, что при появлении нового файла с таким же названием старый не стирается. На других машинах, как я слышал, старые файлы сразу стираются, чтобы не захламлять диск. У нас ты старый файл могла бы списать, если бы знала больше об операционной системе. Директивой OLD ты умеешь списывать только последний вариант.

— Ума не приложу, зачем мог бы понадобиться предыдущий вариант? Мне лично нужен будет только последний. Я просто так спросила, ради интереса. Заговорил ты меня, я и забыла, что еще не ввела эту последнюю строку.

ЧТО ТОНЯ УЖЕ УМЕЕТ

— Порядок. Теперь мы могли бы эту твою последнюю программу «проразмыслить». Смогла бы ты сейчас написать такую программу, которая делала бы наоборот — переводила градусы Цельсия в градусы Фаренгейта?

— Конечно. Там, где написано Фаренгейта, я написала бы Цельсия, вместо Цельсия написала бы Фаренгейта.

— К сожалению, самого главного ты не сказала. Нужно изменить формулу для расчета значения переменной С.

— Ах, да! Такой формулы пока нет, но ее же легко получить из старой.

— Совсем легко. Она получается такой: $F = \frac{9}{5}C + 32$.

Но это совсем неважно, поскольку мы ее сейчас не собираемся программировать. Скажи, ты могла бы сейчас составить программу, которая переводила бы, например, километры в сантиметры?

— О чем речь, товарищ профессор! Я даже знаю формулу, которую здесь нужно использовать. Если пожелаешь, я могу тебе написать программу, которая сантиметры будет переводить в километры и даже дюймы в миллиметры.

— Прекрасно! А секунды в годы?

— Вы меня обижаете, товарищ академик!

— Итак, получилось, что нашу программу очень легко переделать, чтобы она переводила любую единицу измерения в другую независимо от того, что эти единицы измеряют.

— Вряд ли это так уж важно, что мы одни единицы переводим в другие...

— Во! Именно это я и хотел услышать. Важно то, что счет проводится по какой-то формуле, все равно по какой. Это может быть расчет скорости по заданным пути и времени, расчет зарплаты по количеству выработанных норм и расценке; это может быть расчет объема куба по одной грани или объема цилиндра по радиусу и высоте.

— С таким же успехом это могло бы быть и вычисление площади.

— Другой пример. Проявив чудеса изобретательности, ты решила какую-то задачу по математике. Например, такую: из пункта А в пункт В со скоростью v_1 мчится поезд, ему навстречу со скоростью v_2 мчится другой поезд. Через какое время они налетят друг на друга, если расстояние между А и В равно m километров?

— Лучше бы автомашины, они смогли бы разминуться, а то очень уж жуткой получилась у тебя задача.

— Это совершенно все равно. Если тебе нравится, пусть это будут лягушки. Важно то, что с обеих сторон кто-то несется друг другу навстречу. Если решишь эту задачу, то сможешь запрограммировать формулу для расчета времени встречи

$$T = \frac{m}{v_1 + v_2}$$

и тебе останется только ввести значения скоростей v_1 , v_2 и расстояния m , чтобы получить время встречи. И так будет



с любой похожей формулой. И с непохожей тоже. Лишь бы была формула.

— Однако есть и различия. В нашей первой программе нужно было ввести только одно число — градусы Фаренгейта. А чтобы считать по твоей формуле встречи поездов, нужно ввести три числа.

— Да, такое различие имеется. Но все равно мы сейчас можем запрограммировать любую формулу на свете.

Понимаешь, схемы алгоритмов этих программ будут почти одинаковы, а программы будут делать нечто вроде бы совсем, разное.

— Почему же «вроде бы»?

— Потому что различия появляются только тогда, когда ты начинаешь вникать в существо дела: что это такое считается по этой формуле? Что вводят? Что выводят? Когда ты хочешь понять, какой во всем этом смысл. Машина же не может ни хотеть, ни понимать, она только рассчитывает по заданной формуле. И тогда между формулами большого различия нет. Поняла?

— Нет, не совсем.

— Тогда есть смысл порешать задачки.

1. Разобраться в следующей программе:

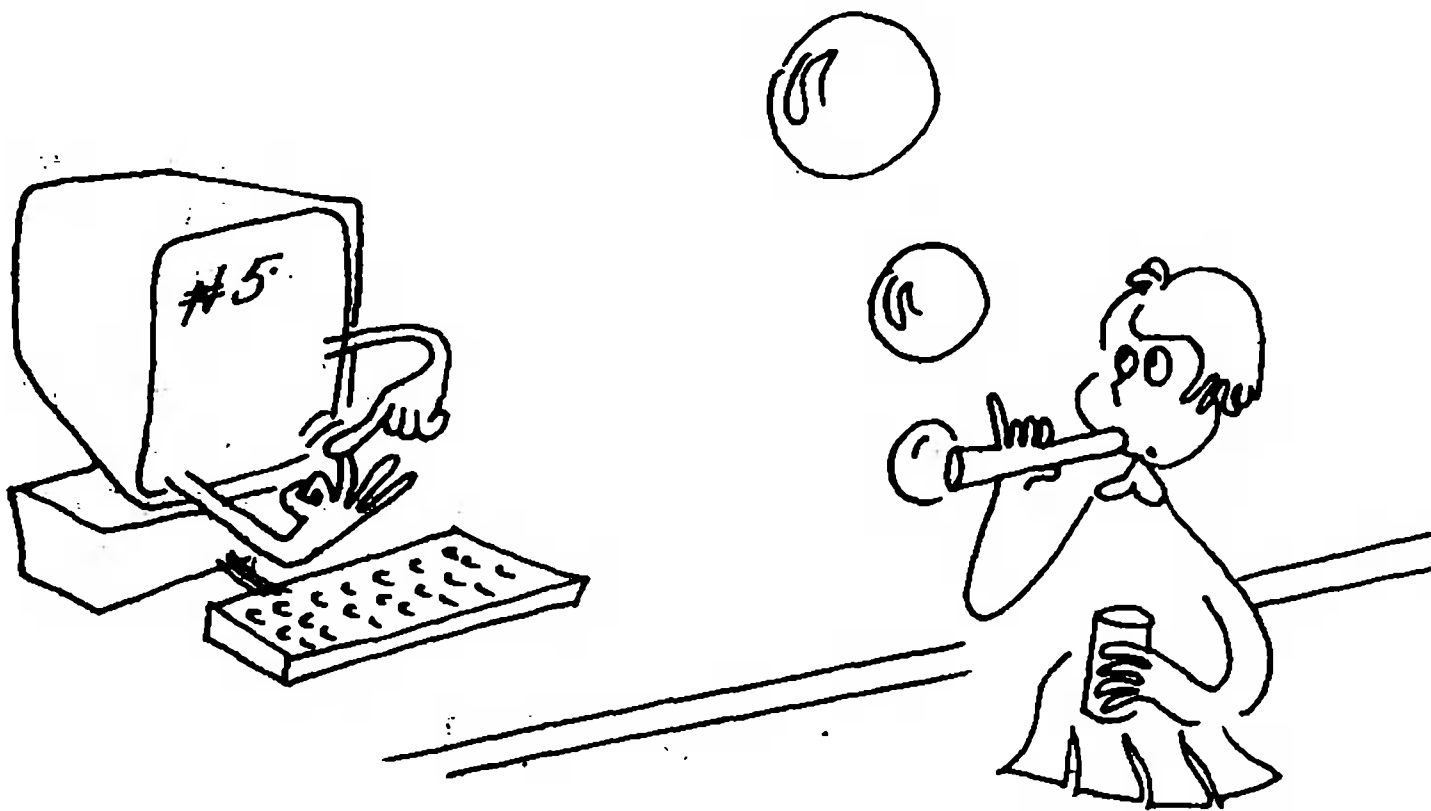
```
10 PRINT "ПРОГРАММА НАХОДИТ СРЕДНЕЕ"  
20 PRINT "ТРЕХ ЧИСЕЛ"  
30 PRINT "ВВЕДИТЕ ПЕРВОЕ ЧИСЛО"  
40 INPUT A  
50 PRINT "ВВЕДИТЕ ВТОРОЕ ЧИСЛО"  
60 INPUT B  
70 PRINT "ВВЕДИТЕ ТРЕТЬЕ ЧИСЛО"  
80 INPUT C  
90 PRINT "СРЕДНЕЕ ЧИСЕЛ"; A, B, C,  
100 PRINT "РАВНО"; (A + B + C)/3  
110 STOP  
120 END
```

2. Программа расчета процентов. Требуется ввода числа, от которого нужно найти проценты и число процентов. Выводит рассчитанные проценты.

3. Программа расчета платы за электроэнергию. Вводятся старое и новое показания счетчика, находится их разность, умножается на 4 к. и делится на 100, чтобы найти сумму в рублях.

4. Составить программу для нахождения абсолютного значения числа P . Оно равно самому P , если число больше или равно нулю, и $-P$, если число отрицательно.

5. Программа спрашивает: «Сколько стоит один самовизг?», а потом: «Сколько у тебя денег?», получив ответы, делит



одно число на другое и сообщает, сколько «самовизгов» может купить собеседник.

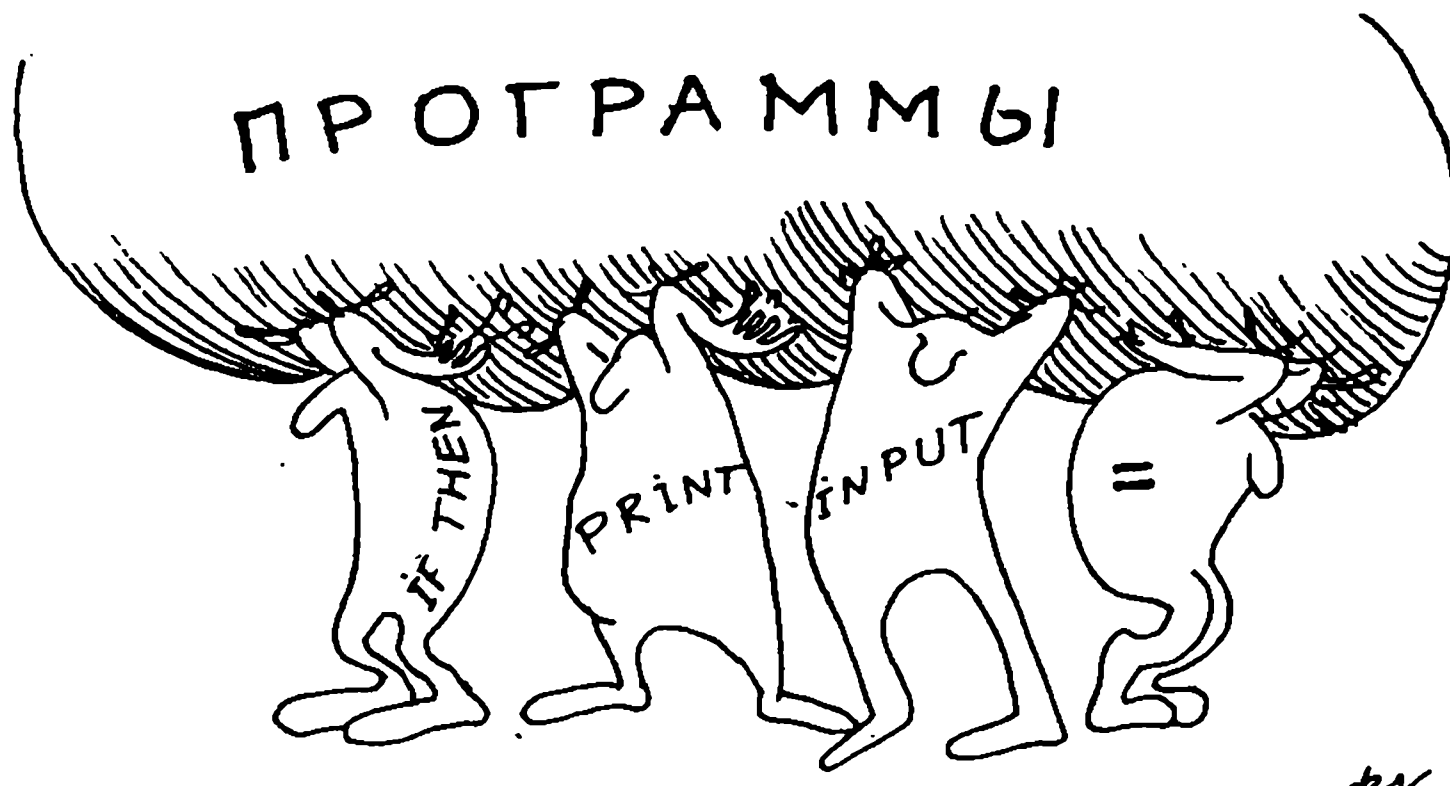
6. Программа спрашивает: «Сколько у вас бензина?», а потом сообщает, сколько километров пользователь может проехать на «Жигулях», сколько — на «Запорожце», сколько — на Ил-62, и в конце добавляет, что на велосипеде можно проехать, сколько захочется. Еще можно, если уметь, проверить, не введено ли отрицательное число или ноль, и если так, то вежливо (обязательно вежливо!) указать на ошибку.
7. Программа кассира. Кассир магазина в конце рабочего дня подсчитала количество денежных купюр различных достоинств. Программа спрашивает ее поочередно, сколько имеется купюр разного достоинства — 1, 3, 5, 10, 25, 50 и 100 р. Выводится общая сумма денег.

10. ТОНЯ УЖЕ ЗНАЕТ О ПРОГРАММИРОВАНИИ ВСЕ

Весело насвистывая, Тоня появилась у машины. Петя ее уже ждал.

— Опоздала. А в это время здесь был Алгоритмыч. Спросил, как дела, я рассказал, и он высказался, что твои занятия пора кончать.

Нельзя сказать, чтобы Тоня очень увлеклась новым делом, но она предполагала, что именно она скажет, что ей все уже надоело, когда действительно надоест. Но так! Вдруг! Ее отстранить! Нет, к такому Тоня не привыкла.



— Что это за новости? Кому я здесь мешаю? Я что, кого-нибудь тут обидела?! Или этого твоего Змея-Алгоритмыча оскорбила? Или...

— Не шуми. Он просто сказал: раз человек освоил четыре кита программирования — оператор ветвления, операторы ввода и вывода и плюс еще оператор присваивания, то в программировании он знает все существенное и ему осталось освоить только мелочи, несущественные детали. Одним словом, сейчас ты можешь уже учить других.

— Ну знаешь ли, Бейсиков! Ты меня не дразни. Лучше поведай, неужели в программировании больше ничего нет, кроме этих четырех операторов?

— Да уж нет! Еще будет масса различных приемов, условий и правил, и интересных, и не очень, но ничего принципиально нового не будет. Если вот только понятия подпрограммы и функции, но и их легко освоить. Ах да, осталось еще понятие массива чисел, и его мы как раз сегодня и разберем.

Массив

О МАССИВАХ ЧИСЕЛ

Массив чисел — это, считай, таблица чисел, или последовательность чисел, причем эти числа находятся в памяти машины и пронумерованы, чтобы их можно было друг от друга отличить. Можно и так сказать: массив — это переменная, имеющая несколько перенумерованных значений. И как всякая переменная, массив имеет свое имя, которое можно образовывать совершенно так же, как имя обычной переменной, — из буквы или из буквы и цифры.

Если, допустим, мы создали массив с именем *M*, то первое число этого массива, или первое значение переменной *M*,

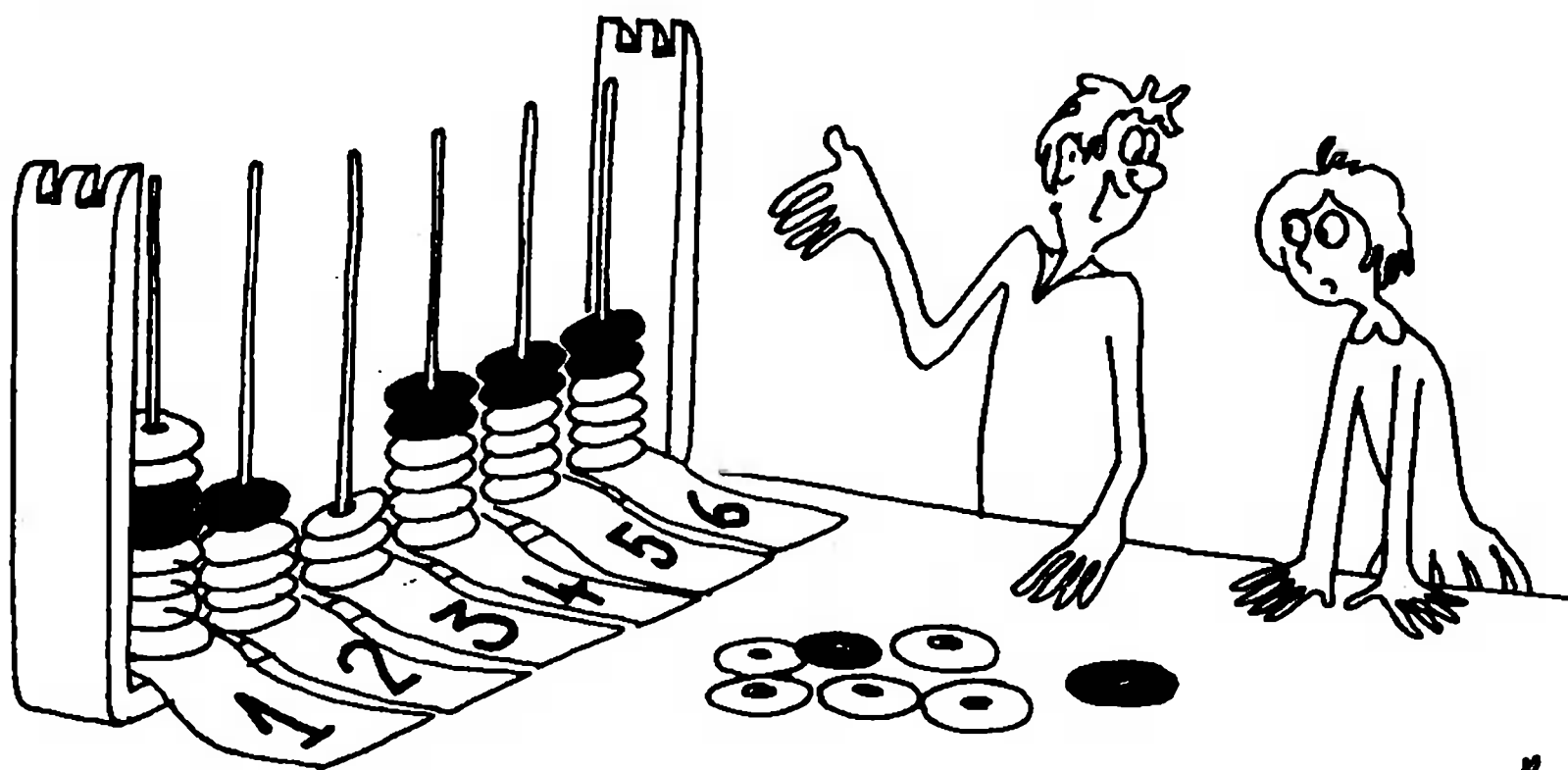
или, как чаще всего говорят, первый элемент массива, обозначают $M(1)$, второй — $M(2)$, третий — $M(3)$ и т. д. Такими обозначениями, как ты понимаешь, нужно пользоваться в том случае, если необходимо соответствующее значение элемента использовать в каком-нибудь арифметическом выражении или хочется присвоить новое значение этому элементу. Например, запись $M(6)=18$ означает, что шестому элементу массива M присваивается значение 18. Выражение $B=5 \cdot M(6)$ означает, что для получения результата значение шестого элемента массива M умножается на 5.

— А ты не смог бы дать хоть один пример, где бы эти массивыгодились?

— Конечно, могу. Но пока не забыл, скажу, что в скобках после обозначения массива необязательно писать число, чтобы иметь номер элемента. Можно писать, например, $M(K)$, тогда интерпретатор найдет элемент с номером K . Разумеется, до этого переменной K должно быть присвоено нужное значение. Можно сложнее: $M(K+1)$ или $M(5 \cdot K+3)$, и вообще в скобках может быть любое арифметическое выражение. Интерпретатор это выражение вычислит и найдет требуемый элемент массива.

Сейчас о том, зачем нужны такие массивы. В общих словах, они нужны, если твоей программе надо работать с наборами однотипных чисел, заданных в таблицах. Скажем, тебе дали или результаты каких-нибудь измерений — средней дневной температуры в течение года, или имеющееся в магазине количество одинаковых пар обуви по размерам, или зарплату каждого сотрудника какого-нибудь учреждения. Важно, чтобы все сотрудники в списках были в одном неизменном порядке, тогда по порядковому номеру сотрудника всегда можно будет найти в массиве зарплаты, сколько ему причитается. А вот другой пример. В массиве можно хранить оценки, полученные учениками за контрольную по английскому языку. Опять

Элемент массива



Небольшой массив с шестью элементами

важно, чтобы номер ученика в классном журнале был неизменным: он будет и номером элемента массива. Путаницу здесь допустить нельзя, некоторые не обрадуются, если им припишут чужую оценку. Как видишь, подобных примеров тьма-тьмущая.

— Да эти примеры уже надоели. Говори, что еще надо знать, и начинаем работу.

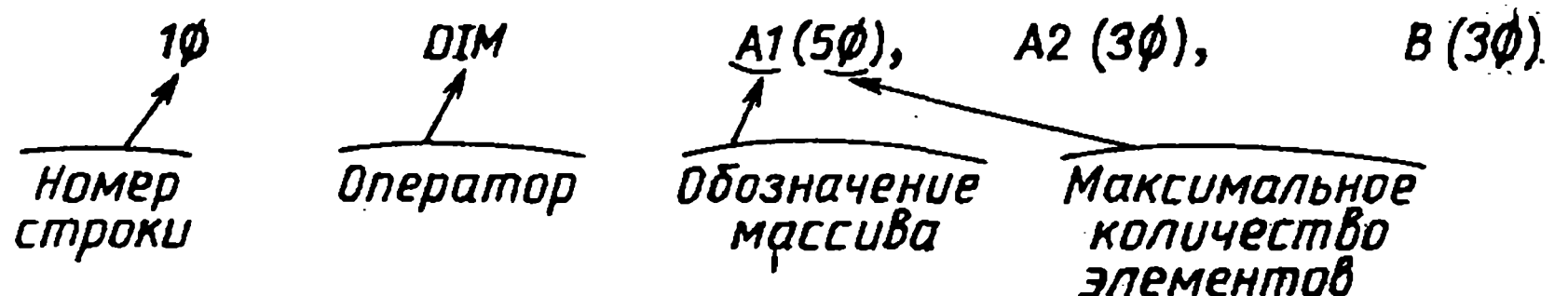
КАКОЙ НУЖЕН МАССИВ, ДОГАДАЙСЯ САМ

— До начала работы нужно описать массив — определить его обозначение и длину. Как обозначать массивы, я уже говорил, добавлю, что обозначения простых переменных и массивов могут совпадать. Например, может быть массив *A* и переменная *A*. Имена одни и те же, а значения, конечно, разные. *A* — длина массива, как легко догадаться, — это максимальное количество элементов в массиве.

Описывают массив специальным оператором описания массивов *DIM*. Эти три буквы являются сокращением английского слова *dimension* — размерность. После *DIM* пишут имя массива и в скобках задают длину массива, или, как я говорил, максимальное количество элементов.

— А что это такое — максимальное количество элементов?

— Сейчас скажу. Закончим только об операторе *DIM*. Он может быть, например, таким:



Как видишь, после *DIM* можно писать несколько массивов сразу, разделяя их описания запятыми. При выполнении этого оператора интерпретатор в памяти отведет место для хранения значений этих массивов. Притом столько места, чтобы можно было бы запомнить количество чисел, равное максимальному количеству элементов. Может статься, что в программе нужен массив, но точная его длина тебе неизвестна. Тогда этот массив можно описать, задав максимальное количество элементов заведомо большим, чем тебе может понадобиться. Если ты и описала более длинный массив, чем тебе надо, невелика беда, важно, чтобы он не получился короче.

— А если я совсем, совсем не представляю, сколько нужно будет элементов?

— Тогда тебе еще рано составлять программу. Обычно



Объем массива нужно представлять заранее

все-таки можно предвидеть, сколько элементов понадобится. Например, ты хочешь создавать массив оценок по контрольной, и притом такой, чтобы был пригоден для любого класса. Сразу видны трудности — в каждом классе количество учеников различно.

Тогда можно описать массив так:

```
20 DIM A(50)
```

Я думаю, что ни в одном классе не будет больше пятидесяти учеников. В этом случае 50 — это максимальное количество элементов в массиве, а фактическое ты введешь в память отдельно во время работы программы.

— А почему бы не взять 100?

— Видишь ли, фактически оператор DIM является приказом интерпретатору выделить в памяти место для хранения какого-то количества чисел. И чем больше элементов задашь, тем больше памяти интерпретатор отведет для хранения их значений. Конечно, в память нашей машины запросто войдет даже десять тысяч чисел, так что жадничать нечего — 50 или 100, все равно мелочь. Но в больших программах приходится думать и об экономии места в памяти.

МАЛЕНЬКАЯ ПРОГРАММА ДЛЯ ВВОДА ОЦЕНОК

— Так как машину сейчас ремонтируют и работать нельзя, составь, пожалуйста, программу, которая вводила бы оценки в какой-нибудь массив. Такая программа большой самостоятельной ценности не будет иметь, но ее можно представлять как составную часть большой программы.

— Пожалуйста.

```
10 DIM A(50)
20 PRINT "ВВЕДИТЕ ОЦЕНКУ"
30 INPUT A(1)
40 PRINT "ВВЕДИТЕ ОЦЕНКУ"
50 INPUT A(2)
```

Послушай! Что это такое? Мне так и надо будет писать все эти PRINT и INPUT пятьдесят раз?

— Ну не-ет! В таком случае программирование было бы сплошной писаниной. Вспомни один из первых наших алгоритмов — алгоритм обучения в школе. Там был счетчик классов, здесь ты можешь сделать счетчик учеников — сначала присвоить счетчику первое значение, потом в нужном месте добавлять единицу или другое число, на которое надо увеличить счетчик.

— Да-да, я вспомнила! Ага! Если вставить строку 15 $H=1$, то значение счетчика H становится равным 1. Далее в строке 40 $H=H+1$ я увеличу номер школьника на единицу. И это почти все.

— Не забудь в строке 30 заменить $A(1)$ на $A(H)$.

— Да. А то все время будет вводиться только первый элемент. А вот сейчас мне нужно, чтобы после увеличения значения счетчика опять выполнялась строка 20.

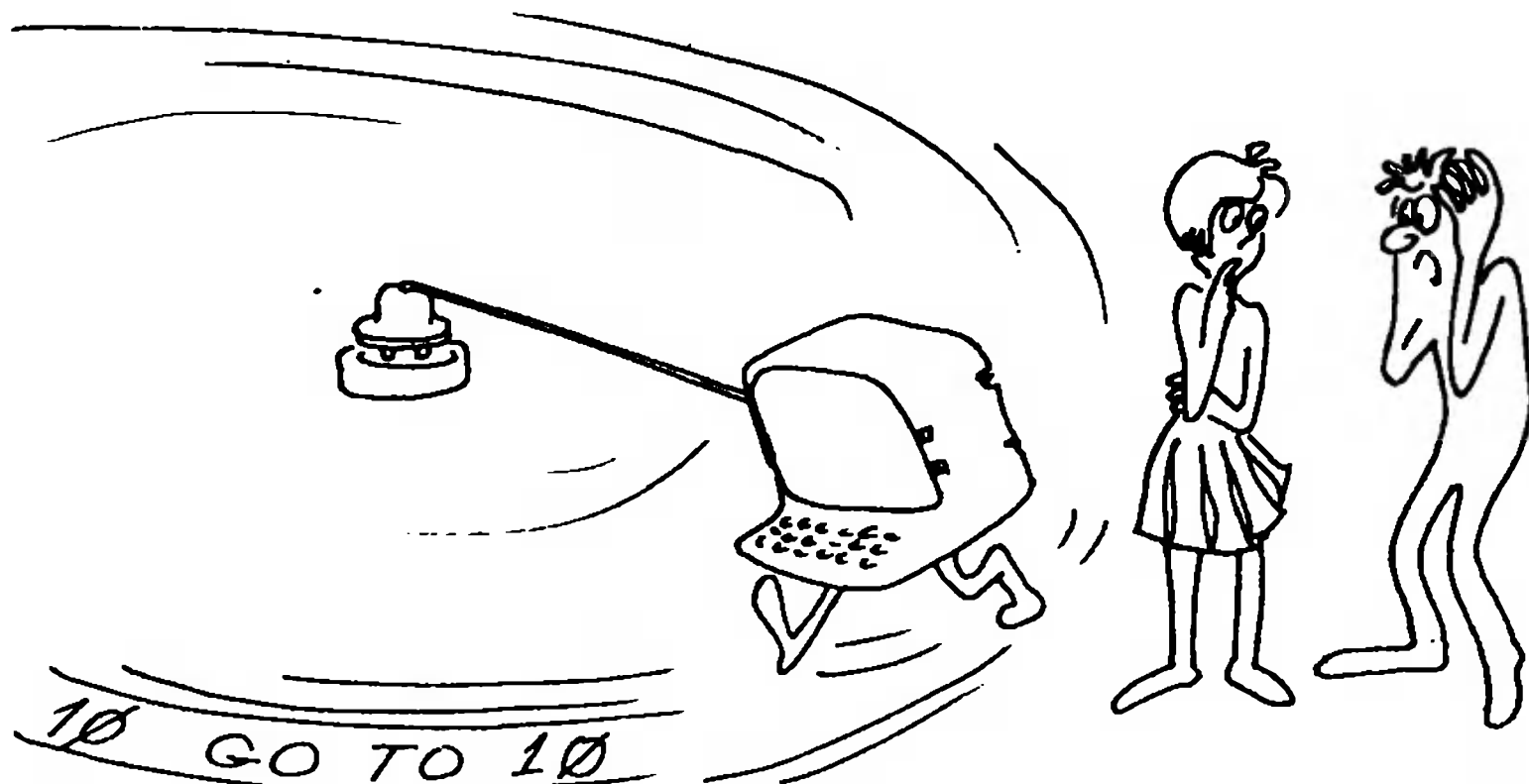
ОПЕРАТОР, ВСЕГДА ПЕРЕДАЮЩИЙ УПРАВЛЕНИЕ

— Раз уж ты хочешь делать именно так, то можешь использовать оператор безусловной передачи управления GO TO, что произносится «гоу ту».

— И что означает «иди на», но был уговор, что больше новых операторов не будет.

— Такого уговора не было. Речь шла о том, что не будет ничего существенно нового, так что несущественно новые операторы могут еще быть. Закончим про оператор GO TO. После него надо писать номер той строки, которая должна быть выполнена следующей. Например, 50 GO TO 20 означает, что следующей будет выполнена строка 20 и потом по порядку строки, следующие за нею, до тех пор пока не встретится строка с оператором GO TO. А вот доказательство, что оператор GO TO не является существенно новым. Его можно записать при помощи давно известного оператора ветвления. Например, таким образом:

```
50 IF H=H THEN 20
```



Он же никогда не остановится!

Этот оператор сравнивает значения переменной N со значением той же переменной, и если они равны — а они, конечно, всегда равны, — то управление передается на строку 20.

— Зачем же тогда такой оператор нужен?

— Для краткости, только для краткости. И яснее будет, если запишешь `GO TO`, а не какой-то оператор `IF`, в котором еще надо разбираться. Правда, есть и такие программисты, которые считают, что этот оператор не нужен.

— Тогда программа готова!

```
10 DIM A (50)
15 N=1
20 PRINT "ВВЕДИТЕ ОЦЕНКУ"
30 INPUT A (N)
40 N=N+1
50 GO TO 20
60 END
```

Совсем неплохо. Правда?

— Да, неплохо, но кое-чего не хватает. Машина все будет и будет спрашивать оценки.

— Ой, да! Я же все время передаю управление со строки 50 на строку 20, а между этими строками никаких передач управления нет. Этот цикл никогда не кончится.

— Нет, в принципе все кончится при вводе значения пятидесятого элемента массива. Пятьдесят первый уже не введется — будет сообщение об ошибке. Сообщение об ошибке выдается всегда, когда мы хотим использовать значение элемента массива, номер которого больше максимального количества элементов в массиве.

— Что же мне делать? Может быть, поступить так же, как при переводе градусов Фаренгейта в градусы Цельсия? После каждого ввода оценки еще дополнительно вводить либо ноль, либо не ноль для выяснения, кончается ли работа?

— Да, так можно было бы. Но так ты фактически удвоишь работу пользователя. Ему придется вводить и оценку, и, кроме того, указание, кончать или не кончать работу. Сколько оценок, столько указаний. А если оценки перепутаются с указаниями?

— Не перепутаются. Напишем на экран информацию, что именно надо вводить.

— Как пить дать перепутаются. В таких случаях пользователь всегда хочет вводить все быстрее и быстрее и кончается это ошибками.

— Сам виноват.

— Это еще неизвестно. Может быть, виноват программист, который так нескладно запрограммировал. Лучше здесь делать так: в самом начале вывести сообщение, что сигналом конца ввода служит ввод нуля вместо оценки. Ясно, что оценки могут меняться только в пределах от 1 до 5. А в программе проверяем введенную оценку — ноль это или не ноль.

Еще может быть и другой способ: в самом начале спрашиваем, сколько в классе учеников, и каждый раз до увеличения — я подчеркиваю, до, а не после увеличения счетчика N в строке 40 — проверяем, стало ли N равным количеству учеников. Если не стало, продолжаем ввод. Если стало, кончаем.

— А почему ты так усиленно подчеркивал, что эту проверку нужно выполнять до увеличения N ?

— До увеличения N нужно проверять, равно ли значение счетчика количеству учеников, а после увеличения — больше ли оно количества учеников. Отличие только в этом.

— Ну, а какой же вариант ввода данных лучше?

— Может быть, второй, когда после увеличения счетчика в строке 40 мы делаем сравнение «не больше ли N количества учеников», и если не больше, то уходим на строку 20.

— Насколько я помню, такого логического условия «не больше» в Бейсике не было.

— Да, не было. Но что такое «не больше»? Это меньше или равно. А такое условие у нас было. Если обозначать количество учеников буквой K , то новая строка 50 будет такой:

```
50 IF N <= K THEN 20
```

— И значение переменной K нужно ввести где-то в начале программы?

ОПЕРАТОР-СЧЕТЧИК

— Да, но до того как это делать, расскажу про оператор цикла, который придуман специально для описания подобных операций со счетчиком. Только счетчик здесь называется переменной цикла.

Переменная цикла

— Подожди, подожди! Чтобы получился счетчик, нужно знать, с какого числа начинать и каким числом кончать?!

— Ты удивительно сообразительна! В операторе цикла действительно указывают первое значение переменной цикла и конечное тоже. И еще нужно указать шаг, с каким изменяется переменная цикла.

Посмотрим на пример оператора цикла:

FOR N=1 TO K STEP 1

Это, правда, только первая его часть; но сначала, что означает «фор»?

— Это означает «для», а «ту», которое следует за $N=1$, могло бы означать «к» или «на».

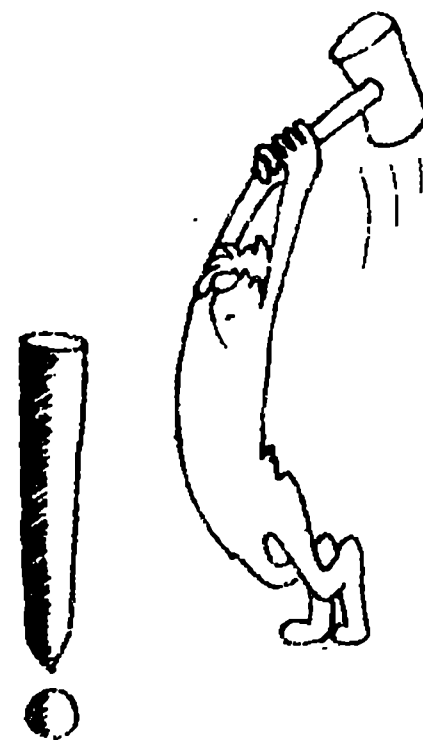
— Да. Но здесь оно означает скорее «до». Сразу после FOR идет имя переменной, в нашем случае N. Это будет та знаменитая переменная цикла. Потом за знаком равенства пишется первое значение переменной цикла. В этом операторе знак «=» нужно читать «с», т. е. совсем иначе, чем в операторе присваивания. Кстати, как там мы его читали?

— «Присваивается значение», а в операторе ветвления мы его читали так, как надо,— «равно». Вот так! Кое-что и мы помним.

— Молодец! После TO пишется конечное значение переменной цикла. В этом примере значение переменной K. А за словом «стэп», что означает «шаг», «ступенька», указывают значение шага. Что это такое было — шаг?

— Наверно, это величина, которую мы будем все добавлять и добавлять к первому значению переменной цикла. . .

— Да, ты права. Расскажу еще об одном трюке, который я сам придумал. Пустое место в самом конце оператора читается «выполнить последующие операторы». Так, сейчас мы этот пример оператора цикла можем перевести на русский язык:



FOR	N	=	1	TO	K	STEP	1	
Для	значений переменной N, начиная	от	1	до	K	с шагом	1	выполнить последующие операторы

На Бейсике

По-русски

И ЧТО ВСЕ ЭТО ОЗНАЧАЕТ

— Теперь поговорим о второй части оператора цикла, которая в программе может находиться совсем в другом месте и в нашем примере будет выглядеть так:

NEXT H

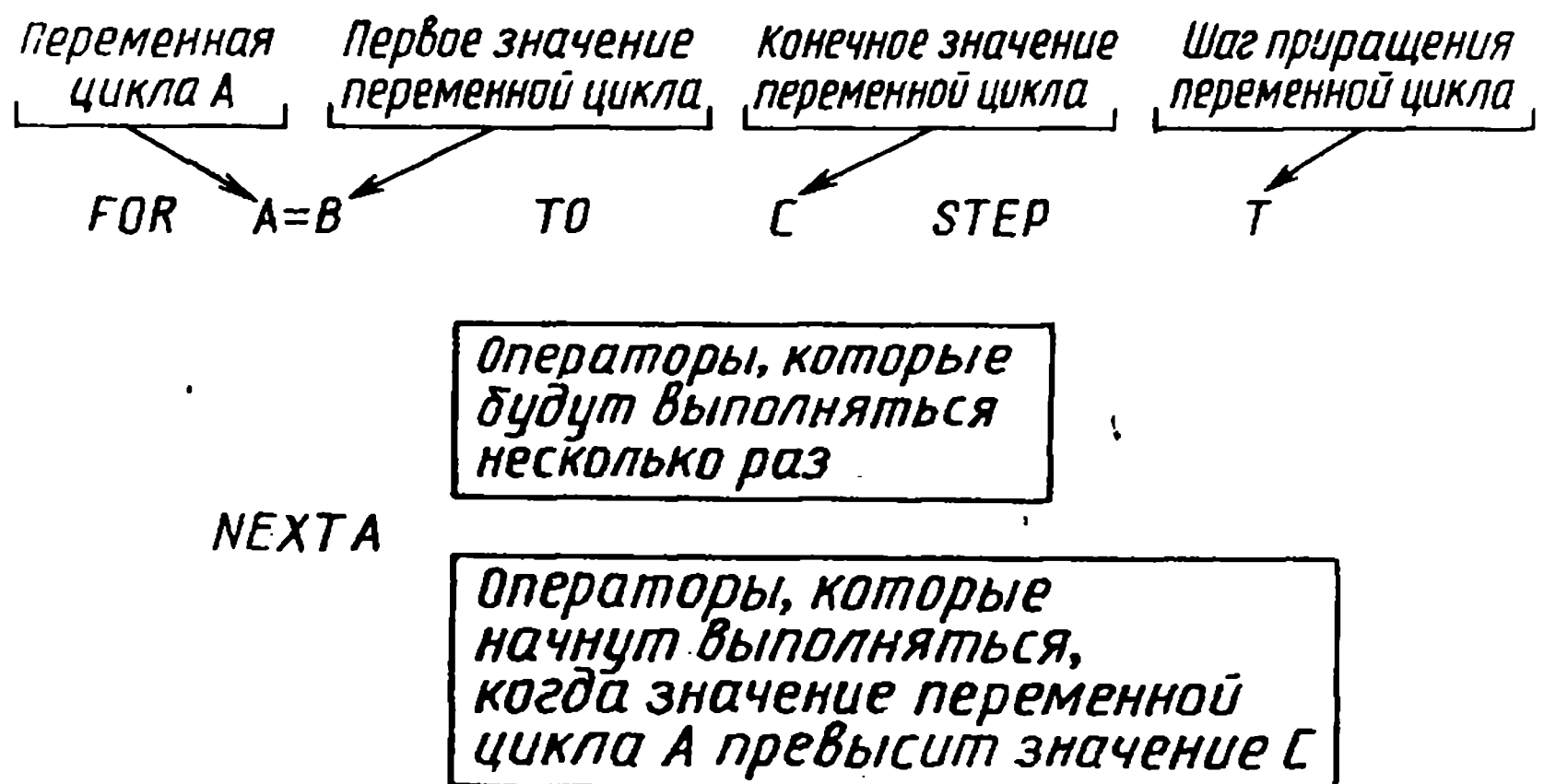
— «Нэксст» означает «следующий».

— Запомни, что после слова NEXT должно быть имя переменной цикла. Между этими двумя частями оператора цикла, совсем как между скобками, находятся те операторы, которые необходимо выполнить при всех допустимых значениях переменной цикла.

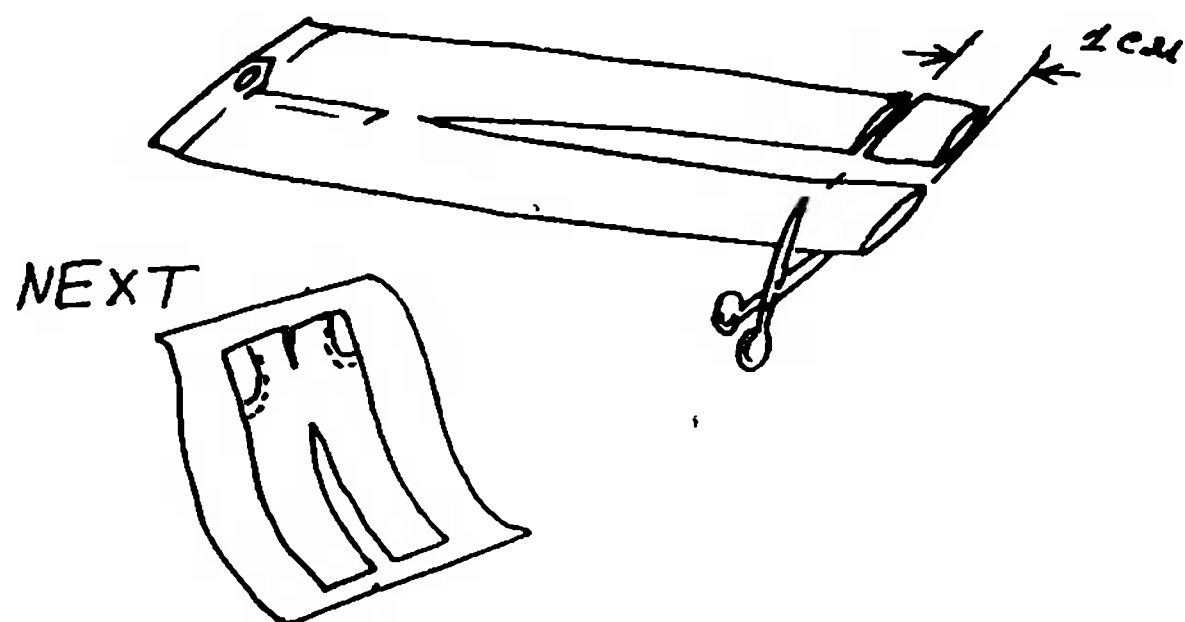
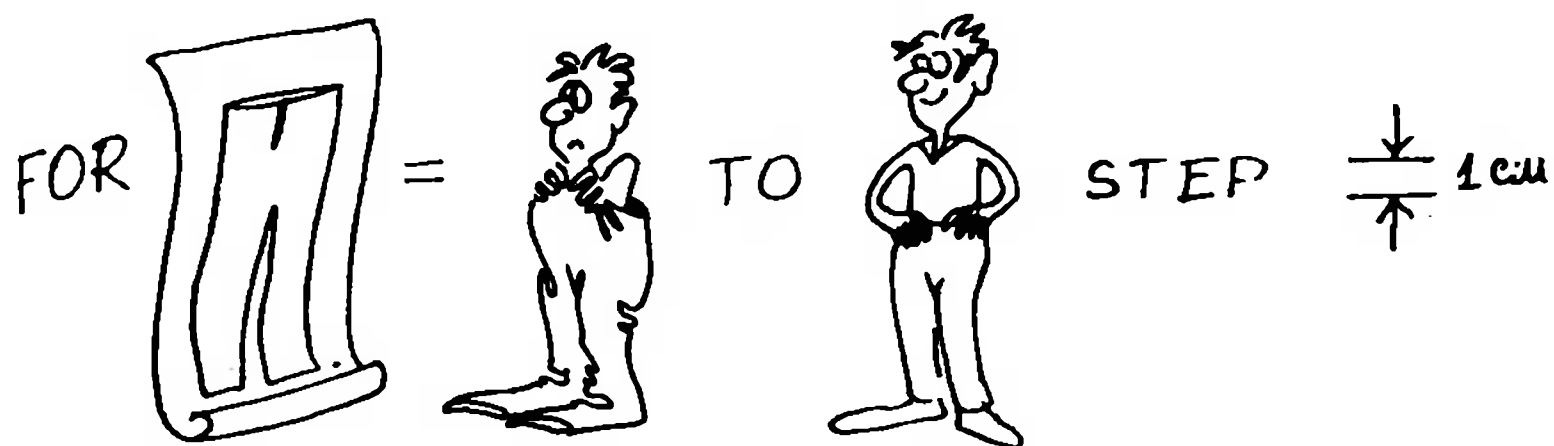
— Все допустимые значения цикла — это те, которые получаются, если прибавлять и прибавлять значение шага к первому значению переменной цикла?

— Да, но не забудь, что прибавление закончится тогда, когда значение переменной цикла станет больше конечного значения. Может случиться, что уже первое значение будет больше конечного, и тогда операторы внутри цикла не выполняются ни разу.

Сейчас я тебе еще раз поясню оператор цикла такой схемой:



Поизучай эту схему, кое-что станет яснее. Схема упрощена, ибо на самом деле вместо переменных B, C и T может быть любое арифметическое выражение. В то же время возможен и более простой вариант оператора цикла: если шаг приращения равен единице, то STEP 1 можно и не писать. В такой сокращенной форме наш оператор цикла из первого примера будет иметь вид:



Программа для начинающих закройщиков

```
FOR H=1 TO K
NEXT H
```

Это обеспечивает выполнение операторов внутри цикла для значений H , начиная с 1, потом 2, 3 и т. д. до K включительно. Разумеется, и в этой сокращенной форме вместо единицы и K можно писать арифметические выражения.

ПРАВИЛЬНЫЕ И НЕПРАВИЛЬНЫЕ ОПЕРАТОРЫ ЦИКЛА

Пора переходить к примерам. Правильен ли такой оператор цикла?

```
FOR H=100 TO 200 STEP K
...
NEXT H
```

— Думаю, что правилен. Переменная H будет пробегать значения, начиная с сотни, с шагом K до тех пор, пока очередное значение не станет больше 200.

— Найди последнее значение H , если K равно 6?

— Шести? У меня получается, что последнее значение будет $100 + 96$, т. е. 196.

— Точно. А этот оператор цикла правильный?

```
FOR Z=K+200 * H TO M/K + P
...
NEXT K
```

— Тоже правильный.
 — Нет, неправильный.
 — Почему это нет? Ты же сказал, что в операторе цикла можно использовать любые арифметические выражения.
 — Да, это я говорил. Но ошибка здесь в другом. После NEXT обязательно должна стоять переменная цикла — то же обозначение, что стоит после слова FOR, в нашем случае Z. Писать другое нельзя, и ничего не писать тоже нельзя.

— Ну и хитрец!
 — Есть немножко. О такой ошибке интерпретатор сообщение выдаст, но только встретив неправильный NEXT, т. е. выполнив все предыдущие операторы цикла. А сейчас посмотри на такой оператор цикла. Правильен ли он?

FOR K=6+A*B/C TO ((X+K)/A-X*C)*2

...

NEXT K

— Нет, неправилен.
 — Почему?
 — Не знаю, но правильных операторов ты больше показывать не станешь. Что я учителей не знаю?
 — Надо признать, что ты близка к истине, но все-таки посмотри и скажи, какой недостаток имеет этот оператор.

— Арифметические выражения быть могут. . . шаг, если он равен единице, указывать необязательно. . .

— Спору нет. Дело в другом.

— Не знаю. Сдаюсь.

— В этом примере для определения конца цикла используется выражение, содержащее переменную цикла K. Это плохой стиль. Человеку трудно в этом операторе разобраться, но интерпретатор выполнение программы не прекратит, он сначала вычислит первое значение переменной цикла K, т. е. $6 + A \cdot B / C$, потом станет вычислять конечное значение, но оно будет рассчитано с использованием именно этого значения K, так что фактически конечным значением будет $((X + (6 + A \cdot B / C)) / A - X \cdot C) \cdot 2$. Позже, при выполнении оператора цикла, когда переменная K будет приобретать новые значения, никакого пересчета не будет.

— Все ясно. Но что же здесь плохого?

— Это, разумеется, не ошибка. Это — трюкачество, нарушение общего принципа программирования: программировать по возможности проще, без выкрутасов, так, чтобы программу легко могли бы понять другие, да и ты сам, если придется к ней возвращаться через несколько месяцев.

Вот и последний пример. Как он тебе нравится?

```
FOR X=365 TO 0 STEP -1
```

```
NEXT X
```

— Вот это да! Неужели шаг может быть отрицательным?

— Да, может. В этом случае прибавление шага к значению переменной цикла фактически даст уменьшение этого значения, и, когда оно станет меньше конечного значения, цикл прекратится. Это бывает нужно, хотя и не очень часто.

— Все равно мне такая возможность нравится.

И ЕЩЕ РАЗ ТА ЖЕ САМАЯ ПРОГРАММА

— Посмотри сейчас на нашу программу ввода оценок. Теперь, когда ты знаешь оператор цикла, строку 15 можно заменить первой частью оператора, а вместо строк 40 и 50 вставить NEXT N. И тогда программа будет выглядеть так:

```
10 DIM A (50)
12 PRINT "ВВЕДИТЕ КОЛИЧЕСТВО УЧЕНИКОВ";
13 INPUT K
15 FOR N=1 TO K
20 PRINT "ВВЕДИТЕ ОЦЕНКУ";
25 PRINT "УЧЕНИКА НОМЕР"; N;
30 INPUT A (N)
40 NEXT N
60 END
```

Вывод порядкового номера ученика я вставил в строке 25 для того, чтобы пользователь точно знал, которому из учеников он вводит оценку. А точки с запятыми на концах всех операторов вывода появились для того, чтобы следующий выводимый знак, а это будет вопросительный знак, который покажет оператор ввода, был сразу после сообщений в той же самой строке. Да, и добавился ввод количества учеников в строке 12.

— Одного я не понимаю: стоило ли так возиться, если мы сэкономили только одну строчку? Если раньше, чтобы сделать цикл, нужно было написать три строчки, то теперь только две. Мелочь.

— Как посмотреть. Я лично думаю, что оператор цикла изобрели не из-за стремления к экономии. Он необходим человечеству по другим причинам. Важна сама идея цикла! Важно понять, что программы состоят из циклов. Что в задачах их надо искать и находить. И что в программах они должны быть ясно видны. Программист должен думать циклами, как говорит наш Алгоритмыч. Это надо понимать так, что циклы

должны лежать в основе алгоритмов и программ. Ну и экономия все-таки есть. Настоящие программы очень длинные, и если что-то можно укоротить, то это и надо делать, но ни в коем случае не за счет ясности программы.

ПРОГРАММА СУММИРОВАНИЯ ЧИСЕЛ

— А сейчас составь программу суммирования чисел от единицы до ста, конечно, используя оператор цикла.

— Ха! Не на ту попал! Я отлично знаю эту сказку об одном великом математике, которому учитель дал сложить все числа от единицы до ста, чтобы он какое-то время сидел тихо и не шалил, но ничего не вышло, потому что он мгновенно сообразил, что надо сначала написать все числа от единицы до ста, а потом под ними все числа от ста до единицы и попарно сложить.

$$\begin{array}{r} + \quad 1, \quad 2, \dots, 100 \\ \quad 100, \quad 99, \dots, \quad 1 \\ \hline 101, 101, \dots, 101 \end{array}$$

Всюду получатся одинаковые суммы 101, и таких сумм будет, конечно, 100. Сложив сейчас все эти суммы, мы получим 10 100, а так как каждое число в сумме участвовало 2 раза, надо эту сумму поделить на 2, и получится 5050. Это и есть искомый результат, так что нечего зря затруднять машину.

— Все равно составь программу, но бери не конкретное число 100, а произвольное N, которое введет пользователь.

— Ладно уж, ладно, хотя и для этого случая имеется формула: сумма равна $N(N+1)/2$. Так что программу я пишу только из уважения к тебе. Учти.

— Не трать зря время. Постарайся составить до того, как мы попадем на машину. Учти, что переменную цикла можно использовать в арифметических выражениях точно так же, как любое другое число.

Тоня взялась за работу, и в результате появилась следующая программа:

```
10 PRINT "ПРОГРАММА СУММИРУЕТ ЧИСЛА";
15 PRINT "ОТ 1 ДО N"
20 PRINT "ВВЕДИТЕ ЧИСЛО N"
30 INPUT N
40 S=0
50 FOR I=1 TO N
60 S=S+I
70 NEXT I
80 PRINT "СУММА РАВНА"; S
90 STOP
```

— Ух и намучилась, пока сообразила, что переменной для суммы S вначале надо присвоить нуль, а потом добавлять числа по одному. Ну как?

— Ничего. Программа работать будет.

Прождав еще некоторое время, ребята попали на машину.

— Ну, Тоня, вызывай интерпретатор!

>BAS

READY

Сначала будущую программу надо как-то назвать при помощи директивы NEW. Выполнение этой директивы хорошо еще и тем, что очищается память. Сейчас, когда мы только начали работу, это не имеет значения, но было бы важно, если бы ты уже вводила какую-нибудь программу в память. Введи NEW и название новой программы.

NEW SUMMA

— Ты доволен?

БЫСТРО И ПЛОХО!

ЭТО НЕ СТИЛЬ ПРОГРАММИСТОВ!

— Да, вполне. Начинай вводить программу, и побыстрее. У нас сегодня мало времени.

Тоня взялась за работу и вскоре сообщила:

— Готово. Пускаем.

— Сначала посмотрим.

— Нечего смотреть. Какие могут быть ошибки при вводе такой коротенькой программки?

RUNNN

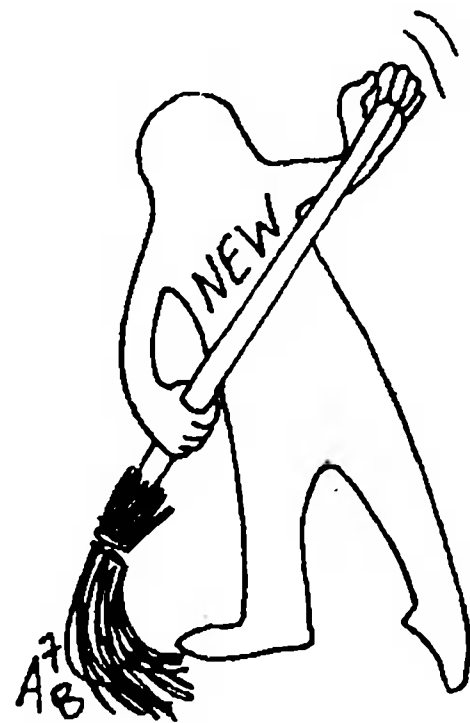
ПРОГРАММА СУММИРУЕТ ЧИСЛА ОТ 1 ДО N
ВВЕДИТЕ ЧИСЛО N

SYNTAX ERROR AT LINE 30

Это что за шуточки?!

— Ну, конечно! В строке 30 синтаксическая ошибка, т. е. такая строка в Бейсике невозможна. Скорее всего перепутаны буквы. Сумеешь посмотреть эту строку?

— Конечно, нужно использовать директиву LIST.



— Лучше LISTNN и номер строки.

```
LISTNN 30  
30 INPT N
```

— Да, действительно, там почему-то появился INPT, где должен быть INPUT.

— Появился, появился. . . и почему-то.

— Не страшно. Введу эту строку заново.

```
30 INPUT N  
RUNNN
```

```
ПРОГРАММА СУММИРУЕТ ЧИСЛА ОТ 1 ДО N  
ВВЕДИТЕ ЧИСЛО N  
?100
```

```
?FOR WITHOUT NEXT AT LINE 60
```

— Опять чушь какая-то!

— Ошибка, дитя, ошибка. Интерпретатор не смог найти соответствующее NEXT для какого-то FOR. В какой строке должен был быть NEXT?

— В строке номер 70.

— Посмотрим ее.

```
LISTNN 70
```

```
70 NXT I
```

Ну вот! Эта строка тоже ошибочно введена.

— Так, исправила.

— Тогда пускай еще раз.

```
RUNNN
```

```
ПРОГРАММА СУММИРУЕТ ЧИСЛА ОТ 1 ДО N  
ВВЕДИТЕ ЧИСЛО N  
?100  
СУММА БУДЕТ 5050
```

```
STOP AT LINE 110
```

```
READY
```

- Фу-у-у... Наконец-то все в порядке.
- Теперь быстро сохрани свою программу на диске, как я тебя учил.
- Все в порядке. Бежим домой.

11. ДЕЛАЙ, КАК Я, ДЕЛАЙ ЛУЧШЕ МЕНЯ!

Не каждый день можно попасть работать на вычислительную машину, поэтому Петя предложил посмотреть свои старые программы, которые он составлял в ученические годы.

— Знаешь, совсем не вредно посмотреть программы, составленные другими. Думаю, ты потом почувствуешь, что смогла бы сделать не хуже.

— Ну уж! Куда мне тягаться с такими корифеями программирования, как Петя Бейсикив!

ПРОГРАММА РЕШЕНИЯ САМОГО ПРОСТОГО УРАВНЕНИЯ

— Хорошо, хорошо. Вот первая программа, которая решает уравнение

$$AX=B,$$

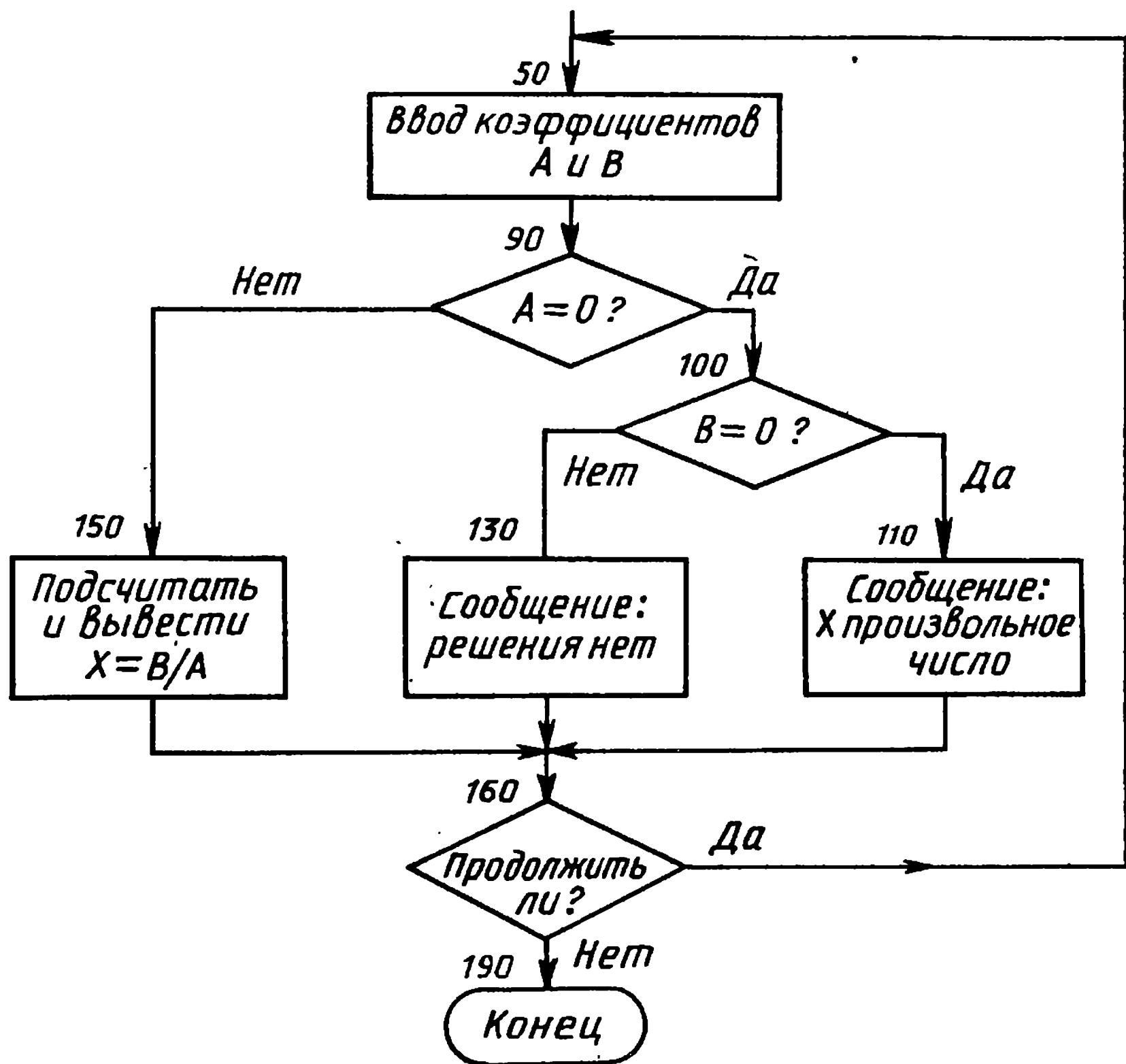
А и В — известные коэффициенты, X — неизвестная величина.

— Чтобы решить такое уравнение, институты кончать не надо.

— Не надо. Единственное, надо помнить, что проверяется равенство А нулю. Если равно, то проверяем значение В, и только в случае, если и это значение тоже равно нулю, уравнение будет иметь решение, даже бесконечно много решений. Если же В нулю не равно, то все — решения нет. Ведь что бы ни умножать на ноль, результат не может быть отличным от нуля.

— Да, приятно вспомнить молодость, когда это все казалось непостижимой премудростью.

— Программа составлена по такой схеме:



А сама программа выглядит так:

```

10 PRINT "ПРОГРАММА РЕШАЕТ УРАВНЕНИЕ А*Х=В"
20 REM ПЕРЕМЕННАЯ А-КОЭФФ. А; В-КОЭФФ. В
30 PRINT \ PRINT "ВВЕДИТЕ КОЭФФИЦИЕНТ А";
40 INPUT А
50 PRINT "ВВЕДИТЕ КОЭФФИЦИЕНТ В"; \ INPUT В
55 \
60 REM ПОСЛЕ ВВОДА А И В СРАВНИВАЕМ А С 0
70 REM ЕСЛИ А=0, ТО ВЫВОДИМ РЕЗУЛЬТАТ
80 REM В ЗАВИСИМОСТИ ОТ ТОГО В=0 ИЛИ НЕТ
85 \
90 IF А<>0 THEN 150
100 IF В<>0 THEN 130
110 PRINT "ОТВЕТ: Х ПРОИЗВОЛЬНОЕ ЧИСЛО"
120 GO TO 160
130 PRINT "ОТВЕТ: РЕШЕНИЯ НЕТ"

```

```

140 GO TO 160
150 PRINT "ОТВЕТ:      X=";B/A
155 \
160 PRINT \ PRINT "КОНЕЦ РАБОТЫ - ВВЕДИТЕ 0"
170 INPUT K
180 IF K<>0 THEN 30
190 STOP
200 END

```

Смотри, в схеме алгоритма я у каждого блока приписал номер строки, с которой начинается выполнение этого блока в программе.

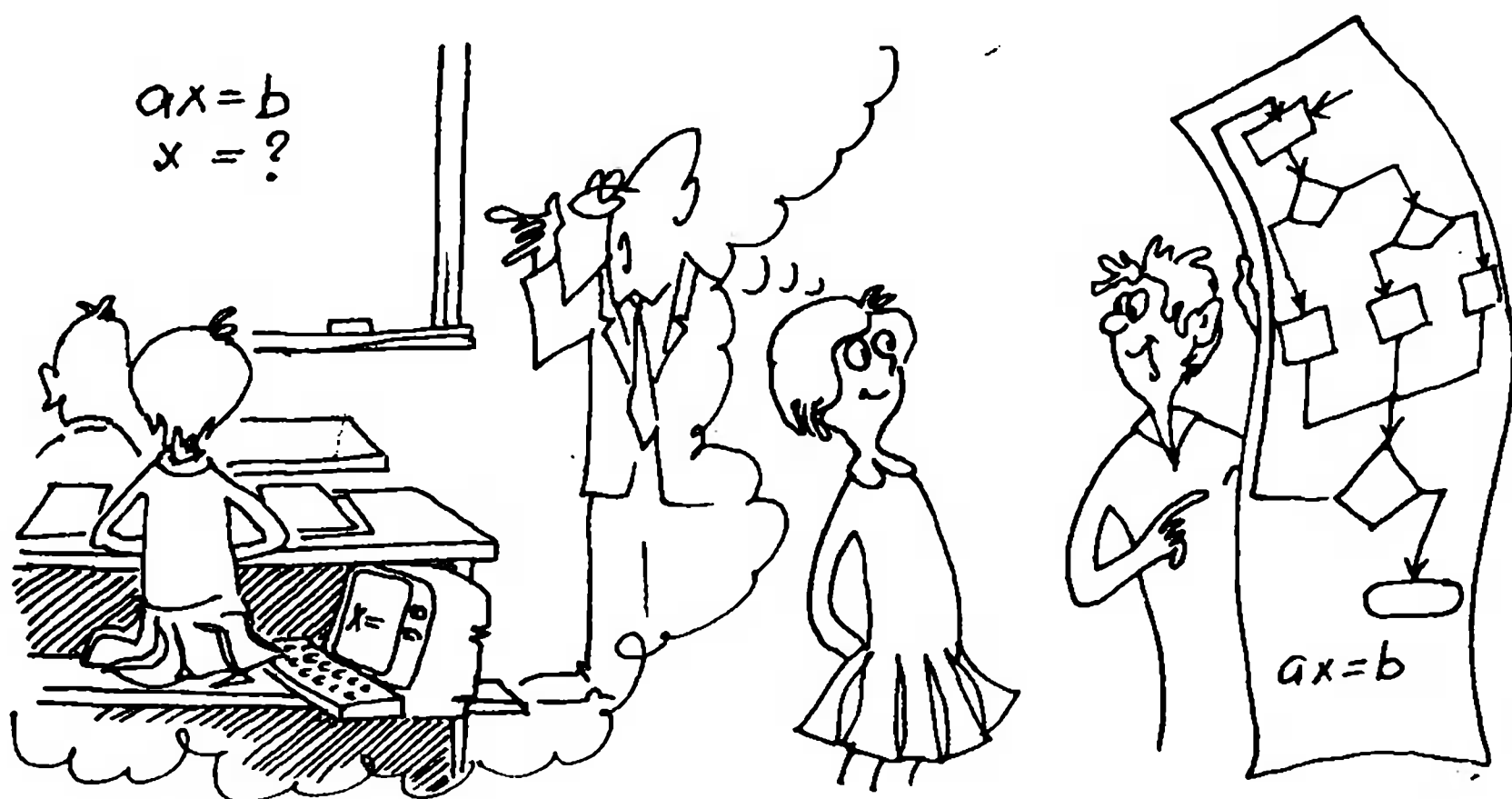
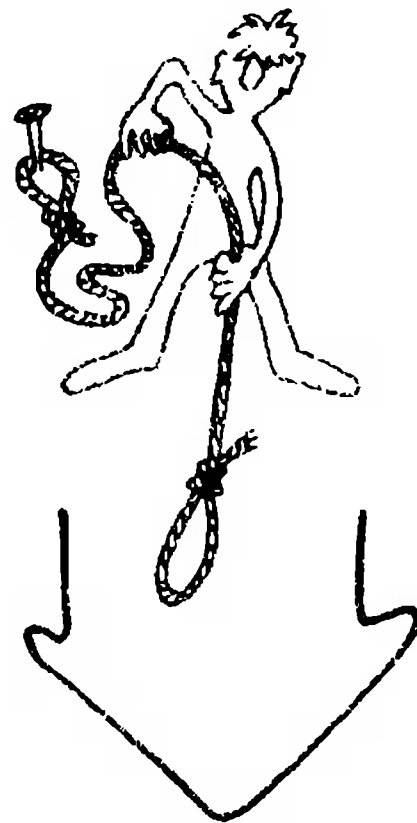
— Ну что ж. Начинаем изучать твою программу. В первой строке я ничего особенного не вижу. Выводится на экран сообщение, и все дела. А вот строка 20 начинается странно...

КАК ПОЯСНИТЬ СВОЮ ПРОГРАММУ

— Там появляется новый оператор — оператор примечаний, который записывается буквами REM, что является сокращением слова *remark* — «римак» — «примечание». Работа этого оператора крайне проста — он ничего не делает.

— Да, ничего не делать просто. А зачем же тогда нужен такой оператор?

— После REM в программе можно записать все, что хочется. Почти все, как ты поймешь из следующей строки про-



Контрольную всегда начинай со схемы алгоритма

граммы. Это могут быть пояснения работы программы: что она делает, каков ее алгоритм? В отдельных местах, сложнее, можно пояснить, что там делается.

— Это что, опять для пользователя? Но пользователь же сам программу не видит.

— Всекие бывают пользователи. Однако такие примечания нужны не только пользователю, но и самому программисту, когда через некоторое время он изрядно позабудет свою программу. Поэтому так важно, чтобы в программе были примечания. Не столь важно, чтобы их было много, но важно, чтобы они давали что-то новое по сравнению с самой программой. Например, без толку писать что-нибудь такое:

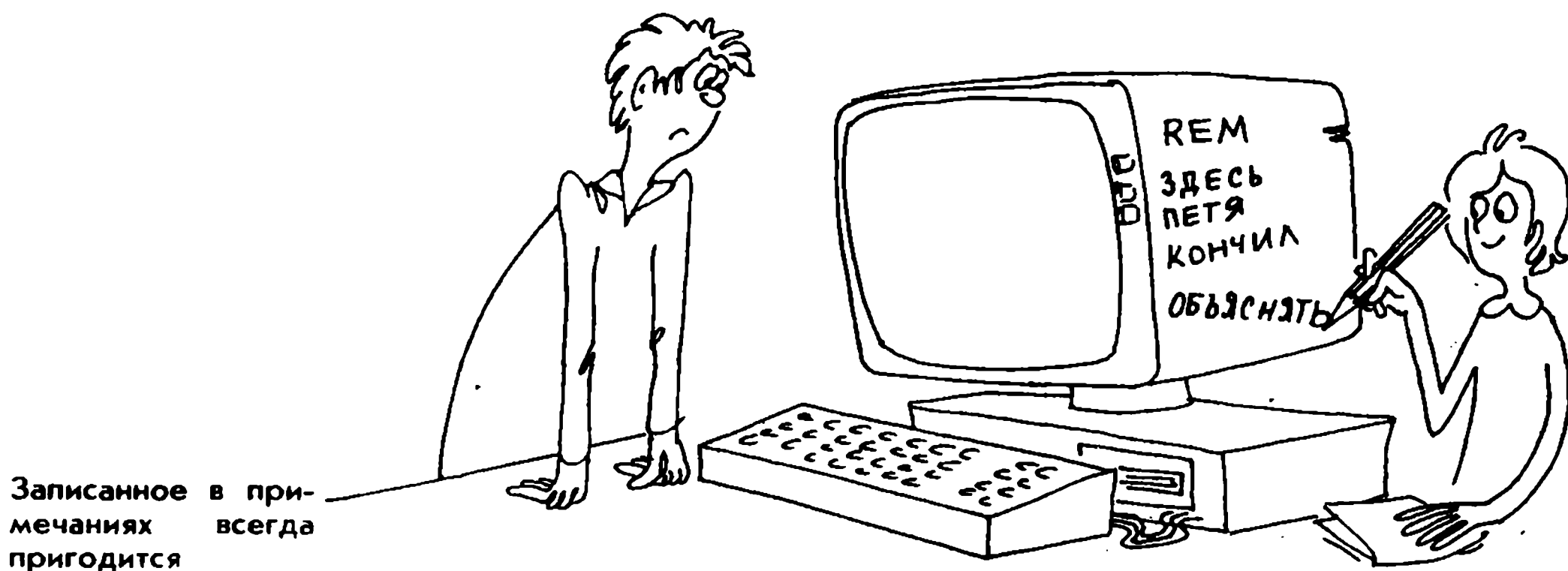
```
10 REM ПРИСВАИВАЕМ ПЕРЕМЕННОЙ А ЗНАЧЕНИЕ 7  
20 A=7
```

Здесь в примечании описан оператор Бейсика. И все.

— И как же было бы лучше?

— Лучше было бы написать, какой смысл имеет переменная А, например количество дней в неделе. А во второй строке своей программы я сделал примечание о том, какие переменные встречаются в этой программе и каков их смысл. Правда, говорю я только об основных переменных А и В, значения которых равны коэффициентам А и В.

В следующей строке появляется символ «\». Как я тебе уже говорил, это обратная косая черта. Ею отделяют один оператор от другого, если хотят записать в одной строке несколько операторов. В этой строке я сначала записал оператор PRINT без ничего — он выведет на экран пустую строку. А за ним следует другой оператор PRINT с текстом. Сейчас должно быть ясно, что в тексте примечания после оператора REM эта обратная косая черта не должна появиться. Ее появление означало бы, что примечание кончается и начинается следующий оператор. Обратную косую черту я использую



еще для отделения одной части программы от другой, как, например, в строке 55.

— Строку 40 можешь не пояснять, но следующая... Там тоже два оператора: PRINT и INPUT. Ничего страшного, понять можно. Однако хорошо, что в одной строке можно писать несколько операторов.

— Хорошо-то хорошо, но увлекаться этим не советуют, потому что теряется наглядность программы. И еще, управление можно передать только на начало строки. На середину нельзя. Если мне когда-нибудь попозже понадобится бы при помощи оператора IF или GO TO передавать управление на оператор INPUT в строке 50, то пришлось бы эту строку переделать в две строки. Но я знаю, что этого мне не придется делать, так как и PRINT, и INPUT в строке 50 тесно связаны между собой.

— Далее идут примечания. Целых три строки. Там ты пишешь о том, как программа делает свое дело.

— Да, это я кратко описываю, и больше примечаний в этой программе не будет. Вообще-то нужно было бы, но у меня много операторов PRINT с достаточно понятными текстами для вывода, которые вполне можно читать как примечания. Смотри дальше.

— Ага! Теперь значение A сравниваешь с нулем. Если A не нуль, и это было бы естественно, ты идешь на строку с номером 150, но ее я смотреть не буду. Буду смотреть следующую строку, а там у тебя опять ветвление. Ну да, так это предусмотрено в схеме алгоритма. Если, например $B=0$, то выводится одно сообщение, а если не так, то другое. Думаю, это понять можно. После вывода обоих сообщений ты идешь на строку 160, в которой...

— Сначала, может быть, надо посмотреть строку 150, на которую попадаем, если A не равно нулю.

— Чего там смотреть, выводится частное B/A , и все. А потом прямым ходом на строку 160! Как же иначе, ведь и в схеме алгоритма после всех работ предусмотрен переход на блок с вопросом: что делать, продолжать работу или нет? Значит, сейчас должна решаться эта проблема. Что и делается в строках с номерами от 160 до 190. Прекрасная программа, Петушок! Поздравляю!

— Если начать «проразмысливать» эту программу, то сразу станет ясно, что здесь фактически запрограммирован расчет по одной формуле, и притом по очень простой.

— Не скромничай! Здесь же имеется столько сравнений.

— Да, эти сравнения действительно немножко осложняют программирование, но не настолько, чтобы особенно переживать. А вот проверка правильности работы программы из-за этих операторов сравнения становится и вправду труднее.

— Какая это еще проверка?

— Проверка того, что программа в самом деле делает то, что она, по моему замыслу, должна была делать, что она действительно решает уравнение с одним неизвестным.

— Это сделать легко! Дашь ей решить, например, уравнение $4x=8$, и если она получит ответ 2, то все в порядке.

— Нет, не все в порядке. Может быть, программа не справится с уравнением $4x=0$, ведь в этом случае счет пойдет по совсем другому пути, иначе сработает оператор сравнения коэффициента B с нулем и...

— Вот в чем дело! Тогда же надо будет еще и решить пример, когда нулю равен коэффициент A .

— Да. И кроме того, еще и пример, когда нулю равны оба коэффициента.

— Сложная наука!

— Зато, если все проверил, можешь быть полностью уверен в правильности работы своей программы.

Об этой программе хватит. Смотрим следующую.



12. НЕБОЛЬШОЙ ЭКСКУРС В БУХГАЛТЕРСКОЕ ДЕЛО

— Эта программа рассчитывает подоходный налог по заданной величине зарплаты трудящихся государственных предприятий.

— Ужас, какие серьезные дела! Я, например, и знать не знаю, сколько мои предки получают и какие налоги платят.

— Могла бы и поинтересоваться. Правда, я и сам, только работая с этой программой, узнал, по какому алгоритму рассчитывали подоходный налог до 1990 г. И этот алгоритм такой: с зарплаты, меньшей 70 р., никаких налогов не удерживают — это так называемый необлагаемый минимум. Налог удерживают только с полных рублей, копейки отбрасываются. Если тебе положено 71 р. 99 к., то налог будешь платить только с 71 р. Для суммы 71, 72 или 73 р. налог N рассчитывается по формуле

$$N=0,34A-23,89,$$

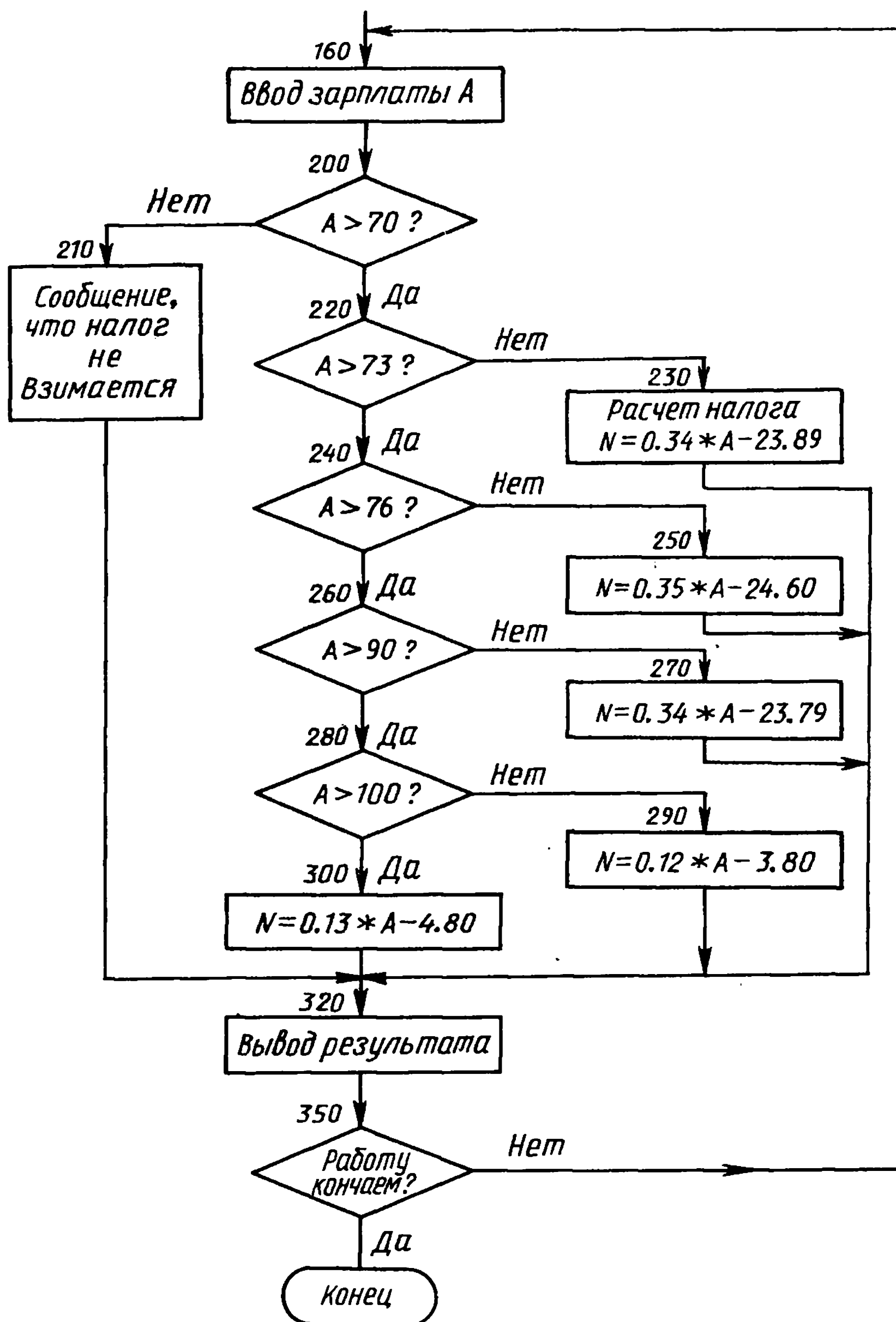
где через A обозначена зарплата. Для следующих сумм формулы будут такими:

$$N=0,35A-24,60, \text{ если } 74 \leq A \leq 76;$$

$$N=0,34A-23,79, \text{ если } 77 \leq A \leq 90.$$

Дальше начинается такая наука: с зарплаты больше 90 р. и меньше 101 р. налог рассчитывают как с суммы 90 р. плюс еще 12% от суммы выше 90 р. Получится формула

$$N = 0,12A - 3,80.$$



Наконец, начиная со 101 р., налог платят как со 100 р. плюс еще 13% от суммы выше 100 р. Для этого случая получается такая формула: $N=0,13A-4,80$.

— Поняла я не так уж много.

— Тогда посмотри еще схему алгоритма.

А теперь текст программы.

```
10 PRINT "ПРОГРАММА СЧИТАЕТ ПОДОХОДНЫЙ НАЛОГ"
20 REM A - ДОХОД, N - НАЛОГ
30 M=70 \ \ REM НЕОБЛАГАЕМЫЙ МИНИМУМ ЗАРПЛАТЫ
40 R1=.34 \ \ REM КОЭФФ. В ФОРМУЛЕ N=R1*A-Z
50 R2=.35
60 R3=.12
70 R4=.13
80 S1=73 \ \ REM КОНСТАНТЫ ДЛЯ СМЕНЫ ФОРМУЛ
90 S2=76 \ S3=90
100 Z1=23.89 \ \ REM КОЭФФ. В ФОРМУЛАХ
110 Z2=24.6
120 Z3=23.79
130 Z4=3.8
140 Z5=4.8
150 S4=100
155 \
160 PRINT \ PRINT "ВВЕДИТЕ ЗАРПЛАТУ В РУБЛЯХ"
170 INPUT A1
180 A=INT(A1) \ REM НАЛОГ НАСЧИТЫВАЮТ ТОЛЬКО С РУБЛЕЙ
185 \
190 REM НАЧАЛО РАСЧЕТА НАЛОГА *****
200 IF A>M THEN 220
210 PRINT "ЗАРПЛАТА ";A;" РУБ. НЕ ОБЛАГАЕТСЯ" \ GO TO 350
220 IF A>S1 THEN 240
230 N=R1*A-Z1 \ GO TO 310 \ \ REM 0.34*A-23.89
240 IF A>S2 THEN 260
250 N=R2*A-Z2 \ GO TO 310 \ \ REM 0.35*A-24.60
260 IF A>S3 THEN 280
270 N=R1*A-Z3 \ GO TO 310 \ \ REM 0.34*A-23.79
280 IF A>S4 THEN 300
290 N=R3*A-Z4 \ GO TO 310 \ \ REM 0.12*A-3.80
300 N=R4*A-Z5 \ \ REM 0.13*A-4.80
305 \
310 REM НАЛОГ РАССЧИТАН *****
320 K=INT(N*100+.5) \ N=K/100
330 R=INT(N) \ K=K-R*100 \ REM ОТДЕЛЯЕТ РУБ. ОТ КОП.
340 PRINT \ PRINT "НАЛОГ БУДЕТ ";R;" РУБ. ";K;" КОП"
350 PRINT \ PRINT "ВВЕДИТЕ 0 ДЛЯ ОКОНЧАНИЯ РАБОТЫ"
360 INPUT Z
370 IF Z<>0 THEN 160
380 STOP
390 END
```

— Давай сюда эту программу, сейчас я ее разнесу в пух и прах. Так, так. . . Первая строка печатывает, что программа собирается делать. Очень хорошо. Во второй строке примечания — зарплата, налог. . .

— В следующих строках я определяю необходимые для счета постоянные. Например, через *M* обозначаю необлагаемый минимум. Когда этот минимум повысят, в программе нужно будет поменять только строку 30 и не придется по всей программе искать, где стоит число 70 и что именно это 70 там означает. То же самое и с другими постоянными.

— Понятно! Это происходит в строках с номерами с 30 по 150. Потом пользователь вводит величину зарплаты. Это происходит в строке 170. А в строке 180? Что там делается?

НУЖНЫЕ ВСЕМ ФУНКЦИИ

— Здесь? Самое интересное в этой программе. Ты знакомишься с первой встроенной функцией в своей жизни программиста. Иногда человек должен со значениями своих переменных сделать что-то особое, отличающееся от четырех арифметических действий. Для таких нужд имеются встроенные функции. У них, как и у переменных, имеется обозначение, или имя, например *INT*, после которого в скобках пишут арифметическое выражение, с которым надо делать то, что функция делает. Таких функций не очень много. . .

— А почему их называют встроенными?

— Потому что они уже имеются в интерпретаторе Бейсика независимо от того, нужны они программисту или нет. И еще для того, чтобы отличить от тех, которые в случае необходимости может образовать сам программист, но об этом позже.

Встроенные функции



Целую часть находит
встроенная функция
INT

— И что бы это INT значило?

— INT — это сокращение английского слова integer, которое произносится «интэджер» и означает «целый», конечно, если речь идет о числах, а не о горшках. Функция INT находит целую часть числа, точнее, целую часть того арифметического выражения, которое записано в скобках за словом INT. Например, если $A = 4.56789$, то $INT(A/2)$ будет 2. Разве не просто?

— Великолепная вещь — программирование! Человек совершенно незаметно для себя повторяет и закрепляет арифметику.

— Возвратимся к строке 180. Как ты уже знаешь, налог начисляют только из рублей, копейки отбрасываются, что на языке математики означает: взять целую часть зарплаты, выраженной в рублях.

— Блестяще! А это что за звездочки в строке 190?

— Да просто для красоты! Отделяют вводную часть программы от основной. Оператор примечаний можно использовать и в таких целях.

— А в строке 200 зарплату уже сравнивают с необлагаемым минимумом. Если зарплата оказалась больше, то перепрыгивают строку с номером 210, которую надо было выполнить, если бы зарплата оказалась меньше минимума. Так оно и есть! В этой строке выводится сообщение, что зарплата не облагается, и управление передается куда-то вперед. Потом увидим; куда именно.

Дальше. В строке 220 проверяют, не будет ли зарплата, случаем, больше 73 р. А на строку 230 попадают только тогда, если зарплата больше необлагаемого минимума, т. е. 70 р., и кроме того, не больше 73 р. Кажется, все идет по формулам.

— И совершенно так же проводятся расчеты и в других случаях. Все в программе было тщательно проверено, чтобы налог определялся с точностью до копейки.

— А вот когда налог рассчитали, происходит что-то необыкновенное. Я имею в виду строку 320.

ОШИБКИ ОКРУГЛЕНИЯ В КОМПЬЮТЕРЕ НЕИЗБЕЖНЫ

— Видишь ли, при расчетах в машине возникают ошибки округления и, скажем, вместо правильного результата 5 может получиться 4.99999, что, конечно, весьма близко к 5. Но если машина выведет, что налог равен 4.99999 р., любой бухгалтер со смеху умрет. К сожалению, первые варианты программы так и делали, и поэтому понадобилась строка 320. Здесь я делаю следующее.

Сначала, умножив на 100, превращаю рубли в копейки и потом добавляю полкопейки, чтобы в случае, если вместо 500 к. расчет дал 499.999 к., получить 500.499 к., взяв целую часть которых я получу 500 к., как и надо. А если результат и так был правильным, то добавление полкопейки и взятие целой части его не испортят.

Еще я хотел в рассчитанном налоге отделить рубли от копеек: например, если получилось 5.42 р. налога, то я хочу отдельно вывести 5 и 42. Понять это отделение рублей и копеек не так-то просто. Смотри, здесь в строке 330 я от общего количества копеек К вычитаю количество полных рублей, помноженное на 100. В результате будет...

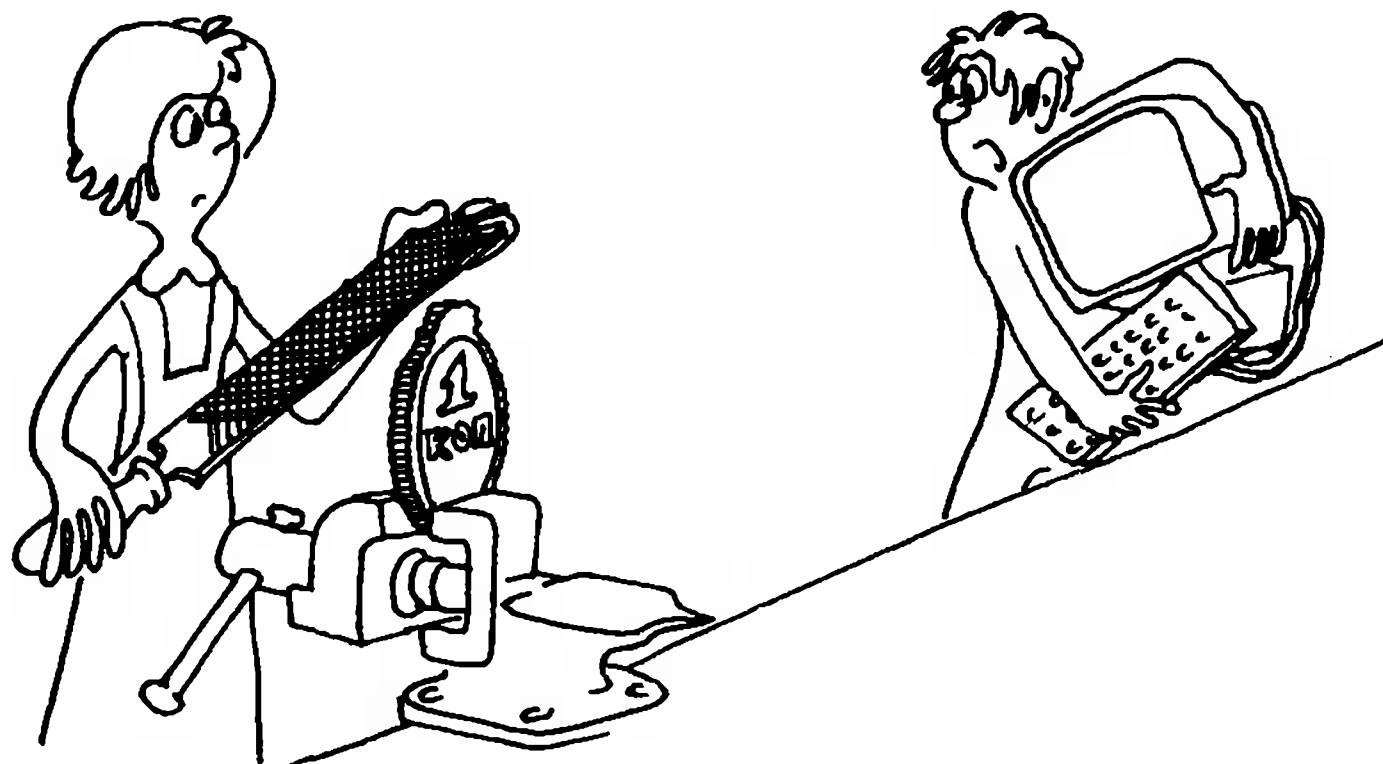
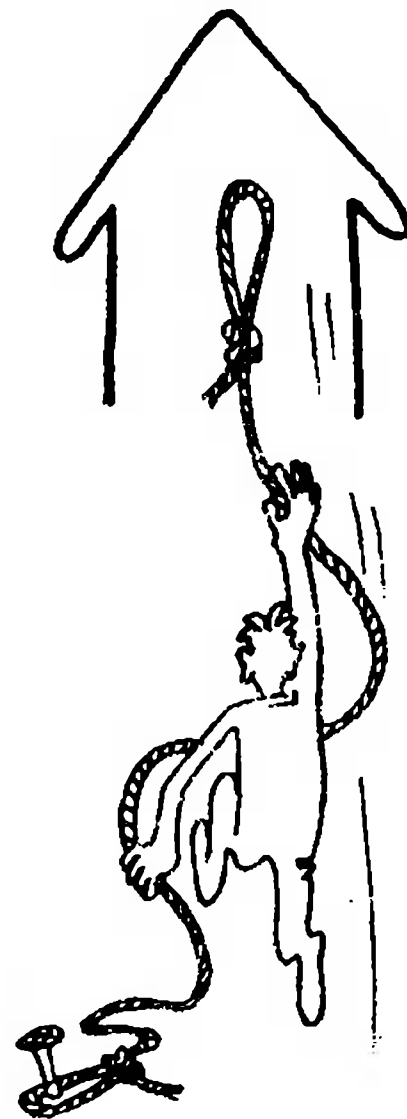
— Количество копеек! Это совсем несложно!

— Еще остались вывод результатов и запрос, хочет ли пользователь работать с этой программой дальше. Для такой программы очень важно, чтобы она всегда выдавала правильные результаты, поэтому я ее тщательно проверил, пропустив для всех значений зарплаты от 71 до 90 р., для некоторых — между 90 и 100, а также для некоторых, больших 100. Так много вариантов надо было пропускать, чтобы убедиться, что каждая из используемых формул считается верно. У меня значение рассчитанного налога всегда совпадало с тем, которое получается по таблицам, используемым настоящими бухгалтерами. Те таблицы и составлялись по этим формулам, но только вручную.

— Ну ладно. Вообще мне кажется, что можно твои программы и вправду посмотреть: кое-что проясняется. Может, завтра другие посмотрим, если у тебя еще есть?

— Есть, есть. Посмотрим завтра, а сегодня прошу тебя обратить свой взор на следующие задачи:

1. Составить программу, которая генерировала бы таблицу подоходного налога для всех сумм в пределах от 71 до



Стой! Копейки округляются не так!

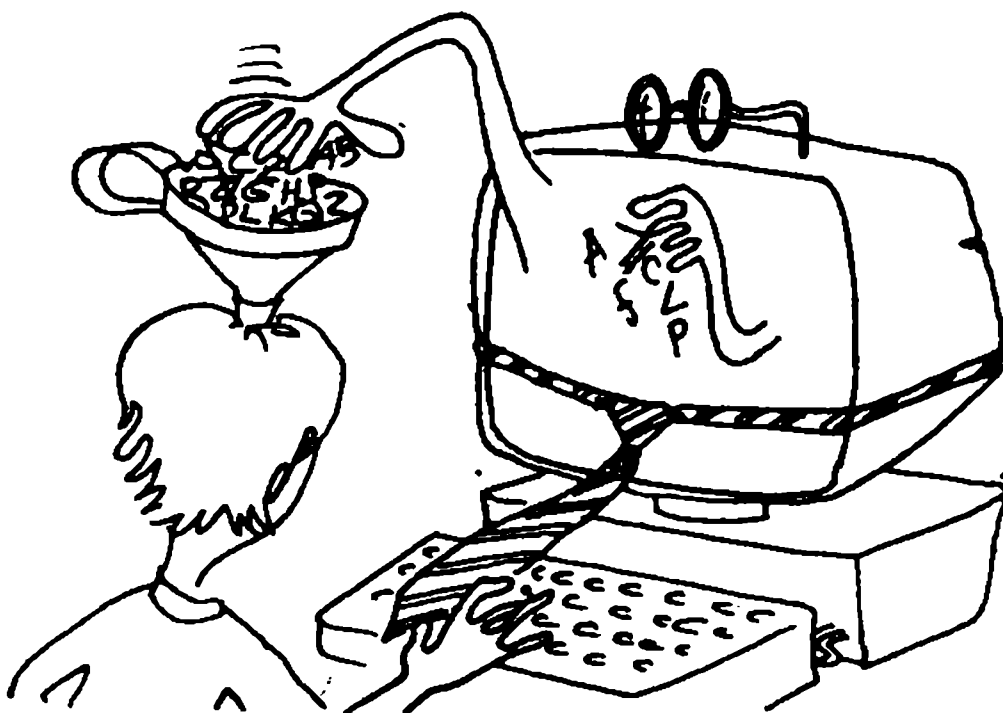
300 р. Добиться, чтобы программа выводила на каждой строке экрана несколько значений исходной суммы и соответствующего ей подоходного налога.

2. Составить программу, вычисляющую подоходный налог, который следует взимать с человека, занимающегося индивидуальной трудовой деятельностью. Необходимые данные нужно узнать самостоятельно, поскольку в каждом городе они могут быть различными.
3. Составить аналогичную программу для члена кооператива.
4. Составить программу для расчета комсомольских взносов, исходя из суммы доходов.
5. При пересылке по почте простой бандероли за первые 50 г веса нужно платить 20 к. За каждые последующие полные или неполные 50 г требуется уплатить еще 5 к. Составить программу для расчета платы за пересылку бандероли по заданному весу.

13. КОМПЬЮТЕР ИЛИ УЧИТЕЛЬ?

На следующий день Петя принес программы посложнее.

— Ты, конечно, уже слышала, что, как считают некоторые, компьютер может не хуже учителя обучать детей и что скоро всех учителей заменят машинами. Другие тут возражают, думают, что машина ничему толковому не научит. Пока они спорят, изучим одну обучающую программу. Разумеется, совсем простую. Да обучающие программы и не бывают особенно сложными. Считается, что самое важное — не запрограммировать, а все продумать, как программа будет обучать, чтобы чему-то научить.



НАУКА ОБУЧАТЬ НАЗЫВАЕТСЯ МЕТОДИКОЙ

— Моя программа обучает таблице умножения, точнее, повторяет ее. И методика обучения такова. Берутся два целых случайных числа. Потом о случайных числах расскажу поподробнее. Если первое равно единице, то заменяем его на 7. Если же второе, то на 8. Оба числа показываем обучаемому, он их должен в уме помножить и ввести результат. Если результат оказался правильным, то хорошо, если же нет, то отмечаем в памяти, что ученик ошибся. Так делаем столько раз, сколько ученик желает. В самом конце выводим оценку.

— А где тут методика?

— Методика в том, что мы совсем не даем примеры умножения на единицу, такие все умеют решать. Зато мы увеличиваем количество примеров с умножением на 7 и 8.

— Значит, ты считаешь, что примеры с девяткой легче решить, чем примеры с семеркой или восьмеркой?

— Я в этом не сомневаюсь. Умножая на 9, нужно число помножить на 10 и отнять само число, например:

$$7 \times 9 = 7 \times (10 - 1) = 7 \times 10 - 7 = 63.$$

— Да ладно уж, что я таблицу умножения не знаю?

— Теперь можешь посмотреть программу.

```
10 REM ПРОГРАММА ДЛЯ ПОВТОРЕНИЯ ТАБЛИЦЫ УМНОЖЕНИЯ
20 REM A И B - СЛУЧАЙНЫЕ ЧИСЛА, X - ИХ ПРОИЗВЕДЕНИЕ
30 REM P - КОЛИЧЕСТВО ПРИМЕРОВ, P1 - ОЦЕНКА,
32 REM K - КОЛИЧЕСТВО ОШИБОК
35 \
40 FOR I=1 TO 24 \ PRINT \ NEXT I
50 PRINT "ПРИВЕТ, ДРУЖОК!" \ PRINT
60 PRINT " СЕЙЧАС ПОВТОРИМ ТАБЛИЦУ УМНОЖЕНИЯ"
70 PRINT "Я БУДУ СПРАШИВАТЬ, ТЫ ОТВЕЧАЙ"
80 PRINT " СКОЛЬКО ПРИМЕРОВ ТЕБЕ ПОКАЗАТЬ"; \ INPUT P
85 \
90 IF P>=5 THEN 120
100 PRINT "ЭТО МАЛО, БЕРЕМ 5" \ P=5
110 GO TO 140
120 IF P<=100 THEN 140
130 PRINT "ЭТО СЛИШКОМ МНОГО, БЕРЕМ 100" \ P=100
140 PRINT \ PRINT \ PRINT " ВНИМАНИЕ !!! СТАРТ !!!"
150 K=0 \ \ REM ВНАЧАЛЕ КОЛИЧЕСТВО ОШИБОК 0
155 \
160 FOR I=1 TO P
170 A=INT(RND*9)+1 \ \ REM ВЫБОР ПЕРВОГО ЧИСЛА
180 IF A<>1 THEN 200
190 A=7 \ \ REM ЕСЛИ A=1, БЕРЕМ 7
200 B=INT(RND*9)+1 \ \ REM ВЫБОР ВТОРОГО ЧИСЛА
210 IF B<>1 THEN 230
```

```

220 B=8 \ \ REM ЕСЛИ B=1, БЕРЕМ 8
230 PRINT "СКОЛЬКО БУДЕТ ";A;" УМНОЖИТЬ НА ";B;" ";
240 INPUT X \ \ REM ВВОД ОТВЕТА
250 IF X=A*B THEN 280
260 K=K+1 \ \ REM УВЕЛИЧИВАЕМ КОЛИЧЕСТВО ОШИБОК
275 PRINT " НЕВЕРНО! ПРАВИЛЬНЫЙ ОТВЕТ ";A*B
280 NEXT I
285 \
290 IF K=0 THEN 360
300 PRINT "БЫЛО ";P-K;" ПРАВИЛЬНЫХ ";K;
305 PRINT " НЕПРАВИЛЬНЫХ ОТВЕТОВ"
310 P1=INT((P-K)/P*5+.5) \ \ REM ВЫБОР ОЦЕНКИ
320 IF P1<>0 THEN 340
330 P1=1 \ \ REM ОЦЕНКИ 0 НЕТ
340 PRINT "МОГЛО БЫТЬ ЛУЧШЕ! СТАВЛЮ ";P1
350 GO TO 370
360 PRINT "ОТЛИЧНО !!! МОЛОДЕЦ !!! ПЯТЬ !!!"
370 PRINT \ PRINT
375 \
380 PRINT "ХОЧЕШЬ ПРОДОЛЖИТЬ, ВВЕДИ 1" INPUT Z
390 IF Z=1 THEN 80
400 PRINT \ PRINT "КОНЕЦ РАБОТЫ! ДО СВИДАНИЯ!"
410 STOP
420 END

```

— Посмотрим, посмотрим, как ты собираешься детей мучить. Этот последний оператор REM мне очень нравится: в нем легче легкого разобратся. А вот этот оператор цикла — в строке 40, вот он-то потруднее будет. Ого! И целых три оператора в одной строке!

— Ничего особенного! Этот цикл выводит на экран 24 пустые строки. Так сказать, очищает экран, потому что на экране имеются ровно 24 строки.

— Дальше идет милое сюсюканье. Усыпляешь бдительность ребенка. Так и надо! Что-то существенное начинается в строке 80: вводится количество примеров. И сразу в следующей строке его сравнивают с 5...

— Это для проверки ввода. Ввод всегда нужно проверять. Мало ли где человек может ошибиться. Я считаю, что правильное количество примеров от 5 до 100. Если пользователь по ошибке или нарочно вводит другое число, я это число заменяю либо на 5 в строке 100, либо на 100 в строке 130.

— Прекрасно! Ты, мол, дружок, введи сколько хочешь, а мы скажем, сколько «берем». Вот это действительно смахивает на педагогику.

— Но ведь это же разумно: брать больше одного или двух примеров и не брать слишком много!

— Разумно, разумно. Я просто дурачусь. Здесь не класс,

где нужно сидеть спокойно. Смотрим дальше. Строку 140 я понимаю полностью — она для запугивания ребенка. В строке 150 ты какой-то переменной K присваиваешь значение нуль. Что такое K? Смотрим примечания в начале программы. Количество ошибок. А-а! В самом начале этот бедняга еще никаких ошибок не наделал, поэтому их количество и равно нулю. Дальше цикл... начинается в строке 160... Интересно, а где он кончается? В строке 280. И очевидно, все происходит именно в этом цикле. А это что такое — в строке 170? Ничего не понимаю.

ГЕНЕРАТОР СЛУЧАЙНЫХ ЧИСЕЛ — ФУНКЦИЯ RND

— Это ты увидела новую встроенную функцию, которая выдает случайные числа. Ее обозначение RND получено сокращением английского слова random — его произносят «рэн-дом», и означает оно «случайный». Сейчас самое время поговорить о случайных числах. Наиболее простой способ их получения — бросать игральную кость. Если бросать много раз, то все числа от 1 до 6 выпадут примерно одинаковое количество раз. А то, какое число выпадет в следующий раз, предугадать нельзя. Единственное, что можно сказать: вероятность выпадения каждого числа равна одной шестой.

— А случайные числа? Что же это такое?

— Если, бросая кость, ты будешь записывать выпадающие числа, то и получишь равномерно распределенные случайные числа от 1 до 6: 2, 3, 1, 5, 4, 3, 2, 1 и т. д.

А сейчас представь себе игральную кость с двадцатью гранями, и на гранях написаны числа от 1 до 20. Бросая эту кость, ты получишь равномерно распределенные натуральные случайные числа от 1 до 20.

Случайные числа



Выбор случайного числа

Но вернемся к функции RND. Она не имеет аргумента. Обращаясь к этой функции, получаем случайные действительные числа, равномерно распределенные в интервале $[0; 1]$.

— Нет, этого я не понимаю.

— Хорошо, попробую попроще. Когда я обращаюсь к этой функции, она закрывает глаза и острым карандашом тычет в интервал от 0 до 1. И попадает в какое-то число, например в 0.76812. В следующий раз, когда я обращаюсь к ней, она попадает совсем в другое число, например в 0.38631.



При этом попасть в любое место у нее равная вероятность. Ее совсем не тянет тыкать ближе к концам или к середине интервала. Это и означает, что получаемые числа — «равномерно распределенные». Если же выражаться совсем просто, для детей, то можно сказать, что при помощи функции RND компьютер «задумывает» одно число от 0 до 1.

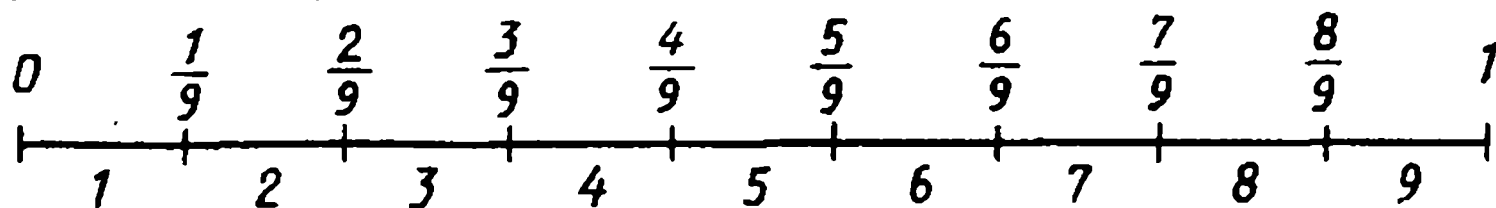
— Наконец я начинаю понимать.

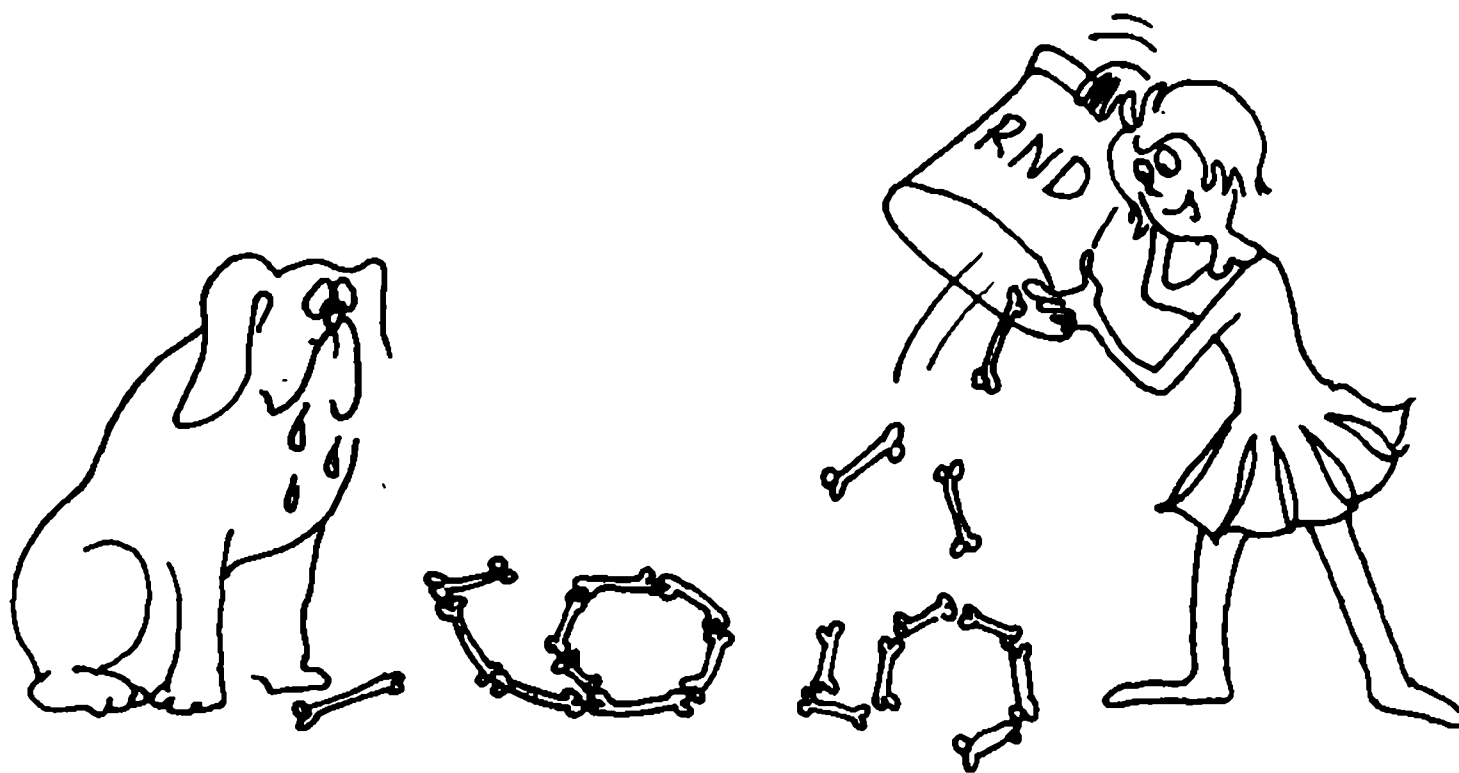
— Может быть, поймешь еще лучше, когда узнаешь, зачем мне это.

ЗАЧЕМ НУЖНЫ СЛУЧАЙНЫЕ ЧИСЛА

Представь, что я хочу сделать игральную кость с девятью гранями, чтобы иметь случайные числа от 1 до 9, которые я буду давать ученику перемножать. А такой кости нет и вообще быть не может. Поэтому я интервал от 0 до 1 разбиваю на девять одинаковых частей и даю функции RND тыкать в этот интервал. Если она попадает в пятую часть, то я говорю, что мне выпала пятерка, если в третью, то тройка и т. д. На самом деле мне функция RND дает только число C между нулем и единицей и мне самому надо определить, в какую из частей оно попадает. Это делается так: полученное число делим на $\frac{1}{9}$ и узнаем, сколько девятых оно содержит.

Например, если $C=0.76812$, то, деля на $\frac{1}{9}$, или, что то же самое, умножая на 9, получим 6.91308. Значит, это число находится за шестью девятыми, но ближе семи девятым, т. е. в седьмой части.





RND выбрасывает
кости

Как найти эту семерку или в крайнем случае шестерку? Здесь неоценимую помощь оказывает наша знакомая — функция INT. Скажи, чему равна целая часть 6.91308?

— Да шести же!

— Осталось немного. Чтобы рассчитать, в какую часть интервала попала функция RND, нужно добавить единицу к этой целой части. Все это делается в строке 170. И в строке 200, где выбирается второе число. Подчеркну, что, повторно обращаясь к функции RND, мы всякий раз получаем другое случайное число. Почти так же мы поступили бы и в случае, если нужно было бы иметь целое случайное число от 1 до произвольного числа M . Тогда вместо 9 в строке 170 подставили бы переменную M .

— Ты слишком далеко умчался. Я так и не поняла, что происходит в строке 180... Так-так, ты полученное число сравниваешь с единицей, и поскольку единицы тебе не нравятся, то ты переделываешь значение A с единицы на семерку согласно своей методике обучения.

— И подобным образом я поступаю с переменной B . Смотри строки 200, 210 и 220.

— Ладно. И строка 230 хочет у бедного ребенка узнать, какое же все-таки получается произведение.

— Да, действительно. Но остановись, эту строку полезно поизучать. Видишь пробелы после слов СКОЛЬКО БУДЕТ и также после и перед УМНОЖИТЬ НА? Они нужны потому, что значения переменных выводятся сразу после этих текстов; видишь точки с запятыми после A и B ? И еще одна мелочь: после значения B выводим пробел, так как следующим идет оператор INPUT, который в той же строке покажет вопросительный знак. И будет красивей, если значение B и вопросительный знак отделим пробелом.

Кстати, этот вопросительный знак оператор INPUT показывает всегда, и отменить его показ нельзя, поэтому я в выводе перед вводом обычно задаю вопросы. Как в этом слу-

чае: СКОЛЬКО БУДЕТ, а не НАЙДИ или как-то еще по-другому.

— Прекрасно. Ученик ввел ответ, и ты в строке 250 сравниваешь этот ответ с правильным результатом.

— Если все правильно, я сочиняю очередной пример. При этом, как видишь, я ухожу на оператор NEXT, чтобы увеличить значение переменной цикла — счетчика примеров и вообще чтобы цикл работал как надо.

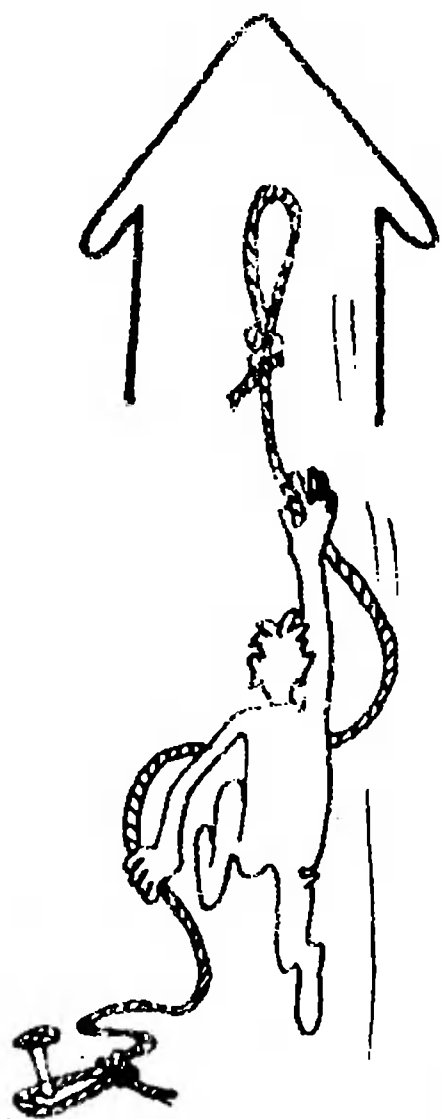
— А если ответ неправильный, увеличиваешь счетчик сделанных ошибок и выводишь неприятное сообщение.

— Еще и правильный ответ, чтобы человек знал.

— Сейчас цикл уже закончен, все примеры рассмотрены и мы находимся на строке 290. Что такое было K? Количество ошибок. Проверяем, не нуль ли количество ошибок... Если нуль, то идем на строку 360, а если не нуль... А-а! Выводится количество правильных и неправильных ответов. Да?

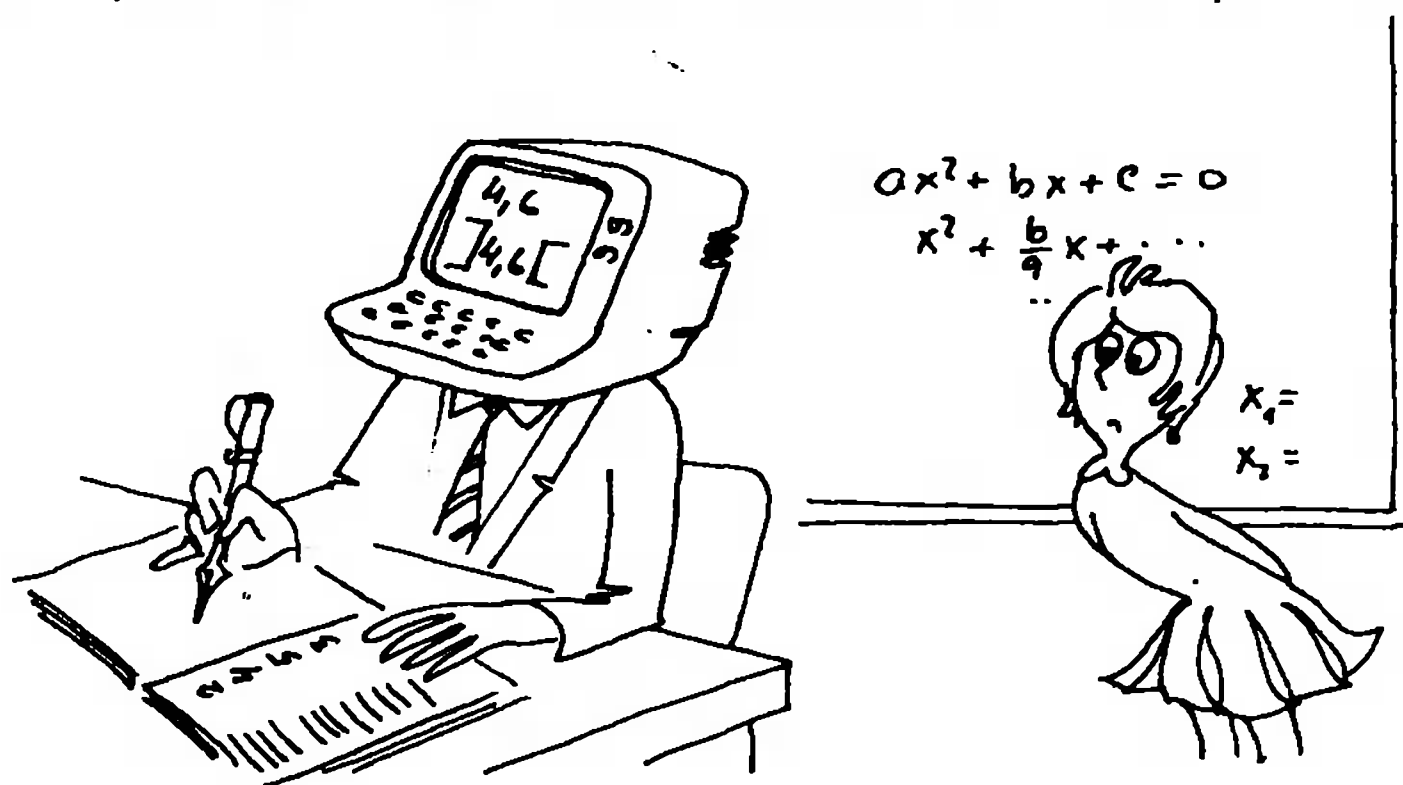
— Совершенно верно.

— И что это за P1 в строке 310? Оценка? И так ты вычисляешь оценку?!



ОЦЕНКИ — ВЕЩЬ НУЖНАЯ

— Да. Я делю количество правильных ответов на общее количество примеров и узнаю долю правильных ответов. Моя система оценок такая: если ты правильно ответил на 4/5 вопросов, то получаешь четверку, если на 3/5, то тройку и т. д. до кола. Поэтому я должен узнать, сколько пятых из всех ответов были правильными. Чтобы это узнать, я делю долю правильных ответов на 1/5, так же как я делал, когда искал случайное число от 1 до 9. Таким способом я получаю оценку, но с десятичными дробями за запятой, например 2,8 или 4,6. Целую часть этого числа в качестве оценки брать нельзя,



Какую оценку я получу?..

потому что оценка получится заниженной: при 2,8 вместо тройки, что было бы более справедливо, ученик получит двойку, при 4,6 — четверку. Чтобы так не произошло, я добавляю 0,5 и только потом беру целую часть.

— Ты очень любезен.

— И еще что. Если правильных ответов очень мало или даже нет ни одного, то может получиться оценка нуль, что в наших школах не предусмотрено. Поэтому в строках 320 и 330 этот возможный нуль переделывается в единицу. И потом...

— И потом вывод оценки. Это прекрасно! Я говорю о предложении «Могло быть лучше!». Оно служит неопровержимым доказательством, что компьютер может быть учителем. Если кто и сомневался, то, увидев эту фразу на экране, бесспорно согласится, что машина может полностью заменить учителя.

— Не знаю, но мне кажется, что компьютер может только чуточку помочь учителю, подучивая школьника каким-то мелочам.

— Ну знаешь ли, Бейсигов! Я сама отлично понимаю, что ни в коем случае нельзя заменять учителей машинами. Кто же захочет дразнить машину? И разве ты сможешь машине втемашить, что вчера вечером жутко болела голова и ты не смог выучить таблицу умножения и ставить тебе пару бессердечно?

— Вернемся к программе.

— А там ничего не осталось. Строка 360 выводит... Что же она выводит? А-а! На эту строку приходят, если ошибок вообще не было. И потом запрос о продолжении работы. Это как всегда.

— И теперь...

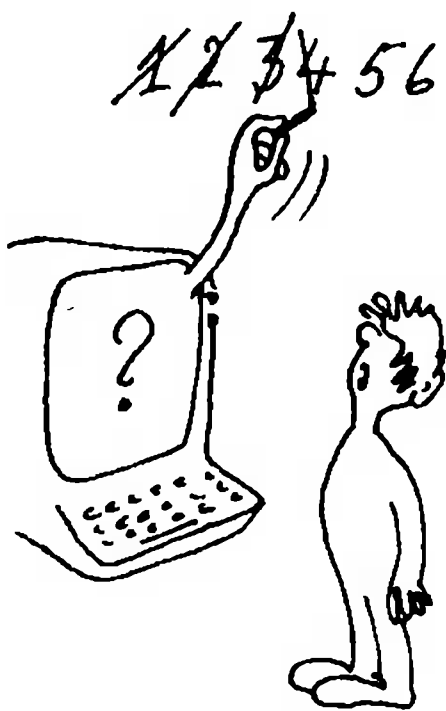
— Теперь надо отдохнуть. Было совсем не так легко изучать творения твоего гения. Пошли во двор, порезвимся. Я только прихвачу ракетки для бадминтона.

14. ГЛАВНОЕ — ЭТО УЧЕТ

Вернувшись, ребята стали бодро заниматься, так как Тоне хотелось кончить изучение следующей программы пораньше: по телевизору ожидался концерт Раймонда Паулса.

— Лучше не будем спешить. Мы же не крепостные? Не кончим сегодня, продолжим завтра.

— Знаешь что, Бейсигов, твое образцовое мышление иногда дает позорные сбои. Любой классный руководитель, и электронный, и этот... э... физиологический, тебе скажет, что начатое дело нельзя откладывать на потом.



ЧЕМУ ЕЩЕ МОЖНО УЧИТЬ ДЕТЕЙ!

— В любом случае нечего терять зря время. Давай «про-размыслим» последнюю программу обучения таблице умножения. Берешься ли ты составить, например, программу, которая обучала бы сложению в объеме первой десятки?

— А эту твою программу с таблицей умножения можно смотреть?

— Отчего же нет?! Это же не контрольная. В программировании ты можешь смотреть все, что хочешь. Главное, чтобы программа работала как надо. Хотя нет, по программированию тоже могли бы быть контрольные, но на сей раз это не контрольная.

— Если так, то я, конечно, составила бы. В текстах для вывода я бы заменила все про таблицу умножения на все необходимое для сложения, и еще, чтобы полученный школьником результат можно было бы сравнивать с правильным, я этот правильный рассчитала бы, складывая оба числа.

— Хорошо, но как со сложением в пределах сотни?

— В пределах сотни? Это будет потруднее. Тут надо понимать, как работать с функцией RND.

— Как же иначе! Но если все-таки не понимаешь, то запомни: когда нужно действительное число в пределах от нуля до X, пиши так:

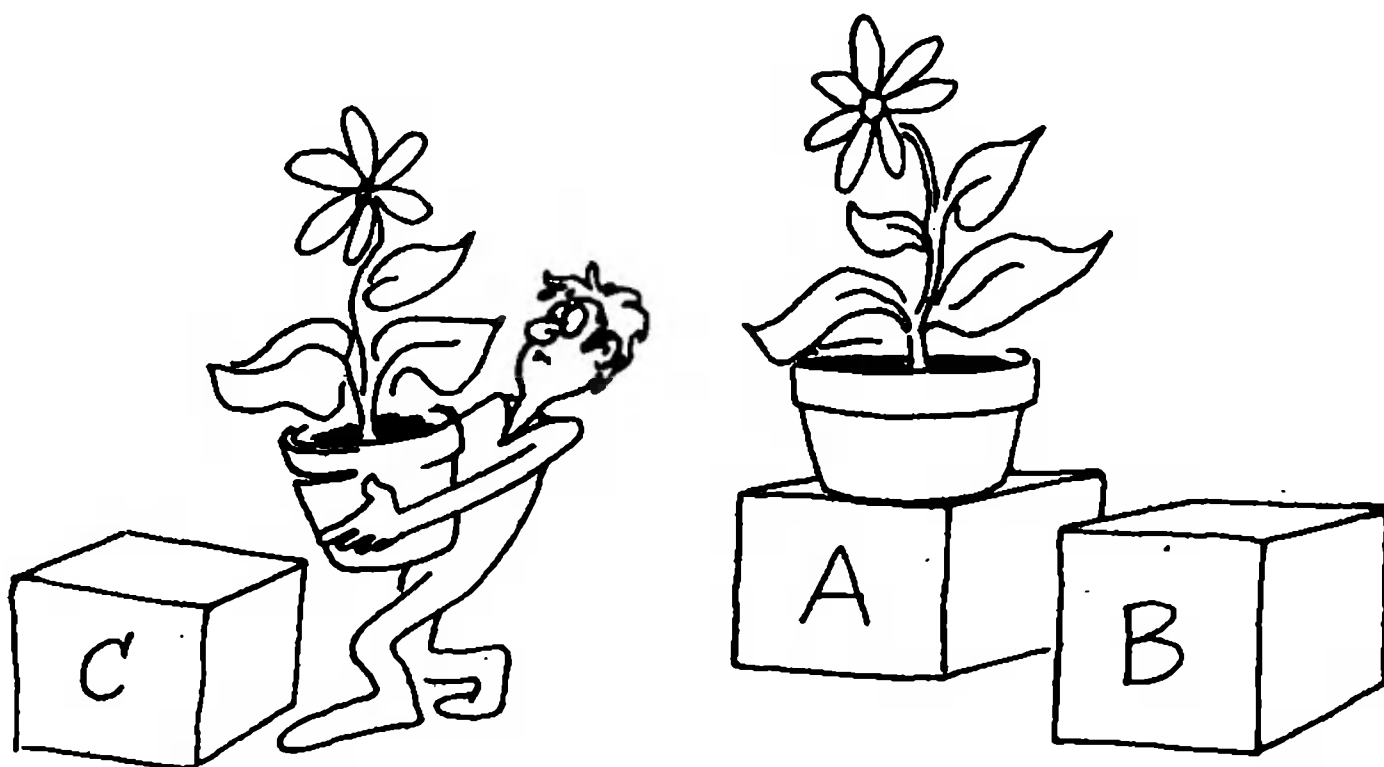
$$X * RND$$

Ну, а когда нужно целое число в пределах от единицы до N, пиши так:

$$INT(N * RND) + 1$$

— Значит, случайные числа в пределах сотни я смогу получить, написав выражение

$$A = INT(100 * RND) + 1$$



— Без сомнения.

— Тогда я такую программу составить берусь! Более того, я берусь с этими бедными детьми повторить вычитание в пределах сотни.

— А это уже будет не столь просто! Как ты поступишь, если разность случайных чисел окажется отрицательной? Маленькие дети про отрицательные числа ничего не слыхали!

— Что же делать?... Может быть, в таком случае поменять числа местами?

— Меняй смело. Сейчас расскажу, как это делать. С ходу можно и не догадаться. Поменять местами значения двух переменных А и В можно только при помощи третьей переменной, например С. Это будет выглядеть так:

$C = A$

$A = B$

$B = C$

Но такая замена не единственное, что можно делать с этими отрицательными числами. Еще можно брать следующую пару чисел. Рано или поздно попадутся такие два числа, разность которых окажется положительной. Само собой, в программе должна быть проверка разности, не отрицательна ли она?

— Что-то можно было бы предпринять и с делением.

— Еще бы! Например, пусть ученик определяет, сколько раз второе число входит в первое и каков остаток.

— Вычислить остаток при делении я не берусь...

— А мы это уже один раз делали. Когда зарплату в копейках выражали в рублях. Тогда мы определяли, сколько раз в соответствующее число входит сотня и каков остаток. Тебе это еще казалось очень простым делом.

— Ну, ну... Все равно немножко надо подумать.

— Немножко надо подумать всегда, а иногда даже и побольше. Вообще эта программа может служить болванкой для всяких программ повторения разнообразных формул. Например, превращения минут в секунды или наоборот, суток в часы или наоборот и т. д., и т. п.

— Похоже, такие программы превращения были бы даже проще, так как для них нужно только одно случайное число.

— Точно! А если бы и понадобились даже три, то ничего ужасного не произошло бы. Функция RND выдала бы их без замедления.

Еще стоило бы подумать об улучшении методики обучения. Программа могла бы следить за тем, какие примеры для школьника самые трудные: если, например, он путается в умножении на 7, то чаще давать такие примеры. Или через некоторое время повторить те примеры, в которых он ошибался. Такая программа была бы намного сложнее.

— Кажется, у тебя была еще какая-то программа, которую стоило бы посмотреть?

— Да, есть. И эта программа решает совсем другую проблему. Представь себе магазин тканей, и в нем какая-то ткань имеется в нескольких рулонах, или, как говорили в старину, штуках. Продавщицы отрезают ткань от всех штук, и им важно знать, сколько ткани в каждой штуке осталось, чтобы в конце не остались от штук маленькие куски, которые никому продать будет нельзя и придется списывать или уценивать.

— Что же продавщица может сделать?

— Она может немножко подсчитать. Если, например, в штуке с тканью для брюк осталось три метра, продавщица может отрезать три раза по метру, т. е. для коротких брюк, или же два раза по полтора метра, т. е. для длинных брюк. Если же она, не ровен час, отрежет метр двадцать, то останется метр восемьдесят, и такой отрез целиком никто не купит. Придется отрезать, допустим, метр сорок, а остаток в сорок сантиметров останется вообще непроданным. Поэтому, если покупателю нужно метр двадцать, правильнее будет брать другую штуку.

— Очевидно, ты редко бываешь в магазинах. В таких случаях продавцы говорят: вот вам метр восемьдесят, покупайте целиком, резать мы не будем.

— Мы играем в магазины высокой культуры обслуживания, выражаясь по-научному, их моделируем.

— Пусть будет так. Но чем можешь помочь ты со своим компьютером?

— Не так уж и много. Может быть, и можно придумать такой алгоритм, который по информации о наиболее часто покупаемых длинах отрезков подсказывал бы продавщице, от какой штуки отрезать требуемую длину. Но я такого алго-



ритма не знаю. Попробуй ты придумать, коль сильна! Я смог только составить программу, которая ведет учет остатка ткани в штуках. Программа имеет четыре режима работы. Первый — это начальный режим, когда вводим в память начальные длины штук. Во втором — основном режиме длины штук меняем, указывая, сколько от каждой отрезано. В третьем режиме показываем, сколько ткани в каждой штуке осталось, а последний режим заканчивает работу. И сама программа, естественно, делится на четыре части, которые достаточно просты, что позволяет не рисовать для тебя их схемы. Начнем сразу изучать программу.

```

10 REM ПРОГРАММА УЧИТЫВАЕТ КОЛИЧЕСТВО ТКАНИ В ШТУКАХ
20 REM G - ДЛИНЫ ШТУК, B - ИХ КОЛИЧЕСТВО, A - ДЛИНА
30 REM ТЕКУЩЕГО ОТРЕЗА, N - НОМЕР ТЕКУЩЕГО ОТРЕЗА
35 \
40 DIM G(50) \ D1=50 \ REM D1 - МАКСИМ. КОЛИЧЕСТВО ШТУК
50 REM В СТРОКЕ 40 ЗАМЕНИТЬ ЗНАЧЕНИЯ D1, ЕСЛИ ШТУК
52 REM МОЖЕТ БЫТЬ БОЛЬШЕ 50
55 \
60 PRINT "СКОЛЬКО ШТУК ТКАНИ ВСЕГО"; INPUT B
70 IF B<=D1 THEN 90
80 PRINT "СТОЛЬКО НЕЛЬЗЯ, ИЗМЕНИТЕ СТРОКУ 40" \ STOP
85 \
90 PRINT "РЕЖИМЫ РАБОТЫ:"
100 PRINT "      1 - ЗАДАВАНИЕ ДЛИН ОТДЕЛЬНЫХ ШТУК"
110 PRINT "      2 - ВВОД ДЛИН ОТРЕЗОВ"
120 PRINT "      3 - ПОКАЗ ДЛИН ОТРЕЗОВ"
130 PRINT "      4 - КОНЕЦ РАБОТЫ"
140 PRINT " КОТОРЫЙ РЕЖИМ "; \ INPUT R \ PRINT
145 \
150 ON R GO TO 160,220,370,460
155 \
160 PRINT "ЗАДАЕМ ПЕРВОНАЧАЛЬНЫЕ ДЛИНЫ ШТУК" \ REM1111111111
170 FOR I=1 TO B
180 PRINT "ДЛИНА ";I;"-й ШТУКИ В МЕТРАХ";
190 INPUT G(I)
200 NEXT I
210 GO TO 90
215 \
220 PRINT "ВВОД ДЛИН ОТРЕЗОВ" \ REM 22222222222222222222222222222222
230 PRINT \ PRINT "ПРИ ВВОДЕ НУЛЯ В КАЧЕСТВЕ НОМЕРА"
240 PRINT "БУДУТ ПОКАЗАНЫ ДЛИНЫ ШТУК"
250 PRINT "КАКОВ НОМЕР ШТУКИ"; \ INPUT N
260 IF N<>INT(N) THEN 280
270 IF N<=D1 THEN 290
280 PRINT "ВВОД НЕПРАВИЛЬНЫЙ, ПОВТОРИТЕ" \ GO TO 250
290 IF N=0 THEN 370 \ \ REM ХОЧЕТ ПОСМОТРЕТЬ ДЛИНЫ

```


с D1. И если штук оказалось больше, то выводим сообщение о заменах в строке 40.

— Потом начинается вывод основных сообщений. Пользователь узнает, какие существуют режимы работы, и получает предложение выбрать нужный режим, другими словами, ввести номер нужного режима. Надеюсь, что все это ты хорошо понимаешь. Неясности начнутся со строки 150. Здесь используется незнакомый оператор — оператор вычисляемого перехода ON. Это слово произносят «он», оно означает «на», хотя мы будем его переводить иначе — «в соответствии с».

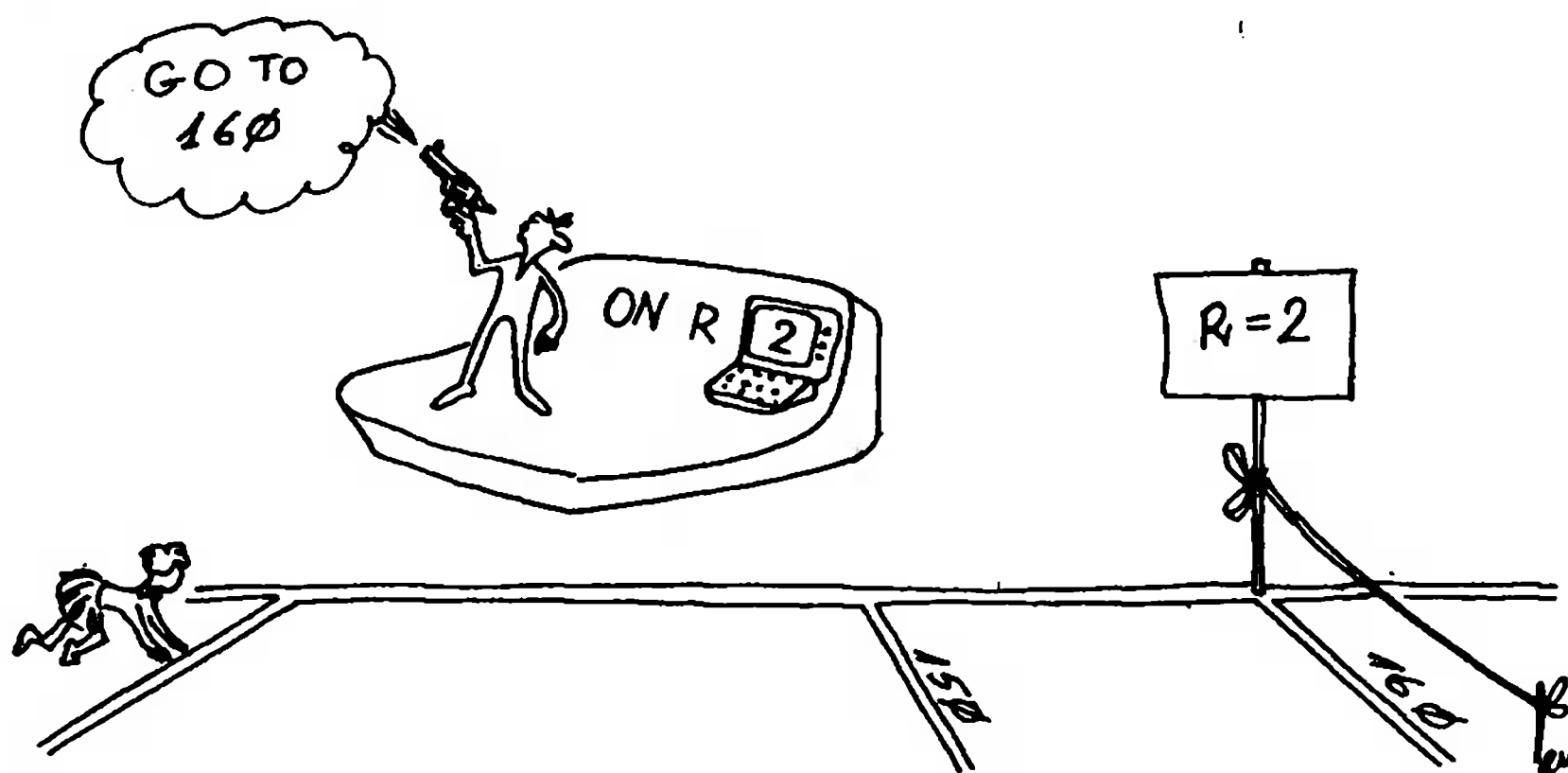
ЕЩЕ ОДИН ОПЕРАТОР ПЕРЕХОДА

— Запись самого оператора выглядит так:

ON	R	GOTO	160, 220, 370, 460	На Бейсике
В соответствии со	значением арифметического выражения R	иди на	строки с номерами 160, 220, 370 и 460	По-русски

Оператор вычисляемого перехода работает просто. Сначала проверяется значение арифметического выражения, в нашем примере значение переменной R. Если $R=1$, то управление передается на строку, номер которой идет первым после GOTO, в нашем примере 160; если же R получится равным 2, то на строку, номер которой указан вторым, и т. д.

Если же значение R получится таким, что столько номеров строк в операторе нет, например у нас всего четыре



номера, а значение R получилось 5 или —2, то выводится сообщение об ошибке и, что намного печальнее, программа прекращает работу. По этой причине ты сразу можешь указать серьезнейший недостаток моей программы: она не проверяет, ввел ли пользователь правильное значение R между 1 и 4. Если же проверка была бы, то неправильный ввод можно было бы заставить повторить и работа программы не прекратилась бы.

Чтобы ты не подумала, что этот оператор существенно новый, попробуй написать его старыми операторами IF.

— Надо по-по-по-по-думать... Этот самый переписывать?

— Конечно. Не стану же я новый придумывать.

— Хорошо, хорошо. Сейчас. Начну со строки 150.

```
150 IF R=1 THEN 160
```

```
151 IF R=2 THEN 220
```

```
152 IF R=3 THEN 370
```

```
153 IF R=4 THEN 460
```

— Совершенно верно. И вот теперь мы можем изучать первый режим — задание начальных длин штук.

ЧТО ДЕЛАТЬ В НАЧАЛЕ РАБОТЫ

— В тексте программы ты его выделил. Начальное задание длин происходит в строках, начиная со 160 и кончая 210.

— И представляет собой один цикл по всем штукам. В этом цикле вводится длина каждой штуки и засылается в массив G.

— И куда передается управление в конце ввода длин? Ага, ясно! На выбор нового режима. А что же тут выбирать? Надо переходить на ввод длин отрезков.

— Чаще всего, действительно, на второй режим. А если что-то случилось и пользователь хочет повторить ввод, потому что все здорово перепутал, или хочет посмотреть, что введено, а может быть ему все надоело и он хочет закончить?

ЧТО ДЕЛАТЬ ПОСЛЕ КАЖДОЙ ПОКУПКИ

— Второй режим начинается со строки 230 и кончается строкой 360. Длинный.

— Это потому, что ввод нужно проверять, а проверки всегда удлинняют программу. Но без проверок мне программировать не разрешали. Проверки должны быть, и баста!

— Почему такой странный вывод в строках 230 и 240?

— Представь себе ситуацию: человек все вводит и вводит длины отрезов и вдруг ему захотелось посмотреть, сколько в каждой штуке осталось, чтобы решить, от какого отреза сейчас резать. И тогда он введет нулевую длину отреза, что означает переход на показ длин штук. А чтобы этот секрет пользователь узнал, надо ему об этом сообщить. Что я и делаю в строках 230 и 240.

— Далее следует вопрос: каков номер штуки? Очевидно, той, от которой только что отрезали. Ввели... И тут номер штуки сравнивается с собственной целой частью! Интересно бы знать, кому это надо?

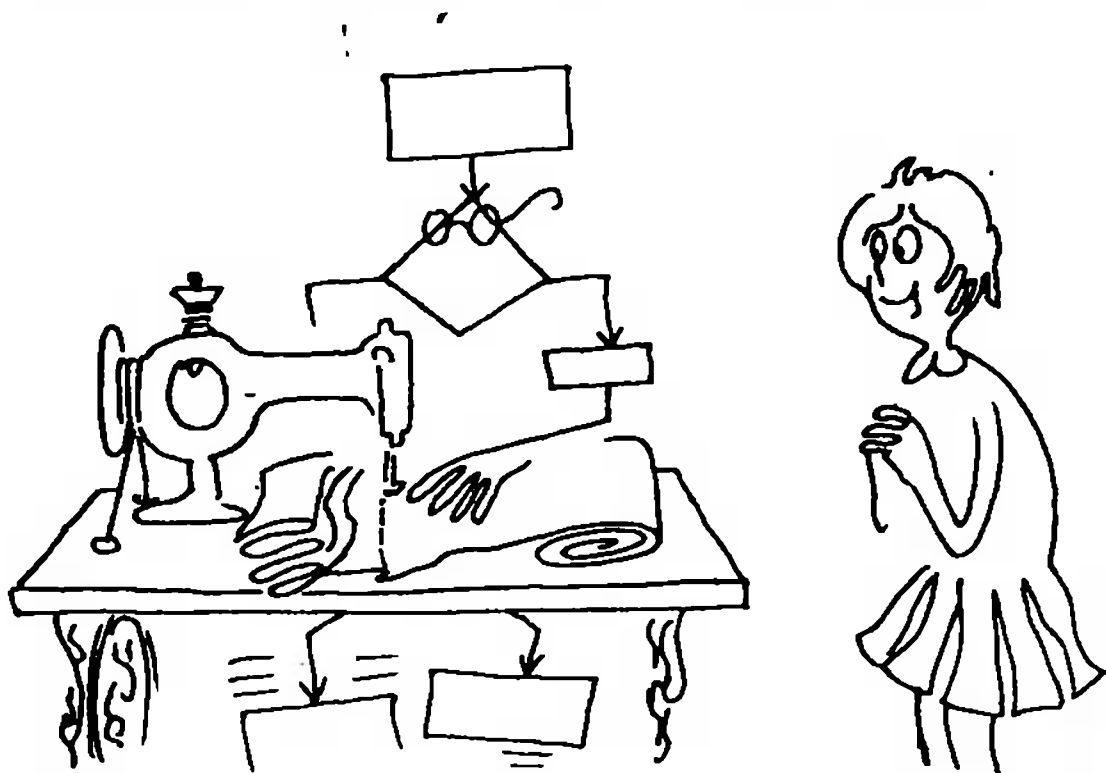
— Это уже проверка. Номер штуки должен быть целым числом. А целая часть целого числа совпадает с самим числом. Так можно убедиться, что ввели целое число.

— А зачем проверять такую мелочь? Кто станет вводить нецелый номер?

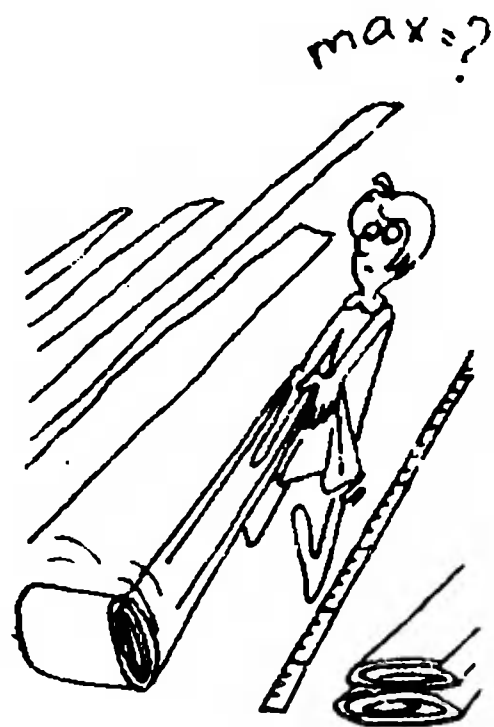
— Опытные программисты уверяли, что в этом месте такое запросто может случиться. Человеку хочется вводить все быстрее и быстрее, и впопыхах он может ввести вместо номера штуки длину отреза.

— Потом номер штуки сравнивается с нулем. Чтобы узнать, не надо ли уходить на показ. Если не надо, то вводится длина отреза, рассчитывается остаток ткани в штуке и сразу сравнивается с нулем. Ясно! Чтобы напечатать в строке 320 «Столько отрезать нельзя», если захотели отрезать больше, чем было в штуке.

— Вот что еще может случиться: человек ошибочно ввел длину отреза не для той штуки, для которой надо было, или ошибся при вводе длины отреза. Надо было вводить 1,45 метра, а он ввел 1,54 метра. И тут же спохватился. Исправление внести очень легко: надо от той же штуки отрезать минус 1,54 метра, т. е. в самом деле добавить эту длину, а потом можно и вычесть нужные 1,45. Не будь отрицательных чисел,



Алгоритм должен уметь исправлять ошибки



мне пришлось бы делать еще один режим — режим исправления.

— Общий принцип ясен. Программист должен признавать за пользователем общечеловеческое право ошибаться и должен предвидеть в своих алгоритмах, что делать, если он действительно ошибся.

— Далее проверяю, не равна ли нулю длина этой штуки, т. е. не продана ли уже вся штука. Если так, то сообщаю об этом пользователю. Если не так, то в строке 350 присваиваю новое значение длине соответствующей штуки. И запрашиваю следующий номер штуки для обработки.

— Этим режим кончается, и можем посмотреть режим показа в строках 380—450. Что это за максимальная длина показываемых штук M в строке 380?

— Все штуки сразу пользователя не могут интересовать. Ему могут понадобиться длины только тех штук, которые близки к концу. Например, те, которые уже короче двух или трех метров. Само собой, можно посмотреть и длины всех штук, задав M больше длины самой длинной штуки.

— Прекрасно. Дальше написан цикл по всем штукам, и похоже, что ты не собираешься показывать нулевые длины проданных штук.

— Да, именно поэтому я в строке 410 длину каждой штуки сравниваю с нулем.

— А в следующей строке ты сравниваешь длину штуки с этой максимальной длиной и обходишь показ в строке 430, если он не нужен. И когда все показано, уходишь на выбор нового режима.

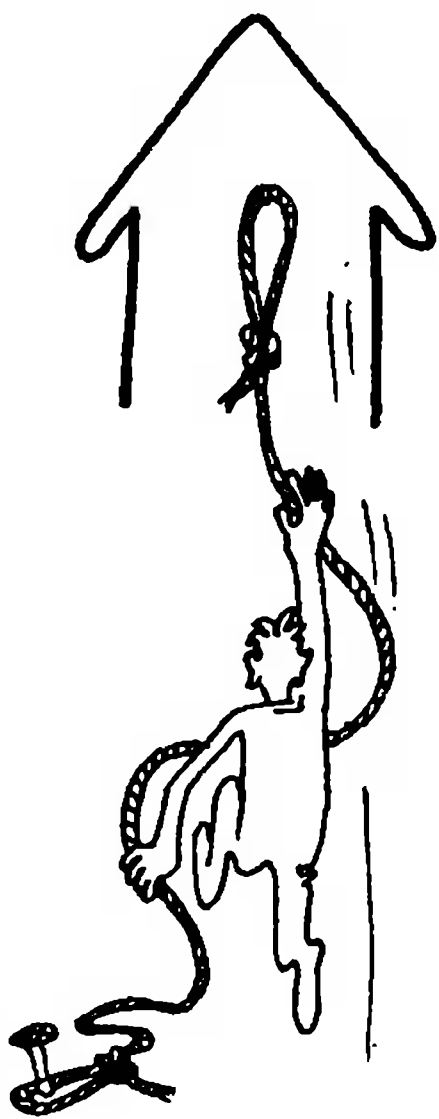
— Это тоже спорное место. Раз я ухожу на показ длин во время ввода отрезов, то возвращаться надо было бы обратно тоже на второй режим. Но тогда пользователь не сможет закончить работу. Сейчас он может из основного режима ввода длин отрезков перейти на режим показа, потом на выбор режимов и выбрать четвертый режим — окончание работы.

О ДАННЫХ И ИХ ОБРАБОТКЕ

Не хочу хвастаться, но эта программа уже намного ближе к настоящим нужным программам, чем предыдущие. Она может сойти за программу обработки данных. Здесь данными являются и длины штук, и длины отрезков.

— А еще что-нибудь похожее на твои штуки можно придумать?

— Думаю, что можно. Труднее всего сообразить, где могло бы быть что-то похожее на продажу ткани в штуках





без остатков. Может быть, на товарной станции, куда приходят вагоны с мешками, ящиками или контейнерами и где их перегружают в грузовики разной грузоподъемности, при этом толком не зная, какие грузовики подадут. Тогда было бы важно, чтобы в конце разгрузки вагона в нем не остались остатки, намного меньшие грузоподъемности машины.

— А что в этом ужасного? В машину загрузят остаток, подъедут к следующему вагону и там догрузят, сколько надо.

— А ты подумала о потерях рабочего времени? Машина должна переезжать, ждать, пока подъедет погрузчик, пока откроют вагон, и, кто знает, что еще!

Намного больше примеров можно найти, если остатки нас не очень волнуют. Например, наша программа могла бы описывать распределение каких-нибудь запасов на складе. Там имеются запасы различных товаров; мы эти товары перенумеруем, введем первоначальные количества в машину и, чуть только из склада что-то выдадим, так сразу на соответствующее количество уменьшим. Или другой случай. Представь, что ты получила зарплату и сразу распределила, сколько тратить на еду, парикмахерскую, мороженое...

— Ну нет! Так я делать не буду. Я все истрачу, а потом видно будет.

— Верю. Но если бы ты все-таки все распределила, то, купив что-то, сразу могла бы побежать к машине и ввести сведения, сколько истрачено.

— А знаешь, я слыхала, что в бухгалтериях некоторых предприятий так и делают. Там расписано, сколько и на что можно тратить деньги. У них это называется — по какой статье платить.

— Вот видишь! И таких примеров можно было бы придумать очень много. Существенно то, что мы имеем массив с числами, которые что-то для нас означают, неважно что, и мы с этими числами что-то делаем...

— ...Неважно что. Здорово у тебя выходит. Что-то делаем, неважно что. Чем-то, неважно чем. Как-то, неважно как.

— Я, наверно, говорю непонятно, но при изучении программирования важно понять его сходство с математикой. Иначе получится, как с Ванечкой, который пришел из школы домой и рассказывает папе, что сегодня проходили сложение. Папа хочет узнать, какие у Ванечки успехи: «Смотри, Ванечка, здесь у меня два апельсина, и я добавляю еще два. Сколько получается вместе?» Ванечка думает, думает и говорит: «Не знаю. В школе мы складывали яблоки».

— Это старый анекдот.

— Старый-то старый, но от этого не менее поучительный. Посмотри сейчас на наш массив в программе! Важно ли, что в нем хранятся именно длины штук? Ясно, что неважно. Или то, что программа от элементов массива что-то именно вычитает? А если было бы сложение, существенно ли это все поменяло? Никак нет! Как говорят, логическая структура программы осталась бы неизменной. Или так: схема алгоритма программы не поменялась. В программировании не только неважно, апельсины или яблоки, но и какие с ними проводят действия: складывают, вычитают или что-либо другое.

— Не понимаю, зачем говорить о сложении, когда у нас вычитание.

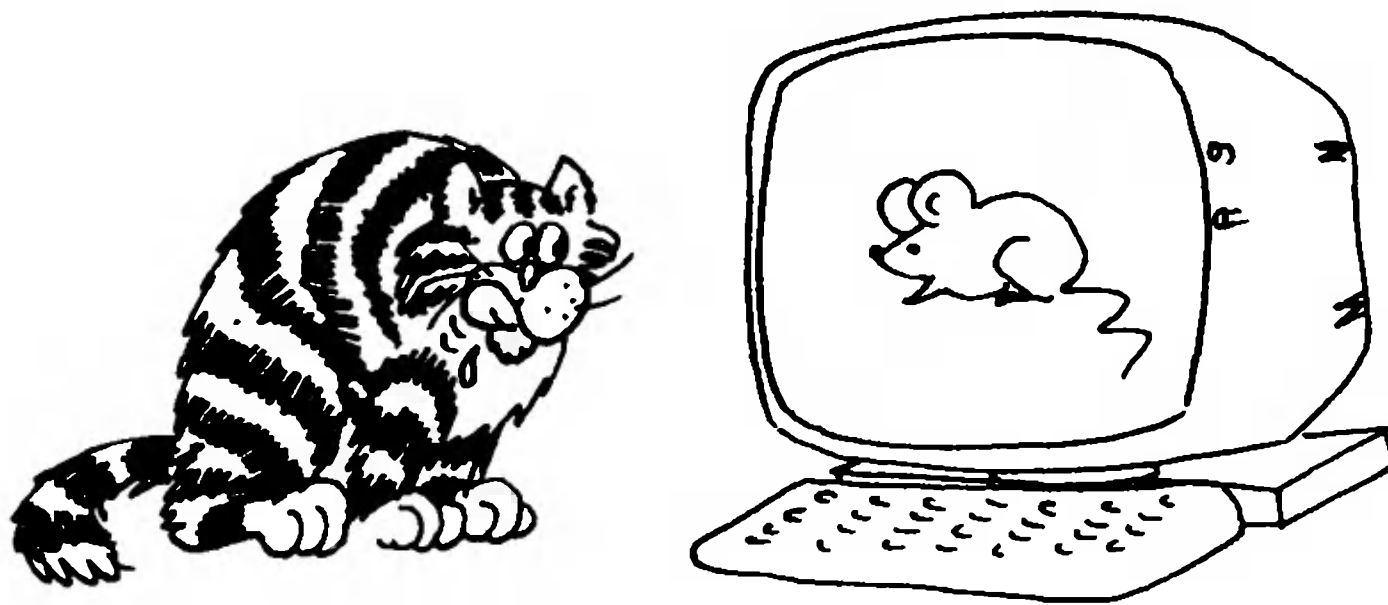
— Если ты собираешься всю свою сознательную жизнь решать только эту задачу, то, конечно, не нужно. А если хочешь что-то усвоить, то придумай пример, где к элементам массива нужно было бы добавлять.

— Да, хватку опытного педагога ты имеешь! Как только ученик возникает, так сразу, шлеп, придумывай пример. Сиди и думай. И не чирикай.

— Думай, думай. И не чирикай.

ЕЩЕ ОБ ОБРАБОТКЕ ДАННЫХ

— Запросто придумаю... Готово! В зоопарке живут слоны, львы, зайцы и другие звери?! Живут. Их кормить надо? Надо. Это денег стоит? Да. И немалых. Я предлагаю составить такую программу. Присваиваем каждому зверю номер и соотносим с ним соответствующий элемент массива. Два слона — два элемента. Четыре тигра — четыре элемента. И т. д. Купили зверю что-то пожевать, сразу в этом массиве отмечаем, сколько потратили, т. е. добавляем сумму денег к элементу массива. Так что наша программа позволит узнать, сколько с начала года каждый зверь съел, т. е. сколько на него потрачено. И если он съел слишком много, то больше есть ему не даем.



— С точки зрения программирования твой пример безупречен. А с точки зрения зверей могли бы быть и возражения.

Придумывать примеры со сложением намного проще. Пригоден учет любых дел. Например, учет работы отдельных бригад с целью начисления заработной платы. Или отдельных тружеников, скажем комбайнеров республики, участников конкурса «Лучший комбайнер». Или учет данных соревнований районов по уборке урожая или учет количества очков, полученных командами, участницами футбольного чемпионата.

Еще лучше, если в программе предусмотрено и сложение, и вычитание. Тогда, например, учет товаров на складе был бы полнее. Товары на складах ведь не только выдают, но и получают. Так что в программе понадобились бы два режима — режим выдачи и режим получения. Подобно можно было бы и описать работу сберкасс, куда люди деньги вкладывают и откуда получают. Ведь использование отрицательных чисел выглядит довольно неестественно.

Все эти разговоры должны раскрыть некоторые проблемы. Первая — это борьба с ошибками. Представь себе: ты приходишь в сберкассу, чтобы снять 100 р., а тебе говорят, что на твоём счету — 200 р. и ты ещё должна сберкассе доплатить, чтобы не быть в долгу. И это все из-за того, что в прошлый раз кассир деньги с твоего счета вычел, а не прибавил к нему, как надо было сделать.

— Нет, такую трагедию я даже представить не могу.

— Следующая проблема. Подобные данные должны храниться на диске, иначе все это без толку. Не забудь, что все содержание памяти пропадает, как только машину выключают. Вот, например, с твоими зверями. Машина должна работать днем и ночью, и в памяти должна быть твоя программа учета, и все это только для того, чтобы ты раз в сутки пришла и ввела, на сколько рублей каждый зверь слопал. Это же безобразие. Должно быть так: ты приходишь, запускаешь программу, она с диска считывает состояние дел на

вчерашний день, ты вводишь сегодняшние данные, и программа записывает новое состояние на диск. Получается, что тебе придется изучать использование дисков. И ещё одна проблема. Нужно, чтобы можно было напечатать информацию на бумаге. Например, у тебя имеется программа, подводящая итоги соревнования комбайнеров республики. Ты же не станешь приводить всех их к дисплею, чтобы они могли ознакомиться с результатами. Значит, надо учиться выводить данные на печать. И все это мы будем делать, но, увы, не в следующий раз.

15. ВЫЧИСЛИТЕЛЬНАЯ МАШИНА НЕ ТОЛЬКО ВЫЧИСЛЯЕТ

В следующий раз Петя позаботился, чтобы машинного времени было достаточно для основательного рассмотрения новых понятий. Как всегда, он начал с общих рассуждений.

— Многие все еще считают, что на электронной вычислительной машине можно только проводить вычисления — выполнить какую-то последовательность арифметических действий: сложения, вычитания, умножения или деления. Польза в том, что ЭВМ делает это чрезвычайно быстро и ошибается редко. Правда, иногда можно прочесть о компьютерах, играющих в шахматы или диагностирующих болезни, и это вызывает удивление. Программировать игру в шахматы мы сможем не очень скоро, но понять, что компьютер отличается от микрокалькулятора не только скоростью счета, можно и сегодня.

— Позвольте, сударь, заметить, что мое знакомство с этим достижением человеческого гения не дает возможности обнаружить те существенные отличия, о которых вы намерены изволили выразиться.

— Надеюсь, это станет яснее сегодня, когда мы рассмотрим символьные переменные.

СИМВОЛЬНЫЕ ПЕРЕМЕННЫЕ

— Когда же кончатся эти новые понятия?!

— Никогда, сударыня. Программист должен учиться всю жизнь. А символьные переменные ничего нового собой не представляют — переменные уже были, символы тоже, сей-

Символьные пере-
менные

час будут символьные переменные, а старые переменные впредь будут называться числовыми переменными. Если честно, то определенное отличие все-таки имеется; большое или маленькое — увидишь сама.

— Ну что ж, начинай.

— Начну с того, что символьные переменные обозначают буквой и «солнышком», а посередине может быть цифра, но не буква! Например, $A \odot$, $B \odot$, $C 1 \odot$ — это правильные обозначения символьных переменных, а неправильные...

— Неправильных не надо. Продолжай.

— Символьная переменная имеет еще и значение — один или несколько символов. Так сказать, строку символов. Кстати, символьные переменные называют еще и строковыми. Имеет ли какой-нибудь смысл эта строка, машине все равно, об этом знает только программист. Немножко потерпи, станет яснее, когда покажу, как выглядит оператор присваивания в случае символьных переменных. Хотя бы так: $A \odot = "A"$. Такая строка означает: символьной переменной с обозначением $A \odot$ присваивается значение — строка символов, состоящая из одной только буквы А. Итак, значения символьных переменных в операторе присваивания ставят в кавычках.

— Что-то похожее делали в операторе PRINT.

ВЫВОД СИМВОЛОВ

— Совершенно верно! Об этом операторе ты вспомнила в нужный момент. Значения символьных операторов можно и выводить. Если бы сейчас ввести операторы... А чего же нет? Так и сделаем.

```
A $\odot$ ="A"  
PRINT A $\odot$   
A
```

Вот видишь, с символьными переменными можно работать в непосредственном режиме. Это во-первых. А во-вторых, все получилось как надо: на экране появилось значение переменной $A \odot$, т. е. буква А, но, ясное дело, без кавычек.

Попробуем еще.

```
A $\odot$ ="ТОНЯ"  
PRINT A $\odot$   
ТОНЯ
```

Следовательно, и вправду значение символьной переменной может быть строкой, а не только одним каким-то

Символьная строка





Строка символов
может содержать
только 255 симво-
лов

символом. В операторе вывода можно писать одновременно и числовые, и символьные переменные, и элементы любых массивов.

```

PRINT A, A
      5

```

Теперь ты должна понять: если буква А используется для имени переменной А X , это не означает, что ее нельзя использовать в той же самой программе для имени переменной А. Это разные переменные. Совсем разные.

А скажи, почему между словом ТОНЯ и цифрой 5 такое большое расстояние?

— Ты что-то говорил о зонах.

— Именно так. Значения переменных А и А X выведены в разных зонах, так как в операторе PRINT между именами стояла запятая.

ЕДИНСТВЕННОЕ ДЕЙСТВИЕ С СИМВОЛАМИ

В отличие от числовых переменных, для которых имеются целых четыре действия, с символьными переменными можно совершать только одно действие. Это единственное действие называется конкатенацией. Если это слово тебе трудно произнести, называй это действие сцеплением. А само действие очень простое: значения символьных переменных записываются друг за другом и результат становится значением другой символьной переменной. Обозначают это действие «+». Вот пример.

```

Вх="СООБРАЖАЛКИНА"
Тх=Ах+Вх
PRINT Тх
ТОНЯСООБРАЖАЛКИНА

```

— Ха-ха! Трюк не получился.
— Сейчас получится.

```

Сх=" "
PRINT Вх+Сх+Ах
СООБРАЖАЛКИНА ТОНЯ

```

Значением переменной C_x является особый символ — пробел. Правду сказать, в этом примере конкатенация особенно нужна не была: то же самое можно было сделать и без нее.

```

PRINT Ах;Сх;Вх
ТОНЯ СООБРАЖАЛКИНА

```

Но в других случаях это действие бывает очень полезным.

КАК СРАВНИВАЮТ СТРОКИ СИМВОЛОВ

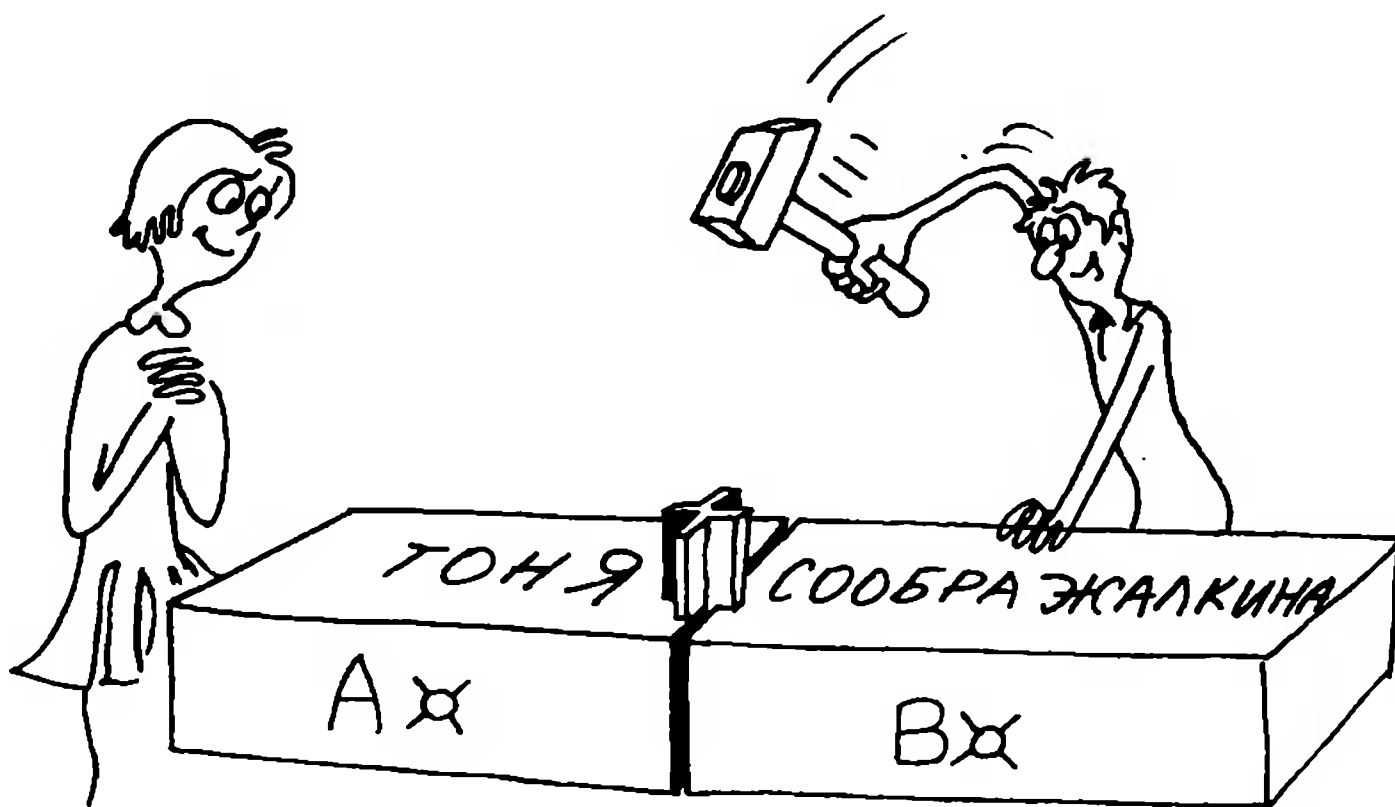
Остается рассказать, как используют символьные переменные в операторе условного перехода. Условный переход может быть, например, таким:

```

40 IF Ах="БОМ" THEN 500

```

В этом примере логическое условие считается выполненным и управление передается на строку 500, если значение



Потихонечку-полегонечку и сцепим целую книгу

переменной $A \boxtimes$ состоит ровно из трех букв, и притом именно БОМ.

— Для числовых переменных можно было определить, которое из них больше или меньше, а для символьных это бессмысленно. Не так ли?

— Да. Для значений символьных переменных, действительно, не существует таких понятий, как «больше» или «меньше», можно только определить, которое из них будет первым при расположении в алфавитном порядке. Для такого дела приспособлены те же самые знаки: $<$, $>$, $=$, $<=$, $=>$.

— Ничего не понять! Что там приспособлено?

— Допустим, мы выполнили операторы $A \boxtimes = \text{«АНЯ»}$ и $B \boxtimes = \text{«ЯНА»}$ и переменные $A \boxtimes$ и $B \boxtimes$ теперь имеют положенные значения. Тогда условие $A \boxtimes < B \boxtimes$ будет правильным, или, выражаясь, точнее, истинным. А истинным это условие будет потому, что при расположении в алфавитном порядке слово АНЯ будет первым, а слово ЯНА — вторым. Условие $A \boxtimes > B \boxtimes$ в этом случае не будет, конечно, истинным, а $B \boxtimes > A \boxtimes$ опять будет.

И для символьных переменных можно написать подобный Бейсико-русский словарь, как для числовых переменных:

$\text{«A»} = \text{«A»}$
 $\text{«A»} < > \text{«B»}$
 $\text{«A»} > \text{«B»}$
 $\text{«A»} < = \text{«A»}$
 $\text{«B»} < \text{«A»}$
 $\text{«A»} > = \text{«B»}$

На Бейсике	По-русски
$=$	совпадает
$< >$	не совпадает
$<$	по алфавиту первый
$= <$	по алфавиту первый или совпадает
$>$	по алфавиту второй
$= >$	по алфавиту второй или совпадает

— Эти «первый» и «второй» пригодятся, если понадобится какие-то слова расставить в алфавитном порядке. И еще?

— Думаю, что это все. Кроме того, следует сказать, что упорядочение слов по алфавиту сравнения «больше — меньше» всегда проходит только для слов английского языка, а для русских слов на других машинах может не получиться. Секрет здесь такой, что интерпретатор сравнивает не буквы, а их коды. Помнишь, а я когда-то тебе говорил, что каждая буква в памяти машины записана числом, называемым кодом?

— Помню. Правда, смутно.

— Так вот, для того чтобы сравнением кодов как чисел получилось сравнение соответствующих им букв, нужно, чтобы коды букв возрастали, когда перечисляем буквы по алфавиту. К сожалению, бывают машины, в которых это не так.

ПРОГРАММА УПОРЯДОЧЕНИЯ ДВУХ СЛОВ

— Думаю, что это все. Попробуй составить программу, которая два введенных в произвольном порядке слова вывела бы на экран в алфавитном порядке.

— Попробую... А как вводить символьные переменные?

— Тем же оператором INPUT. Запиши в нем, значения каких переменных ты хочешь ввести, и вперед. Мы пока использовали оператор INPUT для ввода только одного значения переменной, но с таким же успехом можно вводить и несколько. Надо только написать имена всех вводимых переменных по порядку, отделяя друг от друга запятыми. Так что те два слова можешь ввести одним оператором INPUT S 1 X, S 2 X, где S 1 X — имя переменной для первого слова, а S 2 X — для второго слова.

— Так сразу я этого не сделаю. Надо присесть и подумать.

Тоня села за стол и начала размышлять. Но размышляла она недолго.

— Знаешь, я поняла, как тут надо делать! Вот мое предложение.

```
10 PRINT "ПРОГРАММА РАССТАВИТ 2 СЛОВА ПО АЛФА-  
ВИТУ"
```

```
20 PRINT "ВВЕДИТЕ ДВА СЛОВА"
```

```
30 INPUT S 1 X, S 2 X
```

```
40 IF S 1 X > S 2 X THEN 60
```

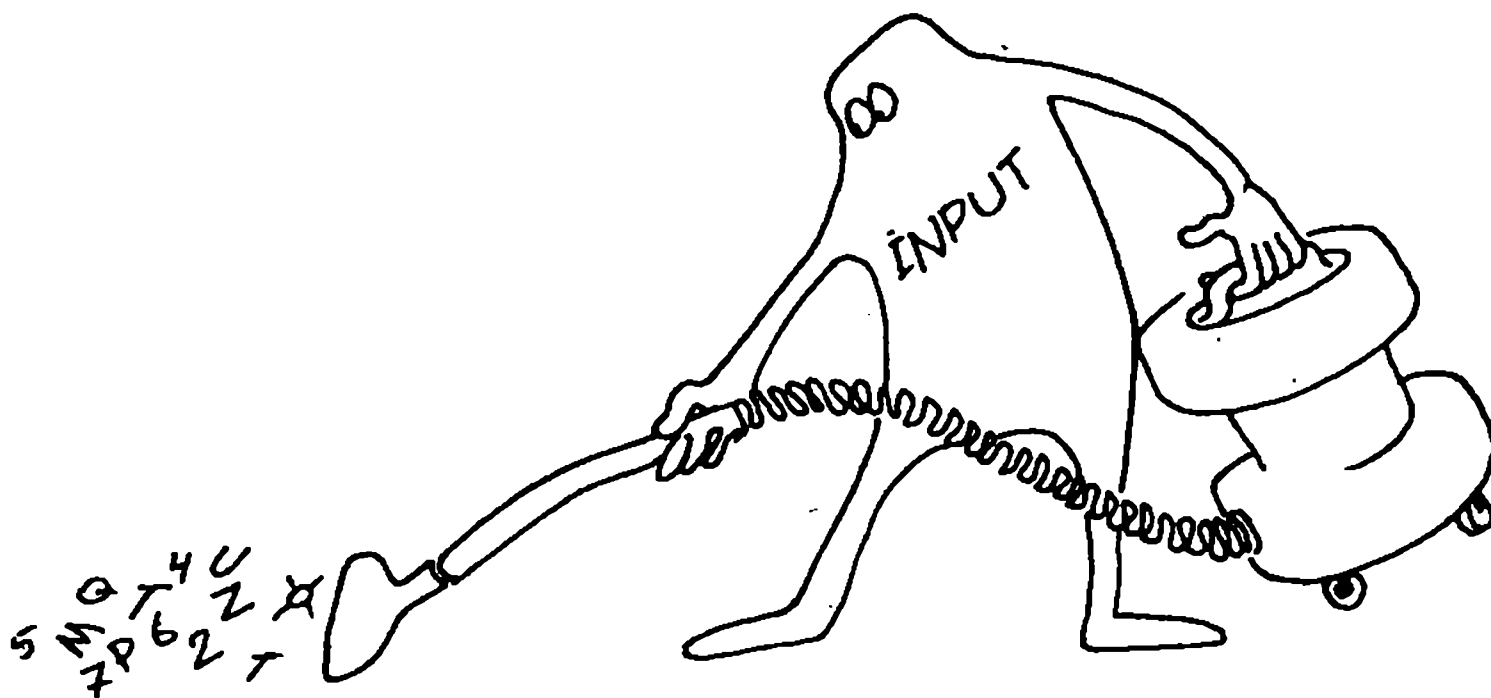
```
50 PRINT S 1 X, S 2 X
```

```
60 PRINT S 2 X, S 1 X
```

```
70 STOP
```

```
80 END
```

— Расскажи, что ты тут делаешь?



Я только собираю — обрабатывают другие

— Сначала сообщая, что программа делает, потом прошу ввести два слова. Потом эти слова ввожу. В строке 40 их сравниваю, и если первое будет по алфавиту вторым, то ухожу на строку 60, где оба слова вывожу — второе перед первым. Но если при сравнении, оказалось, что первое и по алфавиту первое, то в строке 50 вывожу их, так сказать, в порядке поступления...

— А потом? Что будет потом?

— Да, что будет потом?.. Потом еще какой-то вывод.

— Точно, какой-то вывод.

— Ха! Я втисну еще одну строчку:

55 STOP

— Теперь все в порядке.

— Послушай, а я ведь не знаю, как вводить слова во время работы программы, и главное, не получится ли так, что интерпретатор оба слова воспримет как одно большое слово.

— Можно делать двояко. Первый способ: как только показался вопросительный знак, введи первое слово; сразу покажется еще один вопросительный знак, и тогда, конечно, введи второе слово. Больше вопросительных знаков не будет, разумеется, если все запрограммировано правильно. Второй способ: оба слова введи сразу, друг за другом, отделяя запятой.

— Мне все ясно. Я хочу вводить свою программу.

— Хорошо. Только не забудь дать директиву NEW, чтобы очистить.

На сей раз Тоня вводила программу очень тщательно и даже просмотрела текст при помощи директивы LIST. Все выглядело правильно.

— Ну что, я пускаю!

RUNNN

ПРОГРАММА УПОРЯДОЧИВАЕТ 2 СЛОВА ПО АЛФАВИТУ
ВВЕДИТЕ ДВА СЛОВА

? ЯНА

? АНЯ

АНЯ

ЯНА

STOP AT LINE 75

Ур-р-р-ра! Программа работает!

— Почему ты так считаешь?

— Какие могут быть сомнения? Посмотри же, она все вывела правильно!

— Но ты же не знаешь, что будет, если слова ввести в правильном порядке.

— Зачем?! Если слова ввести в правильном порядке, все будет хорошо.

— Все равно нужно проверить. Каждый оператор нужно проверить, и не только просматривая программу, но и в работе. Пускай, пускай еще раз.

— Раз уж ты хочешь...

RUNNN

ПРОГРАММА УПОРЯДОЧИВАЕТ 2 СЛОВА ПО АЛФАВИТУ
ВВЕДИТЕ ДВА СЛОВА

? АНЯ

? ЯНА

АНЯ

ЯНА

STOP AT LINE 55

Ну, что я говорила! Все правильно.

16. РАБОТА СО СЛОВАМИ

Чем ты порадуешь меня сегодня?

— Сегодня хорошо бы поработать с символьными переменными побольше.

— Фу! Они же очень неинтересны. И что там может быть интересного, если их можно только кон... конкан... Мне жутко понравилось это слово, и если бы еще его запомнить!

— Со значениями символьных переменных можно проделывать конкатенацию.

— Вот я и говорю: их можно конкатенировать, или, может быть, правильнее конкатить.

— Так никто не говорит.

— Как это никто, если я только что сказала?

— Ну ладно, пусть. Думаю, что конкатенацию приходится делать даже реже других действий.

— Ты же говорил, что других действий с символьными переменными вообще нет.

— Нет похожих на арифметические действия. Но с символьными переменными можно делать другие — своеобразные действия.

— И какие?



КАК СТАТЬ ХОЗЯИНОМ СВОИХ СЛОВ

— Знаешь что! Придумай хоть одно действие со значениями символьных переменных. Каким действием тебе хотелось бы осчастливить человечество?

— Ты меня застал врасплох. Как в школе говорят, нам этого не задавали.

— Тогда, может быть, так. Что ты делаешь со словами, когда говоришь?

— Я их... произношу...

— Я надеялся, ты ответишь: я их склоняю — «слово», «слова», «слову» и т. д. Вот и получились действия со словом «слово».

— Непривычно... Действия со словами...

— Каким будет алгоритм склонения слова «слово»?

— Надо начинать с именительного падежа — «слово». Дальше родительный. Заменяем окончание: вместо «-о» ставим «-а» — «слова». Так ведь?

— Не сомневался, что ты додумаешься. А теперь скажи: что это такое — заменить окончание? Какие действия требуются?

— Сперва отбрасываем старое окончание и затем добавляем новое.

— Согласен. И вот в Бейсике имеются действия со значениями символьных переменных, позволяющие запрограммировать замену окончаний и подобные мероприятия. Если ты поняла дух программирования, то догадаешься, что таких действий не будет много, но их хватит, чтобы сделать все нужное. Кстати, замена окончания совсем не простое дело. Надо знать, что такое окончание, надо уметь его найти и отделить. Главное, надо иметь алгоритм.

— А что-нибудь поконкретнее об этих действиях не можешь сказать?

— В Бейсике действия с символьными переменными проводят функции. Как они на самом деле работают, посмотрим на машине. Пока скажу только самое важное. Часто приходится из символьных строк выделять подстроку. Конечно, ты спросишь, что такое подстрока.

— Может быть, часть строки?

— Да, но, так сказать, непрерывная часть строки. Например, строка СТРОКА имеет подстроки СТР, СТРО, РОК, ОК, А и еще кучу других. Но строка СОК не является подстрокой, хотя и состоит из символов первоначальной строки.

— Я бы сказала, что СОК — это кон-ка-те-на-ция подстрок С и ОК.

— Действительно! Я об этом не думал.

— Вот так! Знай наших!

— Имеется функция, которая из строки символов вы-

Подстрока

СТРОКА
СТРО
СТР
ТР
Р
РО
КА

таскивает подстроку, и другая функция, которая сообщает, начиная с которого символа в данной строке можно найти другую строку. Например, если даны две строки СТРОКА и РОК, эта функция определит, начиная с какого символа вторая строка входит в первую. Какое число должно в этом случае получиться?

— СТРОКА и РОК? Думаю, что три.

— Так оно и будет.

— А если вторая строка вообще не входит в первую? Скажем, она не РОК, а СРОК?

— Тогда значение той функции будет ноль. На машине увидишь.

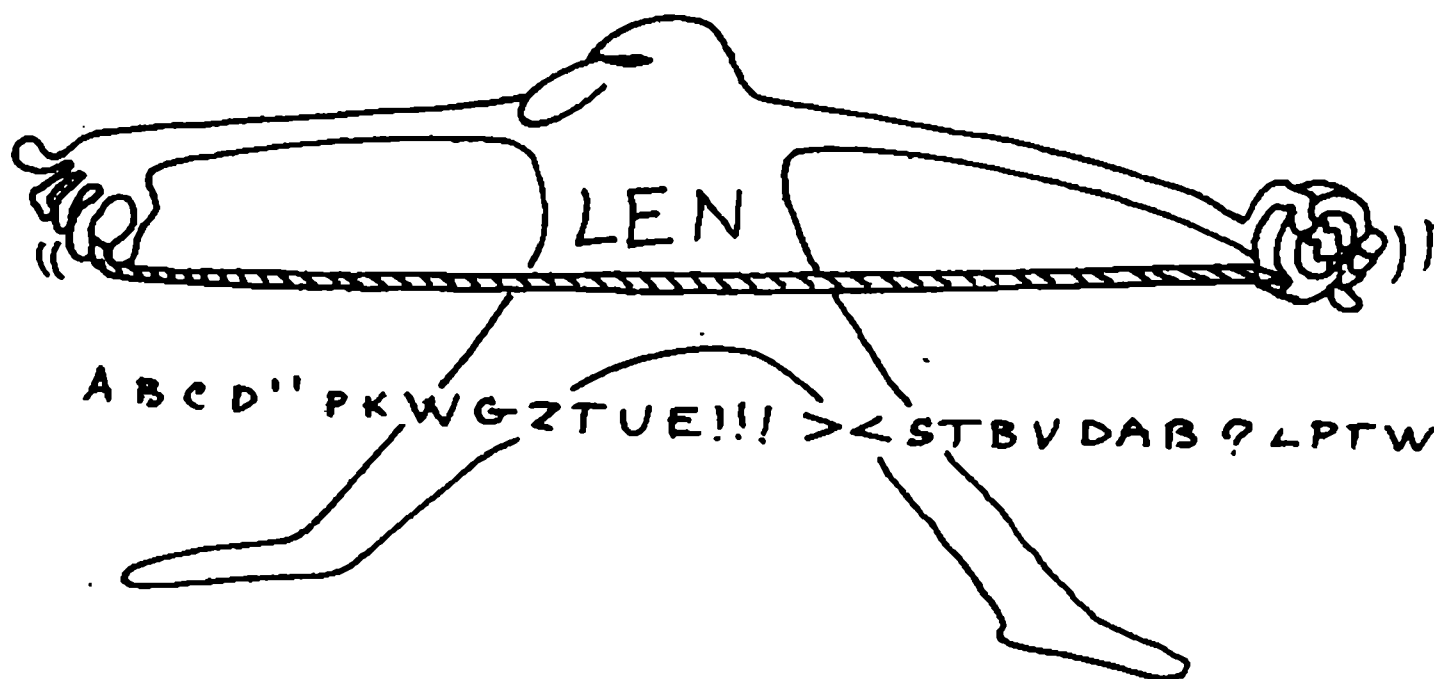
На машине первым делом Тоня вызвала интерпретатор Бейсика.

— Все в порядке. С какой функции начнем?

СКОЛЬКО СИМВОЛОВ В СТРОКЕ!

— Начнем с простейшей. О ней я пока ничего не говорил. Имя этой функции LEN, и она находит длину значения символьной переменной. LEN, кстати, сокращение английского слова length — «ленгс» — «длина». Так как эта функция находит длину строки, т. е. количество символов в строке, то эта строка должна быть ее аргументом. И вызывают эту функцию таким образом:

$A = \text{LEN} (A \text{ } \text{X})$



Переменной A будет присвоено значение, равное длине значения переменной A X — количеству символов в нем.

— Прекратить разговоры. Машина ждет примера.

— Сперва посмотрим в непосредственном режиме.

```

A$="СОДЕРЖАЛКИНА"
A=LEN(A$)
PRINT A
13

```

— Теперь я.

```

A$="БЕИДИКОВ"
PRINT A
13

```

О-хо-хо! Эта функция же не умеет считать!

— Минуточку! Три, пять, восемь. Что-то действительно не так. А-а! Переменной A\$ ты присвоила новое значение, а переменной A нет. Поэтому было выведено старое значение. В непосредственном режиме надо работать осторожно. Здесь надо было делать, например, так:

```

A=LEN(A$)
PRINT A
8

```

или так:

```

PRINT LEN(A$)
8

```

Сейчас все выводится, как надо, а ты, не знаю, что подумала.

— Радуюсь, что не придется ничего менять в программировании.

ЧТО НАЙДЕШЬ В СИМВОЛЬНОЙ СТРОКЕ

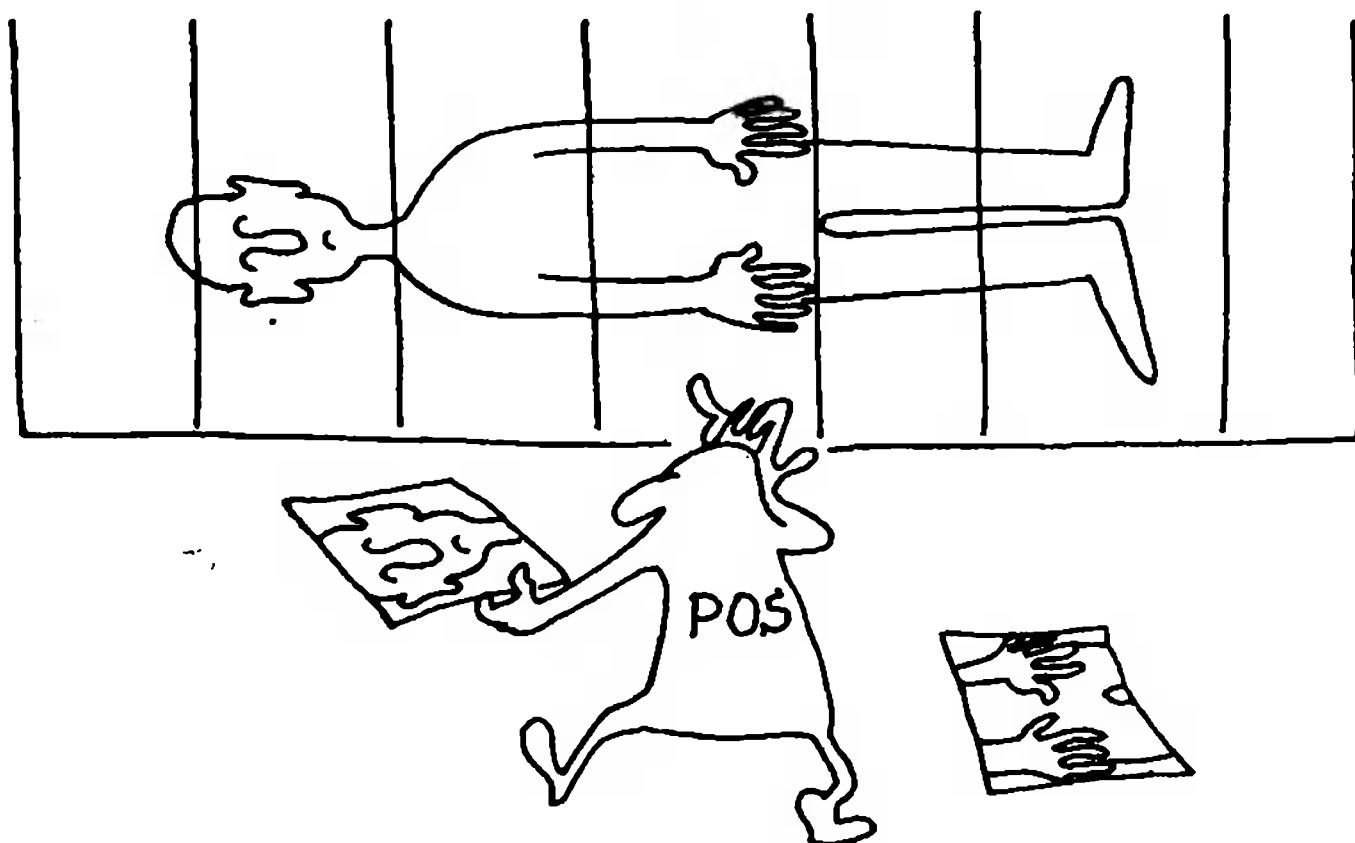
— Теперь можем перейти к другой функции, имя которой POS, сокращение от position — «позишн», что означает «позиция». Эта функция находит местоположение подстроки. Таким вот образом:

```

A=POS(A$, B$, N)

```

Как видишь, она имеет целых три аргумента: первые два — символьные переменные, а третий — числовая. Работает эта функция так: берет значение первого аргумента, ищет в нем значение второго, но начинает поиск с символа, номер которого равен значению третьего аргумента. Значение функции будет равно номеру символа, с которого начинается значение второго аргумента в значении первого, либо нулю, если



значение второго аргумента в значении первого не входит вообще. Посмотри работу этой функции на примере!

```
PRINT POS(A, "ЕЯ", 2)
```

2

Видишь, все получилось. Учти, что третий аргумент этой функции может быть не просто переменной, но и арифметическим выражением. Первые два аргумента могут быть конкатенацией переменных или другой функцией, значение которой — символьная переменная.

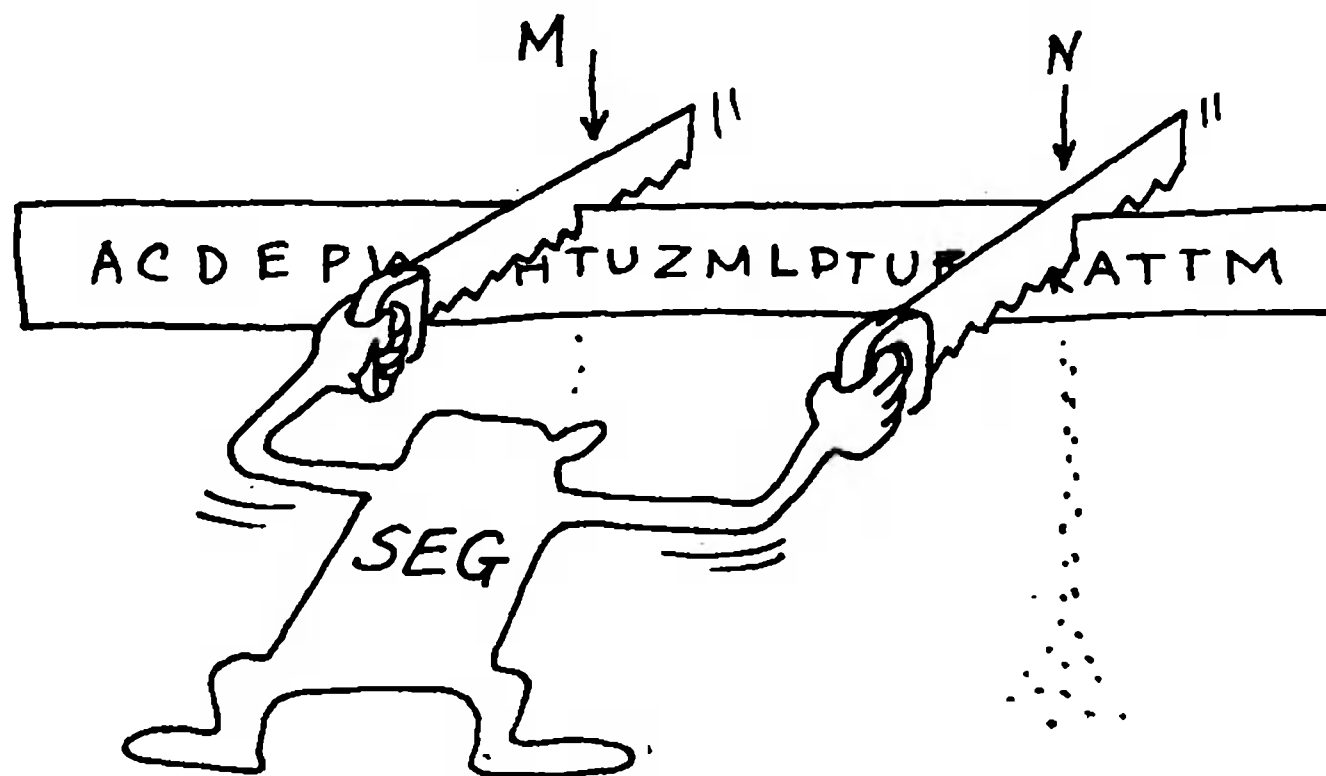
— Таких функций у нас еще не было.

— Скоро будут. Для полноты картины добавлю: аргументами могут быть и прямо задаваемые в кавычках символьные строки, как мы только что записали во втором аргументе, а также цифра в третьем аргументе, тоже как в нашем примере. Другими словами, аргументами могут быть символьные и числовые постоянные.

РАЗБИЕНИЕ НА ЧАСТИ СИМВОЛЬНОЙ СТРОКИ

Теперь о функции `SEG ☼`. Как ты видишь, последний знак в ее имени — «солнышко». Значит, результатом работы функции будет символьная строка. Вот тебе и пример долгожданной функции, дающей в результате символьную строку. Возможно, что `SEG` — это сокращение английского слова `segment`, но точно этого никто не знает. Эта функция выделяет подстроку из значения первого аргумента, второй аргумент указывает, с какого символа это выделение начинать, а третий — на каком кончать. Вот пример для изучения:

```
X ☼ = SEG ☼ (A ☼, M, N)
```



Сейчас попробуем что-нибудь на машине.

```

***= СООБРАЖАЛКИНА"
PRINT "ВВЕДИТЕ 40, 2, 13"
***= СООБРАЖАЛКИНА

```

Ну, поняла, как из СООБРАЖАЛКИНА вышло ВООБРАЖАЛКИНА? Посмотрим быстро одну из моих старых программ SIMBO. Она разделяет на отдельные буквы введенное слово.

```

100 PRINT "SIMBO"

```

Напечатай эту программу!

```

100

```

```

10 REM ПРОГРАММА РАЗДЕЛЯЕТ ВВЕДЕННУЮ СТРОКУ
20 REM НА ОТДЕЛЬНЫЕ СИМВОЛЫ
30 PRINT \ PRINT "ВВЕДИТЕ СТРОКУ СИМВОЛОВ"
40 INPUT A$
50 N=LEN(A$) \ REM НАХОДИТ ДЛИНУ СТРОКИ
60 FOR I=1 TO N
70 B$=SEG$(A$,I,I) \ REM ВЫДЕЛЯЕТ СИМВОЛ
80 PRINT I;" СИМВОЛ - ";B$
90 NEXT I
100 STOP \ END

```

— Первые четыре строки я полностью понимаю, ведь не начинающая же! Труднее со строкой 50, но ты по доброте своей поясняешь, что здесь происходит: значение переменной станет равным количеству букв в слове. Потом начинается

цикл. От единицы до N. Только что нашли N и уже суем границей цикла! Э-э-э! Цикл будет по всем буквам введенного слова, так ведь?

— А как же! Я же хочу все буквы выделить и вывести на экран!

— И выделяешь в строке 70. Это надо посмотреть подробно. Функция SEG $\boxtimes$ выделяет подстроку введенного слова, начиная с I-го символа и кончая тем же самым I-м символом, т. е. одну букву. Прекрасно! Далее выводишь... Сперва выводишь значение I, или, что то же самое, номер буквы, и хитрым образом добавляешь остальной текст и саму букву. Все это делается в цикле. Ничего сложного в этой программе нет! Запустим ее ради интереса.

RUNNN.

ВВЕДИТЕ СТРОКУ СИМВОЛОВ

? СЛОВО

1 СИМВОЛ — С

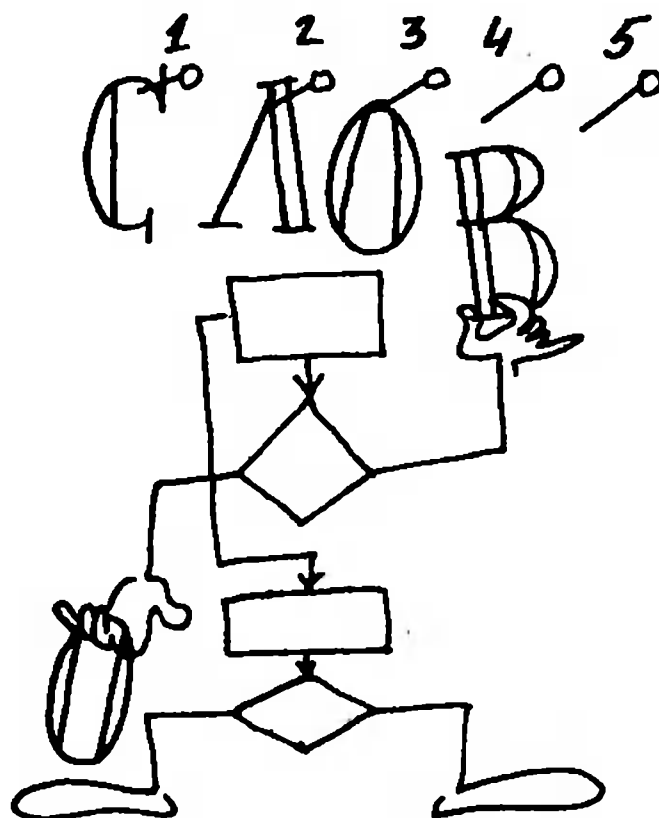
2 СИМВОЛ — Л

3 СИМВОЛ — О

4 СИМВОЛ — В

5 СИМВОЛ — О

STOP AT LINE 100



Жаловаться не могу. Работает правильно.

Слушай, а введенное число ты на цифры разделить можешь?

— Программа работает с символами. С сим-во-ла-ми! И ей совершенно все равно, какую строку символов ты введешь. Введешь строку «123» — программа разделит ее на символы без задержки. Не сомневайся. Ты же видишь, что А $\boxtimes$ — символьная переменная.

СВЯЗЬ МЕЖДУ ЧИСЛАМИ И СИМВОЛАМИ

Задаю вопрос на засыпку. Выполняется такой оператор:

$T \boxtimes = "1" + "2"$

Каково будет значение переменной $T \boxtimes$?

— Ты, может быть, думал, что я скажу три? Ничего подобного. Значением переменной $T \boxtimes$ будет двенадцать.

— Неточно! Значение переменной будет символьная строка «12», что нужно читать скорее «один, два». Символьные строки «12» или там «1234» — это никакие не числа, с ними арифметические действия проводить нельзя. Но если надо, такие строки можно в числа превратить. Для этого нужна функция VAL, имя которой получено сокращением английского слова value — «взлью», по-русски «значение». Ее можно назвать функцией расчета числового значения символьной строки. Вот пример ее использования:

$$K = VAL (B \text{ } \text{X})$$

Значение символьной переменной $B \text{ } \text{X}$ будет превращено в число, и это число станет значением переменной K . Понятно, что значением $B \text{ } \text{X}$ должна быть такая строка, которую в число превратить можно. Например, строки «123», «6789», «3.1414», «0.68E — 06» можно превратить в числа, а строку «ТОНЯ» нет, это такая неполноценная строка.

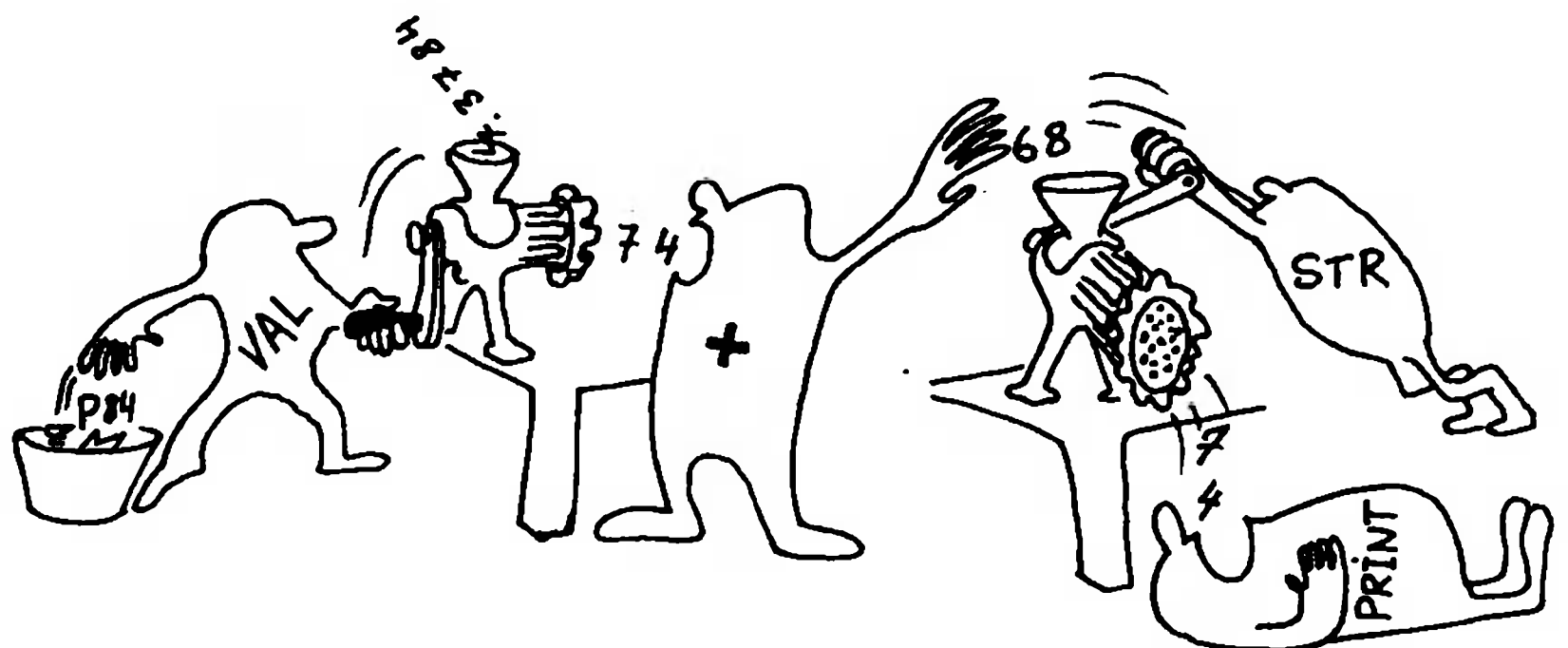
— Ты особенно не выступай! Лучше поясни вот что. Функция полезная, слов нет, а мы говорили о том, как найти цифры заданного числа.

— Раз уж ты жить не можешь без этих цифр, то запомни функцию $STR \text{ } \text{X}$, которая работает совершенно противоположно функции VAL. Она превращает число в строку символов. Полностью STR пишется string, произносится, как пишется, и означает «символьная строка», но только в программировании, в поэзии это означает «струна». Эта функция получается как бы обратной к VAL.

$$A \text{ } \text{X} = STR \text{ } \text{X} (X)$$

— Ясно. Значение переменной X она превращает в символьную строку и объявляет значением $A \text{ } \text{X}$.

— Да, и превращает число совершенно так же, как при выводе, т. е. может превратить в экспоненциальную форму, если надо. Можно сказать, что она печатает значение X в



Вводим символы,
работаем с числами,
выводим символы

значении A ⌘. Может быть, нелишне добавить, что вместо X может быть арифметическое выражение.

— Не надо добавлять. Будет лишне.

КАЛЕНДАРЬ В КОМПЬЮТЕРЕ

— А вот еще две функции, значения которых — символьные переменные, а аргументов вообще нет. Первая функция CLK ⌘ выдает текущее время по часам, а вторая DAT ⌘ — сегодняшнее число. Сначала о числе. Проще всего его узнать так:

```
PRINT DAT⌘  
12-FEB-91
```

Как видишь, это сегодняшнее число, но в строго определенной форме. Сперва день, потом тире, или минус, потом первые три буквы английского названия месяца, потом опять тире и, наконец, последние две цифры года.

— А мне лучше нравится, когда вместо названия месяца стоит его номер: 12—02—90.

— Госстандарт предусматривает вот такую форму записи числа: 90 02 12. А хочешь другую, составь программу, превращающую число из формы Бейсика в твою форму.

— А как это сделать?

— Во-первых, надо из DAT ⌘ выделить название месяца.

— Ага! Функцией... забыла название.

— Функцией SEG ⌘.

— Хорошо. Посмотри, что у меня получится.

— Чуть потерпи. Введем директиву NEW, чтобы очистилась память. Ведь сейчас в памяти компьютера наша программа SIMBO, и если не очистить память, то строки нашей новой программы наложатся на эту программу.

```
NEW CHISLO
```

```
READY
```

Теперь можешь вводить свою программу.

— Программы пока нет, есть только одна строчка.

```
10 A⌘=SEG⌘(DAT⌘,4,6)
```

Будет правильно?

— Кажется, да. Сейчас надо по названию найти номер месяца в году. Для этого я предлагаю ввести такую переменную:

— Тогда давай запиши обращение к этой функции.

$$N = 3(1 - 1) + 1 = 31 - 2$$
$$1 = (N + 2)/3$$

— Так, конечно, было бы красивее. А как туда этот нуль записать?

— Подобное мы только что делали. Введем еще одну переменную.

```
60 C$="010203040506070809101112"
```

И теперь в этой символьной строке по числовому значению номера месяца найди его символьное обозначение.

— Выходит задача, подобная задаче поиска номера месяца. Как-нибудь уж справлюсь.

```
60 J=2*I-1  
70 C1$=SEG$(C$,J,J+1)
```

А сейчас бы сделать новую форму сегодняшнего числа.

```
80 D$=SEG$(DATA,1,3)+C1$+SEG$(DATA,7,9)  
90 PRINT D$  
100 END
```

— Посмотрим, что получилось.

```
RUN
```

```
12-02-91
```

— Ур-р-р-ра! Работает как часы!

ЧАСЫ В КОМПЬЮТЕРЕ

— Теперь функция CLK $\otimes$. Она выдает время по часам в форме... Нет, лучше посмотрим!

```
PRINT CLK$  
17:12:25
```

Сначала часы, двоеточие, минуты, опять двоеточие и, наконец, секунды. Иногда программисту надо знать, сколько секунд прошло с начала суток.

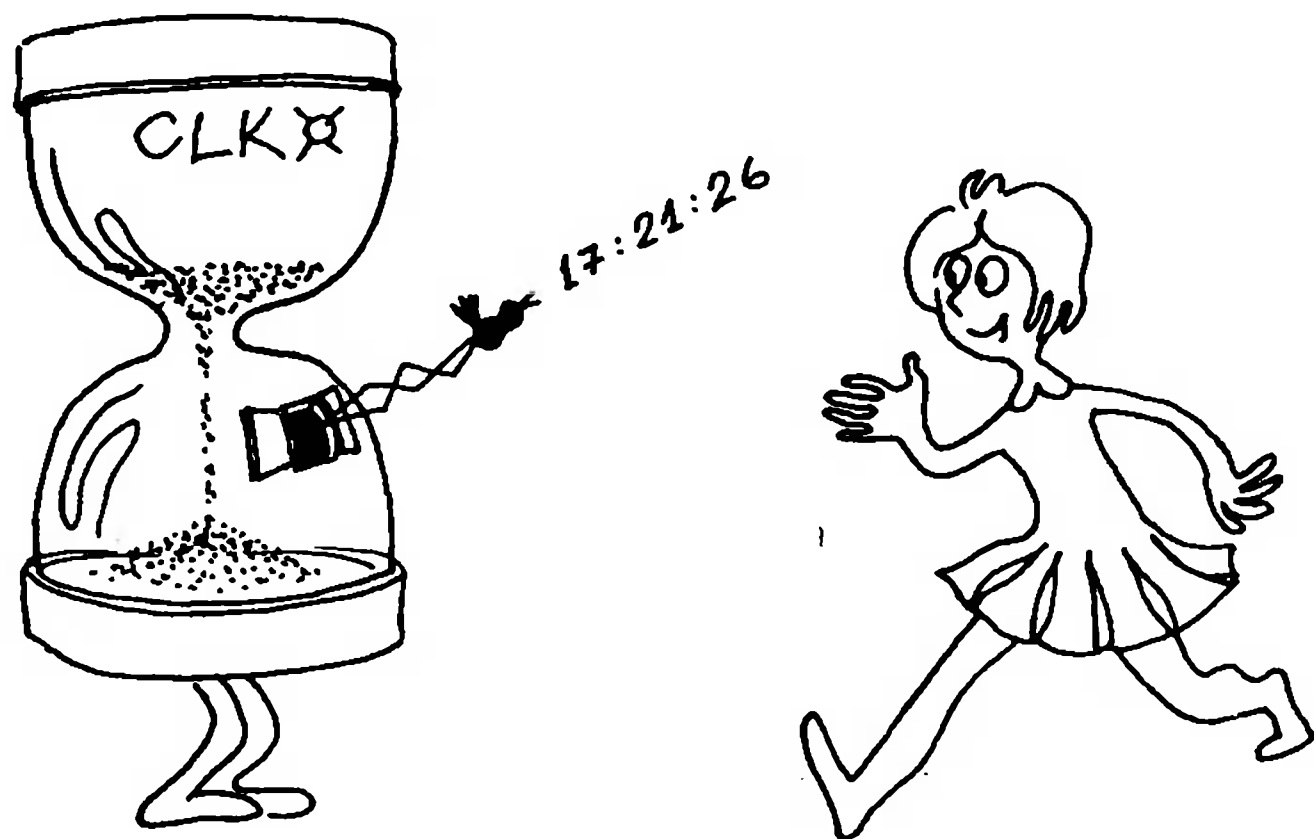
— А зачем?

— Для расчета отрезков времени: например, как долго ученик думал над заданием программы. В начале и в конце обдумывания узнают, сколько секунд прошло с начала суток. Разность этих чисел даст длительность обдумывания.

— Рассчитать, сколько секунд прошло с начала суток,— такая мелочь. Я это берусь сделать.

— Раз берешься, сделай.

— Из символьной строки, которую выдает функция



CLKX, нужно выделить количество часов и умножить его на 3600, потом так же выделить минуты и умножить на 60. И наконец, выделить секунды и сложить все три числа.

— А как это будет выглядеть в программе? Продолжай эту же самую программу для переделки формы даты.

— Тогда смотри.

```
100 A1=VAL(SEG$(CLKX,1,2))
```

Подсчитываем количество часов. Видел, как ловко я воспользовалась функцией VAL для превращения символьной строки в число?!

— Видел. Это и в самом деле было ловко. Твердо различать символьные и числовые переменные не так уж и легко.

— Дальше.

```
110 A2=VAL(SEG$(CLKX,4,5))
120 A3=VAL(SEG$(CLKX,7,8))
130 A=A1*3600+A2*60+A3
140 PRINT CLKX,A
```

Так будет хорошо?

— Меня не спрашивай. Спрашивай машину.

```
RUNNN
```

```
12-02-91
17:30:40      63040
```

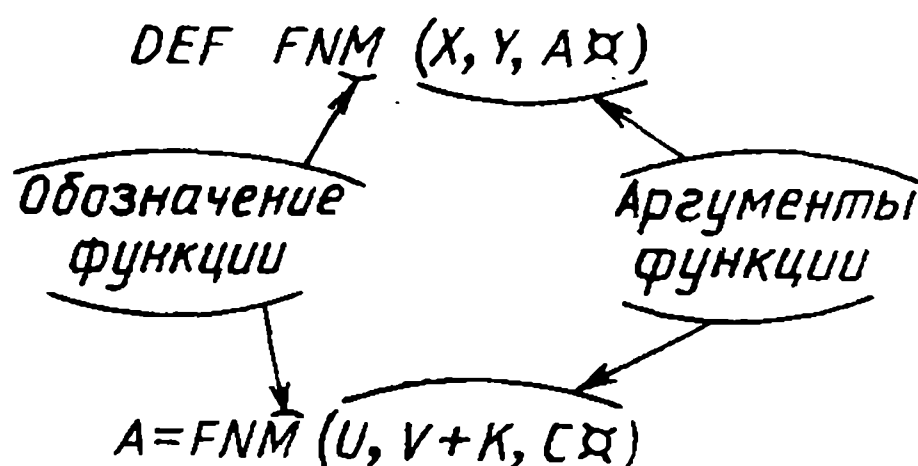
— Машина говорит, что такую замечательную программу она давно не видела.

ФУНКЦИИ ПРОГРАММИСТА — САМОДЕЛЬНЫЙ ИНСТРУМЕНТ

— Это хорошо. Я сейчас подумал, что проще было бы с использованием функций программиста. Для своих нужд программист может определить функции таким образом. Сначала пишет DEF, что является сокращением английского слова *define*, которое произносится «дифайн»...

— ...И совершенно точно означает «определять».

— После DEF пишут FN и какую-нибудь букву, например M. Цифру нельзя. Получается обозначение, или имя, функции. После него в скобках список аргументов, причем хотя бы один аргумент должен обязательно быть. А за знаком равенства и само выражение функции. Это об определении функций. А при ее использовании мы пишем только имя функции и потом в скобках обозначения переменных, для которых функция должна провести свои действия. В схеме это выглядит так:



Как видишь, при использовании функции можем писать в качестве аргумента и целое арифметическое выражение.

— Это как всегда. Но я еще не поняла определения. Напиши-ка какой-нибудь пример!

```
5 DEF FNA(X,Y)=VAL(SEG$(CLK$,X,Y))
```



Присваиваю тебе
звание функции
программиста

— Пожалуйста. Пишу уже для нужд нашей программы. Видишь, я в начале программы определяю функцию, которая сможет заменить три выражения в строках 100, 110 и 120. Меняя значения аргументов, можно будет получить любое из них. Как это делается, увидишь, если самым тщательным образом изучишь такое определение еще одной весьма полезной функции.

```
6 DEF FNP(I)=FNA(1,2)*3+5*FNA(4,5)+6*FNA(7,8)
```

Она рассчитает количество секунд с начала суток. В этом определении я использовал только что введенную функцию FNA. А аргумент I мне нужен только для порядка: если его не будет, интерпретатор выдаст сообщение об ошибке, что нет аргумента.

И еще одна строка в конце программы.

```
150 PRINT FNP(I)
```

Выпечтается значение FNP, конечно, совершенно независимо от значения лишнего аргумента I, т. е. количество секунд с начала суток.

— Ладно. Проверим сейчас твои премудрости.

```
RUNNN
```

```
12-02-91
```

```
17:46:12
```

```
63792
```

```
63793
```

Интересно, почему твое количество секунд отличается от моего?

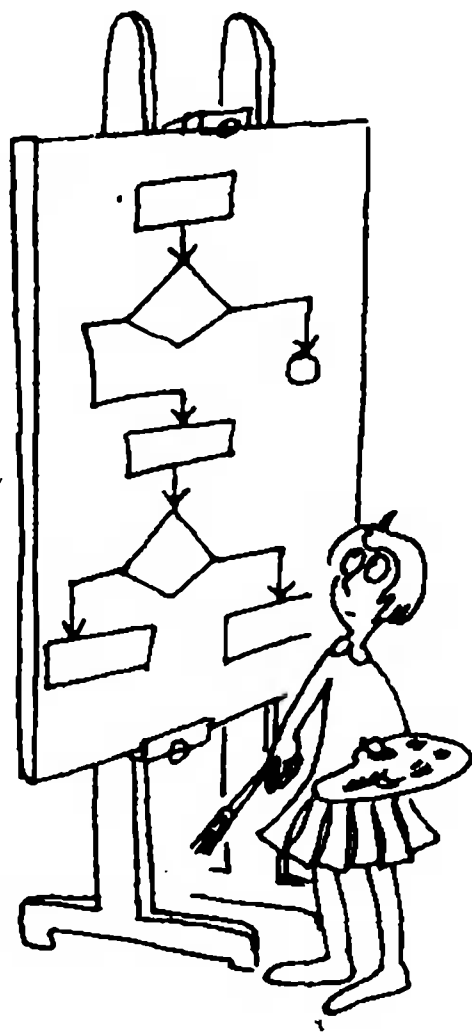
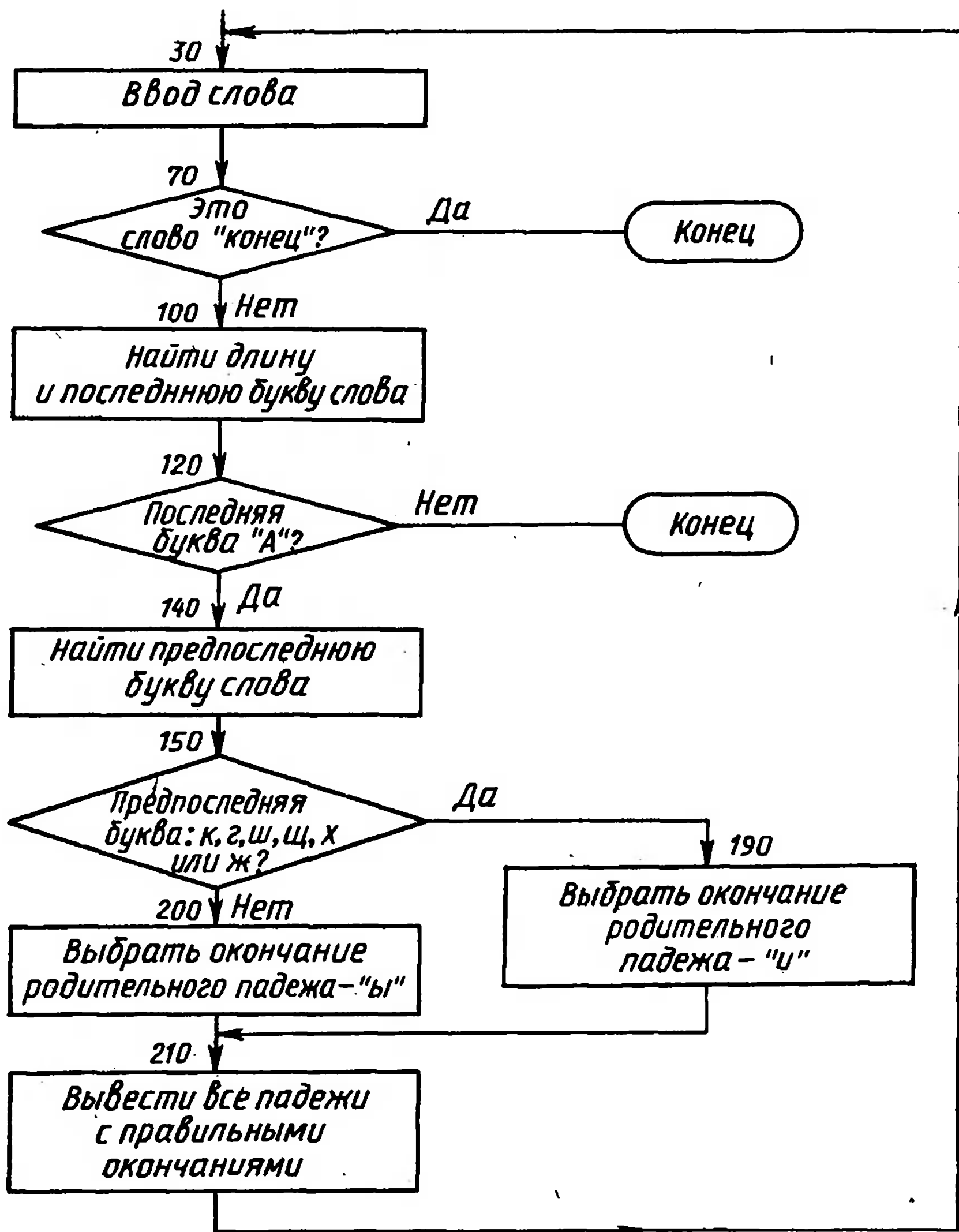
— Пока работали мои функции, счетчик секунд успел перескочить на единицу вперед. Не хочу утверждать, что мой вариант намного лучше твоего, но в случае, если в программе нужно подсчитывать секунды несколько раз, такая функция окажется полезной.

— Вообще-то и действительно нужно. При нахождении отрезка времени по крайней мере два раза, а это в программе может быть нужно будет делать в нескольких, а может быть даже во многих местах.

— Что верно, то верно. А сейчас сделаем так. Ты составляешь программу, которая склоняет русские имена существительные первого склонения, оканчивающиеся на «-а». А я в это время позанимаюсь своими программами.

17. МАШИНА ОСВАИВАЕТ РУССКИЙ ЯЗЫК

Полученной задачей Тоня занималась довольно долго. Даже рисование схемы алгоритма заняло много времени, потому что у Тони появились кое-какие соображения относительно родительного падежа.



ПРОГРАММИРОВАТЬ НАДО УМЕЮЧИ

Программа должна была выводить родительный падеж слова с окончанием «-ы», за исключением слов с предпоследней буквой к, г, ш, щ, х или ж, когда надо «-и». Но вот запрограммировать это...

— Такую программу составить... Это же настоящая мыслящая машина получится.

Петя ничего не ответил. Отдав машину в распоряжение Тони, он умчался консультироваться по своим делам. Тоня начала вводить составленную программу, но, увы, допустила ошибки. Программа со всеми ошибками выглядела так:

```
10 REM ПРОГРАММА СКЛОНЯЕТ ИМЕНА СУЩЕСТВИТЕЛЬНЫЕ
15 REM ПЕРВОГО СКЛОНЕНИЯ С ОКОНЧАНИЕМ -А
20 PRINT \ PRINT \ PRINT \ PRINT
30 PRINT "ВВЕДИТЕ ИМЯ СУЩЕСТВИТЕЛЬНОЕ ПЕРВОГО"
40 PRINT "СКЛОНЕНИЯ НА -А ИЛИ СЛОВО КОНЕЦ,"
50 PRINT "ЕСЛИ КОНЧАЕТЕ"
60 INPUT S$
70 IF S$<>"КОНЕЦ" THEN 90
80 PRINT "КОНЕЦ" \ STOP
90 REM ПРОВЕРКА ПРАВИЛЬНОСТИ ВВОДА
100 D=LEN(S$)
110 C$=SEG$(S$,D,D) \ \ REM ПОСЛЕДНЯЯ БУКВА
120 IF C$="А" THEN 140
130 PRINT "ЭТО СЛОВО НЕ КОНЧАЕТСЯ НА -А" \ STOP
140 C1$=SEG$(S$,D-1,D-1) \ \ REM ПРЕДПОСЛЕДНЯЯ БУКВА
150 IF C1$="К" THEN 200
160 IF C1$="Г" THEN 200
165 IF C1$="Ш" THEN 200
170 IF C1$="Щ" THEN 200
175 IF C1$="Х" THEN 200
180 IF C1$="Ж" THEN 200
190 P$="Ы" \ GO TO 210
200 P$="И"
210 Z$=SEG$(S$,1,D-1) \ \ REM СЛОВО БЕЗ ОКОНЧАНИЯ
230 PRINT
240 PRINT "ИМЕНИТЕЛЬНЫЙ      КТО?      ЧТО?      ";S$
250 PRINT "РОДИТЕЛЬНЫЙ      КОГО?     ЧЕГО?     ";Z$+P$
260 PRINT "ДАТЕЛЬНЫЙ      КОМУ?     ЧЕМУ?     ";Z$+"Е"
270 PRINT "ВИНИТЕЛЬНЫЙ     КОГО?     ЧТО?      ";Z$+"У"
280 PRINT "ТВОРИТЕЛЬНЫЙ     КЕМ?      ЧЕМ?      ";Z$+"ОМ"
```

```

290 PRINT "ПРЕДЛОЖНЫЙ      О КОМ? О ЧЕМ?  ":"0 "+Zx+"E"
300 PRINT  \ PRINT
310 GO TO 30
320 END

```

Тоня решилась и запустила программу самостоятельно без Пети. Было боязно, но очень хотелось узнать, что же получилось.

```

RUNNN

```

```

ВВЕДИТЕ ИМЯ СУЩЕСТВИТЕЛЬНОЕ ПЕРВОГО
СКЛОНЕНИЯ НА -А ИЛИ СЛОВО: КОНЕЦ,
ЕСЛИ КОНЧАЕТЕ
? МАМА

```

```

SYNTAX ERROR AT LINE 70

```

— Что же это такое? Строка 70... Синтаксическая ошибка? Посмотрим.

```

LISTNN 70

```

```

70 IF S<>"КОНЕЦ" THEN 90

```

Так... Все ясно. THEN введен с двумя Н. Это не моя вина, клавиша часто выдает два знака. Сейчас надо исправлять.

```

70 IF S<>"КОНЕЦ" THEN 90

```

Запустим еще раз.

```

RUNNN

```

```

ВВЕДИТЕ ИМЯ СУЩЕСТВИТЕЛЬНОЕ ПЕРВОГО
СКЛОНЕНИЯ НА -А ИЛИ СЛОВО: КОНЕЦ,
ЕСЛИ КОНЧАЕТЕ
? МАМА
ЭТО СЛОВО НЕ КОНЧАЕТСЯ НА -А

```

```

STOP AT LINE 130

```

Какая чушь! Как же не кончается, когда кончается! Кончается «мама» на «-а» и бесспорно является самым натуральным именем существительным первого склонения. Наверно, машина испортилась.

ОШИБКА ИЛИ СБОЙ МАШИНЫ?

Тоня решила подождать Петю.

— Куда ты исчез? Болтаешься где-то, а за это время машина испортилась. Совсем не то выдает, что есть на самом деле.

— Почему ты так считаешь?

— Смотри сюда! Вывела, что «мама» не кончается на «-а». Представляешь!

— Запомни на всю жизнь. Машина портится намного реже, чем ошибается программист. Давай искать ошибку. Твою ошибку. Программа ушла по неправильной ветке, значит, первое, что можно заподозрить, — это операторы перехода. Где ты проверяешь, что введенное слово кончается на «-а»?

— Это я проверяю в строке 120.

— Смотрим эту строку.

LISTEN 120

120 IF C\$="A" THEN 140

Ничего плохого тут нет. Раз оператор перехода правильный, то должно быть неправильным значение переменной C\$. Сейчас мы увидим, что оно не «А», а совсем какая-то другая буква.

PRINT C\$

А

— Получил, провидец! Машина отдала концы, чего там рассуждать!

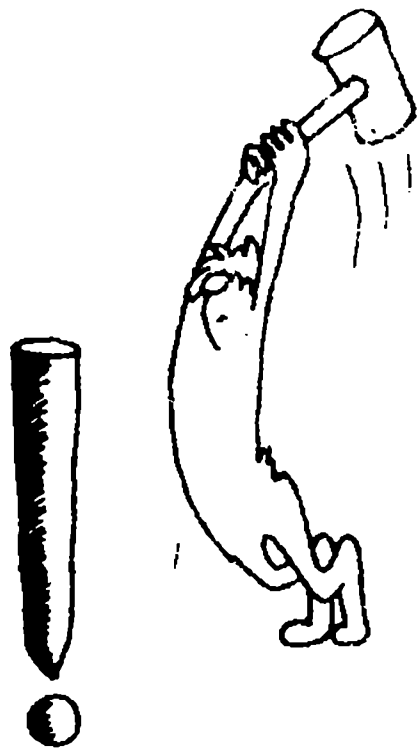
— Ничего не понимаю. . . Больше-то и смотреть нечего. . . А-а! Вот что будет. Дело в том, что интерпретатор отлично различает русские и латинские буквы одинакового начертания — А, М, Т, О или др. И если ты в строке 120 в кавычках вписала не русскую А, а совершенно с нею совпадающую при изображении на экране латинскую А, то все. С русской А она совпасть не сможет.

— Я этого не знала!

— Не знала, так не знала. Впредь будешь знать. Исправляем, посмотрим, что это даст.

120 IF C\$="A" THEN 140

Сейчас запускай.



ВВЕДИТЕ ИМЯ СУЩЕСТВИТЕЛЬНОЕ ПЕРВОГО
СКЛОНЕНИЯ НА -А ИЛИ СЛОВО: КОНЕЦ,
ЕСЛИ КОНЧАЕТЕ
? МАМА

ИМЕНИТЕЛЬНЫЙ	КТО?	ЧТО?	МАМА
РОДИТЕЛЬНЫЙ	КОГО?	ЧЕГО?	МАМЫ
ДАТЕЛЬНЫЙ	КОМУ?	ЧЕМУ?	МАМЕ
ВИНИТЕЛЬНЫЙ	КОГО?	ЧТО?	МАМУ
ТВОРИТЕЛЬНЫЙ	КЕМ?	ЧЕМ?	МАМОЙ
ПРЕДЛОЖНЫЙ	О КОМ?	О ЧЕМ?	О' МАМЕ

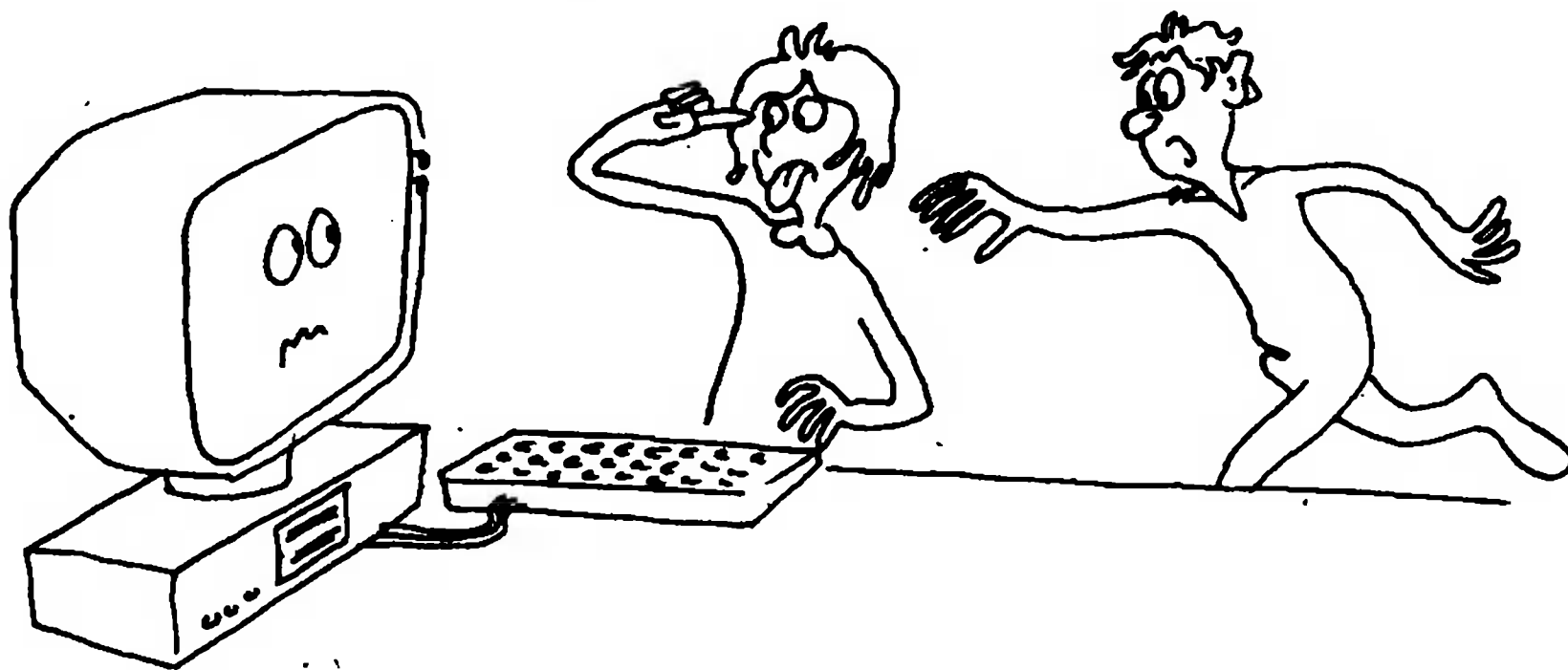
ВВЕДИТЕ ИМЯ СУЩЕСТВИТЕЛЬНОЕ ПЕРВОГО
СКЛОНЕНИЯ НА -А ИЛИ СЛОВО: КОНЕЦ,
ЕСЛИ КОНЧАЕТЕ
? КОНЕЦ
КОНЕЦ

STOP AT LINE 80

— Уф-ф. Теперь-то все правильно. Надо записать программу на диск.

— Что записала программу на диск, хвалю. Но что считаешь все правильным, показывает только твое. . .

— Что, что показывает?!

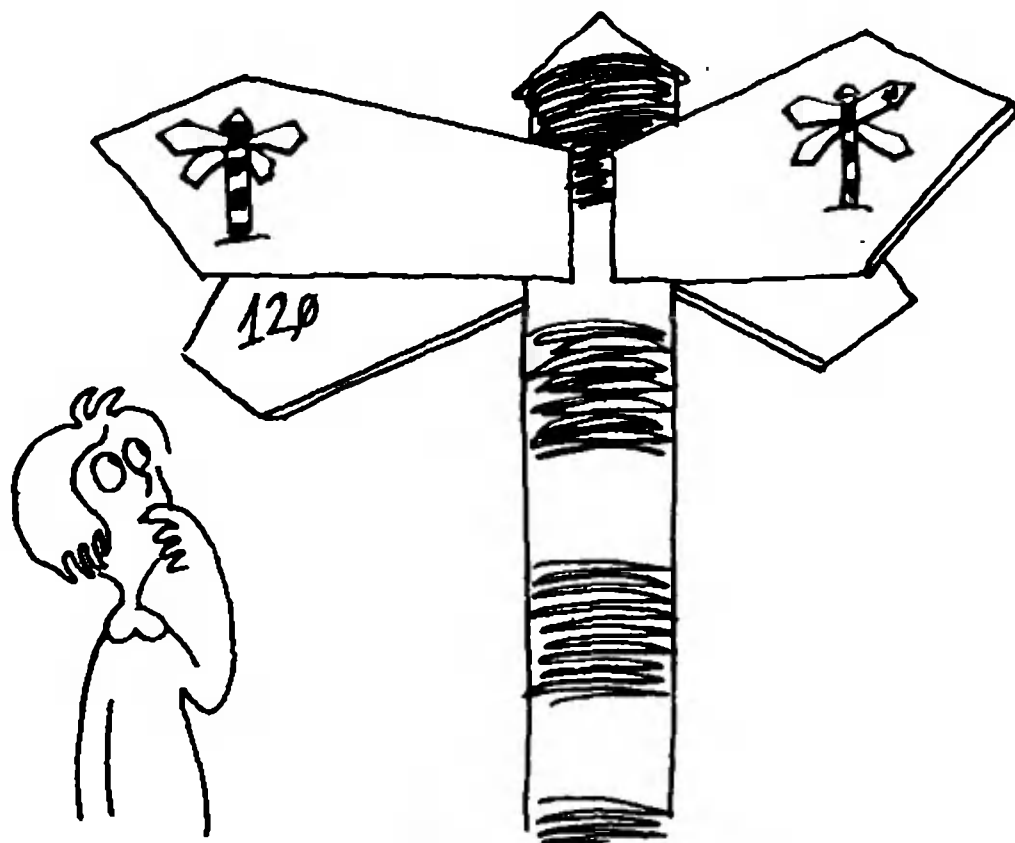


ГОРЬКАЯ ПРАВДА, А НЕ МОРАЛЬ

— Ладно, пусть остается секретом. Я как-то говорил, что программист должен быть уверен, что его программа работает правильно. Но чем опытнее программист, тем меньше

Как пройти все пути в программе?

Отладка



у него такой уверенности. Ты еще не знаешь, что вначале программы долгое время не работают и почти все рабочее время программист тратит на исправление ошибок — отладку программы. Это дело надо полюбить, если желаешь программировать. А поиск ошибок вовсе не такое скучное занятие, как кажется сначала. Здесь нужна железная логика майора Знаменского и, конечно, хорошее знание работы машины, того, как ведет себя интерпретатор в каждой ситуации. К сожалению, поиск ошибки всегда кончается ударом по самолюбию — как ты смог так глупо ошибиться!

— Да, откровенно говоря, такие удары случаются.

— Ну вот! Умнеешь на глазах. Возвращаясь к твоей программе, надо сказать, что ее отладка только началась. Программа правильно сработала со словом «мама».

— Со словом «папа» она тоже правильно сработает!

— Не спорю. Сработает правильно. Но программист должен быть уверен, что его программа сработает правильно с любыми входными данными. Не думай, что я тебе предлагаю проверить программу на всех именах существительных первого склонения, какие только есть в русском языке. Я предлагаю провести проверку программы — тестирование таким образом, чтобы заставить сработать все операторы ветвления как на выполнение логического условия, так и на невыполнение. Другими словами, я хочу пропустить программу с такими входными данными, или тестами, чтобы она прошла по всем возможным путям.

— По всем возможным путям? А говоришь только об операторе ветвления. Разве оператор вычисляемого перехода ON не надо было бы учитывать при рассмотрении этих путей?

— Ах да! Про него я забыл. Он, конечно, тоже создает ветвление в программе. Подай, пожалуйста, программу со схемой алгоритма, и посмотрим теперь, какие слова надо

программе задавать, чтобы все логические операторы срабатывали как на выполнение, так и на невыполнение логического условия.

ПЕРЕД ТЕСТИРОВАНИЕМ ПОЛЕЗНО ПОДУМАТЬ

— Я, пожалуй, здесь не все еще понимаю; но разве пуск программы со словом «мама» не считается?

— Считается, считается. Это пример на стандартное слово с окончанием родительного падежа «-ы». Нам нужно подобрать слова, родительный падеж которых оканчивается на «-и», и притом должны быть учтены все возможности — все возможные здесь предпоследние буквы. Найди слово, оканчивающееся на «-ка».

— Это я могу. «Рука».

— А сейчас слово, оканчивающееся на «-га», чтобы проверить уход при выполнении условия в строке 160. Заодно будет проверено, пропускает ли оператор ветвления в строке 150, если предпоследняя буква не «к».

— «Нога».

— И дальше?

— «Каша», «теща», «муха» и «рожа».

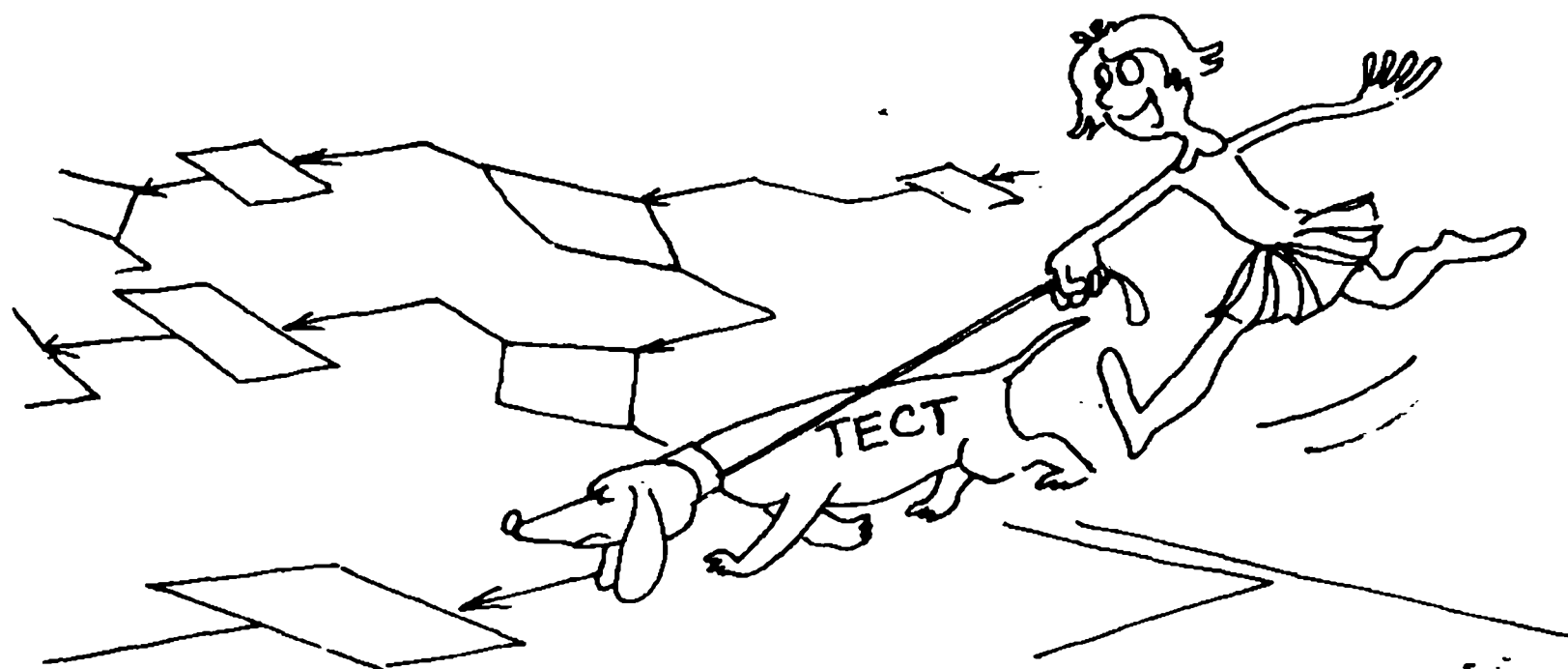
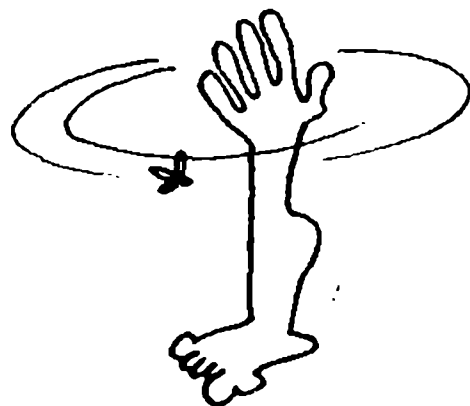
— Итак, для тестирования нам надо пропустить программу со словами «мама», «рука», «нога», «каша», «теща», «муха», «рожа» и с каким-нибудь непригодным словом, например «слово», чтобы проверить, остановится ли программа, если ввести что-то неподходящее. Да, и еще одно слово.

— Какое же? Больше не должно быть.

— А окончание работы программы? Правильно ли она ответит на слово «конец»?

— Нет сомнений. Мы уже ей это слово давали.

— Да, действительно. Но все равно в списке тестов это слово должно быть.



Ошибки отыскиваются тестами

— Должно быть. Должно. Давай введем все эти слова, чтобы ты успокоился и мы могли двинуться дальше.

— Ты особенно не суетись. В программировании надо работать аккуратно, иначе никакого толку не будет. И только для того, чтобы приучить тебя к аккуратности, я настаиваю на этих проверках. Другому они, может быть, и не были бы нужны, потому что операторы перехода очень просты и почти одинаковы.

— Будь аккуратной, приучайся к порядку. Без конца мне это твердят и твердят.

Однако все слова ребята проверили и действительно никаких ошибок не обнаружили.

ЯЗЫКИ И ПРОГРАММИРОВАНИЕ

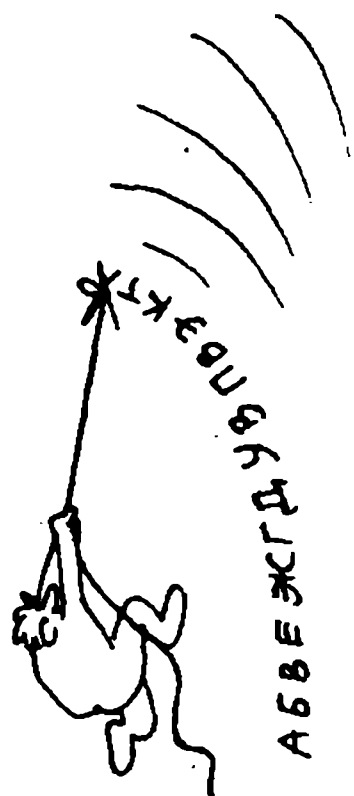
— Ошибок нет, но это не означает, что программу нельзя улучшить. Сразу видно, что программа стала бы короче, если воспользоваться оператором ON. Сначала нужно ввести символьную переменную, содержащую все выделенные согласные: $T \text{ } \text{ } = \text{ "КГШЩЖЧХ"}$. И после того как найдена предпоследняя буква введенного слова, ищем ее в значении переменной $T \text{ } \text{ }$ при помощи функции POS. Равенство значения этой функции нулю означает, что в родительном падеже надо использовать окончание «-ы». Тогда не понадобятся эти сравнения в строках 150—180.

Еще надо сказать, что программа ведь не охватывает все первое склонение. За бортом остались слова, оканчивающиеся на «-я». Главное, программу легко изменить, чтобы учесть и этот случай. После строки сравнения последней буквы слова на совпадение с «-а» ставим строку сравнения с «-я», и если оказалось, что слово действительно кончается на «-я», перейти на строку 200 с присвоением переменной $P \text{ } \text{ }$ значения «И», потому что в этом случае родительный падеж всегда оканчивается на «-и». И все.

— Так уж и все? А винительный падеж от «земля», что вижу? «Землю», а вовсе не «землу».

— Да, верно. Придется иметь еще одну символьную переменную наподобие $P \text{ } \text{ }$, в которую засылать либо «у», либо «ю» в зависимости от того, какой буквой оканчивается слово.

— И это еще не все! В творительном падеже тоже другое окончание — «-ей», а не «-ой», и так просто получается только тогда, когда мы в этом окончании не пишем букву «ё», как надо было бы, скажем, в слове «землём» или «судьёй». А слова на «-ия»? У них ведь в дательном и творительном падеже окончание не «-е», а «-и». Плохо знаешь грамматику родного языка, Бейсиков!



— Этого не скроешь. Но все эти случаи все-таки запрограммировать можно, вводя специальные переменные для окончаний и проверив предпоследнюю букву, которую ты, надеюсь, сумеешь найти.

— Это-то я сумею. А как быть со словами, о которых я только что вспомнила? Существительные среднего рода, оканчивающиеся на «-мя», — «время», «знамя», «имя» и т. д.

— Они относятся, кажется, к третьему склонению?

— Да, к третьему. Но вот, представь себе, приходит какой-нибудь шутник и вводит в программу вместо порядочного слова первого склонения слово «знамя», и мы начинаем его склонять: «знамя», «знами», «знаме», «знаму». . . Представляешь, сколько смеху это вызвало бы? И ничего не делаешь.

— Почему же нет? Можно проверять не одну, а две последние буквы — «мя» они или не «мя». Нужно все эти слова запомнить в особом символьном массиве и введенное слово, если оно кончается на «мя», со всеми ими сравнивать — не совпадает ли.

— Все запомнить! И все сравнить! Это же страшная работа!

— Так уж оно получилось. Когда древние славяне создавали русский язык, то совершенно опрометчиво не учли потребности своей «потомочки» Тони, которая захочет заняться программированием. Слов на «-мя» раз-два и обчелся, а что ты сказала бы, если бы понадобилось различить такие существительные, как «день» и «ночь», которые оканчиваются на «ь» и относятся к разным склонениям и родам? Опять другого выхода нет, и приходится все их запоминать в массивах, которые на сей раз получились бы куда больше.

— Жуть! Что и говорить.

— Ничего не поделаешь. Что-нибудь менять в русском языке уже поздно. Но ты не думай, что эти сравнения машина будет долго делать. Несколько слов она сравнит в одно мгновение.

— Слушай! Что я подумала! А одушевленные и неодушевленные существительные во втором склонении? Как же их различить? Неужели опять все запоминать?

— Можно у пользователя спрашивать, одушевленное это существительное или нет, а если по-честному, то надо запоминать. Многое пришлось бы запоминать, если захотеть сделать программу, склоняющую имена существительные по-настоящему. Ведь мы же ничего не говорили о склонении во множественном числе. Один только именительный падеж множественного числа чего стоит. Например: сук — сучья, котенок — котята. Так что на быстрый успех не рассчитывай. Лучше подумай о том, как твою программу переделать в

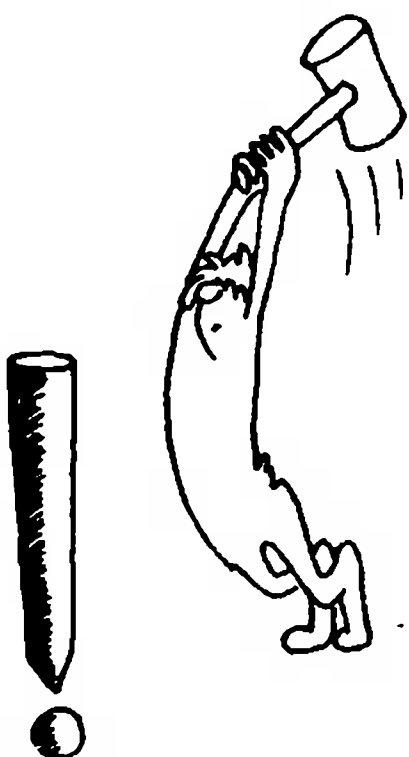


обучающую, чтобы она проверяла знания учеников по склонению существительных первого склонения.

— Да? А как сделать, чтобы. . . ну, чтобы нельзя было заранее знать, какое слово машина будет, например, давать склонять?

— Опять нужно использовать символьные массивы. Ничего не поделаешь, придется в следующий раз о них поговорить подробнее.

18. ЕЩЕ О ТОМ, КАК КОМПЬЮТЕР УЧИТ ДЕТЕЙ



В следующий раз, как и договорились, Тоня и Петя изучали программу, которая использует символьные массивы.

— Эта программа изучает с детьми столицы стран Южной Америки.

— Ой как хорошо! Я никогда не могла разобраться в этих столицах. Точно помню только столицу Бразилии Рио-де-Жанейро, да и то только потому, что туда хотел Остап. . .

— Ха-ха! С тех пор все изменилось, и столица Бразилии сейчас совсем в другой месте! Но ближе к делу. Смотрим программу.

УЧИМ ГЕОГРАФИЮ!

В ней предусмотрены четыре режима работы. В первом режиме на экран выводятся все страны и их столицы. В этом режиме нужно работать в самом начале обучения, когда ребенок еще мало что знает. По-настоящему программа обучает во втором режиме — режиме повторения, когда ученик должен назвать столицу по стране или, наоборот, страну по столице. Третий режим — экзамен.

— Ну как же без экзаменов?

— А как иначе человек сможет узнать, что все освоил? Последний режим — окончание работы. Простой режим. А вот и сама программа (знаки $\times$ и \$ обозначают символьную переменную).

```
10 REM ПРОГРАММА ОБУЧАЕТ СТОЛИЦАМ СТРАН
15 REM ЮЖНОЙ АМЕРИКИ
20 REM R$ - МАССИВ С НАЗВАНИЯМИ СТРАН
```

[illegible]

158

```

850 PRINT "НАЗОВИ СТОЛИЦУ ";R$(N)+I1$(N);
860 INPUT S1$
870 IF S1$<>S$(N) THEN 890
880 PRINT "ОТВЕТ ВЕРНЫЙ" \ GO TO 900
890 PRINT "ОТВЕТ НЕВЕРНЫЙ" \ K=K+1
900 J=J+1 \ E(N)=1 \ REM ОТМЕЧАЕТ СТРАНУ
910 IF J<13 THEN 830 \ PRINT
920 IF K<>0 THEN 940
930 PRINT "ОЦЕНКА - ОТЛИЧНО (5)" \ GO TO 290
940 IF K<>1 THEN 960
950 PRINT "ОЦЕНКА - УДОВЛЕТВОРИТЕЛЬНО (3)." \ GO TO 290
960 PRINT "НУЖНО ПОВТОРИТЬ - ТЫ ДОПУСТИЛ";K;
965 IF K<5 THEN 980
970 PRINT " ОШИБОК" \ GO TO 290
980 PRINT " ОШИБКИ" \ GO TO 290
985 \
990 PRINT "***** КОНЕЦ РАБОТЫ *****" \ STOP
995 \
1000 REM ПОДПРОГРАММА НАХОДИТ СЛУЧАЙНЫЕ НОМЕРА
1010 I=INT(RND*13)+1
1020 RETURN
1900 \
2000 REM ПОДПРОГРАММА ПОВТОРЯЕТ ОТВЕТЫ
2010 IF R1$<>N$ THEN 2030
2020 PRINT "ОТВЕТ ВЕРНЫЙ" \ GO TO 2060
2030 PRINT "НЕПРАВИЛЬНО!!! ПРАВИЛЬНЫЙ ОТВЕТ ";N$
2040 K=K+1
2060 J=J+1 \ RETURN
2070 END

```

— Ужас какая программичка! Такую длинную программу придется неделю разбирать.

— Длина еще ничего не означает. Важно, насколько программа сложна. А эту программу сложной считать нельзя. Сама увидишь, что она в основном состоит из простых наглядных операторов ветвления или вывода. Гарантирую, что все поймешь. Ну, может быть, не с ходу.

— Начало уже поняла. До строки 70. Там ты пишешь о какой-то вероятности.

— Видишь, программа в режиме повторения может задавать вопросы двух типов. Например: Каракас — столица какой страны? Либо: назови столицу Венесуэлы. Ответы ты знаешь. Вопросы этих двух типов чередуются случайным образом. И значение переменной U1 определяет, какого типа вопросы будут чаще. В строке 65 учитель может значение U1 поменять. Если он, например, хочет, чтобы ученик называл только



столицы, то должен присвоить переменной U1 значение 0 или другое какое-то значение.

— С этим ясно. А не ясно с этими массивами окончаний падежей, которые определяются в строке 100. К чему здесь окончания падежей?

— А вот к чему. Мне понадобилось выводить названия стран кроме именительного падежа еще и в родительном, а это оказалось не так просто, потому что почти каждое название склоняется по-своему. Поэтому я пошел по наиболее простому пути — взял и сделал в программе массив для основ названий стран и еще отдельные массивы для окончаний этих названий как в именительном, так и в родительном падежах. А в программе я в нужные места приклеиваю эти окончания к основам, и получаю то, что мне нужно.

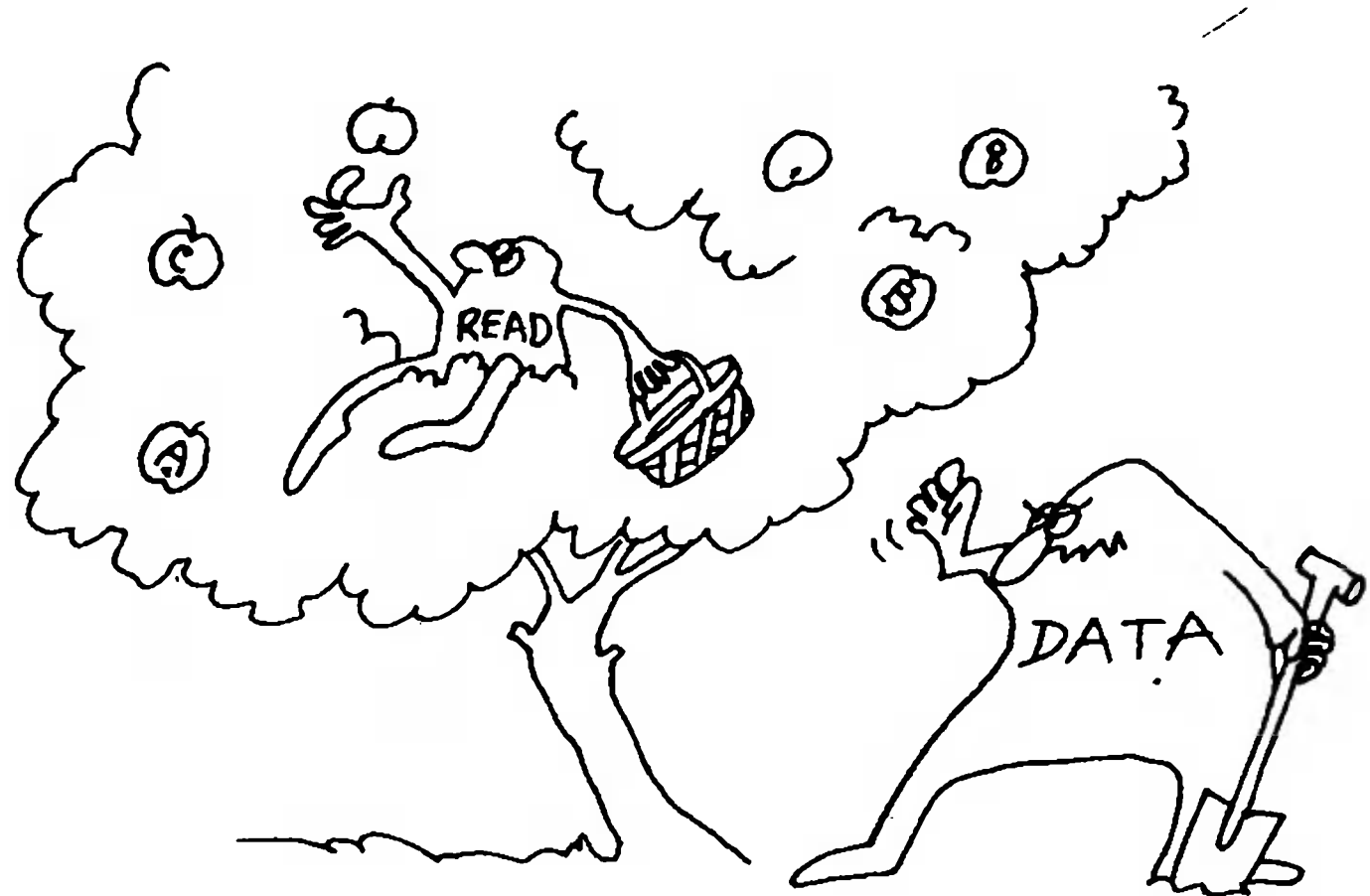
— А здесь в строке 130 и дальше, что это за необыкновенный оператор DATA? Если это оператор.

ХРАНЕНИЕ ДАННЫХ В ПРОГРАММЕ

— Это действительно оператор. Оператор хранения данных. «Дейта» означает данные. Этот оператор только хранит данные. И все. Интерпретатор, встретив такой оператор в программе, перескочит его. В значения переменных данные переносятся другим оператором — READ.

— «Рид» — это «читать». Выходит, что это оператор чтения...

— Или считывания. Из хранилища в операторах DATA он считывает данные в значения переменных. В программе оператор READ появляется в строке 230, где начинается цикл



Кто пришел за данными ко мне?

по всем странам Южной Америки. В этом цикле считываются названия стран, их окончания в обоих падежах и столицы.

Само чтение происходит так. Проходя цикл первый раз, когда $i=1$, интерпретатор, натолкнувшись на оператор READ с элементами символьных массивов $R(i)$, $S(i)$, $I(i)$ и $II(i)$ — сейчас $R(1)$, $S(1)$, $I(1)$, и $II(1)$, начинает искать, где в программе встречается оператор DATA. Находит, что в первый раз в строке 130 и что первые четыре символьные строки, там хранящиеся, — это "КОЛУМБИ", "БОГОТА", "Я" и "И".

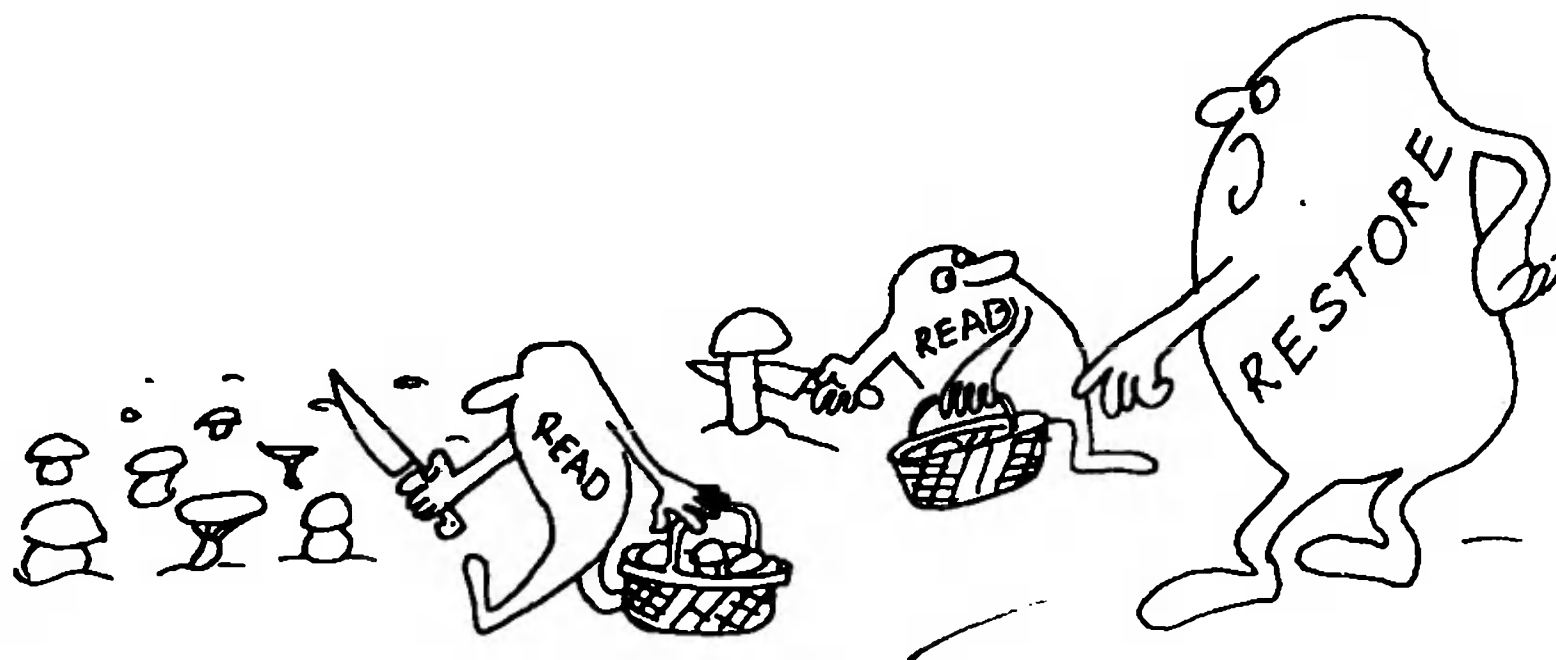
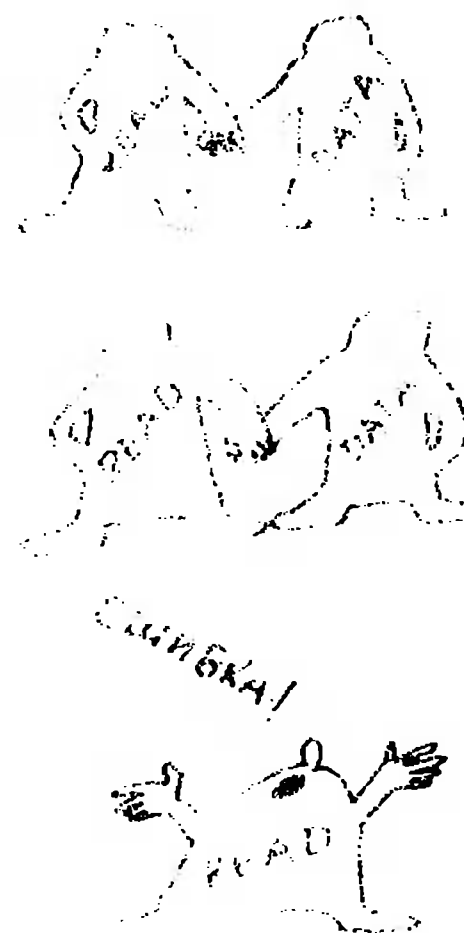
— Понятно! «Я» будет использована, чтобы сделать «Колумбия», а «И», чтобы «Колумбии».

— Правильно, но дай мне сказать, что эти четыре строки и становятся значениями элементов $R(1)$, $S(1)$, $I(1)$ и $II(1)$. Обрати внимание, что в операторе DATA одно значение от другого отделяется запятой.

Дальше, проходя цикл второй раз, свои значения получают элементы $R(2)$, $S(2)$, $I(2)$ и $II(2)$. Оператор READ их считывает из хранилища по порядку. Так что на сей раз это будет "ВЕНЕСУЭЛ", "КАРАКАС", "А" и "Ы". И так до конца цикла. Конечно, в операторах DATA все данные должны храниться в таком порядке, в каком они понадобятся оператору READ.

Во время работы интерпретатор отмечает, до какого места в операторах DATA он дошел. Если все данные уже считаны, но появляется еще оператор READ, то это ошибка.

Иногда данные, хранящиеся в операторе DATA, могут понадобиться программисту несколько раз. Тогда надо привлечь оператор RESTORE. «Ристор» — это «восстановить». Когда интерпретатор найдет в программе этот оператор, он заставит следующий READ считывать данные с самого первого оператора DATA. Сейчас смотри выбор режима в строках 270—340, а потом переходи на режим показа в строках 370—430. На экран выводятся столбиком порядковые номера стран, их названия и столицы. При этом мы и добавляем



Данные собирайте
с самого начала!

падежное окончание к основе названия страны. Это происходит в самом операторе печати в строке 410. Потом идет режим повторения. Строки 440—480 ты, надеюсь, понимаешь?

— В особенности строку 480. Хоть один бы разочек за всю мою долгую жизнь сказали бы: когда тебе надоест заниматься, то... Поверь мне, ни разу!

ВАЖНОЕ ПОНЯТИЕ — ПОДПРОГРАММА

Подпрограмма

— Прежде чем смотреть дальше, поговорим об одном важном понятии программирования. О понятии подпрограммы. Подпрограмма — это вроде узкого специалиста по какому-то конкретному делу. О них надо думать, если в программе что-то надо сделать несколько раз, при этом всякий раз с другими значениями переменных. Скажем, в этой программе мне нужно было несколько раз определить, какой стране соответствует выбранное случайное число, но каждый раз в другом месте программы. Я этот алгоритм запрограммировал отдельно — подпрограммой, начав нумеровать строки с тысячи, хотя с таким же успехом мог быть и другой номер. А там, где надо выполнить этот алгоритм, пишу оператор GOSUB и номер строки, с которого начинается подпрограмма:

GOSUB 1000

— Теперь, конечно, выясним, что это английское слово означает и как его произносить.

— Произносят его «гоусаб», и оно является сокращением английских слов go to subroutine, что означает «идти на подпрограмму». Это еще не все. Очень важно, что этот оператор



Тоня Соображалкина
на в роли под-
программы

передает управление так же, как и оператор GO TO. Выполнение программы продолжается с той строки, на которую было передано управление, до появления оператора RETURN.

— «Ритэн» означало... означало...

— ...«Возвращаться». Интерпретатор возвращается на оператор, который был после GOSUB. Итак, чтобы какую-то часть программы можно было бы считать подпрограммой, там должен быть оператор RETURN. Скажи четко, чем GO TO отличается от GOSUB?

— Ну... если GO TO передает управление куда-то, то это навсегда, а если GOSUB передает управление, то оно вернется назад, как только встретит RETURN. Но, по-моему, пора наконец возвратиться к твоей суперпрограмме.

— Где мы остановились?

— На строке 530.

АВТОМАТИЗАЦИЯ ПЕДАГОГИКИ

— Здесь мы начинаем режим повторения присваиванием нулей переменным, которые означают количество заданных вопросов и ошибочных ответов.

— Сначала выбираем случайное число. Странно, что с ним ничего не делаем. Обычно на что-то умножали и брали целую часть...

— На сей раз нужно случайное число как раз из отрезка $[0; 1]$. Оно определяет, какой зададим вопрос: покажем ли страну и спросим столицу или наоборот. При этом используется значение U1, о котором уже говорили.

— Это ничего. Можно еще поговорить. Хочу разобраться, что здесь произойдет. У тебя значение U1 было 0,5?

— Это означает, что оба типа вопросов будут встречаться одинаково часто.

— Хорошо. А если $U1 = 0$... Тогда полученное случайное число R из отрезка $[0; 1]$ обязательно будет больше U1, и мы уйдем на строку 670.

— То же самое было бы, если задали бы значение U1 меньше нуля.

— Другой случай. Если число R больше U1. Тогда управление всегда перейдет на следующую строку — 580. А если это значение, например, 0,9, то почти всегда будет выполняться следующая строка, и только в одном случае из десяти перейдем на строку 670. Думаю, что в этих трюках разобраться можно. Продолжим.

— Если оператор ветвления в строке 570 пропускает на следующую строку, то мы сразу выводим вопрос и обращаемся к подпрограмме, начинающейся со строки 1000. Она сделает случайный выбор номера страны.

- Посмотреть бы эту подпрограмму.
- Она совсем простая. Выбираем случайное число между 1 и 13. И все.
- И в конце оператор RETURN, как это положено в подпрограммах.
- На сей раз этот оператор передаст управление назад на строку 610, где оператор PRINT выведет нужную столицу, и оператор INPUT в следующей строке обеспечит ввод названия страны.
- Такое уже было. И последующее сравнение тоже достаточно обычное. Смотрим, не ввел ли ученик знак исмера, и если ввел, то переходим на 750. А что там? Ага! Выводим результаты труда и возвращаемся на выбор режима. А если ему еще не надоело? Если не надоело, то почему-то переменной N_Q присваиваем. . . что же присваиваем?
- N_Q — это переменная, значение которой — правильный ответ. В данном случае название страны, которое ученик должен ввести. Опять приходится использовать старый фокус с добавлением окончания в названии страны. Конечно, я считаю, что ученик должен ввести полное название, например: УРУГВАЙ, а не только основу слова, которую я храню в массиве R_Q: УРУГВА.
- И что, потом опять уходим на подпрограмму?
- Да, но сейчас это другая подпрограмма. Она сравнивает введенный ответ с правильным и сообщает, что получилось. Можешь посмотреть текст подпрограммы. Если ответ неправильный, приходится работать дополнительно — нужно увеличить количество допущенных ошибок на единицу.
- И, возвратившись из подпрограммы, идем на строку 560, чтобы все начать сначала.
- Следующая часть программы, где показывают страну и надо ввести столицу, почти такая же, как только что изученная. Немножко отличается вывод, а правильный ответ, конечно, берем из массива столиц.

МАШИНА — ЭКЗАМЕНАТОР

- Теперь уж перейдем на прием экзамена.
- Перед экзаменом расскажу, как он будет проходить. Случайным образом выбираем одну из 13 стран и спрашиваем ее столицу, потом одну из оставшихся 12, потом из 11 и т. д. до конца. Получается, что в отличие от повторения на экзамене один и тот же вопрос не может быть задан два раза. В специальном массиве E отмечаем те страны, о которых уже спрашивали. В начале экзамена в этот массив засылаем нули и после каждого вопроса засылаем единицу

в элемент массива с номером страны, о которой спрашивали.

— Вижу. Цикл засылки нулей имеется в строке 810. В следующей строке ты присваиваешь нули двум переменным.

— Это количество вопросов J и ошибочных ответов K. А что в следующей строке 830?

— Здесь обычный выбор случайного числа от 1 до 13. А потом какой-то странный оператор ветвления.

— Этот оператор как раз выясняет, была ли уже эта страна или нет. Если была, то берем следующее случайное число, а потом, если надо, еще и еще, пока не выпадет такая страна, о которой еще не спрашивали.

— Так ведь эти случайные числа можно брать и брать до бесконечности. Если, допустим, осталась только одна страна...

— Критику признаю справедливой. Придумай алгоритм, у которого такого недостатка не было бы.

— Опять педагогические трюки!

— Никакие не трюки, а предоставление возможности роста. Теперь смотри строку 850.

— Там ничего интересного нет! Заставляют вводить столицу выбранной страны, и, если бедняга-мученик вводит неправильно, программа ругается. Что там другое может быть?

— И вовсе она не ругается. Мои программы никогда не ругаются. Нам таких делать не разрешали. Конечно, начинающему кажется забавным, если программа обзывает пользователя олухом: мол, какие это компьютеры умные, даже ругаться могут. А потом, когда понимаешь что к чему, ничего интересного и удивительного не находишь. Одно только расстройство, потому что, как бы ни понимал, что у машины никакой злости к тебе нет, все равно неприятно, когда тебя ругают. В обучающих программах этого уж никак не должно быть, ученик должен получать одни только радости от общения с компьютером.

— Радости? А посмотри, что происходит дальше. После каждого вопроса в строке 900 отмечается, о какой стране спрашивалось, и, когда опрошены все 13, начинаются радости: если человек не знал всего одну только столицу, ему трояк, а если две, то уже пара. Ты, Петька, предатель общего дела учеников! Надо было делать так: если знает хоть одну — тройка.

— Столицы нетрудно выучить. И какой от тебя толк, если даже столицы стран не знаешь? Алгоритмыч меня заставлял ставить либо пятерку, либо двойку. Троечка уже моя самодеятельность. Лучше «проразмыслим» нашу программу. Как ее можно было бы толково переделать?



КАК МАШИНА МОГЛА БЫ ПОМОЧЬ В ИЗУЧЕНИИ ГЕОГРАФИИ. . .

— Легче всего было бы сделать программу, обучающую столицам стран Северной Америки. Там даже менять ничего особенно не надо было бы.

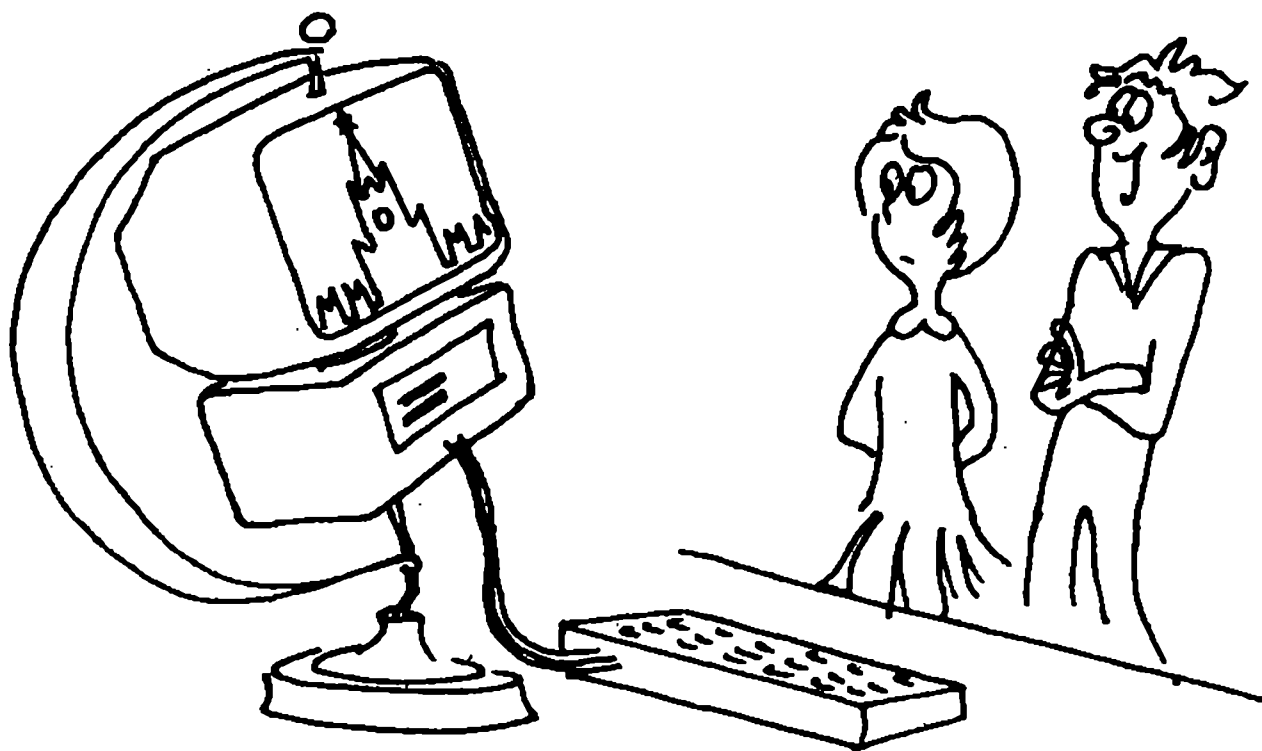
— Кое-что все-таки пришлось бы: заменить все содержание операторов DATA и всюду, где идет речь о Южной Америке, написать о Северной Америке. И еще другие, более мелкие изменения. Подобным же образом можно было бы создать программу обучения столицам всех государств мира.

— Это уже получилась бы капитальная работа! В мире же около двухсот государств. Только ввести названия всех столиц и государств. . .

— Немножко, конечно, пришлось бы повозиться, но живой ты, Тонька, осталась бы. Правда, кажется, что такой экзамен никто не смог бы вынести. Ввести двести ответов — это уж слишком. Экзамен надо было бы переделать так, чтобы спрашивать только, скажем, о двадцати государствах.

— Не двадцати, а ввести постоянную, и пусть учитель ее задает, какую хочет. Подобно тому как ты поступал с постоянной U1. И систему выставления оценок нужно переделать. Она вообще должна быть переделана.

— Здесь тоже можно было бы ввести постоянные. Со столькими-то ошибками ставим четыре, со столькими-то — три, а со столькими-то — уже двойку. И эти постоянные учитель может менять, как ему вздумается. Можно и так сделать, чтобы программа чаще спрашивала те страны, на которых ученик чаще всего ошибается.



...ЯЗЫКОВ...

— А вообще похожими получились бы и программы освоения слов какого-нибудь иностранного языка. В один массив загони тысячу-другую слов, во второй массив — их переводы, и пожалуйста! Правда, одним русским словом перевести слово иностранного языка не всегда можно, так что пришлось бы ограничиться основным значением. Можно было бы учить не только значения слов, но и различные правила, например формы неправильных глаголов в английском языке, да, я думаю, и в других что-то похожее нашлось бы.

— С неправильными глаголами не так-то просто. Они имеют три формы, например *go — went — gone*, а у тебя только два массива.

— Невелика наука добавить еще один массив, но можно и без этого обойтись. В одном массиве я размещаю неопределенную форму *go*, а во втором — две другие: *went* и *gone*, разделяя их запятой или пробелом. Программа никогда не догадается, что это два слова. Это символьная строка, да и только.

— Что это мы все про языки, а как с другими учебными предметами? Здесь такие программы могли бы оказаться полезными?

— Думаю, что да, в любом предмете найдется, чему обучать компьютером.

— И в литературе тоже?

— Сочинения машина писать не сможет, да и по-настоящему обучать этому делу тоже. Я могу предложить составить программу, с помощью которой можно было бы выучить годы жизни писателей. В одном массиве писатель, в другом — годы. Правда, не знаю, как вас, но нас не очень заставляют зубрить эти даты.

...ИСТОРИИ...

Другое дело — история. Здесь смогла бы помочь программа, которая бы повторяла с учеником важнейшие события и годы, когда они происходили. Опять в одном массиве год, в другом — событие.

— Ты хочешь сказать, что значением элемента одного массива могло бы быть целое описание какого-нибудь события, например начала Великой французской революции?

— Почему бы и нет? Но при повторении истории имеет смысл называть только события и спрашивать год. Если делать наоборот, то будет очень трудно догадаться, что именно



произошло, скажем, в 1890 году. В этом году произошло много чего, и совершенно неизвестно, что имел в виду составитель программы.

— Я не могу вспомнить, что произошло в этом году.

— Это я так, для примера. Но если бы и удалось правильно догадаться, то описание нужно было бы ввести точно в такой же форме, как оно дано в соответствующем операторе DATA. Если там будет написано ЛЕВ ТОЛСТОЙ ПИСАЛ РОМАН, то ни в коем случае нельзя отвечать Л. ТОЛСТОЙ. . .

— Придумала, как преобразовать эту программу!

— Как?

...И ЕЩЕ КОЕ-ЧЕГО

— Но только это не совсем для школы. Можно было бы повторить мировые рекорды по легкой атлетике. В одном массиве дисциплина, в другом — рекорд. Если кто желает покрасоваться своими знаниями, может выучить.

— Можно было бы. Но кто же не знает рекорды легкой атлетики?! Лучше бы названия популярных песен и ансамблей, которые их исполняют.

— Вот это-то все знают. Ведь популярный ансамбль — это такой, который все знают.

— А знаешь, что мне самому не нравится в этой программе? То, что ученику приходится вводить очень много — все название страны целиком. Надо было сделать так, чтобы выводился весь список стран или столиц с номерами и ученик должен был указать правильный номер. Правда, такой список работал бы немножко как шпаргалка.

— Долго мы провозились с этой твоей программой! А началось все с вопроса, как использовать мою программу склонения существительных для обучения детей! Получается, что в памяти в операторах DATA нужно хранить кучу существительных первого склонения, потом считывать их в массив, случайным образом оттуда вынимать и предлагать ученику просклонять.

— Думаю, можно было бы получить еще одно случайное число от 1 до 6, которое бы означало номер падежа, и именно этот падеж спрашивать. Разумеется, опять нужно было бы создавать массив с названиями падежей и вопросами; кроме того, другой массив с окончаниями, ну, чтобы построить правильный ответ. Тогда пришлось бы проверять, не выпал ли предложный падеж, потому что в этом случае не только меняется окончание, но и нужно добавить предлог «о» перед словом. Ой, пора кончать! Я еще должен учить русский, и совсем не склонение существительных первого склонения.

19. ТОНЯ УЗНАЕТ О ФАЙЛАХ

Уже не раз Тоня возмущалась, что результаты работы ее программ можно видеть только на экране, а она их охотно бы напечатала, чтобы можно было бы их сохранить и дома показать. А Петя почему-то ее этому не учит. Петя же все время отнекивался, говорил, что еще рано и что лучше всего эти дела осваивать одним махом — выучить сразу все о файлах. Поэтому Тоня обрадовалась, услышав:

— Сегодня начнем углубляться в науку о файлах, или, как я ее называю, файлологию.

— Наконец-то! Давно пора.

— Конечно, можно было бы и раньше, но, на мой взгляд, до того как браться за файлы, тебе нужно было привыкнуть к программированию, к машине. Теперь ты эти файлы, может быть, мгновенно поймешь, но только потому, что твой мыслительный аппарат настроился на формальное мышление, т. е. на такое мышление, какое нужно программисту. А работать с файлами нужно уметь. Бывают такие программы, при том очень важные, которые работают почти только с файлами.

— Да ладно. Ближе к делу!

— Я тебе уже как-то говорил, что файл можно понимать как место на внешнем носителе информации — так называют диски и ленты. Это место имеет обозначение, или имя, по которому его и записанные в нем данные в любой момент можно легко найти.

ФАЙЛ = КНИГА

Каждый файл может быть либо открыт, либо закрыт. Конечно, никто на внешнем носителе информации ничего не закрывает и не открывает. Это такой способ выражения. Открыть файл означает разрешить с ним что-то делать — записывать в него данные или считывать их с него. А закрыть файл означает временно о нем забыть. До него нельзя добраться, его будто нет, и, пока он не открыт, с ним делать ничего нельзя.

Открытие файла

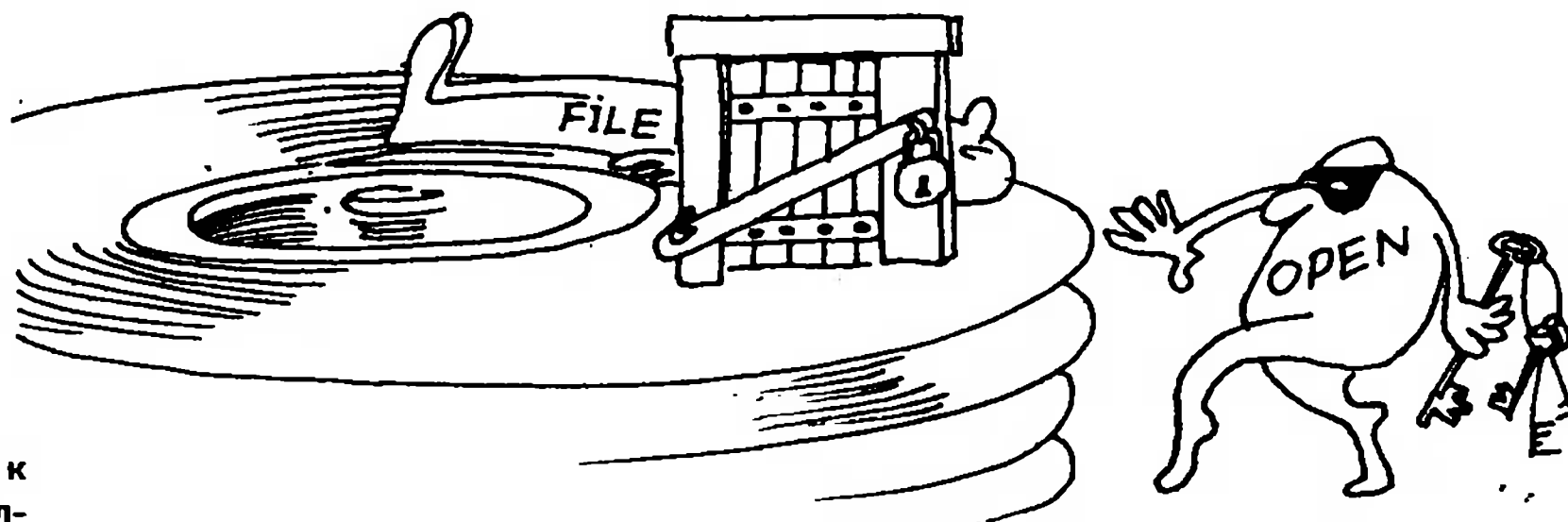
— И как же его открывают?

— А как по-английски «открыть»?

— «Оупен».

— Вот и файлы открывают оператором OPEN. Например, так:

OPEN "DK 1: FAIL" FOR INPUT AS FILE 3



Чтобы попасть к данным, OPEN должен открыть файл

Как видишь, это самый длинный оператор Бейсика. Переведем его на русский язык.

OPEN	"DK1:FAIL"	FOR INPUT	AS	FILE 3	На Бейсике
Открыть	файл с таким описанием	для ввода	в качестве	файла номер 3	По-русски

— Непонятно, почему здесь вылез какой-то «номер 3»?

ЛОГИЧЕСКИЙ НОМЕР ФАЙЛА

— Я это понимаю так. Интерпретатор, сам будучи программой, обрабатывает этот оператор. У интерпретатора в памяти предусмотрено место для хранения сведений об открытых файлах, можно сказать, массив, содержащий символьные строки — описания файлов. А элементы массива положено нумеровать. Вот этот номер и будет номером файла. Еще его называют логическим номером файла. Просто в операторе открытия файла мы указываем номер элемента этого массива, куда надо поместить описание открываемого файла.

— Так что в каждый момент могут быть открытыми несколько файлов?

— Именно так. И притом файл с логическим номером 3 отнюдь не обязательно открывать третьим, его можно открывать и первым, до того как будут открыты файлы с логическими номерами 1 и 2. Открывая файл, интерпретатор проверяет, существует ли на указанном устройстве такой файл, и если существует, то не открыт ли он уже. Если удастся обнаружить какое-то несоответствие, об этом сразу сообщается.

Помнишь ли, что означало, например, такое описание файла: "DK 1 : FAIL"?

Логический номер файла

— Да, помню. DK1 означает дисковое устройство и его номер, а FAIL — имя файла.

— В операторе OPEN описание файла необязательно задавать явно. Его можно задать и в виде значения символьной переменной. Сразу становится понятным, что описание файла еще можно сцеплять из отдельных символьных строк. Посмотри, пожалуйста, такой довольно интересный кусочек программы.

```
100 PRINT "ВВЕДИТЕ НОМЕР ДИСКА"  
110 INPUT N  
120 PRINT "ВВЕДИТЕ ИМЯ ФАЙЛА"  
130 INPUT F  
140 OPEN "DK"+N+":"+F FOR INPUT AS FILE 1
```

Сперва вводим номер диска. Как символ! Потом вводим имя файла.

— Как символы!

— Естественно. А потом сцепляем полное описание файла.

Сейчас о слове INPUT в операторе OPEN. Вместо него может быть и OUTPUT.

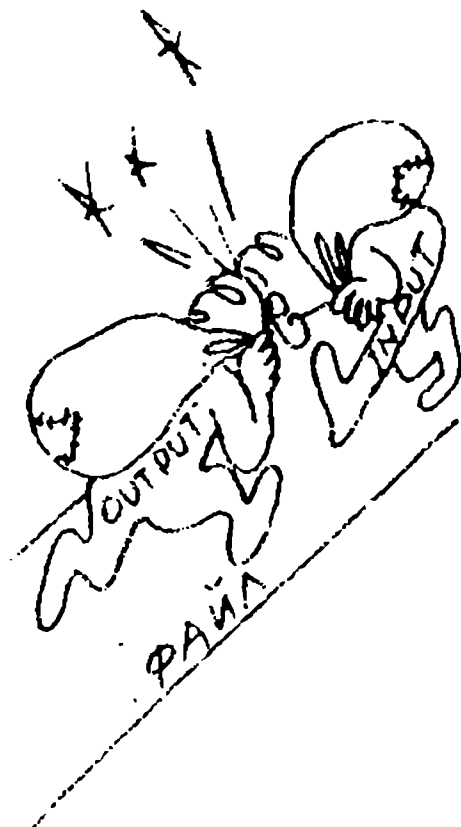
— «Аутпут»... что бы это значило? Может быть, «вывод»? «Ввод» уже был.

— Да, в этом случае файл будет открыт для вывода. Одновременно для ввода и вывода на нашей машине файлы открыть нельзя. Интерпретатор это не позволяет. Чтобы в одном и том же файле сначала записать, а потом считать, нужно, во-первых, файл открыть для вывода; во-вторых, записать в нем; в-третьих, закрыть; в-четвертых, открыть для ввода и, наконец, в-пятых, считать необходимое.

— Как сложно!

— Дело в том, что считывать данные с файла можно только по порядку. И записывать в файл тоже. Когда мы считываем с файла, интерпретатор знает, сколько мы считали — до какого места в файле. И когда записываем в файл, то тоже известно, где находимся. А если в какой-то файл захотим и записывать, и с него же считывать, то может случиться, что мы захотим в один и тот же момент находиться в двух местах. А этого никто не может. Или представь, что мы считываем файл с магнитной ленты. Мы же сможем считать только с того места, где остановили перемотку ленты. Или если мы читаем данные из обычной книги, то ясно, что читать можно только с того места, где кончили листать. Кстати, когда подрастешь, может быть тебя научат записывать в любом месте файла.

— Ясно, что можно считывать с начала файла, если его закрыть и потом снова открыть. Но ты же не собираешься рассказывать, как закрыть файл?



ДЕЙСТВИЯ С ФАЙЛАМИ

- Нет, собираюсь. Его закрывают оператором CLOSE.
- Еще бы! «Клоуз» же означает «заккрыть».
- Этот оператор намного проще оператора OPEN. Нужно только после него написать логический номер того файла, который хочется закрыть:

CLOSE 3

А если номера после CLOSE нет, то закроются все открытые в этот момент файлы. Вот и все про закрытие.

— Прекрасно. Еще один кирпичик в здание Дворца знаний Тони Соображалкиной.

— Еще один очень важный, но чрезвычайно опасный оператор:

KILL "DK1:FAIL"

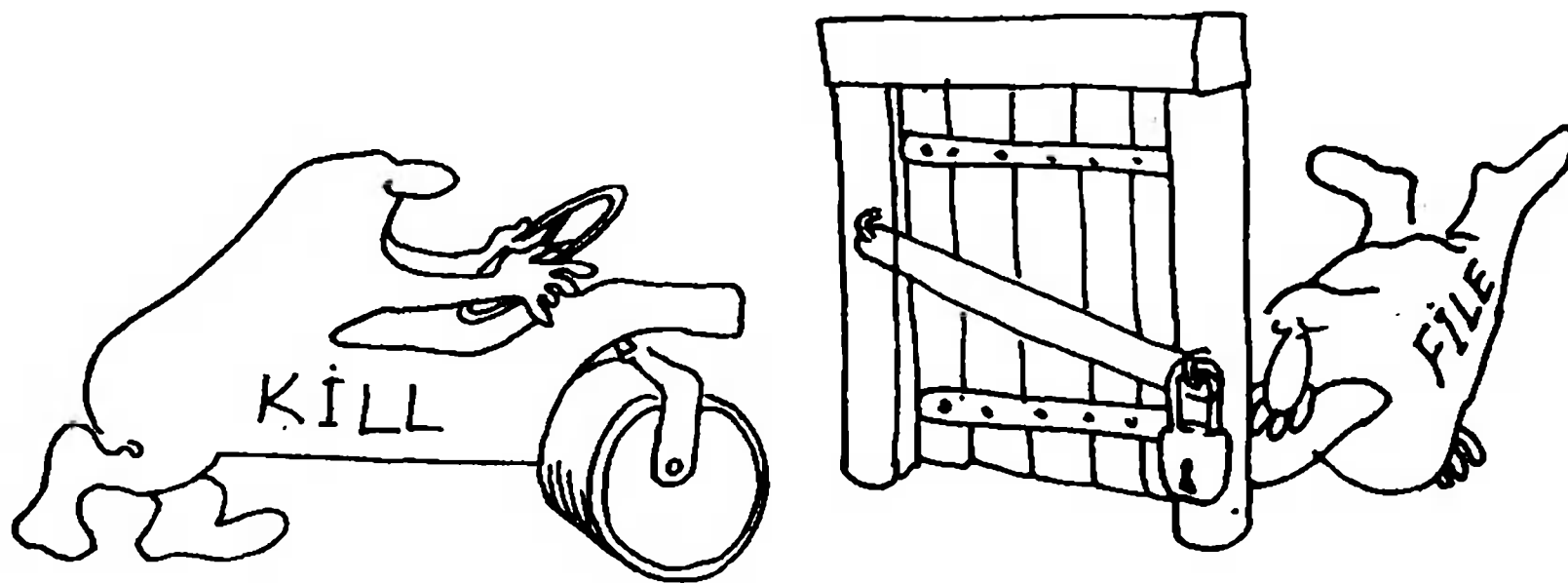
— «Кил»... Не знаю.

— «Убивать»! Оператор, пожалуй, никого не убивает, он просто стирает файл. После выполнения этой строчки на диске номер 1 файла с таким именем не будет. Как видишь, стираемый файл указывается после слова KILL. Либо в явном виде — в кавычках, либо как символьная константа, значение которой — описание стираемого файла.

— Стирает... Просто так — берет и стирает. А если потом опять понадобится?

— Понадобится или нет, но никакими силами ты этот файл не вызовешь из небытия. Кстати, большое преимущество магнитных дисков по сравнению с бумагой то, что с них все легко стереть. А если уж сотрешь то, чего не надо было, пеняй на себя.

— Бр-р-р. Дрожь берет.



ЗАПИСЬ В ФАЙЛ

— Еще следовало бы поговорить о том, как в файл записать и как с него считать. Это делают теми же старыми, добрыми операторами INPUT и PRINT, которые ты хорошо знаешь. Они только чуть дополнены. Смотри, как я пишу:

```
PRINT #3, A,B
```

После слова PRINT идут знак номера «#», номер файла, запятая и список переменных. Этим можно обеспечить запись значения переменных A и B в файл, который в настоящий момент уже открыт для вывода как файл с логическим номером 3. Подобно делается и с оператором INPUT:

```
INPUT #4, B
```

Если напишешь так, получишь значение переменной B с файла, который открыт для ввода с логическим номером 4. Подчеркну, что ввод данных будет продолжен с того места файла, где мы прекратили ввод, когда предыдущий раз вводили. С выводом то же самое.

— Это все про ввод и вывод?

— Нет, не все. Еще ты должна знать, что в этих операторах после знака номера можно писать номер файла не только в явном виде, но и как числовую переменную. Если там написать нуль или значение числовой переменной будет нуль, то ввод или вывод будет проходить на дисплее. Так, как было тогда, когда мы знак номера еще вообще не писали. Получается, будто на дисплее всегда открыт файл с номером нуль. Теперь-то все.

— Вроде бы поняла. Только этот логический номер файла неясен немножко. Но как же все, если ты ничего не сказал про вывод на печать!

— На устройстве печати надо открыть файл для вывода с описанием, состоящим только из кода устройства, подобно тому как мы делали при печати программ:

```
OPEN "LP:" FOR OUTPUT AS FILE 4
```

Разумеется, если для этого файла ты выбрала четвертый логический номер. И дальше выводить в этот файл как обычно, т. е. писать PRINT со знаком номера и номером файла.

Про печать все. Про файлы все. И про программирование тоже все. Сейчас ты, Тоня, в программировании знаешь все существенное и менее существенное тоже, считай, все. Теперь посмотрим еще кой-какие примеры программ — и за работу.

— Так точно! Программировать, программировать и еще раз программировать! Таков пусть будет наш девиз!

КОМПЬЮТЕР ПОЗДРАВЛЯЕТ С ПРАЗДНИКОМ

— Займемся сейчас программой печати поздравлений. Что она должна делать и что она делает? Она должна красиво напечатать поздравление с днем рождения или с каким-то другим праздником. Красиво на сей раз означает, что вокруг текста поздравления должна быть выпечатана рамочка из какого-то символа. Он называется декоративным символом.

В начале поздравления требуется поставить обращение, состоящее из слов «Дорогой», «Уважаемый» и т. п. и из имени поздравляемого человека. Имя поздравляемого программа узнает у пользователя. Разумеется, вводить именно имя необязательно. С таким же успехом могут быть имя и отчество или фамилия. Еще программа узнает у пользователя, как обращаться к поздравляемому человеку — на «ты» или «вы», ученик он или взрослый. Может, начнем смотреть программу, тогда и увидишь, что там происходит.

```
10 REM ПРОГРАММА ПЕЧАТАЕТ ПОЗДРАВЛЕНИЯ
20 R=16 \ REM КОЛИЧЕСТВО СТРОК
30 N=30 \ REM КОЛИЧЕСТВО ЗНАКОВ В СТРОКЕ
40 Sx="*" \ REM ДЕКОРАТИВНЫЙ СИМВОЛ
50 S1x="С ДНЕМ РОЖДЕНИЯ." \ S2x="12"
60 DIM L1x(2,2),Tx(10)
70 L1x(1,1)="ДОРОГОЙ" \ L1x(1,2)="ДОРОГАЯ"
80 L1x(2,1)="УВАЖАЕМЫЙ" \ L1x(2,2)="УВАЖАЕМАЯ"
90 DIM L2x(2) \ L2x(1)="ТЕБЯ" \ L2x(2)="ВАС"
100 DIM L3x(2) \ L3x(1)="ТЕБЕ" \ L3x(2)="ВАС"
105 \
110 DATA " ", "ПОЗДРАВЛЯЮ ", " ", "ЖЕЛАЮ "
120 DATA "БОЛЬШОГО СЧАСТЬЯ, ", "ХОРОШЕГО НАСТРОЕНИЯ"
130 DATA " ", " ", " ", " ", " ПЕТЯ"
135 \
140 PRINT "ПРОГРАММА ПЕЧАТАЕТ ПОЗДРАВЛЕНИЯ"
150 PRINT " ВЫВОД НА ЭКРАН - 1"
160 PRINT " НА БУМАГУ - 2" \ INPUT Fx
165 IF POS(S2x,Fx,1)=0 THEN 150 \ F=VAL(Fx)
170 F=F-1 \ REM СЕЙЧАС F НОМЕР ФАЙЛА
180 IF F=0 THEN 200 \ REM ВЫВОД НА ЭКРАН
190 OPEN "LP:" FOR OUTPUT AS FILE 1 \ REM ПЕЧАТЬ
195 \
200 PRINT "ОБРАТИТЬСЯ НА ТЫ - 1"
210 PRINT " НА ВЫ - 2" \ INPUT Tx
220 PRINT \ IF POS(S2x,Tx,1)=0 THEN 200
230 PRINT "ДОРОГОЙ - 1"
240 PRINT "УВАЖАЕМЫЙ - 2" \ INPUT Cx
250 PRINT \ IF POS(S2x,Cx,1)=0 THEN 230
```

```

260 PRINT "УЧЕНИК          - 1"
270 PRINT "ВЗРОСЛЫЙ       - 2" \ INPUT Ux
280 PRINT \ IF POS(S2x,Ux,1)=0 THEN 260
290 PRINT "      ИМЯ "; \ INPUT Ux
295 PRINT \
300 L=LEN(Ux) \ REM ДЛИНА СЛОВА
310 Px=SEGx(Ux,L,L) \ REM ПОСЛЕДНЯЯ БУКВА
320 \
325 REM НАЧАЛО ОПРЕДЕЛЕНИЯ ПОЛА
330 IF Px<>"А" THEN 370
340 IF SEGx(Ux,L-1,L-1)="Ш" THEN 380
350 IF Ux="ВАЛЕРА" THEN 375
351 IF Ux="СЕРЕЖА" THEN 375
352 IF Ux="СЛАВА" THEN 375
353 IF Ux="ВОВА" THEN 375
354 IF Ux="СЕВА" THEN 375
355 IF Ux="ЛЕВА" THEN 375
356 IF Ux="ЮРА" THEN 375
357 IF Ux="ДИМА" THEN 375
360 D=2 \ GO TO 470 \ REM ЖЕНСКОЕ ИМЯ
370 IF Px="Я" THEN 380
375 D=1 \ GO TO 470 \ REM МУЖСКОЕ ИМЯ
380 PRINT "ПОЗДРАВЛЯЕТЕ МУЖЧИНУ - 1"
390 PRINT "      ЖЕНЩИНУ - 2" \ INPUT Dx
400 D=VAL(Dx) \ IF POS(S2x,Dx,1)=0 THEN 380
440 \
450 REM НАЧИНАЕТСЯ ПОДГОТОВКА ТЕКСТА
460 \
470 RESTORE \ FOR J=1 TO 10 \ READ Tx(J) \ NEXT J
475 REM СНАЧАЛА ФОРМИРУЕТСЯ ОБРАЩЕНИЕ
480 Tx(1)=L1x(VAL(Cx),D)+" "+Ux+"!"
490 Tx(2)=Tx(2)+L2x(VAL(Tx)) \ REM ТЕБЯ ИЛИ ВАС
500 Tx(3)=S1x \ REM В СВЯЗИ С ЧЕМ ПОЗДРАВЛЯЕМ
510 Tx(4)=Tx(4)+L3x(VAL(Tx)) \ REM ТЕБЕ ИЛИ ВАМ
520 IF Ux="1" THEN 540
530 Tx(7)="И КРЕПКОГО ЗДОРОВЬЯ!" \ GO TO 560
540 Tx(7)="И УСПЕХОВ В УЧЕБЕ!"
545 \
550 REM НАЧИНАЕТСЯ ВЫВОД =====
555 \
560 GOSUB 1000 \ REM ПЕРВАЯ СТРОКА РАМКИ
570 R1=INT((R-10)/2) \ REM СКОЛЬКО СТРОК БЕЗ ТЕКСТА?
580 GOSUB 2000 \ REM И СТОЛЬКО ВЫПЕЧАТЫВАЕТ
585 \
590 REM СЕЙЧАС ТЕКСТ В РАМКЕ
600 FOR J=1 TO 10 \ REM ЦИКЛ ПЕЧАТИ ТЕКСТА
610 N1=INT((N-2-LEN(Tx(J)))/2) \ REM СКОЛЬКО ПРОБЕЛОВ?
620 PRINT #F,TAB(5);Sx;TAB(N1+6);Tx(J);TAB(N+4);Sx

```

```

630 NEXT J
635 \
640 GOSUB 2000 \ REM ПЕЧАТЬ СТРОК БЕЗ ТЕКСТА
650 GOSUB 1000 \ REM И ПОСЛЕДНЕЙ СТРОКИ
660 CLOSE \ PRINT \ REM ЗАКРЫВАЕТ ВСЕ ФАЙЛЫ
665 \
670 PRINT "ВЫВЕСТИ ВСЕ КАК ЕСТЬ - 1"
680 PRINT "ИЗМЕНИТЬ ИМЯ ПОЗДРАВЛЯЕМОГО - 2"
690 PRINT "ПОДГОТОВИТЬ ВСЕ ЗАНОВО - 3"
700 PRINT "ЗАКОНЧИТЬ РАБОТУ - 4"
710 INPUT I1
715 \
720 F=0 \ ON I1 GO TO 740,290,150,730
725 \
730 STOP
740 F=1
750 OPEN "LP:" FOR OUTPUT AS FILE 1
760 GO TO 560
770 END

775 \
1000 REM ПЕЧАТЬ СТРОКИ ДЕКОРАТИВНЫХ СИМВОЛОВ
1010 PRINT #F,TAB(5);
1015 FOR I=1 TO N \ PRINT #F,S*; \ NEXT I
1020 PRINT #F \ RETURN
1025 \
2000 REM ПЕЧАТЬ РАМКИ БЕЗ ТЕКСТА
2010 FOR I=1 TO R1
2020 PRINT #F,TAB(5);S*;TAB(N+4);S*
2030 NEXT I
2040 RETURN
3000 END

```

Ой, совсем забыл!

У меня же есть образец напечатанного поздравления.

— Потрясающе! Почему ты мне не прислал на день рождения такое поздравление?

— Как я это мог сделать? А на следующий свой день рождения сама сможешь напечатать хоть сотню или даже две сотни.

— Так будет неинтересно.

— Теперь о самой программе. В самом начале присваиваю значение двум переменным R и N. Эти переменные определяют длину и высоту декоративной рамочки. Потом присваивается значение декоративному символу S* — обычно я беру звездочку. Далее задается праздник, с которым поздравляем, — значение переменной S 1*.

УВАЖАЕМАЯ ТОНЯ!
 ПОЗДРАВЛЯЮ ТЕБЯ
 С ДНЕМ РОЖДЕНИЯ.
 ЖЕЛАЮ ТЕБЕ
 БОЛЬШОГО СЧАСТЬЯ,
 ХОРОШЕГО НАСТРОЕНИЯ
 И УСПЕХОВ В УЧЕБЕ!

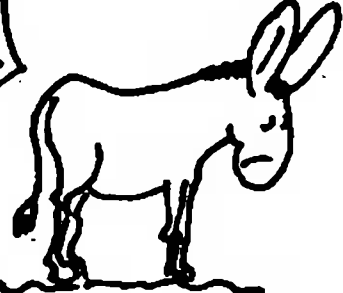
ПЕТЯ



— И это значение придется менять, если захочу поздравить не с днем рождения, а, например, с окончанием школы?

О ДВУМЕРНЫХ МАССИВАХ

— Совершенно верно. В следующей строке начинается кое-что новое. Описывается двумерный символьный массив `L1` тем же самым оператором `DIM`, что и одномерные массивы, но только после обозначения в скобках пишутся два числа, разделенные запятой. Двумерные символьные и числовые массивы появляются, если значения переменных имеют два номера, например оценка в таблице — номер предмета и номер четверти. Предметы, конечно, должны быть заранее пронумерованы. Другой пример — оценки фигуриста в протоколе соревнований с номером судьи и номером фигуриста. Опять же и фигуристы, и судьи должны быть пронумерованы. Обычно двумерные массивы в программах описывают какие-нибудь списки со строками и колонками. Таким списком является и табель, и протокол соревнований. Представь себе еще такой список: в строках — звери зоосада, а в колонках — сколько какого продукта они в день должны съесть. Вот такой маленький фрагмент:

	1	2	3	4
				
1 	1.0 	0.1 	0.0 	0.0 
2 	0.0 	0.1 	15.0 	0.0 
3 	0.0 	0.0 	0.05 	0.15 

Если ты теперь захочешь составить такую программу, которая обрабатывает эти данные, придется нам вводить двумерный массив.

— Не верю, что звери смогут прожить на твои нормы; и вообще, что тут считать!

— Не имею ни малейшего представления, сколько надо вороне, чтобы прожить. И бесспорно, здесь нечего считать, если у нас только три зверя. А если триста? Тогда можно было бы просчитать, каков расход продуктов, сколько они стоят и еще что-нибудь. Подобный список можно было бы составить и для завода, которому надо столько-то гвоздей, столько-то гаек, столько-то интегральных схем и столько-то разных проводов, чтобы сделать один такой, один другой и один, не знаю какой, телевизор.

— А зачем тебе понадобился двумерный массив именно в этой программе?

— В моей программе он предельно прост. Сначала нумерую степень уважения: 1 — дорогой; 2 — уважаемый. Потом род: 1 — мужской род; 2 — женский род. Потом выпи-
сываю двумерный массив:

Род	Степень уважения	
	1	2
1. Мужской	Дорогой	Уважаемый
2. Женский	Дорогая	Уважаемая

— Потом в строках 90, 100 ты еще определяешь символьные массивы и присваиваешь их элементам те значения, которые могут понадобиться при печати поздравления.

СОСТАВИТЬ ПОЗДРАВЛЕНИЕ НЕЛЕГКО

— Да, да. Притом в обоих массивах $L 2 \times$ и $L 3 \times$ первые элементы соответствуют обращению «ты», а вторые — «вы». Можно сказать, что возможные способы обращения я перенумеровал так: 1 — «ты»; 2 — «вы».

— Не понимаю, почему ты так много об этом говоришь? Какие значения хотел, такие присвоил.

— Ты права. Но я хочу подчеркнуть, что я должен хотеть присвоить эти значения, должен выбрать их и придумать, как программа их найдет.

— Ладно. Далее следуют операторы DATA, в которых, судя по всему, ты хранишь текст поздравления.

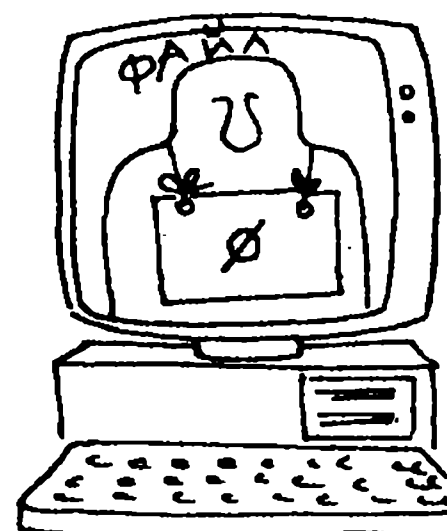
— Я бы сказал, заготовку текста. Потом, когда все символьные строки из операторов DATA перекоچуют в значения переменных, я их дополню и получу окончательный текст.

— Интересно, почему в строке 150 упоминается вывод на экран, если ты собираешься печатать?

— Это нужно для проверки. Не думай, что такую программу я могу сделать в один присест, не допустив ни одной ошибки. Программу надо было отлаживать, и во время отладки лучше выводить на экран, чтобы зря не тратить бумагу. Но вернемся к программе. Ты ввела свое пожелание, куда выводить поздравление: оно содержится в значении переменной $F \times$.

— А почему для хранения чисел 1 или 2 ты выбираешь символьную переменную? С числовой переменной проще.

— Ничего не проще. Это для тебя привычнее числа, чем символы, а машине что числа, что строки символов все одно. А вношу я ответ пользователя в значение символьной переменной для удобства проверки ввода. После ввода мне остается только проверить, входит ли введенный символ в строку символов $S 2 \times$, которую я загодя определил в строке 50. Этим я и занимаюсь в строке 165.



— А поскольку ты все равно потом хочешь иметь число, тебе приходится символ превращать в число при помощи функции VAL. Правильно ли я понимаю?

— Ты понимаешь правильно. Мы получаем число F и сразу вычитаем из него единицу. Зачем? Чтобы получить номер файла. Номер файла вывода на экран всегда должен быть нулем, а файлу печати можем присваивать такой номер, какой хотим. На сей раз лучше всего хотеть единицу.

— Лучше захоти объяснить, на что этот оператор вствления здесь в строке 180!

— Здесь мы вычисленный номер файла сравниваем с нулем. Если он оказался нулем, это означает, что пользователь желает вывести на экран, и мы спокойно уходим дальше на ввод данных. Если же номер файла не ноль, то надо в строке 190 открыть устройство печати — как файл с номером 1.

— Дальнейшее понятно. Пользователь вводит все, что от него требуется.

— Но правильность ввода все время проверяется.

— Это ты делаешь всегда одинаково, потому и ввод у тебя практически всегда одинаков. Ищешь введенный символ в значении переменной S 2 ♂, и все.

— Да, что-то поинтереснее начинается, когда введено имя поздравляемого и мы приступаем к его анализу. Сначала находим длину имени и его последнюю букву в строках 300 и 310. А потом выясняем, кого надо поздравлять — мужчину или женщину.

— Это же просто!

— Тебе просто! А для компьютера неразрешимая задача, потому что нет алгоритма, как по имени определить, мужское это имя или женское.

— Ну, женские имена оканчиваются на «-а»...

— Оканчиваются-то оканчиваются, да не все. И не каждое имя, кончающееся на «-а», будет женским. Более того, существуют же одинаковые имена и для мужчин, и для женщин.

— Ну да!

— Например, Саша может быть и юношей и девушкой.

— Так это же, ну... сокращение.

— Неважно. Программа должна была бы работать для всех случаев. Но алгоритм, который я запрограммировал, сможет уверенно отличить мужчину и женщину только при обращении по имени и отчеству. Конечно, по окончанию — оно «-а» или нет. Если оканчивается на «-а», а не на «-ша», то программа сама пытается выяснить, не мужчина ли это, сравнивая введенное имя с мужскими именами на «-а» — Слава, Савва, Сережа и др. А в случаях, когда обращение оканчивается на «-я» или «-ша», программа сдается и просит пользователя самого ввести, кого он поздравляет. Вот такой алгоритм.

— И запрограммирован он в строках 330—400...

— В результате работы этих строк D получает значение 1, если поздравляем мужчину, и 2, если женщину.

— Об этом хватит. Перейдем к подготовке текста поздравления, которая начинается в строке 470. И сразу вся заготовка текста считается в массив T. А в следующей строке...

— В следующей строке выясняется, почему всюду вводили либо 1, либо 2.

— Смотрим. Переменная S. Ее значение... 1, если надо обращаться «дорогой», и 2, если — «уважаемый». Переменная D указывает пол поздравляемого.

— Важно то, что значения этих переменных...

— ... Используются для получения нужного элемента массива L: «дорогостоящего» или «уважаемого» — в мужском или женском роде.

— Ты мне не даешь похвастаться хорошо сделанным куском. Тебе надо было спокойно все прослушать и в конце воскликнуть: «Ой, Петя, как ты хорошо придумал с этими переменными S и D!»

— Ой, Петя, как ты хорошо придумал, что к найденному значению массива L можно прицепить пробел и имя поздравляемого. И еще восклицательный знак!

— Пробел можно было бы и не ставить, но тогда в массив L нужно было бы заносить не просто «УВАЖАЕМЫЙ», а «УВАЖАЕМЫЙ » с пробелом в конце.

— Я же говорю, что это сногшибательно!

— Можешь смеяться, но я тебе скажу, что программировать интересно хотя бы потому, что пусть и мелочь, но можно хорошо придумать.

— И это дает творческое удовлетворение.

— Одному дает творческое удовлетворение работать, другому — издеваться.

— Ну не надо обижаться! Я же просто пошутила.

— Лучше посмотри, как образуются третий и четвертый элементы массива T. Здесь я сцепляю две символьные строки, а пробел между ними не ставлю, потому что в операторах DATA уже хранятся слова «поздравляю» и «желаю» с пробелом в конце.

— А что ты вытворяешь в строках 520 и дальше? Опять пренебрежительное отношение к ученикам! Взрослым ты желаешь доброго здоровья, а ученикам — успехов в учебе. Успехов в учебе, детка! Зачем тебе доброе здоровье, лишь бы были успехи в учебе.

— Не верю, чтобы кто-то потерял здоровье из-за успехов в учебе.

— Бедный Петя, если бы ты не был так смешон!

— Не понимаю, будем продолжать или отложим?

СОВСЕМ ПРОСТАЯ ПОДПРОГРАММА

— Все. Кончаю. Продолжаем. Смотрим строку 550. Начинается вывод. Строка 560 отсылает к подпрограмме, начинающейся со строки 1000... Что это за подпрограмма?

— Всего будут две подпрограммы. Первая для печати строки с декоративными символами. В ней содержится цикл, который выпечатывает по одному символу столько раз, сколько нужно.

— Вижу цикл. А что это за PRINT перед ним?

— Ах да! О функции TAB ты еще ничего не знаешь. Эту функцию можно вызывать только в операторе PRINT, и она указывает позицию строки, с которой надо продолжать печать. Поясню ее работу на примерах:

```
PRINT "ПРИВЕТ"; TAB (40); "ТОНЯ"
```

Такой оператор слово ПРИВЕТ выведет, начиная с первой позиции, а слово ТОНЯ — с сороковой.

```
PRINT TAB (40); "ПРИВЕТ"; "ТОНЯ"
```

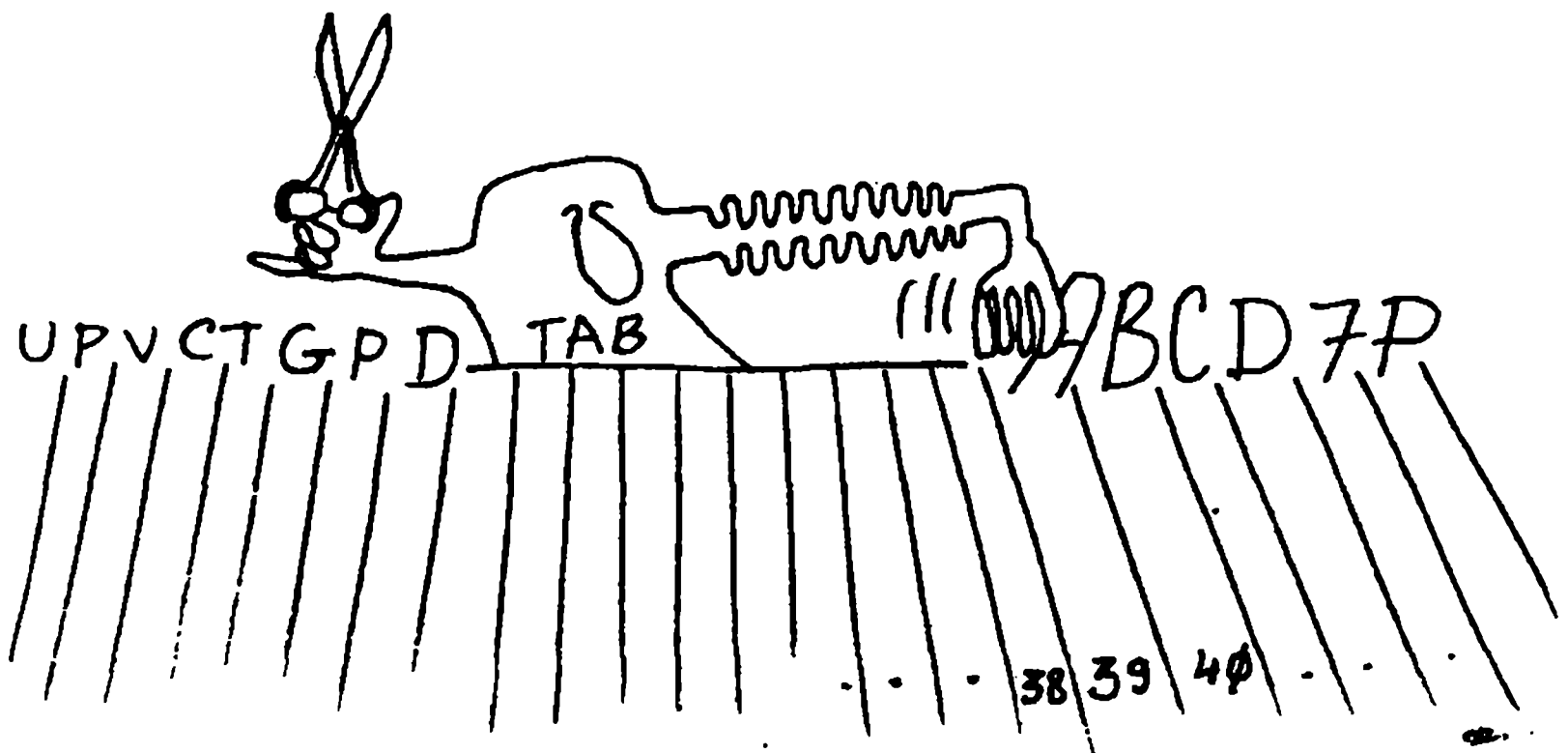
Этот оператор текст ПРИВЕТТОНЯ выведет, начиная с сороковой позиции.

В аргументе функции необязательно указывать число, там может быть любое арифметическое выражение. А сейчас скажи, что делает такой оператор?

```
PRINT TAB(5);
```

— Не знаю. По-моему, он... ничего не делает.

— Не забудь про точку с запятой в конце. Она означает, что следующий оператор PRINT начнет вывод с шестой позиции. Похожий оператор я пишу в строке 1010, чтобы



печать начиналась не с самого края бумаги, а с отступом на пять пробелов.

— Но тогда же и другие строки нужно будет печатать, начиная с шестой позиции.

— Так я и поступлю.

— А на что этот одинокий оператор PRINT после цикла?

— Чтобы напечатать целую строку одинаковых символов, в конце оператора PRINT, состоящего в цикле, ставлю точку с запятой. Влияние этой точки с запятой сохранится и после окончания цикла, так что следующий вывод после цикла продолжит декоративную строчку. Нужный перескок на следующую строку обеспечивает тот оператор PRINT после цикла, о котором ты говорила. Он как бы прекращает действие точки с запятой.

— А возвратившись из подпрограммы, натываемся на какой-то сложный расчет в строке 570.

— Здесь рассчитываем, сколько пустых строк оставить между верхней декоративной строчкой, которую только что напечатали, и текстом поздравления. Пустые строки будут еще и после текста. Их количество я рассчитываю так. Значение переменной R — общее количество строк в поздравлении — мне известно: 16. Отнимаем от него 10 — количество строк в тексте поздравления. Получаем общее количество пустых строк. А сколько из них оставить вверху над текстом? Считаю, что, половину. Потом беру целую часть частного $(R-10)/2$ просто потому, чтобы R1 получилось целым числом, поскольку в дальнейшем я эту переменную собираюсь использовать в местах, где нужны целые числа, ты увидишь.

— Где я это увижу?

ЕЩЕ ОДНА ПОДПРОГРАММА

— Например, во второй моей подпрограмме в строке 2010 я это число использую как границу цикла, которая, конечно, должна быть целым числом. Правда, интерпретатор прекрасно сам понимает, когда нужны именно целые числа, и сам берет целую часть в случае необходимости.

— И где эта твоя вторая подпрограмма? Опять функция TABI. Но оператор PRINT на сей раз куда сложнее. Хотя нет... Понять можно. Сначала ты отступаешь на пять позиций от края бумаги, потом выпечатываешь декоративный символ, потом прыгаешь на конец строки, где выпечатываешь еще один декоративный символ. Всего-то?

— А ты что еще хотела? Больше ничего и не надо. Раз вывели пустые строки, можно начинать выводить текст. Это делает цикл в строках 600—630. Сперва нужно рассчитать,

— Но есть и небольшое отличие. В том случае верхняя и нижняя строки рамки в количестве строк не учитывались: R было количеством строк между строками рамочки, а N — количеством символов в одной строке, включая и декоративные символы слева и справа.

— Вообще... вообще это не я придумал, мне Алгоритмыч рассказал,

— Еще выводятся пустые строки под текстом, потом нижняя строка рамки, и это все о выводе поздравления. Далее оператором CLOSE закрываем все файлы, чтобы действительно началась печать. Нужно учитывать ту особенность, что наша машина печатать на бумаге не начинает, пока файл на печатном устройстве открыт.

A hand-drawn diagram illustrating a complex system architecture. The diagram features several interconnected components:

- Left Side:** A cluster of rectangular blocks and diamond-shaped nodes, with arrows indicating data flow. A large, irregular shape encloses a portion of this cluster.
- Center:** A vertical line acts as a central bus or interface. A hand-drawn arrow points from a block on the right side towards this central line.
- Right Side:** Another cluster of rectangular blocks and diamond-shaped nodes, also with arrows indicating flow. A large, irregular shape encloses a portion of this cluster.
- Bottom:** A horizontal line connects various rectangular blocks, representing a common data path or storage layer.
- Character:** A cartoon character with a large head, wearing a hat and a long coat, stands on the right side of the diagram, looking towards the central structure.

— Поговорим о первой возможности — напечатать все как есть. В этом случае переходим на строку 740 и первым делом присваиваем значение 1 переменной F — номеру файла для вывода. Потом нужно открыть печатающее устройство, потому что все файлы закрыты. Помнишь строку 660?

— А почему ты в строке 720 до выполнения оператора ON присвоил 0 переменной F, чтобы потом сразу присвоить 1?

— А если оператор ON отправил бы нас на строку 290? Тогда не смогли бы выполниться последующие операторы PRINT#F, так как все файлы закрыты, кроме нулевого. Получается, что только в случае, если значение F равно нулю, все выполнится нормально и произойдет вывод на экран.

ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММЫ

Вот что еще надо сказать об этой программе. Когда она была написана, нужно было ее тестировать — убедиться, что она действительно работает со всеми входными данными. Скажи, какие имена мне надо ввести, чтобы убедиться, что пол определится, как было задумано.

— Во-первых, надо пройти весь список имен, конкретно указанных в программе, — Валера, Сережа, Слава и др. Потом... имя, не входящее в этот список, оканчивающееся на «-а», но не на «-ша». Например, Антонина. Потом, конечно, и оканчивающееся на «-ша» — Саша. И еще имя, кончающееся на «-я», — Тоня. Вроде ничего не забыла.

— Забыла. Имя, кончающееся не на «-а» и не на «-я». Например, Петр. Сколько всего вариантов получится?

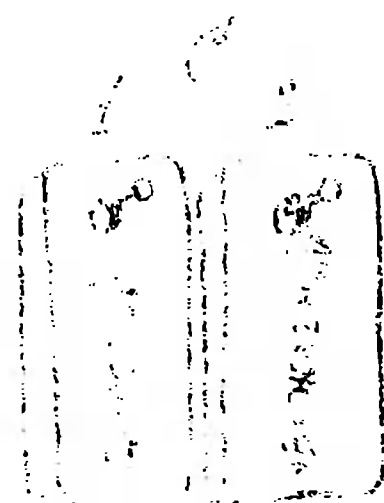
— Восемь плюс четыре. Двенадцать. Значит, для тестирования пришлось бы пропустить программу 12 раз.

— Это для тестирования определения пола поздравляемого. Но еще до ввода его имени нужно четыре раза вводить единицу или двойку. Тестируя сложную программу, пришлось бы проверить любую комбинацию из этих четырех чисел — единиц или двоек и двенадцать имен. Так что всего надо было бы пропустить $12 \cdot 2 \cdot 2 \cdot 2 = 192$ варианта.

— Ужас!

— Не волнуйся. На сей раз этого не придется делать, так как наша программа очень проста и каждый оператор ветвления достаточно проверить один раз.

— Ты, видимо, хочешь сказать, что достаточно проверить комбинации «ученик» — «Антонина» и «взрослый» — «Антонина», чтобы все считать в порядке и не проверять варианты «ученик» — «Петр» и «взрослый» — «Петр»?



— Хочу сказать больше: достаточно проверить варианты «ученик» — «Петр» и «взрослый» — «Антонина», т. е. можно объединить проверку определения пола и проверку обработки ввода «ученик» — «взрослый», другими словами, обработки значения переменной $U \boxtimes$. Это значение мало на что в программе влияет — оно участвует только в одном операторе ветвления в строке 520. Если же это значение и введенное имя участвовали бы в разнообразных вычислениях, то нужно было бы пропустить все возможные варианты — два варианта ввода переменной $V \boxtimes$ и двенадцать вариантов ввода имени.

— По-моему, о значении переменной $S \boxtimes$ можно сказать то же самое. Эта переменная отвечает за то, дорог ли нам поздравляемый или просто уважаем. И эта переменная тоже используется только один раз — там, где из массива выбирается слово «ДОРОГОЙ» или «УВАЖАЕМЫЙ» в нужном грамматическом роде. Если программа правильно выберет слово для Антонины, то и для Тони тоже выберет совершенно правильно.

— И опять, если бы это значение участвовало в каких-нибудь сложных расчетах, то пришлось бы проверить все возможные варианты.

— Очевидно, и о переменной $T \boxtimes$, которая содержит нужное обращение, можно сказать то же самое. И в еще большей степени это можно сказать о переменной $F \boxtimes$, отвечающей за вывод на экран или бумагу. Если один раз запрашивали вывод на бумагу и программа действительно на бумагу вывела, то, запрашивая то же самое в следующий раз, также получим вывод на бумагу, не станет же программа вдруг рисовать на стене.

— Можно надеяться.

— Я совсем запуталась. Сколько всего вариантов получилось?

— Всего двенадцать. Мы должны проверить обработку всех имен и дополнительных переменных. Так как проверка обработки этих дополнительных переменных весьма проста — нужно только ввести два набора чисел: 1,1,1,1 и 2,2,2,2, то ее можно объединить с проверкой имен. Ведь чтобы ввести любое имя, нам придется сначала ввести и значения дополнительных переменных.

Но, как я уже сказал, так просто получилось потому, что мы немножко схлтурили, утверждая, что входные данные не влияют друг на друга.

— Какая же тут халтура! Мы обстоятельно рассудили, что взаимного влияния нет: если один раз программа слово «уважаемый» выбрала правильно, то и в следующий раз тоже выберет правильно. Все логично.

ВОЗМОЖНЫЕ ОШИБКИ

— Все логично с правильной программой. Но программа может быть и неправильной, и притом работать неправильно только при отдельных комбинациях входных данных, а при всех остальных изображать правильно работающую программу.

— Не верю, что такие ошибки могут быть.

— Труднее всего найти ошибки, которых не может быть. Так говорил Алгоритмыч.

— Придумай тогда в этой программе такую ошибку!

— Я не говорю об этой программе, а вообще. Здесь, может быть, и не удастся придумать достаточно «естественную» ошибку, но попробовать можно. Например, для отыскания последней буквы в имени поздравляемого человека мы пишем не

$$P \text{ } \text{ } = \text{SEG } \text{ } (V \text{ } , L, L),$$

а по рассеянности вводим букву G не только в имени функции, но и в аргументах:

$$P \text{ } \text{ } = \text{SEG } \text{ } (V \text{ } , G, G)$$

Что произойдет?

— Что произойдет? Если обращаемся «дорогой», будет найдена первая буква, а если «уважаемый» — вторая.

— Вот представь: ты вводишь имя АНТОНИНА, и если хочешь к ней обратиться «дорогая», то все правильно — первая буква совпадает с последней, и наш алгоритм срабатывает правильно. Если же хочешь обратиться «уважаемая» — не тут-то было, получишь УВАЖАЕМЫЙ АНТОНИНА. С именем КАТЯ будет наоборот — она станет мужчиной, когда будет дорогой. А если такая ошибка?

$$P \text{ } \text{ } = \text{SEG } \text{ } (V \text{ } , 5, 5)$$

— Тогда в имени будет найдена пятая буква. И только с пятибуквенными именами программа сработает правильно; с другими — как бог на душу положит.

— Нужно признать, что наша система тестирования не обеспечивает обнаружения таких ошибок. Скажем, мы могли случайно для тестов взять как раз пятибуквенные слова. Или случайно пропустить вариант «дорогая Антонина», а уважаемым считать Петра. Ведь мы же договорились не перебирать все возможные комбинации входных данных. Тестирование — это сложная наука.

— Ну а если все-таки обнаружили, что программа делает не то? Как найти, где именно ошибка?





— Ой, это нелегко! Если программа вдруг начинает путать, можно подумать, что какой-то из операторов сравнения срабатывает неправильно. Можно их просмотреть и убедиться, что они записаны верно. Если так, то неверны данные, которые эти операторы используют. В нашем случае самое подозрительное данное — это последняя буква Р α , которая присутствует во многих операторах сравнения. Чтобы проверить это значение, можно поступить так. После строки 310 с нахождением Р α вставляем дополнительную строку 315 STOP и пропускаем программу еще раз с теми же входными данными, которые до того давали ошибку. Когда произойдет остановка на строке 315, можем в непосредственном режиме посмотреть значения Р α , D и др., какие захотим. Если обнаружится что-то подозрительное, то посмотрим программные строки, ответственные за это. В конце концов и обнаружим ошибку.

— По традиции мы сейчас должны были бы эту программу «проразмыслить». Но мы столько провозились, что это дело нужно пока отложить.

— «Проразмысливание» отложить можно, но «просмотрение» задач нельзя.

1. В памяти задан массив с оценками ученика по четвертям и учебным предметам. Определить, сколько пятерок, четверок, троек и, увы, двоек было у этого ученика в каждой четверти. Вывести эти сведения, а также сообщение, в каких четвертях он являлся отличником, «хорошистом» или, увы, неуспевающим.
2. Задан массив судейских оценок фигуристов. Составить программу определения, сколько судей дает первое место каждому фигуристу. Сколько судей дает второе место? третье место?

20. ТОНЯ ПРОДОЛЖАЕТ ОСВАИВАТЬ ФАЙЛОЛОГИЮ

Попав на машину в следующий раз, Тоня и Петя напечатали всем родным, близким и друзьям поздравления с днем рождения, именинами, Новым годом, государственными праздниками, началом учебного года, окончанием учебного года — на несколько лет вперед. Наконец Тоня сказала:

— Давай кончим это печатание и поговорим о программировании. Неужто компьютеры изобретены только для печатания?

ЧТО ЕЩЕ МОЖНО НАПЕЧАТАТЬ

— Конечно, нет, но красиво и наглядно напечатать тоже надо уметь. Какой толк, если машина что-то интересное получит, а программист не позаботится о том, чтобы результаты были выведены доходчиво!

— Знаешь что, в порядке «проразмысливания» последней программы скажу, что подобным образом можно было бы программировать печатание разных объявлений. Например, такого-то числа, в таком-то кабинете состоится классное собрание. Явка строго обязательна. И в конце подпись. Программа потребует ввод — где и когда; сами мы изменим значение символьной переменной, которое определяет повестку дня собрания, и вперед. Напечатаем столько, что для всей школы хватит.

— Это ты правильно говоришь. Но ты была права и в том, что компьютеры изобретены не только для печатания. Невелико достижение — что-то ввести и сразу напечатать. Конечно, и это какие-то работы облегчит. Наверно, в любом учреждении приходится печатать, например, приказы о приеме на работу, мало отличающиеся друг от друга. Или приказы об объявлении выговоров.

— Ты считаешь, что выговоры так часто объявляют?

— Понятия не имею. Если не нравится, то, пожалуйста, другой пример. Человек хочет ехать в дом отдыха и идет к врачу за справкой. Врач смотрит: да, здоров, пускай едет. И запускает свой персональный компьютер, который напечатает справку.

— У врачей же такие справки уже имеются. Надо только вписать фамилию, перенесенные болезни, анализ крови и, может быть, еще что-то. Работы не намного больше, чем с компьютером.

— А представь, если у врача на магнитном диске хранится все о здоровье Тони Соображалкиной: болела свинкой, ветрянкой, кровь просто необыкновенная. Соответствующая программа эти сведения с диска считывает и печатает. Вписывать ничего не надо, нужно только ввести в машину редкую фамилию героини справки. Без этого не обойтись.

— В таком случае другое дело.

СПИСОК УЧЕНИКОВ НА ДИСКЕ

— Хотя и нельзя считать, что только программы, работающие с диском, самые серьезные, но уметь считывать данные с диска и записывать их туда очень важно, поэтому посмотрим еще и программу, которая работает с файлом,




```

330 IF U$="#" THEN 280 \ REM КОНЧАТЬ СЧИТЫВАНИЕ?
340 PRINT N;TAB(3);U$;TAB(15);I$;TAB(25);O$;TAB(35);D$
350 GO TO 320
355 \
360 REM ЗАНЕСТИ ИЛИ ВЫЧЕРКНУТЬ УЧЕНИКА 34343434343434
370 IF R=4 THEN 400
380 PRINT "ЗАНЕСТИ ПОСЛЕ КАКОГО НОМЕРА";
390 INPUT N \ GO TO 420
400 PRINT " КАКОЙ НОМЕР ВЫЧЕРКНУТЬ"; \ INPUT N
410 N=N-1
420 OPEN F$ FOR INPUT AS FILE F1
430 OPEN "DK1:RABOT" FOR OUTPUT AS FILE F
440 FOR I=1 TO N
450 GOSUB 2000 \ \ REM СЧИТЫВАЕТ СВЕДЕНИЯ
460 IF U$<>"#" THEN 480
470 PRINT "НОМЕР СЛИШКОМ ВЕЛИК" \ GO TO 670
480 GOSUB 3000
490 NEXT I
500 IF R=3 THEN 540 \ REM УХОД НА ЗАПИСЬ
510 GOSUB 2000 \ \ REM СЧИТЫВАЕТ БЕЗ ЗАПИСИ
520 IF U$="#" THEN 470
530 GO TO 550 \ \ REM ЕЩЕ СЧИТЫВАТЬ ОСТАТОК
540 GOSUB 1000 \ \ REM ВВОД НОВЕШЬКОГО
550 GOSUB 2000 \ \ REM СВЕДЕНИЯ О СЛЕДУЮЩЕМ
560 IF U$="#" THEN 580
570 GOSUB 3000 \ GO TO 550
580 PRINT #F,U$ \ \ CLOSE \ \ REM ВСЕ СЧИТАНО
585 \
590 REM С РАБОЧЕГО ФАЙЛА ПЕРЕПИСЫВАЕТ
600 OPEN F$ FOR OUTPUT AS FILE F
610 OPEN "DK1:RABOT" FOR INPUT AS FILE F1
620 GOSUB 2000 \ \ REM СЧИТЫВАНИЕ С РАБОЧЕГО ФАЙЛА
630 IF U$="#" THEN 660
640 GOSUB 3000 \ \ REM ЗАПИСЬ В НОВЫЙ ФАЙЛ
650 GO TO 620
660 PRINT #F,U$
670 CLOSE \ KILL "DK1:RABOT" \ GO TO 140
675 \
680 \
1000 PRINT "          ФАМИЛИЯ ШКОЛЬНИКА"; \ INPUT U$
1010 IF U$<>"#" THEN 1030
1020 PRINT #F,U$ \ RETURN
1030 PRINT "          ЕГО ИМЯ"; \ INPUT I$
1040 PRINT "          ОТЧЕСТВО"; \ INPUT O$
1050 PRINT "          ДЕНЬ РОЖДЕНИЯ"
1060 PRINT "          ГОД, НАПРИМЕР      86"; \ INPUT D1$
1070 PRINT "          МЕСЯЦ, НАПРИМЕР    02"; \ INPUT D2$
1080 PRINT "          ДЕНЬ, НАПРИМЕР      01"; \ INPUT D3$

```

```

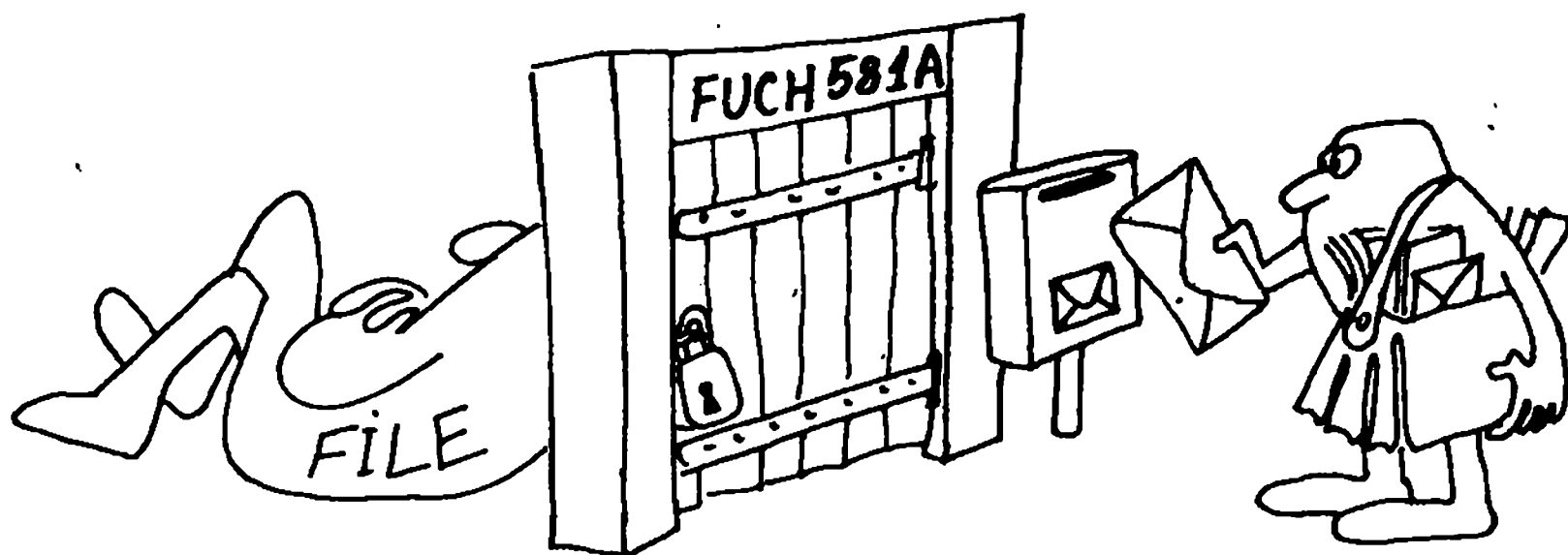
1090 D $\alpha$ =D3 $\alpha$ +"."+D2 $\alpha$ +"."+D1 $\alpha$ 
1100 IF LEN(D $\alpha$ )=8 THEN 1120
1110 PRINT " ДЕНЬ РОЖДЕНИЯ НЕПРАВИЛЬНЫЙ" \ GO TO 1050
1120 GOSUB 3000
1130 RETURN
1135 \
1140 \
2000 REM ПОДПРОГРАММА СЧИТЫВАНИЯ ДАННЫХ
2010 INPUT #F1,D $\alpha$ 
2020 IF D $\alpha$ <>"#" THEN 2040
2030 RETURN
2040 INPUT #F1,I $\alpha$ ,O $\alpha$ ,D $\alpha$ 
2050 RETURN
2055 \
2060 \
3000 REM ЗАПИСЬ ДАННЫХ НА ДИСК
3010 PRINT #F,D $\alpha$  \ PRINT #F,I $\alpha$ 
3020 PRINT #F,O $\alpha$  \ PRINT #F,D $\alpha$ 
3030 RETURN

```

Сперва, наверное, надо пояснить, как в начале работы программа образует описание файла. Пользователь вводит номер устройства, на котором он поставил диск с файлами классов. Этот номер он должен знать. Потом он должен ввести номер того класса, со списком которого он хочет работать. Показываем ему и образец, как может выглядеть этот номер,— 6A, но если он введет его в виде A6 — дело хозяйское, главное, чтобы он всегда вводил одинаково. Потом в строке 130 образуется описание файла. Так как описания файлов могут состоять лишь из букв латинского алфавита, мы должны строго предупредить пользователя использовать только эти буквы.

— Что это такое — FUCH?

— Только что об этом хотел говорить. Это сокращение слов «файлы учеников». По этим буквам можно будет в списках файлов уверенно определить, какие из них содержат све-



Файл находят по имени

дения об учениках. Если пользователь укажет, что диск находится на первом устройстве и речь идет о классе 581А, то получится такое описание файла:

DK1:FUCH581A

— Ха-ха-ха! Класса 581 в школе быть не может!

— Ты же должна понимать, что программа не может иметь ни малейшего представления о классах. Только пользователь может придавать этому классу 581А какой-либо смысл, например: это 1А класс 58-й средней школы.

Посмотрим дальше, как образуется список. Делается это в подпрограмме, начинающейся со строки 1000.

— Смешно. Вводишь фамилию и сразу сравниваешь с решеткой «#». У кого может быть такая фамилия?

— Этот знак, введенный вместо фамилии, означает конец ввода, или, что то же самое, конец списка учеников. Его мы тоже записываем на диск, чтобы потом, получив при считывании с диска такой знак вместо фамилии, знать, что файл на диске кончился. По этой причине всегда при вводе фамилии или считывании ее с диска мы сразу проверяем, не равна ли она символу «#». Ввод остальных данных попроще. Год рождения или, точнее, его последние две цифры. Месяц рождения. День. Потом эти данные собираем в таком виде, как они будут храниться на диске. Интересно, что эта подпрограмма сама вызывает другую подпрограмму для записи данных на диск. Это допускается.

РАБОТА СО СПИСКОМ

Сейчас о другой возможности — посмотреть список. Это делается, чтобы узнать, что в этом файле хранится. Просто показываются фамилии и другие данные. Никакая обработка не производится. Поэтому и показ программируется просто. Можешь сама посмотреть.

— Подожди, не спеши. Посмотрим подпрограмму в строках 2000—2050. Она сначала считывает фамилию и, если она совпадает с символом «#», кончает свою работу. Все ясно.

— Теперь о дополнении списка и вычеркивании из него. Данные о новом ученике вставить где-нибудь в середине файла нельзя. Для этого пришлось бы сдвигать данные об учениках, находящихся в списке после этого нового ученика. При записи на диск это не проходит.

— Еще больше это не проходит при записи на бумагу. А вот если можно было бы слова, написанные на бумаге, передвигать то влево, то вправо, то вверх, то вниз, вот бы облегчилась жизнь ученика!

— Я неточно выразился. Передвигать данные на диске нельзя технически, а средствами программирования это сделать можно: переписываем файл в другое место и во время переписки добавляем данные о новом ученике. Переписывать же на диске можно, потому что на нем очень легко старую запись стереть и на том же месте записать что-то новое, и так хоть двести раз подряд. Между прочим, Алгоритмыч как-то сказал, что файлы можно и так организовать, что удастся вставлять добавки, не переписывая весь файл, но это уже заметно сложнее...

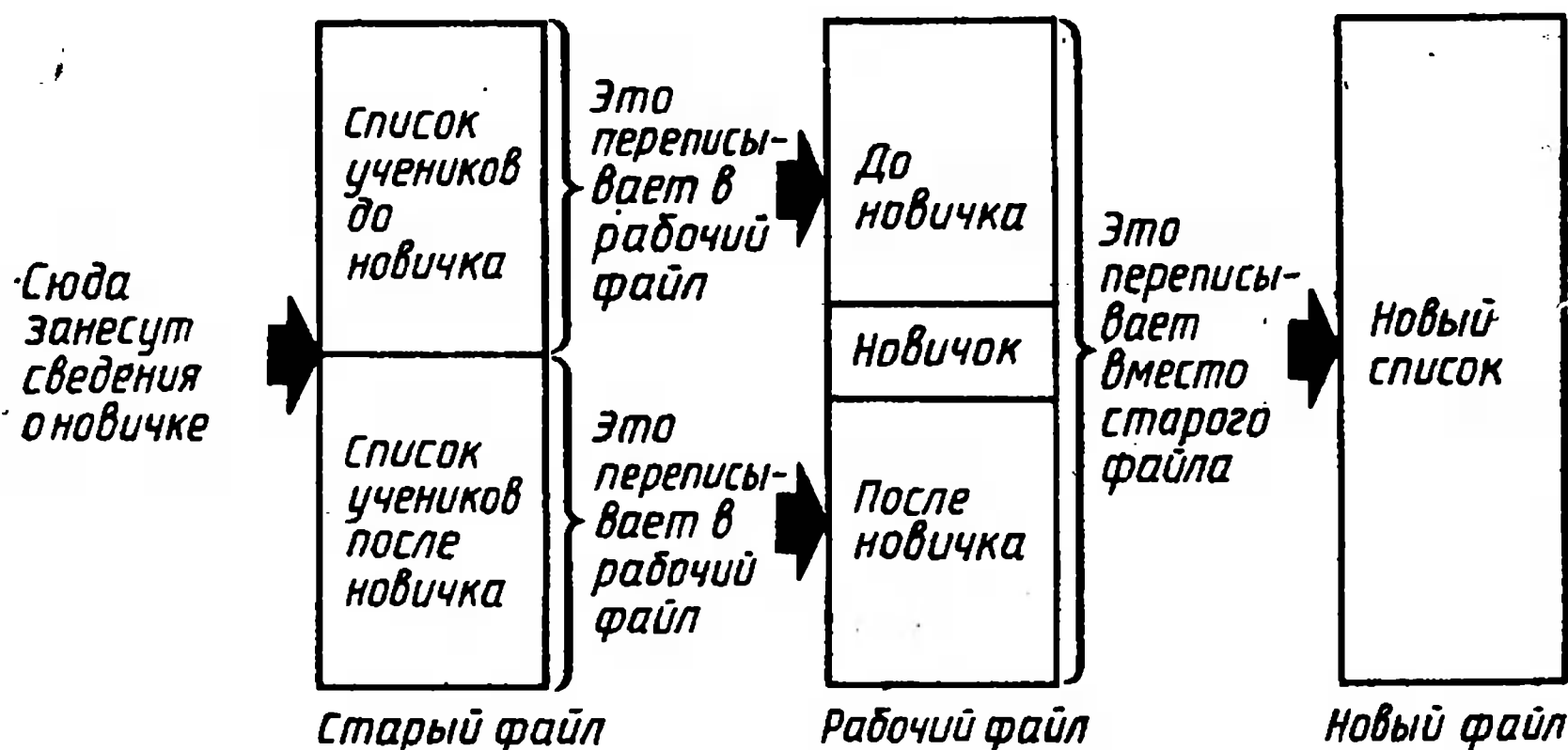
— И чтобы это понять, нужно кончить физмат? Не так ли?

КАК ДОПОЛНИТЬ СПИСОК

— Да, именно так он и сказал! Но вернемся к программе. Вставляем новые данные в файл таким образом. Сначала считываем файл до того места, где в список будем заносить данные о новом ученике. И не только считываем, но и переписываем в новый файл с описанием "DK1:РАБОТ". Потом в этот рабочий файл записываем данные о новом ученике при помощи уже разобранной подпрограммы в строках 1000—1130. И наконец, считываем остаток старого файла, снова с записью в рабочий файл. Когда все сделано, данные из рабочего файла переписываем обратно в старый файл и уничтожаем рабочий файл как ненужный.

— Списываем, записываем, переписываем, переписываем, подписываем, надписываем, выписываем. Как-нибудь по-человечески это изложить нельзя?

— Не знаю. Думаю, что нельзя. Можно только нарисовать схему.



— Сейчас стало понятнее.

— Почти так же вычеркиваем ученика из списка. Только в этом случае данные об этом ученике со старого файла списываем, а в рабочий файл не записываем. И их там не будет. Потом сама посмотри, как это в программе делается. Обязательно посмотри, так как это очень важная программа.

— А что тут такого важного? Обыкновенная программа, и совсем не такая интересная! Ну, записывает на диск. Ну и что?

— А потом же данные можно считать! И с ними сделать все, что хочешь! Ты, конечно, спросишь: что можно сделать со списком учеников класса? Правильно. Ничего грандиозного ты с файлом учеников не сделаешь. Но, научившись работать с файлом учеников, ты сможешь на диске создавать файлы с любой информацией на свете и работать с ними.

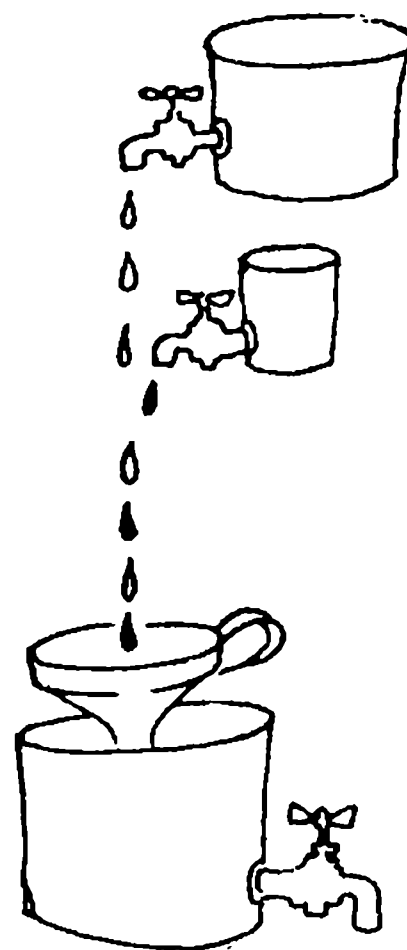
— Какой бы файл еще создать?.. Может быть, учителей?

— Ты, Тоня, лучше думай о том, что нужно сделать, а не о том, что можно было бы сделать. А чтобы это придумать, надо знать и понимать, что с файлами на диске вообще можно делать и, главное, как.

— Но ты же сам говорил, что с файлом учеников ничего особенного сделать нельзя.

— Запомни великую истину, чтобы потом не расстраиваться. Ничего особенного компьютер для тебя не сделает! Но если он сделает вместо тебя что-нибудь не особенное, то у тебя останется больше времени сделать что-то особенное.

— Ну хорошо, и что же мы смогли бы сделать простое?



ДЛЯ ЧЕГО НУЖЕН ТАКОЙ СПИСОК

— Во-первых, напечатать списки учеников. Они бывают нужны. Например, едем на экскурсию, надо брать с собой список. Кто-то ведь должен будет его написать. Пусть печатает машина! И для спортивных соревнований, врачебных осмотров, экранов успеваемости — всюду бывают нужны списки учеников.

— Да... На многих экранах успеваемости приходилось в жизни побывать. Когда я была пионеркой, у нас проводились различные соревнования между звеньями. Можно было бы напечатать списки звеньев, но мы на диске не запоминали номер звена.

— Но никакого запрета не было. Пожалуйста, запоминай. Сделай в программе ввод номера звена: ввод значения переменной Z или же Z α . Изменятся подпрограммы: в них появится новая переменная. Кое-что придется изменить и в основной программе.

— Как добавить ввод номера звена, я как-нибудь и сама догадалась бы. Я ввод и еще двадцати переменных могла бы добавить. Я только не знаю, какие нужны.

— Это зависит от твоих целей. От того, что тебе с учениками надо сделать. Можешь записать адреса, сведения о родителях. Все равно такие данные школа требует.

— Адреса. . . Адреса, к слову сказать, имеют составные части. Город, улица, номер дома и квартиры. . .

— Еще почтовый индекс.

— Да, я тоже его вспомнила. По-видимому, пользователь должен эти составные части вводить по отдельности.

ВВОД ХАРАКТЕРИСТИК УЧЕНИКОВ

— Я тоже так думаю. Посложнее было бы, если бы ты захотела вводить сведения, состоящие из неопределенного количества строк, например характеристику ученика. Тогда можно было бы поступить так. Сначала узнать у пользователя, сколько строк он собирается вводить. А потом написать цикл по строкам. Например, такой:

```
1500 PRINT "КОЛИЧЕСТВО СТРОК В ХАРАКТЕРИСТИКЕ"\
      INPUT N 2
1510 PRINT # F, N 2
1520 FOR I=1 TO N 2
1530 PRINT "ВВЕДИТЕ "; I; " СТРОКУ"\INPUT R 2
1540 PRINT # F, R 2
1550 NEXT I
1560 RETURN
```

Так можно было бы дополнить подпрограмму, начинающуюся в строке 1000. Возможен и другой вариант. Ты требуешь, чтобы в конце характеристики пользователь ввел что-нибудь такое, чего в характеристике заведомо не будет, например комбинацию символов «//» или «# # #» и т. п. В этом случае дополнение к подпрограмме могло бы иметь вид:

```
1500 PRINT "ВВОДИТЕ ХАРАКТЕРИСТИКУ"
1510 PRINT "КОНЕЦ ВВОДА //"
1520 INPUT R 2
1530 PRINT # F, R 2
1540 IF R 2 <> "/" THEN 1520
1550 RETURN
```

Вот так и программируют ввод заранее неизвестного количества строк. Ты заметила, что в первом варианте я количество строк записал на диск?

— Заметила, заметила. Заметила и то, что ты косые черты записал на диск.

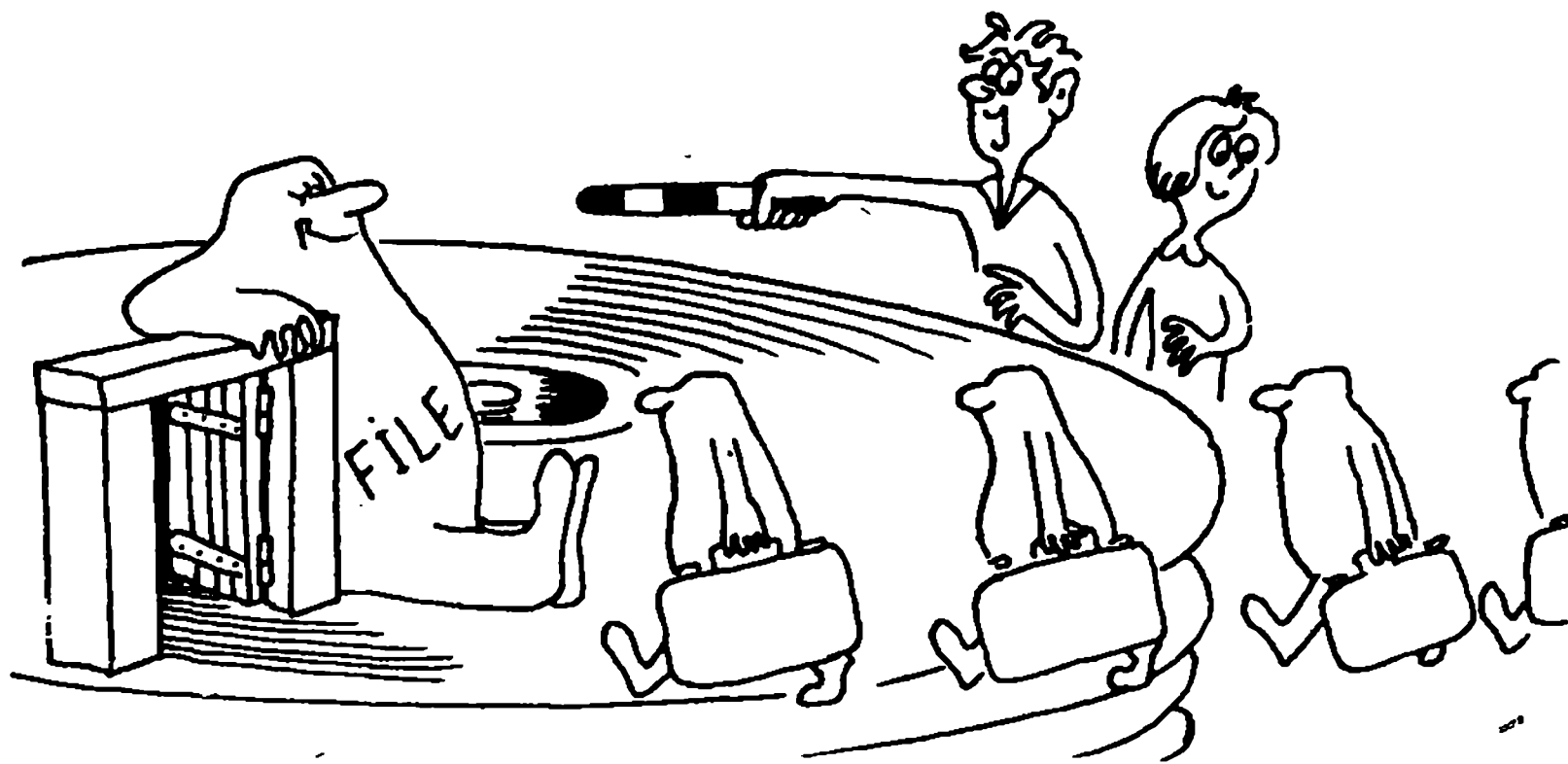
— Это для того, чтобы при считывании с диска понять, где кончается характеристика одного ученика и начинается фамилия другого ученика. Ведь на диск записываем только значения переменных, а смысл этих переменных там никак не отражается. Давай изобразим схематически содержание части файла со сведениями о каждом ученике.

Сведения об ученике						
Фами- лия	Имя	Отчест- во	Дата рож- дения	Первая строка ха- рактерис- тики	Вторая строка ха- рактерис- тики	...//

Как видишь, в конце характеристики я написал «//», а в конце файла, если помнишь, ставился символ «#». Выбор символов — дело вкуса, лишь бы не было путаницы.

— При считывании?

— Так точно! Что ты на диске записала, то и надо будет считывать. Работая с файлом, нужно хорошо знать его структуру. Из чего он состоит, из каких единиц записи. Например, файл со сведениями об учениках FUCH6A состоит из фамилии первого ученика, его имени... и т. д. Все эти сведения образуют четыре строки символов, ничего не означающих для машины, но для нас имеющих смысл фамилии, имени, отчества и даты рождения. Так вот этот набор символьных строк со сведениями об одном ученике, эти четыре строки,



Файл состоит из
единиц записи

вместе взятые, и образуют единицу записи файла. Фактически работа ведется именно с этими единицами записи. Подпрограмма в строках 1000—1130 такую единицу записывает на диск, подпрограмма в строках 2000—2050 ее считывает. Если захочешь составить программу для печати, например, списка учеников, все равно нужно будет запрограммировать считывание целых единиц записи, хотя дату рождения и не собираешься в программе использовать. То, что ты будешь в программе использовать только фамилии и имена, ничего не означает. Изволь считывать всю единицу записи целиком.

ДЕНЬ РОЖДЕНИЯ ДРУГА ЗАБЫТЬ НЕЛЬЗЯ!

Еще мне хочется показать программу, которая проверяет, сколько дней осталось до дня рождения кого-нибудь из одноклассников. Конечно, она использует файл с данными.

```
10 REM ПРОГРАММА ОПРЕДЕЛЯЕТ БЛИЖАЙШИЙ ДЕНЬ РОЖДЕНИЯ
20 PRINT "СЕГОДНЯШНЕЕ ЧИСЛО ДД-ММ-ГГ" \ INPUT D1X
30 N=5 \ REM СКОЛЬКО ДНЕЙ ДО ДНЯ РОЖДЕНИЯ
40 PRINT "НОМЕР ДИСКА С ФАЙЛОМ СПИСКА"
50 INPUT DX
60 PRINT "ВВЕДИТЕ КЛАСС, НАПРИМЕР: 6А"
65 PRINT "БУКВА LATINSKAJA !!!" \ INPUT KX
70 OPEN "DK"+DX+" ":"FUCH"+KX FOR INPUT AS FILE 1
80 SX=D1X \ GOSUB 2000 \ REM ОПРЕДЕЛЯЕТ КОЛ. ДНЕЙ
90 D1=D \ REM И СОХРАНЯЕТ
100 Z=0 \ REM ПОКАЗАТЕЛЬ, НАЙДЕН ЛИ ДЕНЬ РОЖДЕНИЯ
110 GOSUB 1000 \ REM СЧИТЫВАЕТ ЕДИНИЦУ ЗАПИСИ
120 IF UX<>"#" THEN 160
130 IF Z<>0 THEN 150
140 PRINT " В БЛИЖАЙШЕЕ ВРЕМЯ ДНЕЙ РОЖДЕНИЯ НЕТ"
150 CLOSE \ STOP
160 SX=SEGX(DX,1,5)+SEGX(D1X,6,8)
170 GOSUB 2000
175 IF D-D1<0 THEN 110
180 IF D-D1>=N THEN 110
190 Z=1 \ REM СИГНАЛ, ЧТО ДЕНЬ РОЖДЕНИЯ НАЙДЕН
200 PRINT IX;" РОДИЛСЯ ";SEGX(DX,1,5)
210 GO TO 110
220 STOP
1000 REM СЧИТЫВАЕТ СВЕДЕНИЯ О ШКОЛЬНИКЕ
```

```

1010 INPUT #1, U$
1020 IF U$ > "#" THEN 1040
1030 RETURN
1040 INPUT #1, I$, O$, D$
1050 RETURN
2000 REM ОПРЕДЕЛЯЕТ КОЛИЧЕСТВО ДНЕЙ С НАЧАЛА ВЕКА
2010 DATA 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334
2020 DIM A(12)
2030 RESTORE
2040 FOR I=1 TO 12 \ READ A(I) \ NEXT I
2050 DEF FND(I)=VAL(SEG$(S$, 3*I-2, 3*I-1))
2060 G=FND(3)
2070 D=G*365+INT(G/4)+A(FND(2))+FND(1)
2080 IF INT(G/4) <> G/4 THEN 2110
2090 IF FND(2) > 2 THEN 2110
2100 D=D-1
2110 RETURN

```

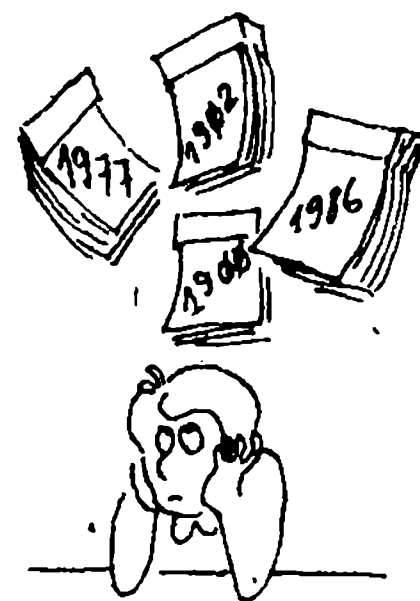
Все операторы этой программы тебе хорошо известны, и ты можешь ее самостоятельно изучить. Единственное, может быть, не поймешь подпрограмму в строках 2000—2110, где рассчитывается, сколько дней прошло с 1 января 1900 года.

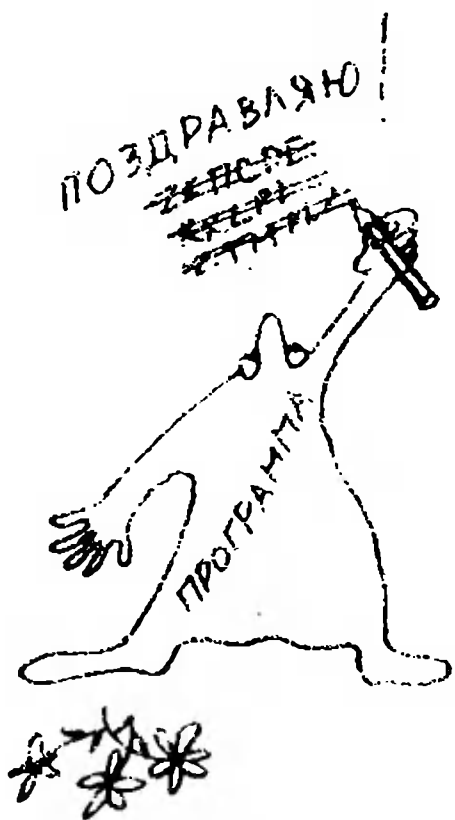
— С 1 января 1900 года! Кому нужна такая старина?

— Наш теперешний календарь довольно путаный. Месяцы длиной то 30, то 31 день плюс еще этот февраль и високосные годы.

— Но тут же ничего нельзя сделать — сам когда-то так выразился.

— Что-нибудь уж можно было бы, но ведь не сделают. Поэтому я должен иметь подпрограмму, рассчитывающую, сколько дней прошло с 1 января 1900 года до сегодняшнего дня и сколько до дня рождения одноклассника, и нахожу разность. Получится, сколько дней осталось. Почему именно с 1 января 1900 года? Думаю, что эта дата совсем необязательна, но так уж было в той программе, которую мне дали для образца. Понять, как же именно рассчитывается количество прошедших дней, можно, если знать, что каждый четвертый год високосный и числа, которые считываются оператором READ в массив A, выражают количество дней, прошедших с начала года до первого числа соответствующего месяца. Вот и все об этой подпрограмме. А вообще говоря, мы можем эту программу, которая считывает сведения с файла, объединить с той, которая печатает поздравление, и автоматически его напечатать, если день рождения уже близок. Правда, в этом случае единица записи файла должна быть другой и содержать сведения о поздравляемом, нужные для печати,—





«взрослый» или «ученик», «уважаемый» или «дорогой» и т. д.

— Давай пофантазируем, что машина печатает поздравления не здесь — в машинном зале, а подключается к персональному компьютеру в квартире поздравляемого и печатает там. И даже не печатает, а записывает на диск, что Петя Бейсиков или кто-то другой поздравил. А у того счастливого имеется программа, которая проверяет, кто поздравил, и посылает ответные поздравления компьютеру, скажем, Пети Бейсикова. Это будет прекрасная жизнь! Никого не надо будет самому поздравлять. Всю дружбу возьмут на себя компьютеры.

— Не хочешь, не делай такие программы поздравлений. Кто тебя заставляет? А представь себе, что на диске хранятся не фамилии, а список работ, которые нужно сделать. И сроки выполнения тоже хранятся там. Тогда машина может печатать не поздравление, а напоминание, что должно быть сделано через столько-то дней. И получится программа контроля выполнения работ. Таких программ уже очень много. Например, я в газете читал о такой программе как о чуде современной техники, и притом чрезвычайно полезном чуде.

— Если говорить о работах на заводах или где-нибудь еще, то количество оставшихся до завершения работы дней мало о чем говорит. Важно, сколько из них рабочих и сколько выходных. А об этом в твоей программе ни гу-гу.

— Рассчитать количество рабочих дней будет потруднее, но ничего невыполнимого нет. То же самое количество дней, прошедших с начала века, позволяет определить день недели.

— Неужели?!

— Конечно. Надо только знать, что первый день века был понедельником. Итак, если остаток от деления количества прошедших дней века на семь будет единицей, то сегодня понедельник, если двойка, то вторник...

— Великолепно! А государственные праздники?

— Их придется разместить в список. Ну, в массив занесешь количество дней с начала года до праздника. Конечно, придется подумать.

— Все это интересно. Но о работе с файлами хотелось бы услышать что-то поконкретнее. Из всего, о чем говорилось, нельзя сделать вывод, что эти файлы действительно важны для реальных задач.

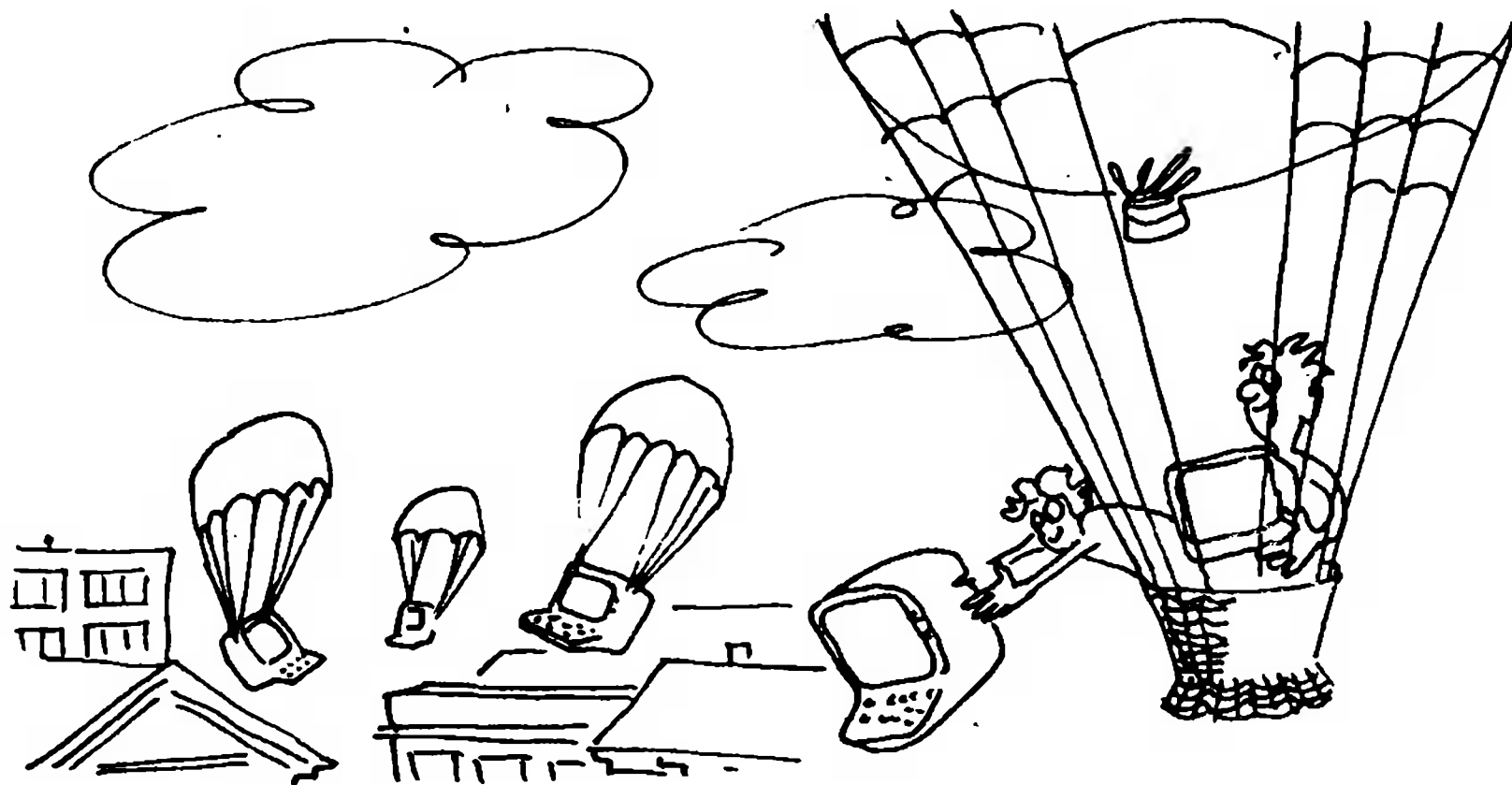
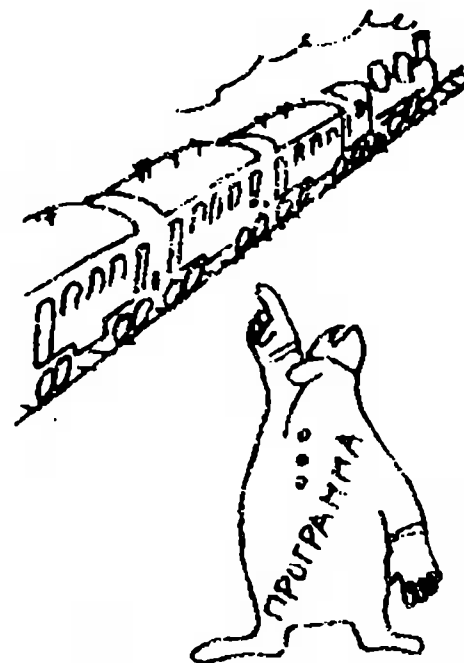
— Неужели! После стольких разговоров?! Сделаем так. Я до следующего раза обдумаю, где еще нужны файлы, и подготовлю задачи по программированию, оставшиеся в наследство от Алгоритмыча.

— Задачи? Может, не надо? Столько уж их было.

— Ну, тогда я их тебе принесу когда-нибудь в другой раз. Но принесу обязательно. Надо хотя бы подумать о том, как их решать.

21. НЕБОЛЬШАЯ ЭКСКУРСИЯ ПО УЛИЦЕ СУВОРОВА

- Тоня, отправимся на экскурсию по улице Суворова.
- Чего это вдруг? А программирование?
- Нет, нет. Никуда ехать не надо. Представим себе, что идем по этой улице, смотрим направо и налево и обсуждаем, где можно использовать ЭВМ и как это было бы лучше делать.
- Это другое дело. Начнем с привокзальной площади.
- На привокзальной площади, каждому ясно, находится вокзал. И там имеется много возможностей использования компьютеров. Например, ты хочешь поехать во Владивосток, но не знаешь, каким поездом, с какого перрона, сколько стоит билет, сколько времени ехать, нужна ли пересадка и т. п. Но, к счастью, в помещениях вокзала ты замечаешь дисплей, на экране которого надпись: ВВЕДИТЕ, В КАКОЙ ГОРОД ХОТИТЕ ЕХАТЬ. Ты сразу хватаешься за клавиатуру и мигом узнаешь все, что нужно. Можешь ли ты после всех своих занятий рассказать, как сделана такая программа информации?
- Там должен быть массив с названиями всех железнодорожных остановок. . .
- Правильно. И еще?
- Еще массив с. . . Нет. Несколько массивов. Много массивов. С названиями и номерами поездов, например Рига — Москва 32, с номерами перронов, с ценами на билеты, со временем отправления и всем остальным. Пользователь вводит название города; программа проверяет, есть ли такой город в массиве. Если есть, то из всех массивов выбирает нужную информацию.
- Так это могло бы выглядеть. На мой взгляд, такая про-



грамма могла бы обойтись без файлов: всю нужную информацию о нескольких сотнях городов можно хранить в самой программе — в операторах DATA и, как всегда, в начале работы считать эти данные в массивы. Файлы, наверное, понадобились бы, если бы пришлось выдавать кассирам сведения о свободных местах в вагонах. Тогда бы мы образовали файлы с местами каждого поезда.

— Файл с местами... Не понимаю.

— Я это представляю так. Единица ввода такого файла охватывает один вагон, состоит из номера вагона, типа (например, нуль — общий вагон, единица — плацкартный и т. д.), еще чего-нибудь и в самом конце содержит числа, описывающие места (нуль, если место свободное, единица, если занято). Таких чисел должно быть столько же, сколько мест. Например, в купейном вагоне 36 мест. Такая программа была бы не очень простой. Пришлось бы подумать о том, как программировать поиск свободного места; может быть, было бы хорошо где-то отметить, что в вагоне уже нет свободных мест. Еще можно было бы в единице записи указать номер первого свободного места, чтобы не искать по всему вагону.

— Искать означает сравнивать эти числа с нулем; если нуль, то свободно, ты сказал...

— Ну да. Ты что, не понимаешь?

— Ничего, ничего. Я только громко размышляю. Я еще вот о чем подумала: было бы здорово, если бы на экране показывали план вагона с указанием, какие места заняты и какие свободны, и ты мог бы указать, какое место желаешь.

— Это лишнее, и так ясно, что все хотят нижние места. Тогда уж лучше показывать карту СССР с маршрутом движения пассажира, с указанием всех необходимых пересадок и другой полезной информации. Но продолжим нашу экскурсию. Рядом с вокзалом почта. Здесь можно предложить информационную программу для облегчения подписки на журналы и газеты. Человек приходит на почту, подходит к дисплею, и ему показывают, на какие журналы по садоводству или программированию он может подписаться и сколько это будет стоить.

— Ты что-то не то говоришь. Как ты сможешь определить, о чем журнал? Показ названий ничего не даст, ведь не всегда можно по названию определить, о чем журнал. Скажем, «Огонек». Если кто-то его не знает, то может подумать, что это журнал для пожарных.

— На названия нечего полагаться. Надо на диск записать сведения о журнале, предусматривая в единице записи файла хранение такой информации: название, цена одного номера, количество номеров в год, о чем этот журнал. Скажем, строка символов МО-ОП-МУ-ДЕ-КР могла бы означать: молодежный, общественно-политический, бывают статьи о музыке, бывают

детективы, бывают кроссворды. Так можно было бы коротко охарактеризовать журнал «Смена». Такие строки мы бы записывали на диск и считывали бы с него, но при показе на экран нужно было бы их расшифровать, чтобы было понятно. Самым трудным при составлении такой программы было бы достаточно полно описать содержание журналов этими строками символов.

— Ладно. Переходим улицу. Там магазины. Гастроном, «Напитки», «Хозтовары», дальше «Ткани».

— С точки зрения программирования все магазины почти одинаковы. В файле хранятся сведения о наличии товаров. Получили товар — отмечают в файле. Продали — опять отмечают. Так всегда можно знать, какие товары имеются в магазине, что лучше всего покупают, сколько сегодня чего продано, на сколько дней каких товаров хватит.

— Ух, и идеализируешь ты сейчас. Там уж так и хотят знать, на сколько дней хватит товара.

— Конечно, идеализирую. Но это ничего. Можно еще больше идеализировать. Допустим, в магазине хозтоваров имеется доступный покупателям компьютер с программой информации о товарах. Покупатель заходит в магазин, подходит к дисплею, и ему показывают сведения о всех слесарных инструментах, какие можно купить в магазине, или о красках, или о гвоздях, или о холодильниках, а также для чего используются, сколько стоят, сколько еще осталось, когда будут в продаже опять. Программа получилась бы очень похожа на ту, которую мы придумывали для журналов.

— Идем дальше. Мастерская, «Оптика», фотоателье...

— Все мастерские для нас программистов чрезвычайно похожи. Для них можно было бы составить программу контроля сроков исполнения заказов, очень похожую на ту, которую я тебе показывал. Ну ту, которая рассчитывала, сколько дней осталось до дня рождения одноклассника. Здесь же она напоминала бы, сколько дней остается до срока выполнения каких-то заказов — один, два или десять. Понятно, программа могла бы только напоминать, сама она ничего сделать не смогла бы. В разных мастерских компьютер мог бы еще и учитывать расход и наличие материалов — стекол там или фотопленки.

— Теперь гостиница, клуб железнодорожников.

— Гостиницы и клубы, с нашей точки зрения, похожи на вокзал. На вокзале были поезда и вагоны, в гостиницах имеются номера, в клубах проводятся концерты. Во всех этих случаях резервируются места; нужно знать, сколько из них занято, сколько свободно. Вот тут-то показ плана мест на экране был бы полезен: человек мог бы из всех свободных мест выбрать наиболее ему подходящее.

— Дальше ничего интересного нет. Магазины, магазины,

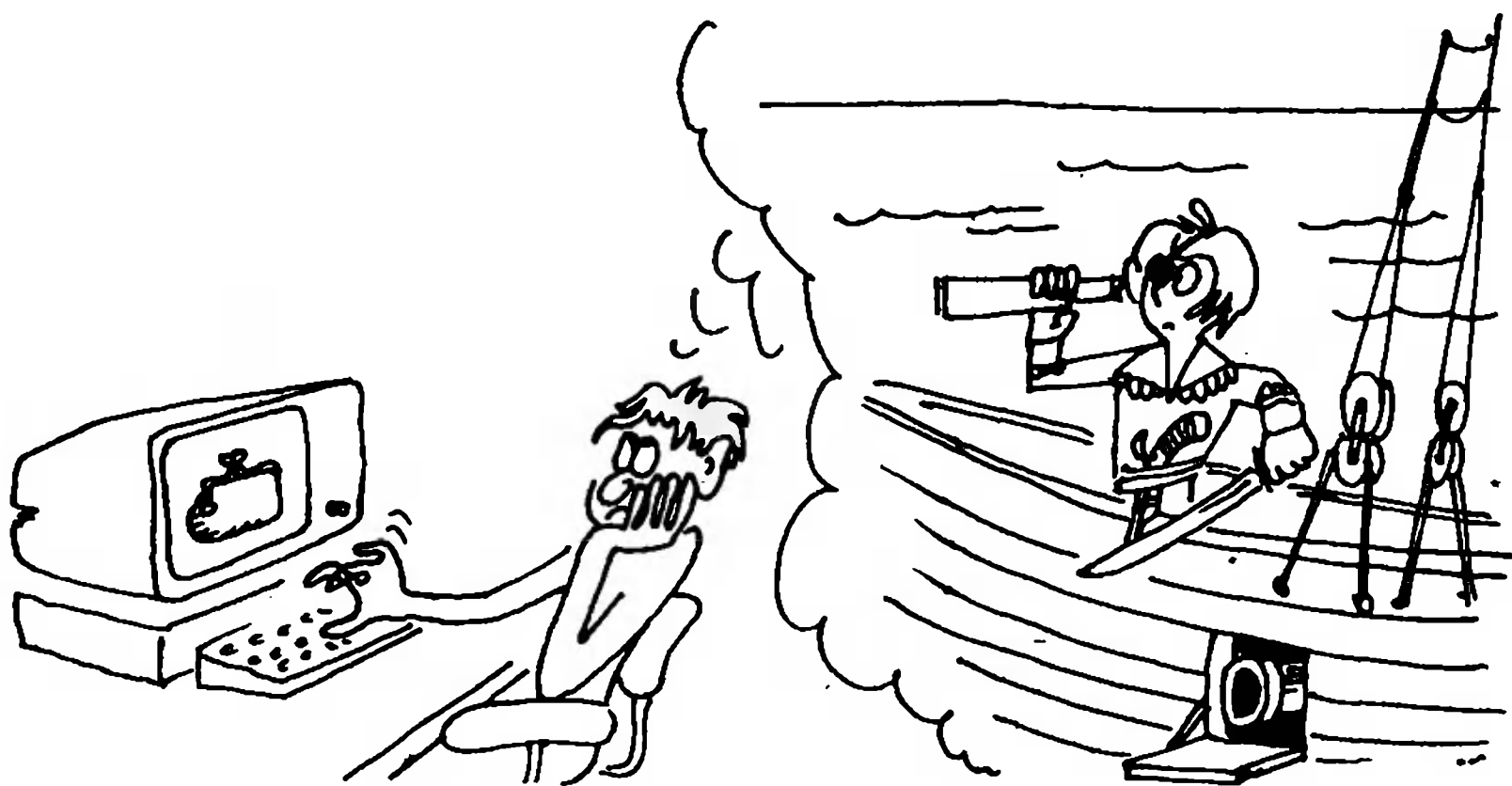
магазины. . . А ты их всех отправляешь в один мешок — мешок информации и учета.

— Разумеется, в каждом месте учет и информация имеют свои особенности. Я о них много знать не могу, потому что меня никуда пускать не станут и рассказывать о своей работе тоже. А эти особенности могут очень сильно повлиять на составление наших программ. Нас учили, что программист должен хорошо знать не только программирование, но и то, что он программирует. Если уж он, например, взялся за программу для гостиницы, то работу гостиниц он должен знать превосходно.

— Там дальше еще виден кинотеатр, но ясно, что ты не усмотришь никакой разницы между ним и вокзалом. Так что смело можешь объявить: «Уважаемые гости нашего города, экскурсия закончена. До свидания!»

22. НА АБОРДАЖ «КОРВЕТОВ»!

Алгоритмыч не забыл про Тоню и ее занятия программированием. И когда, очередной раз поговорив с Петей, узнал, как далеко продвинулась Тоня в компьютерных науках, сказал, что пора ей начать работать самостоятельно и что как раз сейчас появилась такая возможность и необходимость. Оказалось, что в следующем учебном году школа получит класс учебных компьютеров «Корвет», и, следовательно, для их освоения нужны будут знающие люди. Так что надо Тоне отправиться в лабораторию информатики университета, где такие «Корветы» уже есть, и выучить тот вариант Бейсика, который



имеется на этих машинах. И не просто выучить кое-как, а освоить так, чтобы уметь программировать не только расчеты по формулам или обработку символьной информации, но и рисование на экране, чтобы потом можно было сделать интересную игровую программу про каких-нибудь молодцов, которые на корветах отправляются искать сокровища на острове, а пираты им мешают, или что-нибудь другое, но обязательно морское, раз уж речь идет о «Корветах». Конечно, он, Алгоритмыч, обо всем в университете договорится.

Честно говоря, Петя расстроился. Во-первых, распалась их мини-компания, а во-вторых, ему самому тоже было бы интересно освоить новую технику. Но, увы, об этом не могло быть и речи, так как он должен был завершать работу над своей программой, которая предназначалась для расчета зарплаты учителей в зависимости от их нагрузки — количества проводимых ими уроков в неделю. К сожалению, Петя бесспорно отставал от намеченного плана из-за дополнительной педагогической работы по обучению Тони.

Так что в университет Тоня отправилась одна.

ЭТО И ЕСТЬ «КОРВЕТ»

В университете Тоне надо было найти лабораторию информатики и в ней студента Мишу Корветкина.

— Здравствуйте, вы будете Миша Корветкин?

— Михаил Иванович.

— Простите...

— А ты, наверно, будешь Тоня Соображалкина?

— Да, Михаил Иванович.

— И уже знаешь Бейсик?

— Да, Михаил Иванович.

— И какой, если можно полюбопытствовать?

— Как какой?

— Вот так я и думал! Присылают тут всяких недоучек, которые не знают даже, что изучали Бейсик RSX. Запомнила — эр-эс-икс!

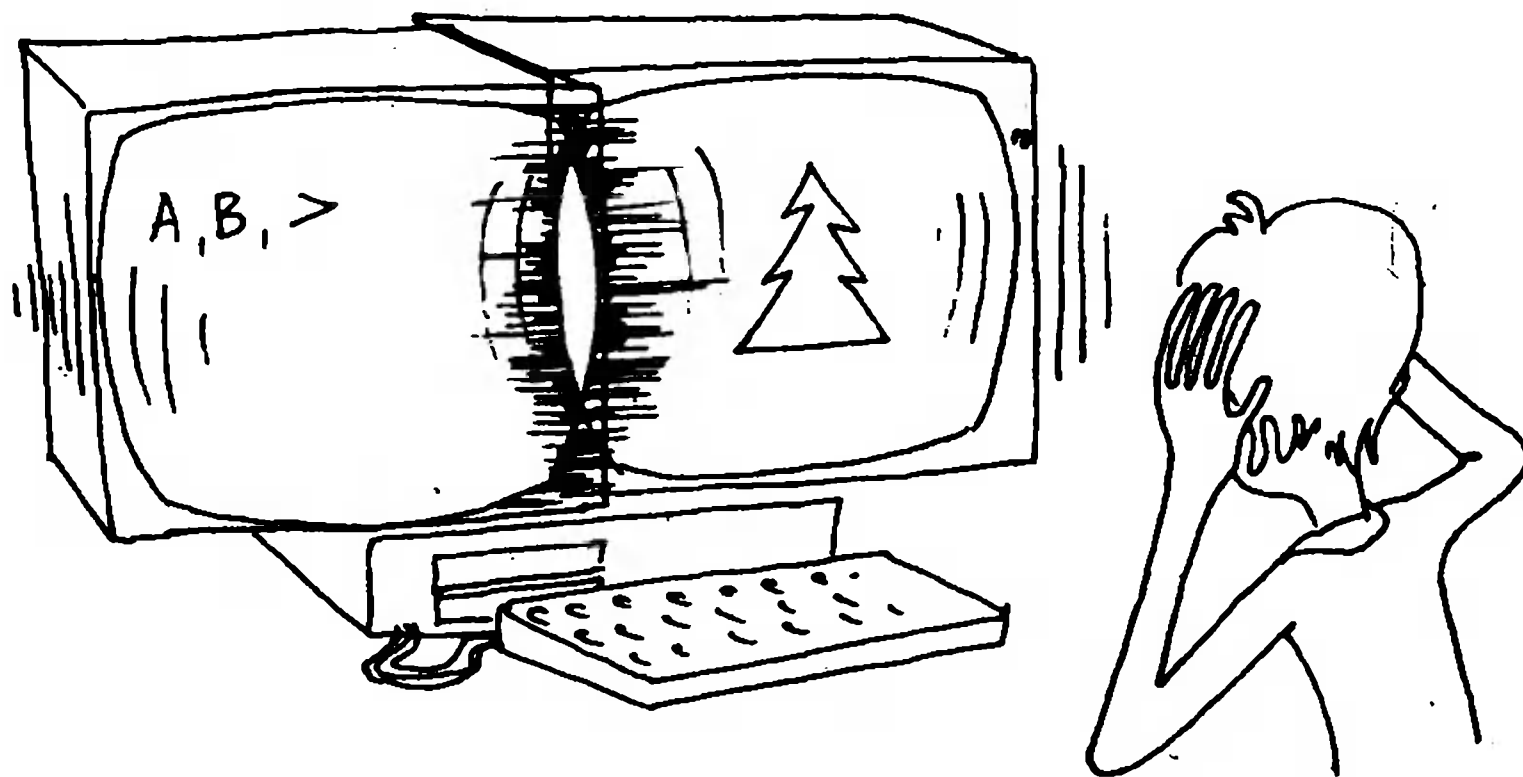
— Запомнила.

— А сейчас будешь изучать Бейсик MSX.

Этот пижон совершенно не понравился Тоне, но она решила временно его потерпеть, потому что боялась как-то подвести Алгоритмыча, ответив этому студентушке так, как он этого заслуживал.

А он продолжал:

— Ну, пошли, покажу я тебе «Корвет». «Корветов» у нас достаточно — целый учебный класс, так что хватит и для



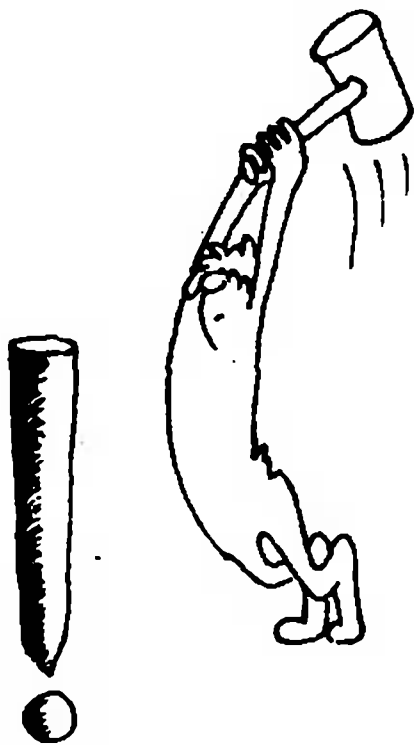
тебя. Но сразу скажу: работать сможешь только на ученической машине, есть еще и учительская, но на ней тебе работать пока не дадим.

Ну, вот она, родненькая. Состоит из системного блока с клавиатурой, из дисплея, который фактически является обыкновенным телевизором, притом, увы, черно-белым, а если бы нам прислали цветные телевизоры, все выглядело бы намного лучше. Но хватит жаловаться. Надо понять, что с точки зрения программиста наш дисплей фактически представляет из себя два дисплея: один символьный, другой графический.

— Будьте любезны объяснить наличие двух дисплеев подробнее, лично я вижу только один.

— Физически это один дисплей, а логически два. Как две программы телевизора, которые можно смотреть одновременно. Каждым из этих двух дисплеев можно управлять отдельно, т. е. изображение на них создается отдельно, хотя и все будет видно одновременно, если мы не позаботимся и один из дисплеев не почистим. Мы отдельно рисуем рисунки, отдельно выводим символы, и, какие бы символы мы ни выводили, наш рисунок останется без изменений. Считается, что даже имеем два курсора: символьный, который, как обычно, показывает место, куда будут выводиться символы при нажатии клавиш, и графический, которого; правда, не видно, но он все равно есть. Можно считать, что графический курсор находится в точке, где мы кончили рисовать. Это все ты легко поймешь, когда начнешь работать.

Под дисплеем находится еще один блок — вспомогательный, рядом на столе стоит обыкновенный кассетный магнитофон. Компьютер может записывать программы и данные на обычную магнитофонную кассету. На учительской машине есть еще устройства печати и дисковод для гибких магнитных дисков. Во-от там. Посмотрела? А сейчас смотри сюда. Чтобы начать работу, надо включить машину выключателем слева сбоку вспомогательного блока. Чик! И включили.



Через несколько секунд Тоня на экране увидела

Бэйсик KORBET в.1

Москва 1986

Ок

— Ой, как здорово! Она пишет по-русски!

— Конечно, но если считать слово «о'кей» русским словом. Это слово в сокращенном виде: «Ок» интерпретатор Бейсика всегда показывает, если все в порядке и можно с ним работать. Как ты уже заметила, на этой машине не нужно вызывать интерпретатор Бейсика, он начинает работать сразу после включения машины. Такую привилегию имеют только ученики, учителям на учительской машине все надо делать самим.

НОВАЯ КЛАВИАТУРА

— А клавиатура-то совсем другая!

— Мелочи жизни. Основные клавиши с буквами ведь расположены на всех клавиатурах одинаково. Правда, «Корвет» существенно отличается от известных тебе машин тем, что позволяет печатать не только заглавные буквы, но и строчные и при этом как буквы русского алфавита, так и латинского. Это очень важно, если делать на ЭВМ системы обработки текстов, которые позволяют на магнитных носителях запоминать тексты, например школьные сочинения, потом их менять, печатать, переписывать и, может быть, еще что-то другое делать. Правда, такой системы у нас пока нет.

Вот посмотри, как выглядят строчные буквы.

— Очень красиво! Но как вы их получили?

— Сейчас поясню, но только скажу, что цифры печатаются в одном регистре с заглавными буквами, а специальные знаки — в одном регистре со строчными буквами.

— Так и было.

— Не совсем. Так было при вводе заглавных русских букв, а при вводе заглавных латинских букв было наоборот. Так что при вводе программ нам приходится чаще переключаться на ввод специальных знаков, но зато проще вводить русский текст.

— Но как же все-таки переключаться на строчные буквы?

— Вот этой большой клавишей с надписью РГ, т. е. «регистр». Таких клавиш даже две — одна слева, а другая справа.

— А что там еще на этой клавише написано?

— То же самое по-английски, наверно, для английских

F6		F7		F8		F9		F10	
F1		F2		F3		F4		F5	
ПРФ	;	1	2	3	4	5	6	7	.
ESC	+	!	//	#	⌘	%	&	/	
ТАБ	Й	Ц	У	К	Е	Н	Г	Ш	[
TAB	}	С	U	К	Е	Н	Г		{
УПР	Ф	Ы	В	А	П	Р	О	Л	
CTRL	F	Y	W	A	P	R	O		
РГ		Я	Ч	С	М	И	Т	Ь	
SHIFT		Q	—	S	M	I	T	X	
	О	Алф							
		ALF							

детей, если они придут, например, в вашу школу. Я советую тебе не забивать голову значением английских сокращений на клавишах. Лучше запомни такую возможность: если нажать одну из клавиш переключения регистров РГ и держать, то можно будет печатать строчные буквы до тех пор, пока не отпустишь эту клавишу РГ.

— Должна же быть возможность как-то освободить обе руки?

— Дай договорить. Если же нажмешь клавишу РГ и еще вот эту клавишу слева с кружочком на ней, то можно будет печатать строчные буквы и после того, как отпустишь клавишу РГ. А когда снова захочешь печатать заглавные буквы, опять нажмешь клавишу РГ и вместе с нею клавишу с кружочком. Эта клавиша с кружочком называется клавишей фиксации.

— А сейчас о том, как переключиться на русские буквы.

— Ты все время норовишь узнать все раньше времени. Я все тебе расскажу, ничего не утаю. Это делается клавишей АЛФ, т. е. «алфавит». Нажмешь ее и будешь держать, сможешь печатать заглавные русские буквы, а если, кроме того, нажмешь и будешь держать клавишу РГ, сможешь печатать строчные русские буквы.

— Ага, а если зафиксировать клавишу РГ, то достаточно будет держать нажатой только клавишу АЛФ.

— Можно так, а можно дополнительно к клавише РГ



СТОП STOP						ИЗ DEL	ВЗ INS	СТР CLS
8 (	9)	0	=	• *	←	↖ 7	▲ 8	↘ 9
Щ }	З }	Х Z	Ъ H	—	↙	◀ 4	МЕНЮ MODE 5	▶ 6
Д L	Ж D	Э V	\ I	• >	↘	← 1	2 ▼	→ 3
Б В	Ю Q ▼	,	/	Р РГ SHIFT	↕	⇐	⇒	•
		ГРАФ GRAPH		СЕЛ SEL				

зафиксировать еще и клавишу АЛФ, нажав ее и вместе с нею клавишу фиксации. Еще пример, как можно использовать эту клавишу. Видишь здесь справа дополнительную клавиатуру для управления курсором и для ввода цифр? Обычно этой клавиатурой управляют курсором, а для того чтобы с нее ввести цифры, надо нажать клавишу СЕЛ и держать, и держать...

— Или нажать клавишу фиксации и не надо будет держать.

— Так. Молодец! А СЕЛ, если интересно, возникло от английского слова «select», что означает «выбрать».

Дальше. Здесь справа имеется клавиша с надписью ГРАФ. Если нажать ее и держать, то при нажатии клавиш вместо букв на экран будут показываться так называемые псевдографические символы, т. е. как бы куски рисунков, состоящие из закрашенных прямоугольников, из которых ты можешь составить большие картины, конечно, не рассчитывая на большой художественный успех.

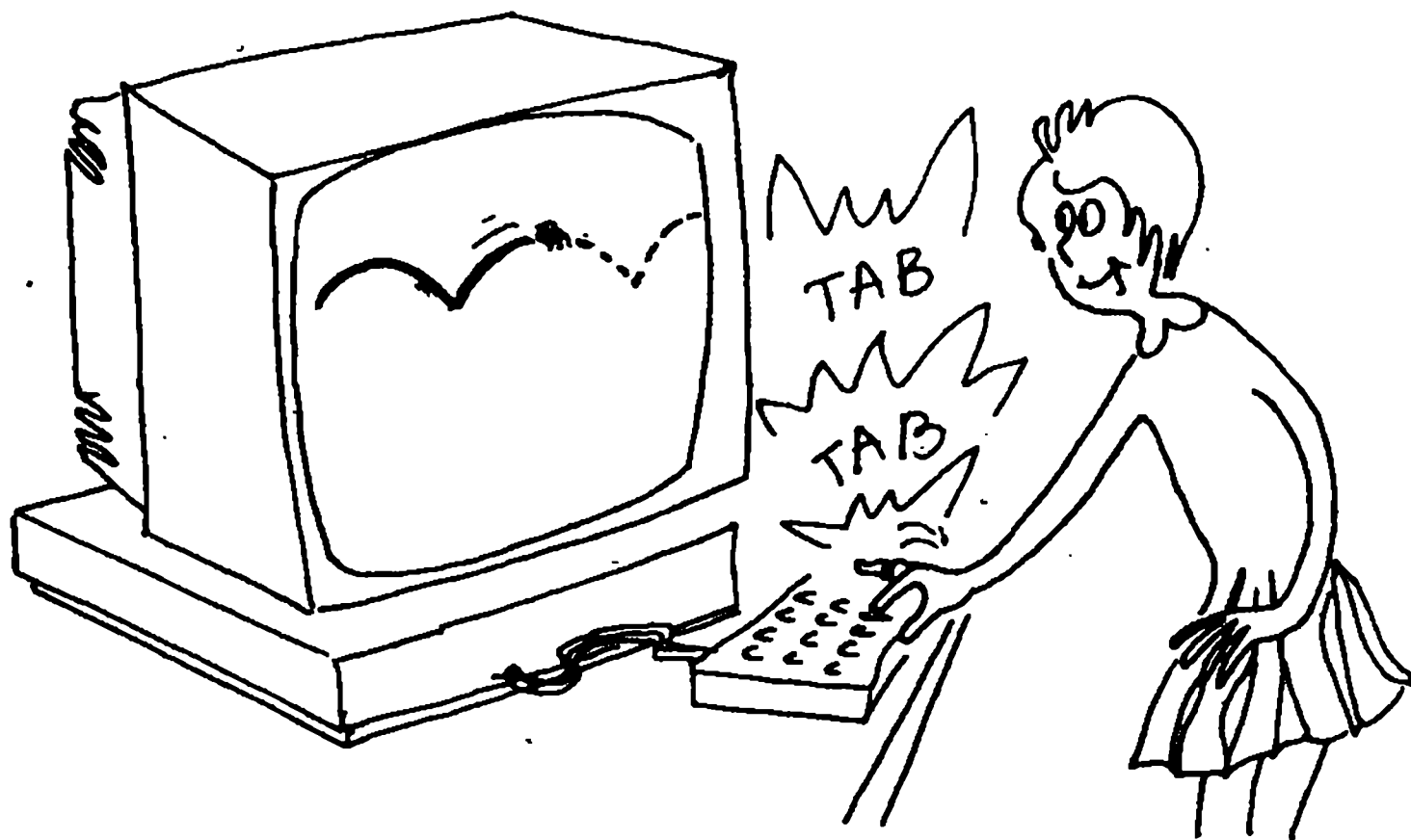
— Это надо будет попробовать.

— Подожди, подожди. Это искусство освоишь еще не так скоро. Так вот, клавишу ГРАФ тоже можно зафиксировать и рисовать свои картины, имея обе руки свободными.

И последняя клавиша, которую тоже можно зафиксировать. Это клавиша управления УПР, если ее нажать, то вводятся управляющие символы, о которых тебе знать еще рано.

— У меня получилось, что каждая клавиша может дать пять различных символов.





— Даже шесть, если посчитать и управляющие символы. Но смотри, мы почти уже все клавиши освоили. Про клавишу ввода — вот здесь справа большая клавиша с ломаной стрелкой — рассказывать тебе нечего, как и про клавишу исправления над клавишей ввода, которой можно стереть последний символ во вводимой строке.

— А зачем эта клавиша ТАВ?

— Ах, эта... Ею при вводе можно передвигать курсор на несколько позиций вперед. Называется клавишей табуляции. Будешь работать, попробуешь. И клавишу прерывания ПРФ освоишь по ходу работы. А вот про эти пять клавиш слева вверху немножко скажу. Они называются функциональными клавишами, по-русски можно было бы сказать «рабочие клавиши». Этим клавишам сам программист может назначать функцию или работу, может запрограммировать реакцию программы на нажатие этих клавиш. Правда, он может с таким же успехом запрограммировать и реакцию программы на нажатие любой буквенной клавиши тоже... Но все-таки функциональные клавиши бывают полезными. Скажем, в интерпретаторе Бейсика, запрограммирован вывод на экран команд Бейсика при нажатии этих клавиш.

— На них какие-то двойные номера...

— Там написаны номера этих функциональных клавиш. Их обозначают F1, F2 и т. д. А двойные они потому, что нажатие каждой такой клавиши при нажатой клавише РГ вызывает ввод в машину другого кода... Ой, ты же, наверное, не знаешь, что такое код. Просто можно запрограммировать разную реакцию программы на нажатие функциональных клавиш в зависимости от того, нажата ли клавиша РГ или нет. Поэтому считается, что этих клавиш в самом деле десять.

— А не пора ли перейти на изучение Бейсика?

ОБЩЕЕ МЕЖДУ ДВУМЯ БЕЙСИКАМИ

— Конечно, конечно. Но не думай, что я тебе смогу изложить весь Бейсик целиком. Потом сможешь взять книжку с описанием, которая приходит с машинами, и узнать все подробности. Только учти, что описание может и не совпасть с реальной жизнью. Всегда надо проверять неясные и новые вещи, пропуская простейшие программы, чтобы убедиться: да, все идет, как должно. Сначала я тебе расскажу о том, что ты уже знаешь или, точнее, должна была бы знать о нашем Бейсике, т. е. об общем между этими двумя видами Бейсика. Понятие переменной, конечно, для всех одно, но у нас имена переменных могут состоять из произвольного количества знаков...

— Значит, в имени может быть несколько букв?

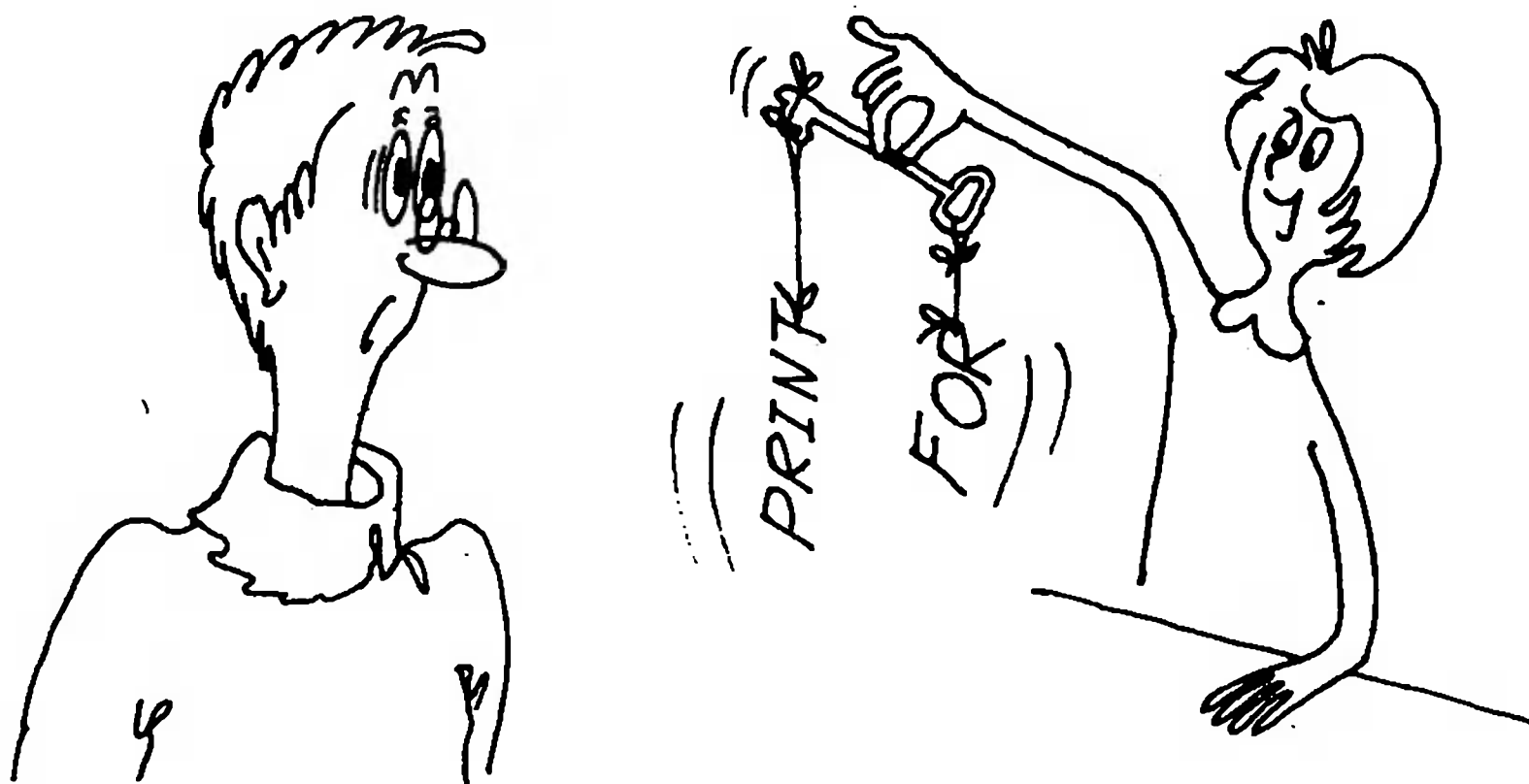
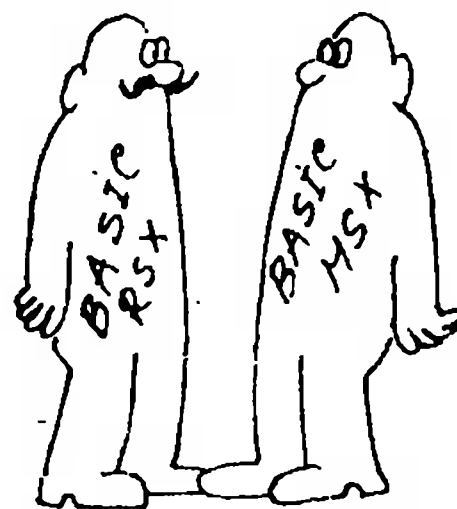
— Да, но может быть еще и несколько цифр, однако первый знак — буква, это во-первых, а во-вторых, важны только первые два знака: если два первых знака у двух имен совпадают, то интерпретатор эти имена воспринимает как имя одной и той же переменной, хотя остальные знаки могут и отличаться. И в-третьих, нельзя использовать в качестве имен слова, используемые в Бейсике, например PRINT, FOR или...

— Что эти слова называются ключевыми, вы, наверное, не знаете...

У Миши от неожиданности даже дыхание перехватило, и он не смог сообразить, как среагировать на такой наскок на его авторитет. А Тоня очень спокойно продолжала:

— А что вы, Михаланыч, можете сказать про имена символьных переменных?

— Все остается по-старому. В конце имени должно быть «солнышко», а в других отношениях они такие же, как и имена числовых переменных. Что касается операторов, то опера-



тор присваивания такой же, оператор вывода PRINT такой же, оператор ввода тоже такой же. Есть, правда, дополнительные возможности, но нет необходимости спешить о них рассказывать, потому что все можно делать и так, как ты умеешь.

Совершенно так же, как и прежде, работает оператор цикла FOR...NEXT, оператор перехода GOTO, оператор вычисляемого перехода ON, оператор перехода на подпрограмму GOSUB и возвращения из нее RETURN, операторы STOP, END, DATA, READ. Ах да, здесь обязательно писать GOTO вместе! Оператор описания массивов имеет существенное дополнение: размеры массива при определении необязательно должны быть числами, можно писать в скобках после имени массива произвольное арифметическое выражение, например DIM (M+4).

Совпадают и имена встроенных функций, работающих с числами INT, ABS, SIN и др. Так же определяются функции программиста с использованием оператора DEFFN. И такими же будут основные директивы, которые, правда, здесь называются командами, — LIST, RUN, NEW.

— Прекрасно!

КОМАНДЫ И ФУНКЦИОНАЛЬНЫЕ КЛАВИШИ

— Людям ленивым бывает приятно узнавать, что ввод самых важных команд запрограммирован в функциональных клавишах. Это выглядит так. Ты нажимаешь функциональную клавишу F4, и сразу на экране выскакивают буквы LIST. Ты сразу можешь нажать клавишу ввода, и тебе будет на экране показана твоя программа. А можешь после слова LIST задать номера строк, которые ты хочешь посмотреть, и только потом нажать клавишу ввода.

— Написать первый номер, потом через тире последний, кажется так?

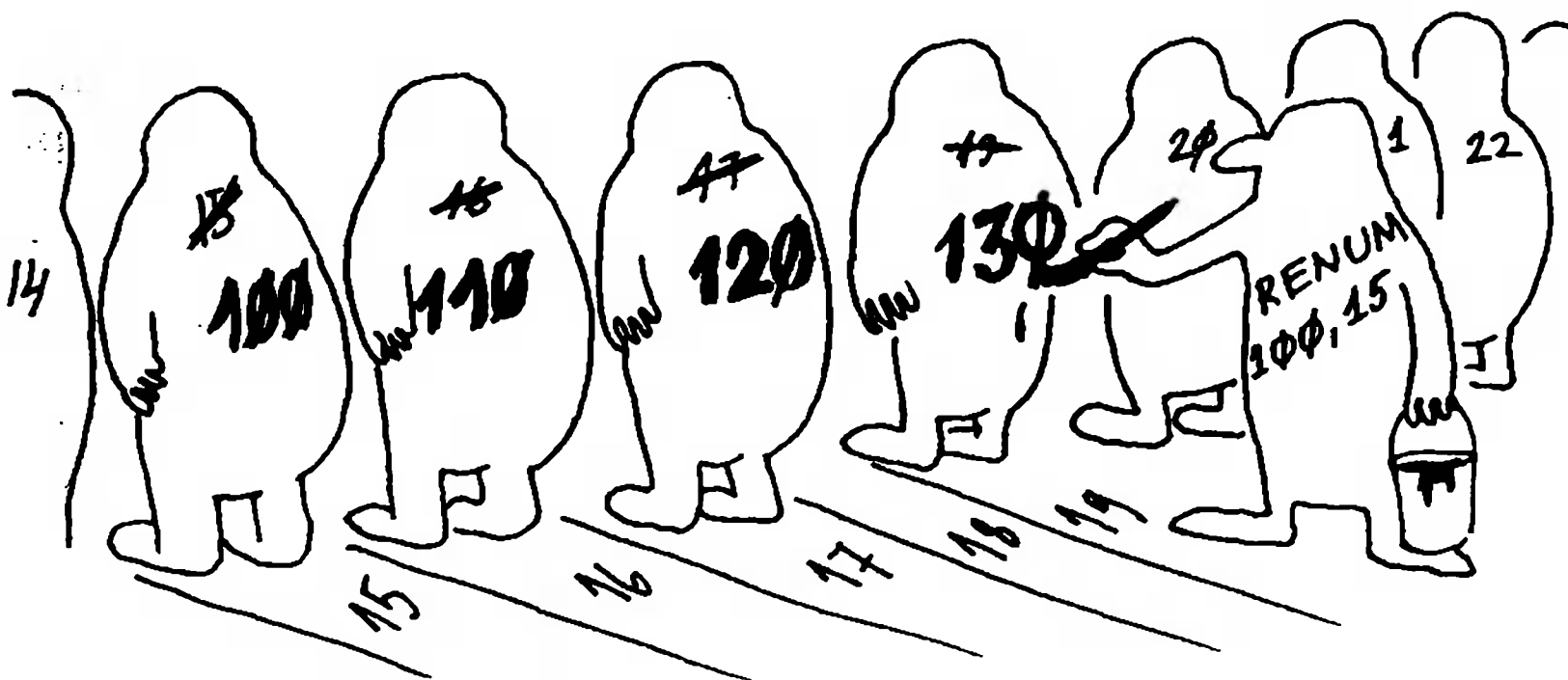
— Да, именно так.

— Кстати, такая клавиша действительно очень удобна и не только ленивому человеку.

— Если нажать клавишу F6, т. е. ту же самую клавишу F1 при нажатой клавише РГ, то на экране появится слово RENUM. Это сокращение английского слова renumber — «ринамбэ», что означает «перенумеровать».

— Ясно. Это аналог команды RESEQ.

— Ты права. Если выполнить эту команду, то все строки перенумеруются. Первая строка получит номер 10, вторая — 20 и т. д. с шагом 10. Если ты хочешь, чтобы первая строка получила другой номер, то запиши этот номер после слова RENUM. Например, RENUM 100. Если же ты хочешь пере-



нумеровать программу не с начала, а начиная с какой-то строки, то делается так. После **RENUM** пишется номер, какой ты хочешь дать этой строке, как я уже говорил, потом через запятую номер, который эта строка имеет сейчас. Например, команда **RENUM 100, 15** перенумерует все строки, начиная со строки 15, и даст им номера 100, 110 и т. д.

— А если я хочу, чтобы номера шли не через десять...

— Тихо! Не мешай говорить. Шаг нумерации задается третьим числом.

— Получается, что если нумеровать с шагом, отличным от 10, придется писать все три числа — новый номер, потом номер, откуда начинать нумерацию...

— Нет, не получается. Любое из первых двух чисел можно опустить, но не опуская запятые. Например, можешь написать **RENUM, 5**, и все будет перенумеровано с шагом 5.

— А первый номер будет пять?

— Нет, десять. Значение первого номера по умолчанию — десять. Значение параметра или постоянной по умолчанию, как ты знаешь, — это то значение, которое интерпретатор берет, когда программист никакого значения не задает. Скажем, шаг нумерации по умолчанию был 10, а номер строки по умолчанию, от которого начинается нумерация, будет равен номеру первой строки программы.

— Вообще-то я этого не знала.

— Еще могу кое-что новое рассказать. Нажатие клавиши **F1** вызывает перенос курсора в левый верхний угол и чистку всего экрана. Это потому, что вводится и сразу выполняется команда чистки экрана **CLS**. А если это действие выполнить при нажатой клавише **PG**, тогда очистится графический экран. По-другому, введется и выполнится команда **PCLS**.

— Можно подводить итоги. Команда **CLS** стирает с экрана выведенные символы, а команда **PCLS** — нарисованные рисунки. Так?

— Так. Но если нарисовать графически букву **A**, то ее придется стирать командой **PCLS**. Обе эти команды можно ис-



пользовать и в программах. Например, перед началом рисования хорошо выполнить команду CLS, для того чтобы на экране не было символов.

Клавиши F2 и F7 тоже дают полезные команды. F2 дает команду LOAD, а F7 — команду SAVE.

— SAVE я знаю, это, наверное, сохранение программы, только я не знаю, где на этой машине сохраняют программы.

— На магнитофонной кассете. А LOAD — команда считывания с кассеты. После LOAD и SAVE надо писать название файла в кавычках, поэтому клавиши F2 и F7 дают не только слова LOAD и SAVE, но и первую кавычку, после которой ты сама можешь написать нужное тебе название файла. Но о записи на кассетный магнитофон подробнее я расскажу, когда у тебя будет что записывать. Про команду редактирования строк программы EDIT тоже расскажу потом. Сейчас я скажу, что в клавише F5 живет команда RUN, которую ты знаешь, а в клавише F10 — команда CONT, которая вызывает продолжение работы программы, остановленной оператором STOP. Обе эти последние команды вводятся и выполняются сразу после нажатия соответствующих клавиш.

РАБОТА С СИМВОЛЬНЫМИ ПЕРЕМЕННЫМИ

А вот набор функций для работы с символьными переменными здесь совсем другой.

— Как это другой!

— Прошу не волноваться. Ничего страшного нет. Сделать можно то же самое, что и старыми функциями, но только писать надо будет иначе. Главное все-таки понимать, что и как ты должна сделать. Как определить длину символьной переменной?

— У нас определяли функцией LEN.

— Здесь тоже. Никакого отличия. Такими же остались и функции преобразования чисел в символьную строку STR.↯ и символьной строки в число VAL. А функция POS, которая определяла, с какого знака начинается одна строка в другой, здесь называется иначе — INSTR, и порядок аргументов другой: первый аргумент — номер знака, с которого начинать поиск, второй аргумент — имя переменной, в значении которой проводится поиск, третий аргумент — имя переменной, значение которой ищется.

— Это ясно.

— Ясно будет и с функцией LEFT ↯, которая хотя и совсем новая, но очень понятная. Она отцепляет кусок с левого конца символьной строки.

— Значит, в аргументах этой функции надо указать, от какой строки и сколько символов отцепить?

— Именно так. Обращение к функции будет иметь вид:

$A \propto = \text{LEFT} \propto (B \propto, M)$

Если значение M равно 5, а значение $B \propto$ — АНТОНИНА, то значение A получится АНТОН.

— Да оч-ч-ч-чень интересно получается.

— Слово «лефт» по-русски означает «левый», а раз есть левый, то будет и правый, следовательно, должна быть функция $\text{RIGHT} \propto$ — произносится «райт», которая отделяет символы с конца строки. Обращение к ней такое же, но только второй, числовой аргумент означает, сколько символов с конца значения первого аргумента следует отправить в значение результата.

$A \propto = \text{RIGHT} \propto (B \propto, M)$

Если M равно 4, а $B \propto$ — АНТОНИНА, то результатом будет НИНА.

— Михаил Иванович, вы не могли бы оставить в покое это бедное имя?

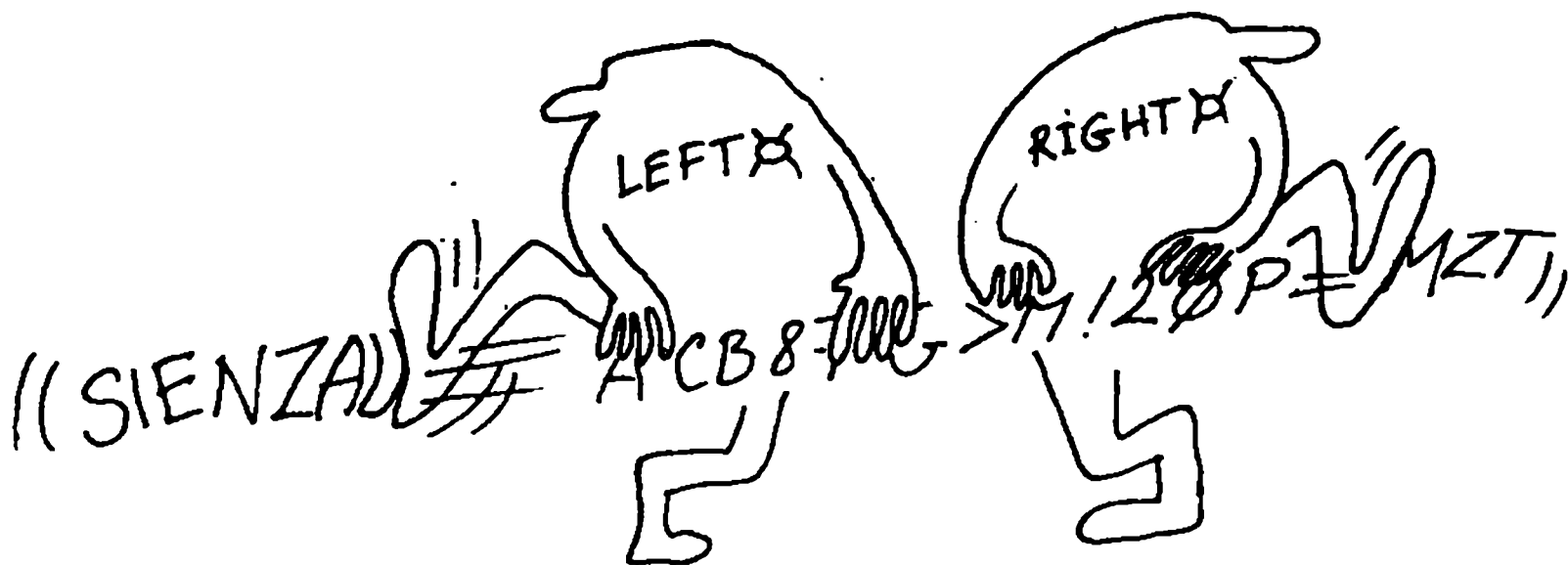
— Пожалуйста, в следующий раз возьмем другую строку символов. Это когда вы, Антонина, освоите следующую функцию — $\text{MID} \propto$, если, конечно, это произойдет.

— Произойдет, произойдет. Работайте, Михаланыч, работайте, не отвлекайтесь!

— Имя этой функции возникло от слова *middle*, которое надо произносить «мидл», что означает «средний»...

— И которая выделяет середину заданной строки символов. Ее аргументами будет обрабатываемая строка, номер символа, с которого начать выделение подстроки и количество выделяемых символов. Так ведь?

— Ты угадала. Могло быть и иначе. Эти аргументы нужно писать в следующем порядке: первым — символьную переменную, вторым — номер символа, с которого начинаем выделение, а третьим — количество выделяемых символов:



$$A \text{ } \bowtie = \text{MID } \bowtie (B \text{ } \bowtie, M1, M2)$$

Последний аргумент может отсутствовать:

$$C \text{ } \bowtie = \text{MID } \bowtie (K \text{ } \bowtie, M5)$$

В этом случае отделяется конец строки, начиная с указанного места, и независимо от того, сколько там оказалось символов, становится строкой результата.

— Ага! В этом случае функция $\text{MID } \bowtie$ совпадает с функцией $\text{RIGHT } \bowtie$.

— Нет, нет! Функция $\text{RIGHT } \bowtie$ отделяла указанное количество символов от конца строки, а $\text{MID } \bowtie$ без третьего аргумента отделяет весь конец строки, начиная от указанного места. Подумай хорошенько, это разные вещи.

— М-м-м... Да, разные.

— Эту функцию еще можно писать слева от знака равенства, тогда она становится новым оператором. Можно его назвать оператором замены подстроки. А справа пишется та строка, на которую заменяют.

— Да, действительно ничего не понять при таком объяснении.

— Еще ничего не потеряно. Пишем пример.

$$\text{MID } \bowtie (X \text{ } \bowtie, K1, K2) = C \text{ } \bowtie$$

Символы из значения первого аргумента $X \text{ } \bowtie$ заменяются на символы строки $C \text{ } \bowtie$, записанной справа от знака равенства. Замена начинается с $K1$ -го символа, и всего заменяются $K2$ символа. Символы для замены берутся, начиная с первого символа строки $C \text{ } \bowtie$.

Если в нашем случае значением $X \text{ } \bowtie$ будет строка **ВАНЬКА**, значением $C \text{ } \bowtie$ — строка **ТОНЯ**, а $K1=1$ и $K2=2$, то строка $X \text{ } \bowtie$ — **ВАНЬКА** превратится в строку **ТОНЬКА**.

— Очень остроумно!

— Прошу запомнить, что этим оператором нельзя удлинить символьную строку. Например, если $X \text{ } \bowtie$ будет **ВАНЯ**, $C \text{ } \bowtie$ — **ЕЧКА** и мы зададим $K1=4$, $K2=4$ в надежде из **ВАНЯ** сделать **ВАНЕЧКА**, то не тут-то было — получится только **ВАНЕ**.

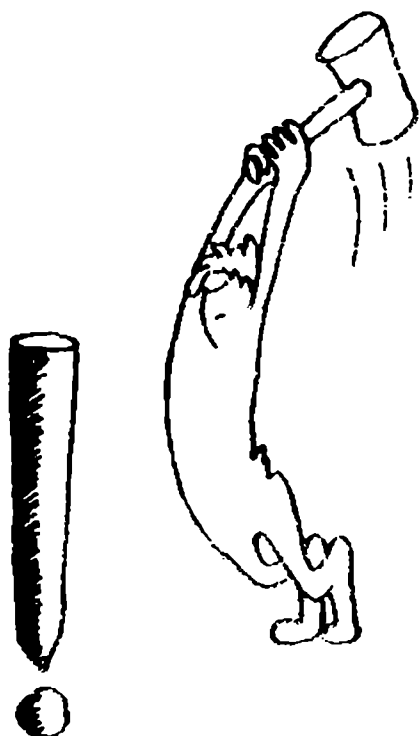
Разберись, что получится после выполнения такой строки:

$$\text{MID } \bowtie (M \text{ } \bowtie, 1, 2) = \text{MID } \bowtie (M \text{ } \bowtie, 3, 2)$$

если значение $M \text{ } \bowtie$ — **ТОНЯ**.

— Что получится... Опять уже надоевшие упражнения в остроумии!

— Не можешь сообразить — так и скажи! Ответ такой: получится **НЯНЯ**. Очень просто. Дальше, если в операторе $\text{MID } \bowtie$, т. е. слева от знака равенства, не писать последнего



аргумента, то из строки для замены справа от знака равенства будут взяты все символы, какие только там есть, но так, чтобы обрабатываемая строка не удлинялась.

Насчет длины символьной строки должна быть полная ясность. Пока символьной переменной ничего не присвоено, ее длина равна нулю, а если ее значение определить равной, скажем, шести пробелам, то длина станет равной шести.

— Еще бы!

— Не так-то это просто. Надо четко понимать, что пробел является обычным символом. Имеется еще и специальная функция, которая заполняет пробелами значение переменной, — функция `SPACE X`, произносится «спейс» и означает «пространство» или «пробел». К этой функции обращаются так:

$$K\ X = \text{SPACE } X\ (M)$$

После выполнения такой строки переменная `K X` получит значение, состоящее из `M` пробелов. И еще одна похожая функция — `STRING X`, которая заполняет значение символьной переменной одинаковыми символами.

$$K\ X = \text{STRING } X\ (M, P\ X)$$

Число `M` указывает, сколько раз нужно взять для значения переменной `K X` первый символ значения переменной `P X`.

Это, пожалуй, все про символьные переменные. . . На сегодня хватит. Иди домой и все хорошо обдумай! Я в следующий раз буду спрашивать, и чтоб не было никаких отговорок, что голова болела, много задали или что-нибудь еще. Поняла?

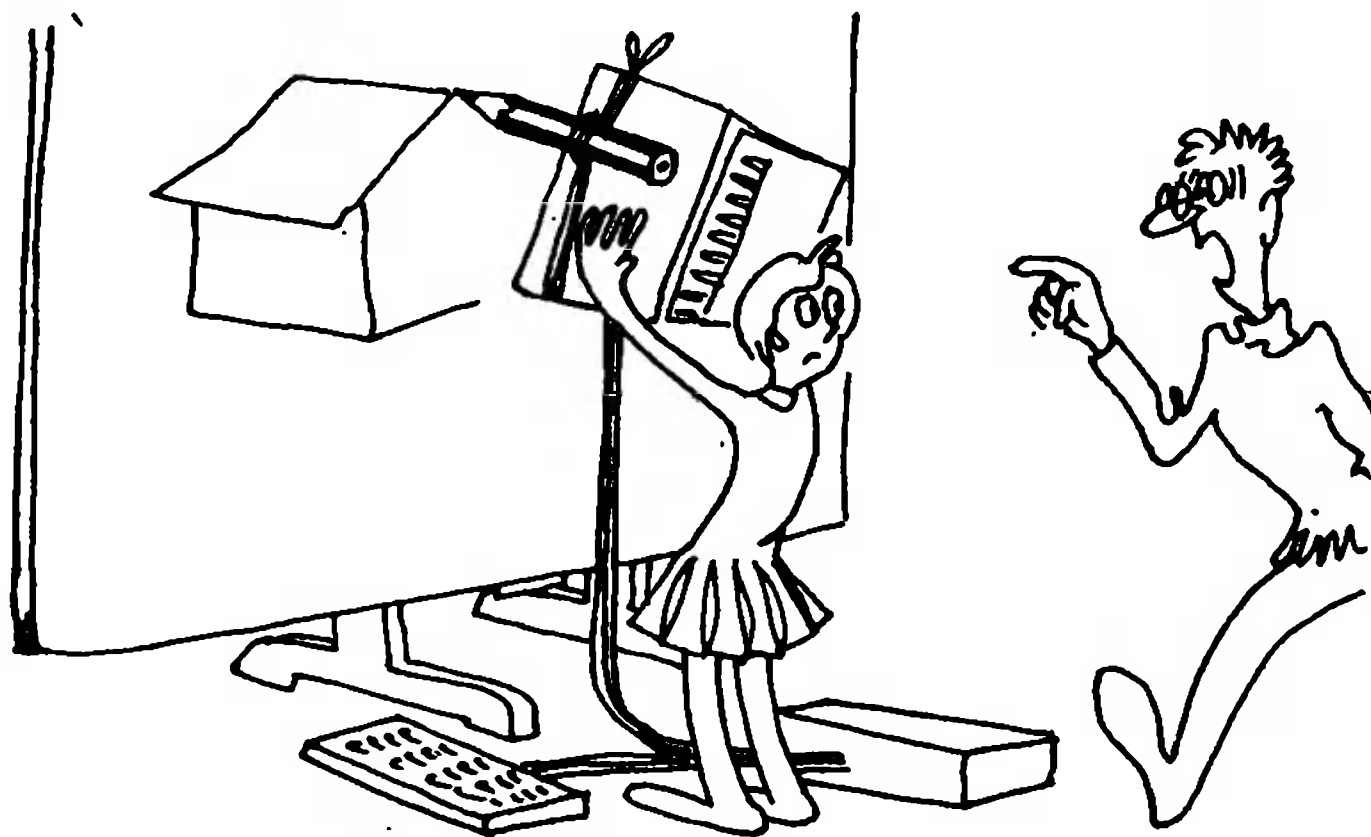
— Поняла, Михаланыч. До свидания.

23. КАК РИСОВАТЬ КОМПЬЮТЕРОМ

— Сегодня тебе предстоит счастливая возможность познать азы компьютерной графики. Компьютерная графика сегодня это уже отдельная специальность, но кое-что о ней надо знать и таким пигалицам, как ты, а то ведь никакого представления не будешь иметь, как делают, скажем, современные игры. . .

— Извините, Михаланыч, неужели работа на «Корветах» так стимулирует болтливость?

— Нет, я не могу! Последний раз! Больше я не стану заниматься со школьниками. Эта современная молодежь. . .



— Но все-таки, может, что-нибудь расскажете про графику?

— Слушай и запоминай, я не стану повторять. Рисунок на экране телевизора, как, впрочем, и в книге, состоит из отдельных точек. И если телевизор приспособить в качестве дисплея, ничего не меняется, все, что хочешь рисовать, придется рисовать точечками, но так как их достаточно много, то что-то и удастся нарисовать. Получается нечто похожее на вышивание крестиком. Ты любишь вышивать крестиком?

— Нет, это немодно!

— Конечно! Станет она заниматься чем-нибудь, что немодно! А зря. Это помогло бы тебе понять первый оператор, который я тебе буду объяснять,— оператор задания цвета фона COLOR. Это слово означает «цвет» и произносится «калэ».

— При чем тут цвет, если дисплей у нас бесцветный?

— Дисплей у нас черно-белый, т. е. допускает изображение как минимум двумя цветами — черным и белым. На самом деле можно получить изображение и серым цветом разных оттенков. А оператором COLOR мы задаем, какой будет цвет заднего фона, т. е. цвет ткани, на которой ты собираешься вышивать крестиком, и цвет переднего фона, т. е. цвет ниток, которыми ты будешь вышивать. Отличие в том, что здесь ты оба эти цвета можешь менять по ходу работы программы, а при вышивке могла менять только цвет ниток.

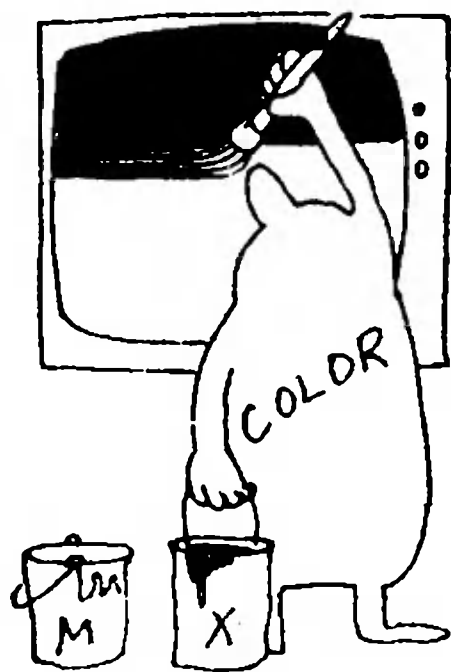
— Похоже, вы очень любите рукоделие.

— Тш-ш-ш! После слова COLOR указываются два числа:

COLOR M, X

Первое означает номер цвета переднего фона, а второе — номер цвета заднего фона.

— Что это за номера такие?



— Все цвета, которые на нашем дисплее можно было бы получить, будь он цветным, или оттенки серого цвета, которые мы получаем на самом деле, пронумерованы номерами от 0 до 7. Следовательно, всего возможны 8 цветов. Запомни основное: черный цвет имеет номер 0, а белый — номер 7, так что получаются всего четыре возможности записи этого оператора. Какие?

— Это очень просто.

COLOR 0, 7

даст возможность рисовать черным по белому;

COLOR 7, 0

белым по черному. А еще?

— Учти возможность рисовать белым по белому и черным по черному.

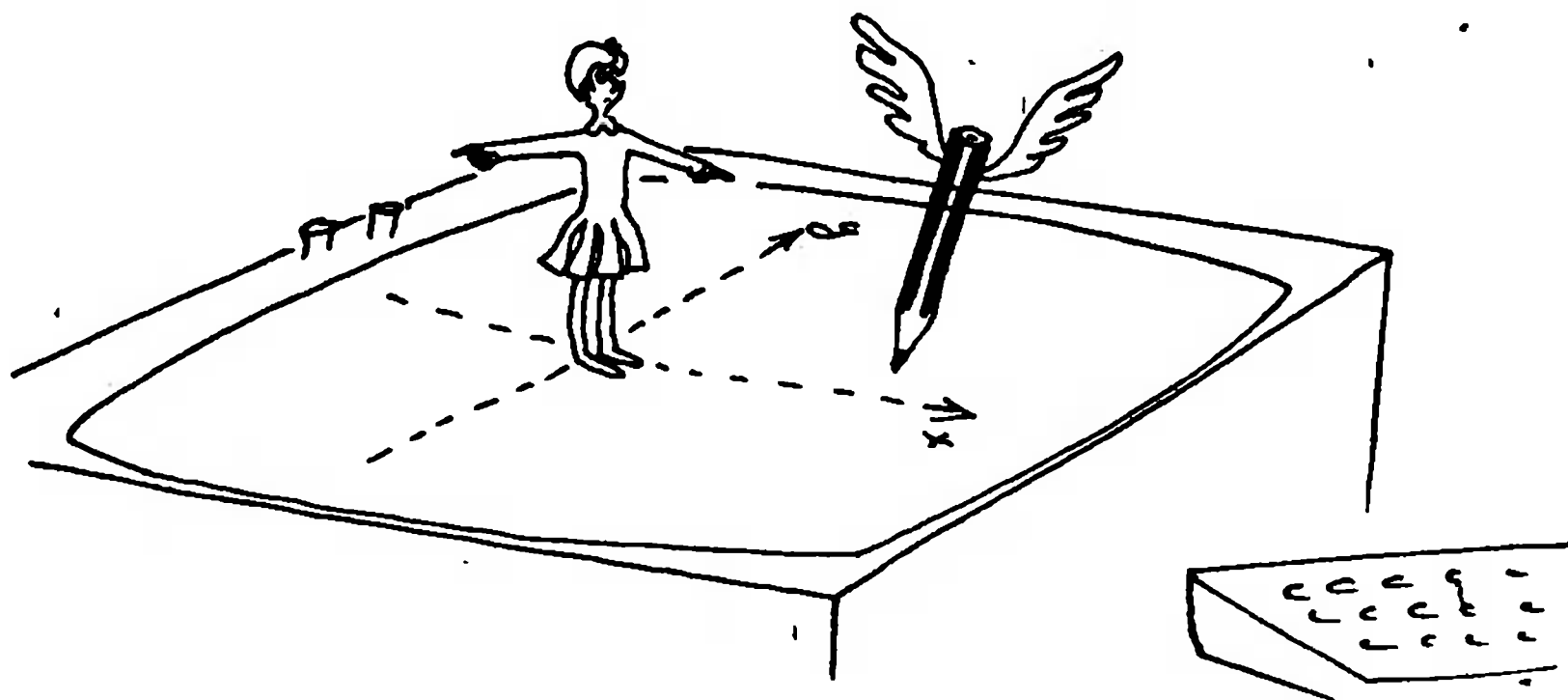
— Это еще зачем?

— Чтобы стирать нарисованное. Вот представь себе: ты рисуешь и рисуешь белым по черному, и вдруг захотелось стереть часть нарисованного. Ты переключаешься на рисование черным по черному оператором COLOR 0, 0. Понимаешь, это переключение рисунок не изменило, он остался таким, каким был. Если нарисуешь что-нибудь сейчас по уже нарисованному месту, то белое закрасится черным, а будет выглядеть, что рисунок стирается.

— А я вот думаю, думаю и не могу понять, зачем надо стирать, ведь лучше же это вообще не рисовать.

— Ты права. Это нужно только в одном случае, если хотят запрограммировать движение рисунка по экрану, другими словами, мультипликацию. Тогда в одном месте рисуем, потом стираем, потом рядом рисуем и стираем, еще дальше рисуем и стираем, а выглядит, будто происходит движение.

— Поняла. Но как же все-таки рисовать?



— Чтобы это понять, ты должна быть достаточно грамотной — знать, что это такое — координаты на плоскости.

— О координатах на плоскости я что-то слышала.

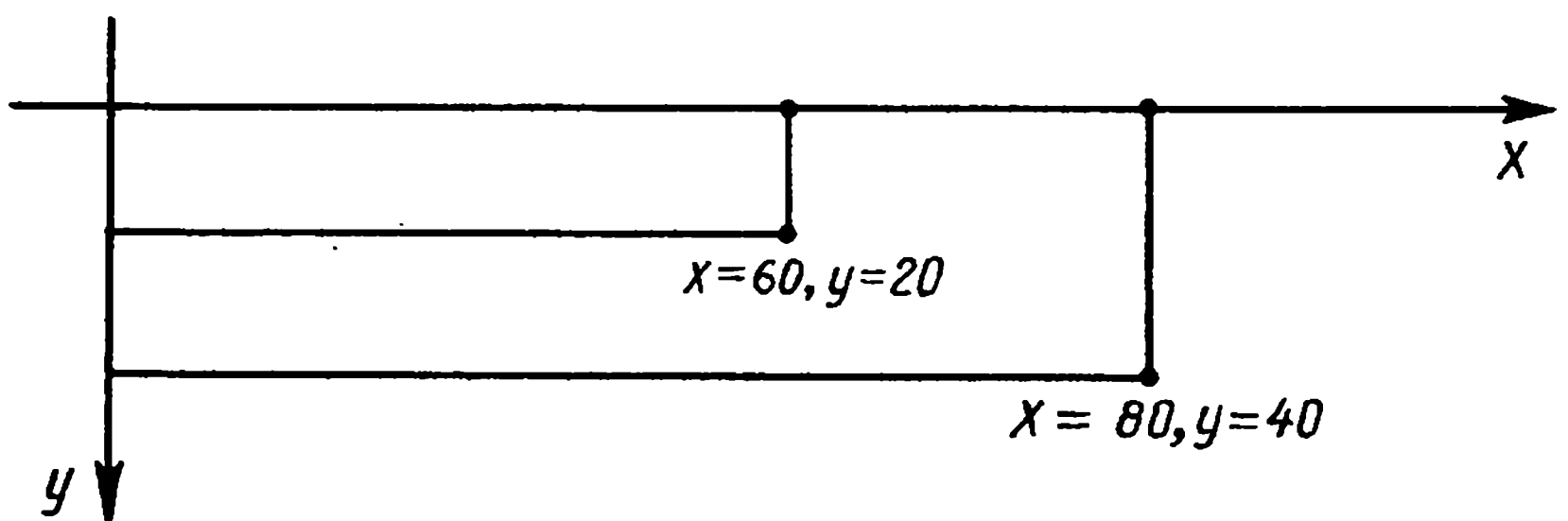
— Все слышали, но небольшое повторение не повредит. У нас на дисплее, если речь вести о рисовании, имеются 256 строк точек, а в каждой строке содержатся 512 точек. Одна из этих 131 072 точек объявляется особой — ей присваивается звание центра координат. Считай, что мы становимся в этом центре координат. Еще объявляется, какое направление будет «вперед», а какое — «направо», так сказать, определяется, как мы встанем на экране.

— Еще надо сказать, что встанем мы не кое-как, а так, чтобы смотреть перпендикулярно одной из сторон экрана.

— Да, что-то такое надо было бы сказать. После включения наших «Корветов» центр координат у них в левом верхнем углу экрана, и смотреть надо вниз. Так вот, координаты точки на экранах наших «Корветов» означают следующее. Представь себе, что от центра координат ты переходишь к точке, координаты которой надо определять. И двигаешься ты сначала налево, а потом вниз. Количество точек, пройденных при движении налево, даст тебе первую координату — x , а количество точек, пройденных при движении вниз, даст тебе координату y .

— Вы очень интересно рассказываете, Михаланыч.

— Совсем запутался. Лучше нарисую. Вот такова наша система координат. Нарисую еще несколько точек с координатами.



— Как же все-таки рисовать?

— Она какая-то нетерпеливая. Я же уже открыл рот для сообщения об операторе рисования отрезков прямых. Это оператор LINE.

— Понятно. «Лайн» означает «прямая».

— Ничего подобного. Это означает «линия», а какая именно, прямая или кривая, это еще надо уточнять. Но это разговоры. Суть в том, что после этого слова пишут координаты концов отрезка, который надо нарисовать.

— Первыми надо писать координаты начала отрезка, а вторыми — координаты конца?

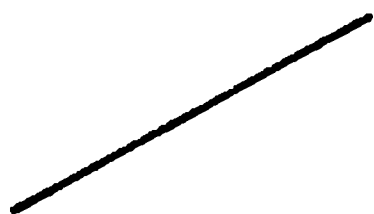
— Я бы сказал не так. Точка, координаты которой напишешь первыми, будет считаться началом, а точка, координаты которой запишешь вторыми, станет концом отрезка. Но пора тебе сказать, как же записываются эти координаты. Их записывают сразу после LINE в скобках, отделяя одни от других тире. Вот так:

LINE (100, 50) — (200, 25)

Учти, в скобках первыми пишутся x-координаты. И еще учти, что координаты необязательно задавать в виде чисел, конечно, в скобках можно писать имена переменных и арифметические выражения. А сейчас выполним эту строку в непосредственном режиме.

Видела, какой красивый отрезок получился?

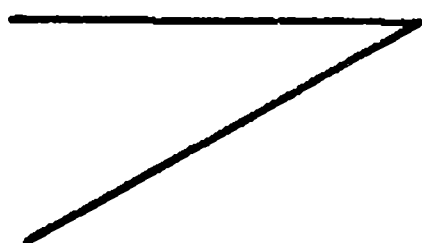
— Ну уж красивый! Весь такой угловатый.



— Это потому, что он фактически состоит из точек. Горизонтальные и вертикальные прямые получаются красивее.

Если я координаты начальной точки не напишу вообще, то новый отрезок будет начинаться там, где кончился предыдущий, т. е. там, где находится графический курсор. Вот посмотри, что выйдет.

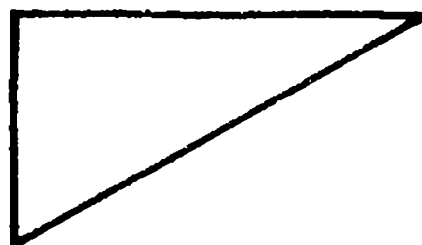
LINE — (100, 25)



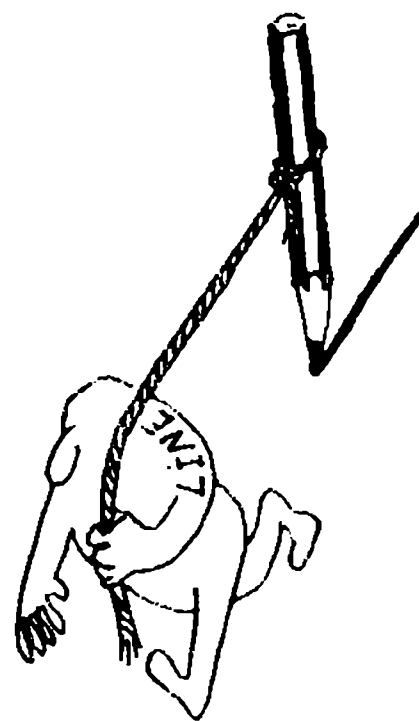
Сейчас прошу уважаемую ученицу попробовать нарисовать недостающую сторону треугольника.

— С превеликим удовольствием.

LINE — (100, 50)



— Редко встречаются такие быстросоображающие дети. У оператора LINE имеется целый ряд дополнительных возможностей. Первая из них такая. Можем использовать так



называемые относительные координаты, когда последняя нарисованная точка, или местоположение графического курсора, объявляется новым, временным центром координат, относительно которого вычисляются координаты, если перед ними в операторе LINE написано слово STEP. Кстати, эти координаты могут быть отрицательными, если мы указываем точку правее или выше места нахождения графического курсора. Это может выглядеть, например, так:

LINE (0, 0) — STEP (50, 0)

Что получится? От начала координат мы проведем отрезок в точку на 50 единиц левее нашей последней нарисованной точки. Какую точку мы рисовали последней?

— Эту... ну, с координатами (100, 50). Получилось, что графический курсор сейчас находится в точке (50, 50).

— Да. Далее, в операторе LINE после координат можем задать цвет рисуемой линии. Через запятую пишем номер цвета в виде числа или переменной.

— Ага! Определяем, рисовать или стирать.

— Вот именно. Если цвет не указывать, как это мы все время делали, то отрезки рисуются цветом переднего фона, каким уж он у нас задан. Продолжаем. После цвета, опять через запятую, можем писать букву B, и, если напишем, все поменяется. Сейчас оператор будет рисовать не отдельный отрезок, а целый прямоугольник, и задавать сейчас надо не координаты концов отрезка, а координаты углов прямоугольника. Первыми указываются координаты левого верхнего угла прямоугольника, вторыми — правого нижнего угла. Существенно еще то, что цвет сторон прямоугольника вообще можно не задавать, а после координат писать две запятые и потом букву B. Тогда прямоугольник будет нарисован цветом переднего фона, который мы могли где-то раньше определить. Каким оператором мы можем определить цвет переднего фона?

— Оператором COLOR. Первый параметр в нем задает цвет переднего фона.

— Хвалю за прекрасную память. Ну что, нарисуем какой-нибудь прямоугольник?

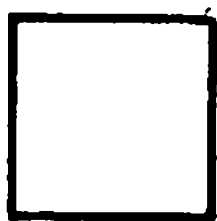
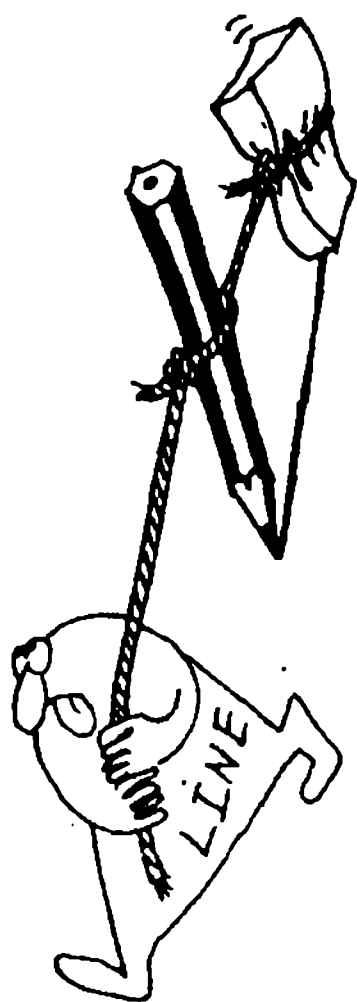
— Нарисуем.

— Но чтобы все было красиво, необходимо очистить экран от рисунков. Это делается оператором PCLS. На символы, выведенные операторами PRINT, LIST и т. п., этот оператор не действует. Он очищает только графический экран.

PCLS

LINE (20, 20) — (60, 0) ,, B

Особую ценность этому оператору придает то, что не только можно нарисовать прямоугольник, а еще можно его и



закрасить. Достигается это добавлением после В буквы F. И тогда нарисованный прямоугольник закрасится цветом сторон прямоугольника.

— То есть цветом переднего фона.

— Не путай! Цветом переднего фона он закрасится, если никакого цвета в операторе LINE не указать. А если укажешь, и притом отличный от цвета переднего фона, то весь прямоугольник закрасится этим цветом. Тогда этот оператор будет выглядеть, например, так:

LINE (20, 20) — (60, 0), 3, BF

— А если надо закрашивать, скажем, треугольник?

— Можно закрасить любую замкнутую область, используя оператор PAINT, что произносится «пэйнт» и означает что-то вроде «малевать», что ли. После слова PAINT указывается координата точки внутри области. Все равно какой точки, лишь бы внутри закрашиваемой области. Потом указывается цвет закрашки, а после него можно указать цвет границы, а можно и не указывать, ошибки не будет. Выглядит это все вместе, например, так:

PAINT (P1, P2), K1, K2

Что произойдет?

— Область, внутри которой находится точка с координатами P1 и P2, будет закрашена цветом, номер которого дает значение переменной K1, а значение переменной K2 даст... даст...

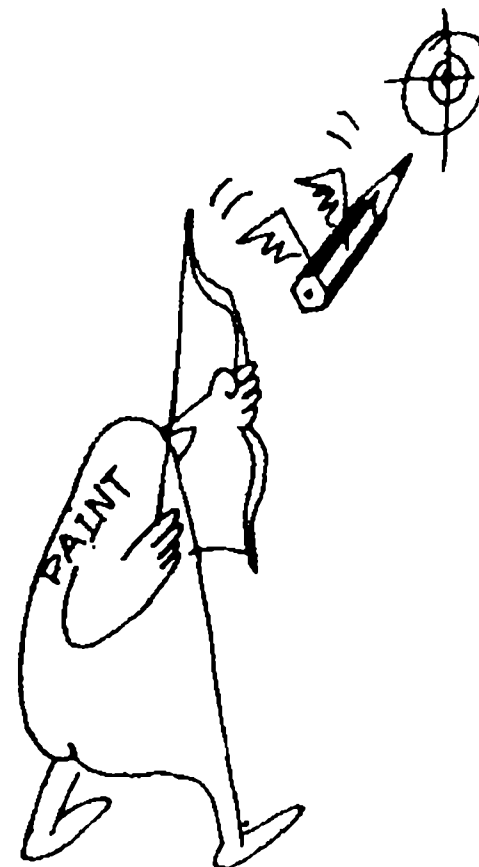
— Окраску линии границы. Еще некоторые мелочи. Как я уже сказал, цвет границы можно не указывать. В этом случае граница закрасится тем же цветом, что и сама область. Можно не указывать и цвет закрашки области, в этом случае область закрасится цветом переднего фона. И наконец, можно задавать относительные координаты. Что для этого надо сделать?

— Где-то надо написать STEP. А вот где...

— После слова PAINT и перед самыми координатами. Вот так:

PAINT STEP (100, 50)

Можно рисовать не только отрезки прямых и прямоугольники, но и отдельные точки. Это можно делать при помощи двух операторов: PSET и PRESET. Сами слова, видимо, являются сокращением от английских выражений point set и point reset, что означает «установить точку» и «заново установить точку». Произношу я эти сокращения «писет» и «пирисет». После PSET и PRESET пишется координата точки, которую надо рисовать, а потом цвет, которым ее надо рисовать.



— А какая же разница между этими двумя операторами?

— Да никакой, если писать полностью. Разница появляется, если не задавать цвет. Тогда оператор PSET нарисует точку цветом переднего фона, а оператор PRESET — цветом заднего фона.

— Пример будет?

— Да, будет. Вот такой:

PSET STEP (50, ТОЧКА), C 1

Как видишь, в этом операторе координаты точки можно задавать и в относительных координатах. Если хочешь задавать как обычно, то слово STEP не пишется.

И наконец, последний графический оператор, о котором я собираюсь тебе рассказывать, — оператор рисования окружностей. Это достаточно сложная вещь, так что даже не стану тебе все подробности излагать, если захочешь, потом сможешь в книжках посмотреть. Пишется слово CIRCLE, которое означает «окружность» и произносится «сэклъ», после него указываются координаты центра рисуемой окружности, при этом можно указывать и относительные координаты, написав после CIRCLE, как всегда, слово STEP, потом через запятую задается радиус этой окружности.

— В сантиметрах?

— Какая чушь! В единицах графического экрана, в точках. После радиуса через запятую можно задать цвет рисуемой линии.

— А если не задать, то будет нарисована окружность цвета переднего фона?

— Да. Это пока все. Смотрим на примере.

PCLS

CIRCLE (250, 100), 50, 2

Вот так можно нарисовать окружность.

Сейчас прошу вас, уважаемая Антонина, нарисовать что-нибудь красивое и поучительное.

Тоня взялась за дело и, изрядно попотев, получила следующую программу. Какой рисунок у нее получится?

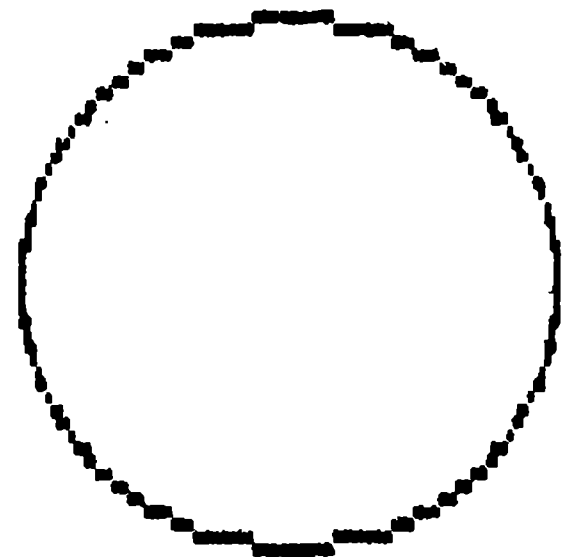
10 PCLS

20 LINE (0, 100) — (150, 0), B

30 LINE (20, 90) — (70, 70), B

40 LINE (80, 90) — (130, 70), B

50 LINE (30, 30) — (120, 20), B



Миша посмотрел ее рисунок и, усмехнувшись, сказал:
— Такой шедевр мирового программирования нужно запомнить на ленте. Я тебе на время работы выделяю страшный дефицит — магнитофонную кассету, не потеряй ее. На ней ты сможешь записывать свои программы.

— А данные можно?

— Нет, нельзя. Таких операторов у нас нет.

Скажу честно, записать программу на магнитофонную кассету может только технически грамотный человек — знаток магнитофонов. Он должен знать, что у магнитофона имеются клавиши перемотки ленты вперед и назад, клавиша записи и клавиша воспроизведения записи, или, как мы ее будем называть, клавиша считывания.

— Ну, таких специалистов нетрудно будет найти.

— Это еще не все. Надо еще понимать две команды: **MOTOR ON** и **MOTOR OFF**. Слова эти означают «запустить мотор» и «заглушить мотор».

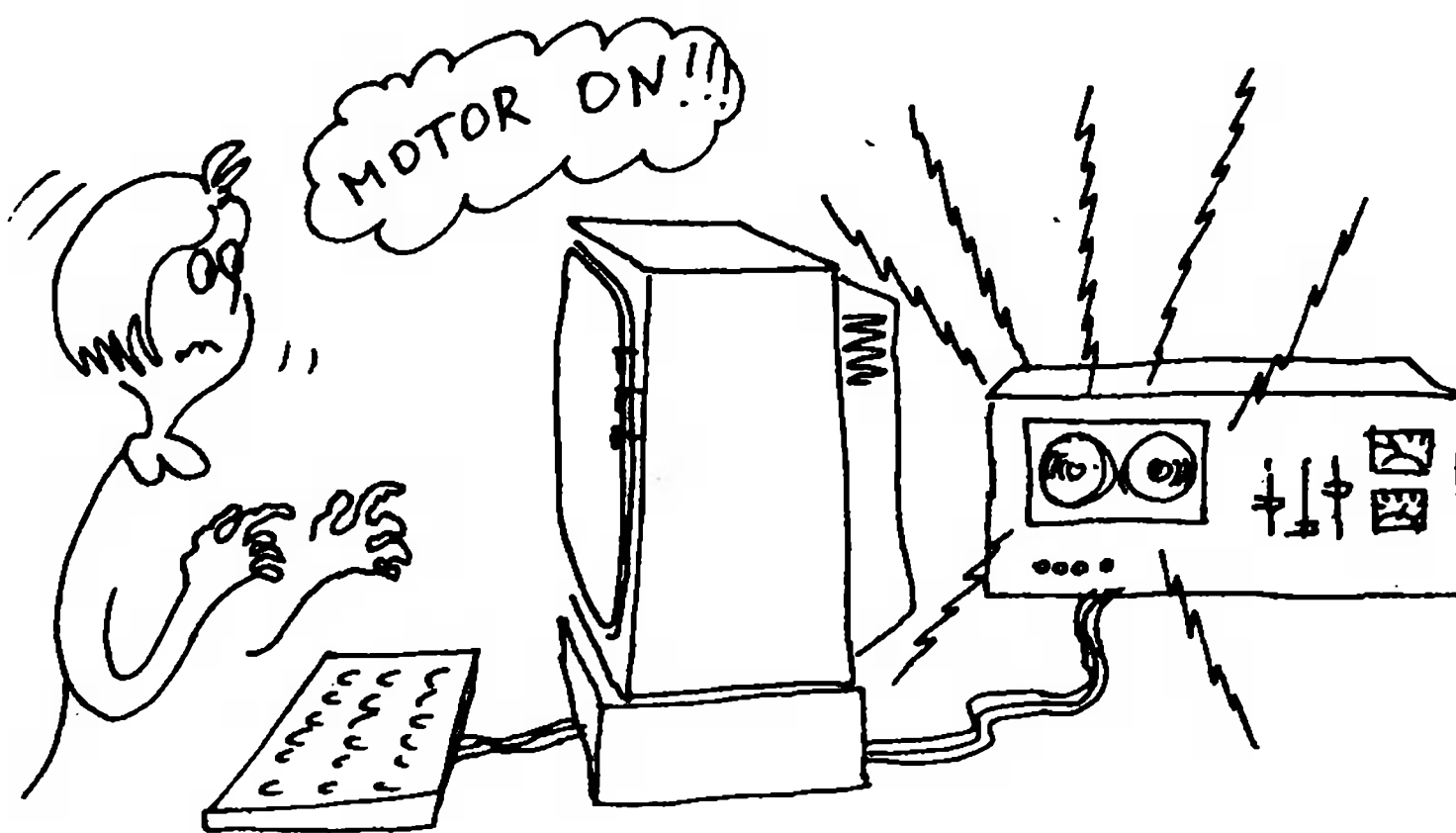
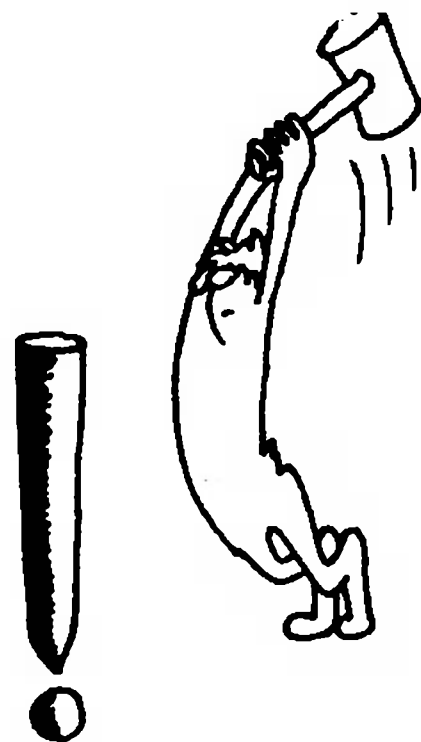
— Это что, команды Бейсика?

— Именно так. Учти, что магнитофон присоединен к машине и она может управлять записью и считыванием информации. Но перемоткой ленты или, скажем, установкой кассеты машина не управляет. Это должна делать ты сама. И когда такое желание у тебя появляется, ты должна сообщить об этом машине командой **MOTOR ON**.

— То есть запустить мотор?

— Да. Когда мотор запущен, ты можешь нажимать на клавиши магнитофона, и он будет делать то, что надо. Например, перематывать ленту. Когда мотор заглушен, нажать на клавиши тоже можно, но никаких действий магнитофона это не вызовет, пока не поступит команда управления от машины.

Итак, первое, что надо сделать при записи программы на магнитофон, — это дать команду **MOTOR ON**. Второе — вста-





вить свою кассету и перемотать до свободного места на магнитной ленте.

— А как я буду знать, где у меня свободное место?

— Ты что, магнитофона не видела? У него имеется счетчик перемотанной ленты. В каких единицах измеряется перемотанная лента, я не знаю, но это неважно, зная показатели этого счетчика до записи программы и после, ты сможешь в следующий раз перемотать ленту так, чтобы не записать два раза на одно и то же место. Поэтому я тебе от всей души советую завести какую-нибудь бумажку, на которой будешь отмечать, где у тебя что записано.

— Заведу, Михаланыч, заведу.

— Эти записи тебе помогут и при считывании нужной тебе программы. Дело в том, что машина начнет ее искать на ленте, начиная с того места, где лента стоит. И если ты сможешь перемотать до нужного места, считывание пойдет быстрее.

— Получается, что поиск на ленте идет медленнее, чем перемотка?

— А как же! Поиск идет с такой же скоростью, как и считывание. Это и есть считывание, машина считывает и «смотрит», то ли считано, что ты хочешь, или нет. Если еще не считано, то продолжает работать. Считывание, если ты забыла,— это то же самое, что воспроизведение музыки, и происходит оно много медленнее перемотки.

— Ладно. Я уже ввела команду MOTOR ON и вставила кассету. Что дальше?

— Установи счетчик на нуль и перемотай до отметки 010. Обычно программисты не начинают записывать с самого начала ленты. Это просто так, на всякий случай.

— Перемотала.

— Теперь надо отключить мотор.

— Помню. Это делалось командой MOTOR OFF.

— Сейчас нажмем клавишу записи. Нажмем и увидим, что ничего не происходит. Это потому, что мотор отключен. Будь он включен, мы бы сразу увидели, что лента в кассете начинает крутиться и вроде происходит запись.

— Ничего не понимаю! Запись чего? И почему вроде?

— Реальная запись чего-нибудь может начинаться только после ввода команды SAVE. И при этом будет записана та программа, которая в данный момент «сидит» в памяти. Файл, который в результате записи образуется на ленте, нужно как-то назвать и название написать после SAVE в кавычках.

— Назовем так:

SAVE "MORDA"

— Можно и так. Что кому нравится. Видишь, идет запись — крутится лента в кассете и слышен шум в динамике магнитофона.

— Уже все. На экране показалось ФАЙЛ MORDA ЗАПИСАН.

— Сейчас перемотай ленту до отметки 020 и запиши еще раз, лучше с другим именем.

— Хэ! Я нажала на клавишу перемотки, а перемотка не началась!

— Вот так! Говоришь, говоришь — и все впустую. Надо включить мотор.

— Ах, да! Но зачем мне записывать второй раз?

— Жизнь такая. Вполне может случиться, что в следующий раз ты не сможешь первый файл считать. В этом случае можно будет попробовать считать второй файл.

— Что, вторые файлы лучше считываются?

— Нет, конечно. Бывает, что и второй файл не считывается, тогда хорошо, если есть еще и третий.

— Мне хватит двух.

Через минуту Тоня сообщила:

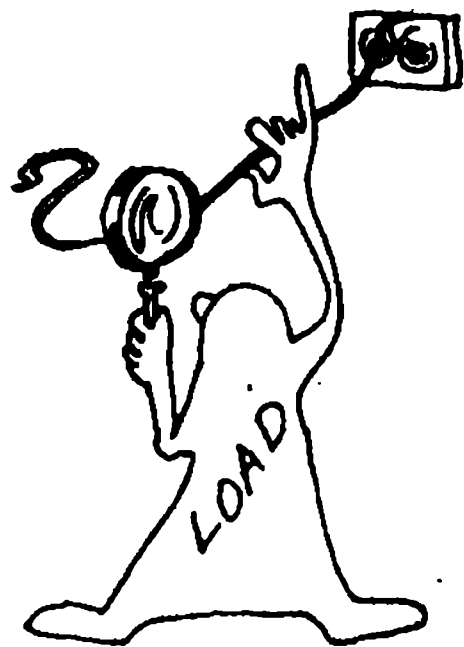
— Порядок! Сейчас о том, как считывать файлы?

— Все делаем похоже, но только нажимаем клавишу считывания и вводим команду LOAD. Я кратко повторяю. Даем команду MOTOR ON, вставляем кассету, перематываем до нужного места, даем команду MOTOR OFF, нажимаем клавишу считывания, вводим команду LOAD с именем считываемого файла, заключенного в кавычки, и готово. Можешь попробовать.

— Некогда, Михаланыч. Нужно домой.

— Нужно, так нужно. Запомни только еще такую вещь, что содержание ленты можешь узнать, вставив ленту в магнитофон и задав команду LOAD с именем файла, заведомо отсутствующим на ленте. Тогда перемотается вся лента в поисках этого файла, а попутно будут выведены на экран имена встреченных файлов.

— Запомнила. До свидания.



24. ТОНЯ УЧИТ ПЕТЮ

Петя, хотя и был сильно занят учебой и программированием, все-таки решил немножко подучиться и Бейсику, применяемому на «Корветах». Он думал, что раз уж он достаточно хорошо разбирается в одном варианте Бейсика, то должен суметь освоить и другой без больших затруднений. И еще ему в глубине души казалось, что Тоня перестанет его уважать и страшно возгордится, если будет знать больше его. Поэтому он достал у знакомых ребят описание корвет-

ского Бейсика и самостоятельно его изучил. Не полностью, конечно, потому что не все понял.

А Тоня об этих специфических Петиных трудностях совсем не догадывалась и, когда они после уроков случайно встретились, она сразу стала тараторить о том, как интересно работать на новом Бейсике и как много всего можно на нем сделать.

— Знаешь, как там здорово сделано, что можно считывать символ прямо с клавиатуры. Понимаешь, когда мы работаем обычным оператором INPUT, он показывает знак вопроса и ждет, пока мы введем нужные символы и нажмем клавишу ввода, чтобы сообщить, что больше ничего не хотим вводить. А здесь можно сделать так, что машина мгновенно реагирует на нажатие только одной клавиши, можно предусмотреть различное поведение программы в зависимости от того, какую клавишу ты нажал.

— Это оператор INKEY δ ?

— А ты откуда знаешь?

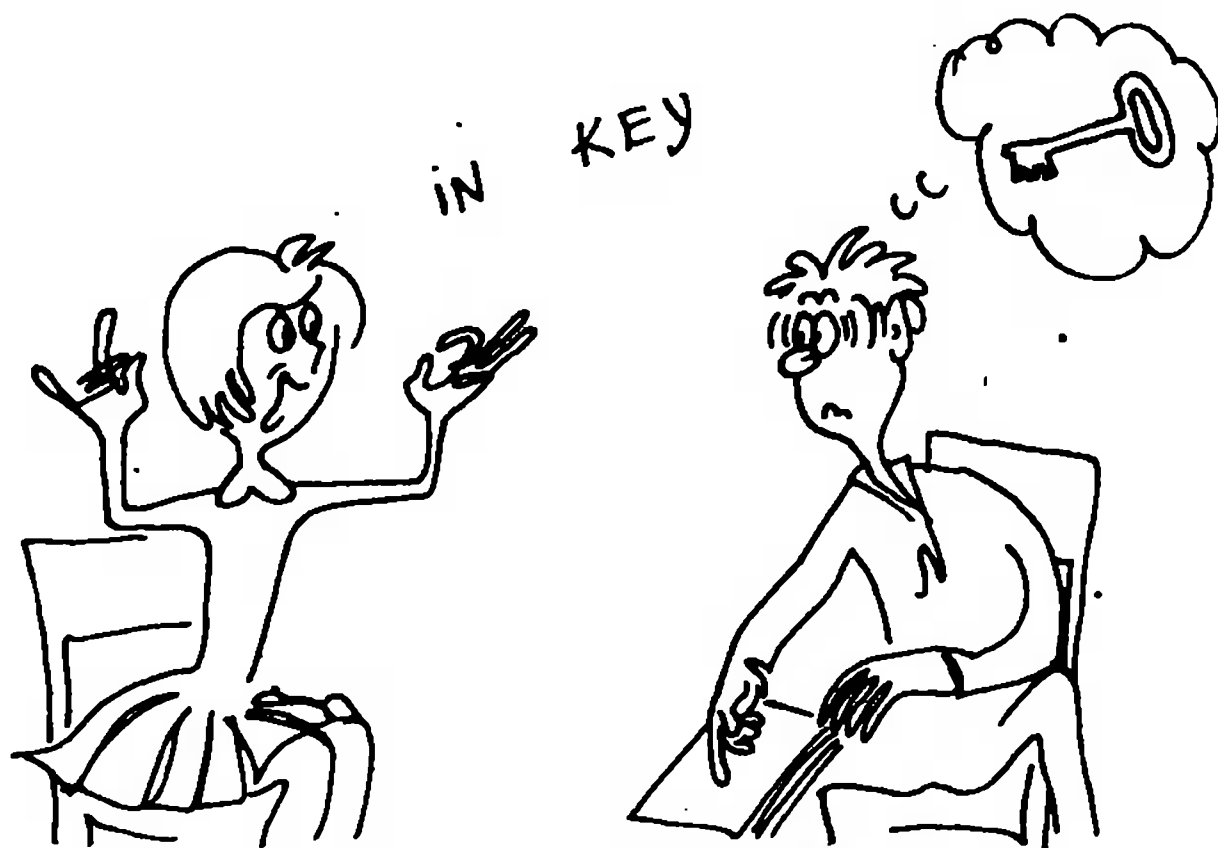
— Мне захотелось тоже узнать, как работают на «Корвете», и я посмотрел описание тамошнего Бейсика, но это место я не понял.

— Прекрасная идея, но это место ты действительно не понял, потому что говоришь «оператор». На самом деле это у них считается переменной. Название этой переменной произносится «инки» и состоит из двух слов: «in», которое означает «в», и «key», которое означает «клавиша». А понять эту переменную проще простого. Ее значением является последняя нажатая на клавиатуре клавиша.

— Это уж ты брось. Клавиша не может быть значением переменной.

— Конечно, конечно. Значением переменной будет код нажатой клавиши. А ты знаешь, что такое код?

— Что-то об этом знаю. В памяти машины фактически



хранятся не буквы, а цифры, которыми эти буквы обозначены. А цифра, которой обозначена какая-нибудь буква, называется кодом этой буквы. Эта цифра, или код, станет буквой только при выводе на экран или печать.

— Не только символам сопоставлены коды, но и каждой клавише. Нет, не так. . . Надо учесть, что при нажатии одной и той же клавиши можно получить различные символы, например строчные или заглавные, латинские или русские буквы. Любой из этих букв сопоставлен свой код. Но, главное, имеются и клавиши, которые никакую букву не дают, например клавиши, предназначенные для управления курсором.

— Да знаю я это. . .

— А знаешь ли ты, как узнать код клавиши?

— Надо посмотреть в таблице кодов. Такая таблица должна быть.

— У кого-то такая таблица есть, но у тебя нет. Что делать?

— Не знаю.

— А я это уже освоила. Надо ввести в машину и исполнить такую маленькую, но интересную программку:

```
10 A X=INKEY X: IF A X="" THEN 10
20 IF A X="" THEN STOP
30 PRINT ASC (A X), A X
40 GOTO 10
```

— Что в одной строке программы можно писать несколько операторов, отделяя их двоеточием, я знаю, но не знаю, что это за функция ASC.

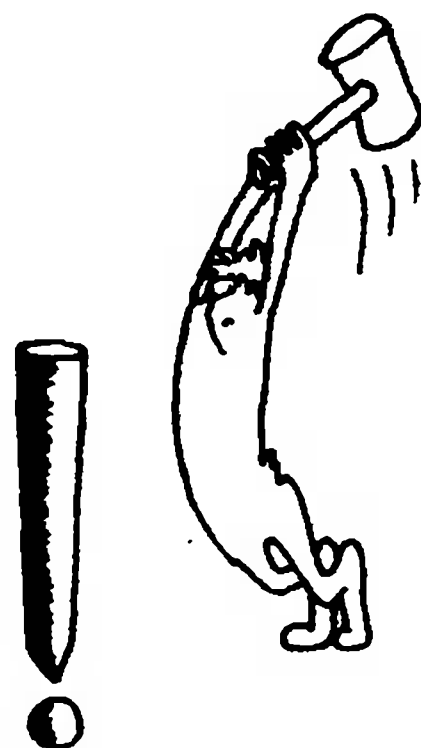
— Подожди же! Посмотри сначала, как используется переменная INKEY X! Видишь, я в строке 10 присваиваю переменной A X значение этой переменной! А этим значением будет код последней нажатой клавиши. И в той же строке я сразу узнаю, нажималась ли какая-нибудь клавиша, ввелось ли что-нибудь в переменную A X.

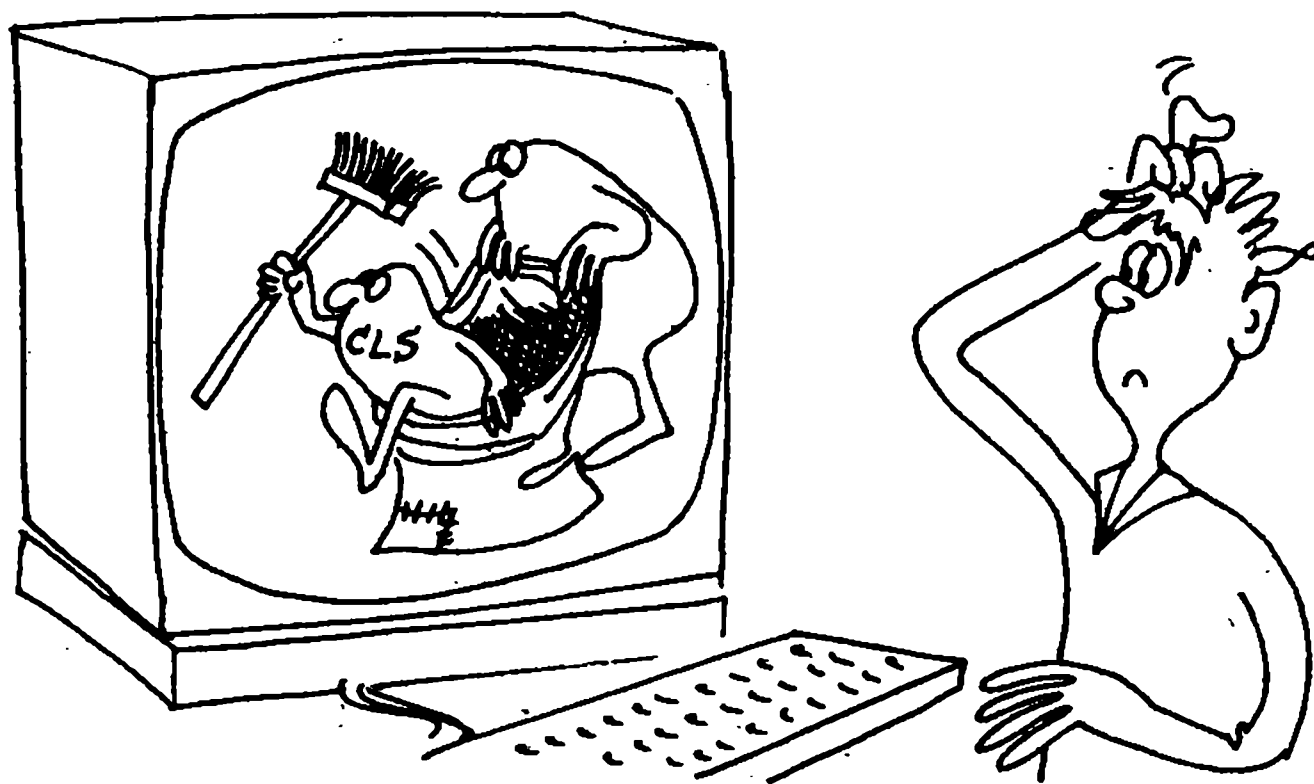
— По-научному говорят: пуста ли строка, являющаяся значением A X?

— Здесь вот что важно. Переменная INKEY X тем отличается от других, что если мы ее значение передавали в какую-нибудь другую переменную, то переменная INKEY X опустошается — ее значением становится, как ты говоришь, пустая строка, будто ничего там никогда не было. Со значениями обычных символьных переменных такого не случается, их надо специально очищать — заслат в них пустое слово.

— Это-то я знаю. Вернемся к программке.

— Ну, вот. Если ни одна клавиша не нажималась, управление передается на начало строки 10, и все повторяется.



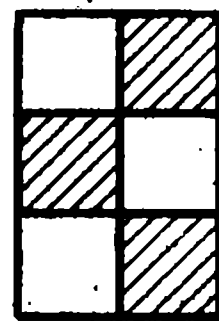
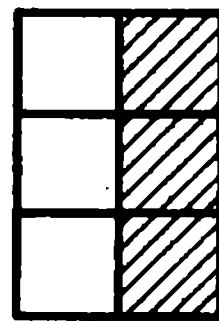
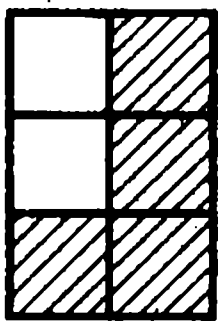
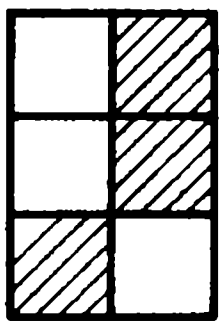


— Э-э-э! Ты ждешь, пока кто-то нажмет клавишу!

— Точно! И если нажал, я печатаю код нажатой клавиши. Для этого мне нужна функция ASC, которая смотрит в то место памяти, где должен храниться первый символ аргумента функции ASC, и вычисляет код. Код символа, если там был символ, или просто код, если это не символ.

— Дай сообразить. Выходит, что значением символьной переменной может быть и вовсе не символ.

— Да, да! Там сидит код, символа или не символа — это станет известным только тогда, когда ты выведешь его на экран. Если это был код буквы, то при выводе на экране появится буква, если это был управляющий код, то произойдет какое-нибудь управляющее действие, например очистится экран, или если это был код псевдографического символа — есть у них такие, то на экране появится маленький прямоугольник с черно-белым орнаментом, состоящим из закрашенных квадратиков. Они получаются, если взять три клетки по вертикали и две по горизонтали и эти шесть клеток по разному закрашивать. Например, так:



— Зачем такие штуки нужны?

— Я толком не знаю. Говорят, из них можно скомбинировать большой рисунок, если надо. Это получится значительно быстрее, чем рисовать графическими операторами. Учти, что между строками обычных символов на экране имеются про-

светы, а псевдографические символы лепятся на экране без промежутков. А какие псевдографические символы существуют в природе и каковы их коды, можно узнать при помощи нашей программы. Видишь, строка 30 выводит не только код символа, но и сам символ.

— Все хорошо, но я так и не могу понять, зачем тебя заставили учить эту науку о кодах.

— И не поймешь, пока я тебе не покажу свою программу-рисовальщик. А то всю жизнь ты все хвастался своими сверхмудрыми программами.

— Ну что ж. Покажи.

— Пожалуйста.

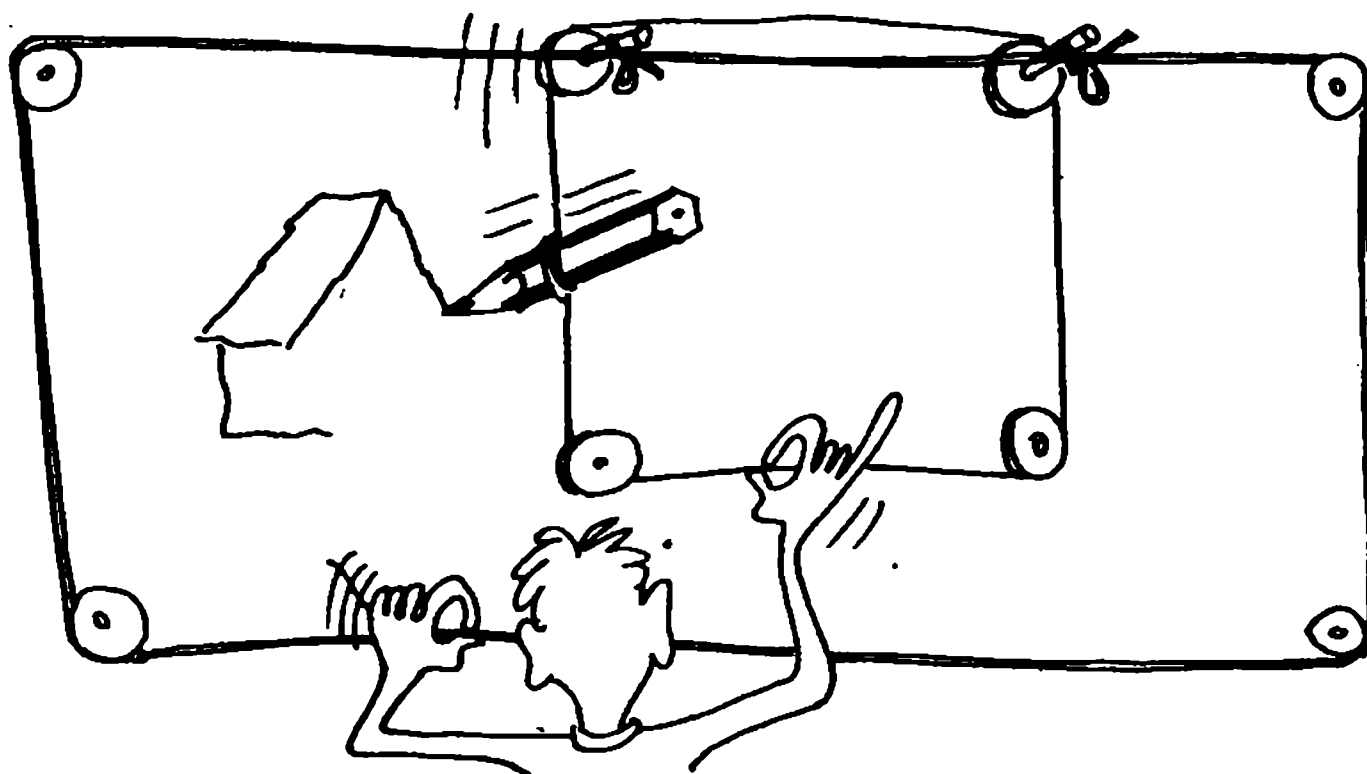
```
10 CLEAR: CLS: COLOR 7,0: PSET (50,50),0: CO=7
20 RIS=7: GOSUB 1000
30 AX=INKEY$: IF AX="" THEN 30
40 A=ASC(AX)
50 RIS=0: GOSUB 1000
60 IF A<>16 THEN 70 ELSE CO=7-CO: COLOR CO,0: BEEP: GOTO 20
70 IF A<>24 THEN 80 ELSE LINE -STEP ( 1, 0): GOTO 20
80 IF A<>8 THEN 90 ELSE LINE -STEP (-1, 0): GOTO 20
90 IF A<>25 THEN 100 ELSE LINE -STEP ( 0,-1): GOTO 20
100 IF A<>26 THEN 110 ELSE LINE -STEP ( 0, 1): GOTO 20
110 IF A<>12 THEN 120 ELSE LINE -STEP (-1,-1): GOTO 20
120 IF A<>4 THEN 130 ELSE LINE -STEP ( 1,-1): GOTO 20
130 IF A<>23 THEN 140 ELSE LINE -STEP (-1, 1): GOTO 20
140 IF A<>21 THEN 150 ELSE LINE -STEP ( 1,-1): GOTO 20
150 IF A<>31 THEN 160 ELSE PCLS: PSET (50,50): GOTO 20
160 IF A<>29 THEN 170 ELSE STOP
170 GOTO 20
1000 IF CO=7 THEN RETURN
1010 LINE -STEP (-2,2),RIS
1020 LINE -STEP ( 4,0),RIS
1030 LINE -STEP (-2,-2),RIS
1040 RETURN
```

— Хэ! Какой примитив! Какое отсутствие хорошего стиля! Нет даже ни одного примечания!

— Примечания у меня были. Но когда я готовила программу для печати на учительской машине, их пришлось убрать, так как они все были по-русски, а печатное устройство не умеет печатать русские буквы. А я не умею писать примечания по-английски. Ты особенно не переживай, все, что надо, я тебе расскажу. Программа рисует на экран любой рисунок.

— Как это любой?!

— Любой, который пользователь программы сам может нарисовать. Для рисования используются клавиши управления курсором, ну те, на которых нарисованы стрелочки. Нажмешь



на клавишу со стрелкой направо, и на экране начнет рисоваться прямая в правую сторону. И так, пока будешь держать клавишу нажатой. Если видишь, что нарисовано достаточно, нажимаешь другую клавишу, например налево вверх по диагонали. И так, пока рисунок не будет готов.

— Да у тебя могут получиться только рисунки, состоящие из непрерывной линии!

— Будь спокоен! Такие мелочи Тоня Соображалкина в состоянии предусмотреть. Запрограммировано так, что нажатие центральной клавиши с цифрой 5 и надписью МЕНЮ переводит программу в режим, когда графический курсор передвигается, не рисуя. А чтобы знать, куда мы передвинулись, в этом режиме я рисую маленькую стрелочку там, где находится графический курсор. Смотри программу, все увидишь.

— CLEAR в строке 10 я понимаю. Этой командой очищается память — значения числовых переменных становятся равными нулю, а значения символьных переменных — равными пустой строке.

— Это не все. После слова CLEAR можно написать арифметическое выражение, оно будет подсчитано, и результат определит, сколько символов ты сможешь хранить во всех своих символьных переменных. Это может иметь значение, если ты собираешься обрабатывать очень много символов. Если ничего не делать, интерпретатор отводит место для 100 символов.

— Что команда CLS очищает экран, я знаю...

— Символьный, чтобы ничего не мешало рисовать.

— Устанавливается белый цвет переднего фона и черный цвет заднего фона оператором COLOR. А этот PSET я не помню.

— Это оператор рисования точки. После PSET пишут координаты рисуемой точки и цвет этой точки.

— Стоп! Ты же ее рисуешь как раз цветом заднего фона. Она же не будет видна!

— Да, не будет! А мне и не надо. Я просто хочу начинать рисование с точки (50, 50). Это так, каприз. А вот переменная C0 — это уже серьезно. Она означает цвет, которым я рисую основной рисунок, а не стрелочку. Сейчас я собираюсь рисовать и поэтому задаю этот цвет равным цвету переднего фона — белому, а когда захочу просто передвигать графический курсор без рисования или стереть неправильно нарисованную часть рисунка, сделаю значение C0 равным нулю, т. е. буду рисовать цветом заднего фона — черным.

— А переменная RIS в строке 20 для чего?

— Это цвет, которым я рисую маленькую стрелочку, отмечающую местоположение графического курсора, когда основной рисунок не рисуется. Эту стрелочку я передвигаю вместе с нерисующим курсором. А что такое передвигать рисунок по экрану? Это нарисовать в одном месте, потом стереть, а потом нарисовать в другом, стереть... И т. д. Рисует эту стрелочку подпрограмма, а переменная RIS определяет, получится ли в самом деле рисование или стирание. Сейчас ее значение 7, так что получится рисование.

— Вот и не получится. Ты сразу уходишь на подпрограмму, начинающуюся со строки 1000, а там возврат, если C0 равно 7, как раз как у тебя сейчас.

— Получится рисование, если я перейду на режим передвигания курсора. Признаком этого режима служит переменная C0.

— В строке 30 ты определяешь, какую клавишу нажал пользователь.

— Ты неточно выражаешься. Я здесь просто жду нажатия клавиши. Определение нажатой клавиши получается потом, после того как в строке 40 я нашла код нажатой клавиши с помощью функции ASC.

— В строке 50 ты переменной, которая означала цвет стрелочки, присваиваешь значение нуль и сразу эту стрелочку рисуешь.

— Я ее стираю. Потом анализирую введенный код, передвигаю графический курсор, куда надо, и возвращаюсь на строку 20, в которой рисую стрелочку уже в новом месте, и перехожу на строку 30 ждать нажатия клавиши.

— Логика понятна, так что можно смотреть, как ты изучаешь полученный код. Эти операторы ветвления в последующих строках мне непривычны. Я, конечно, знаю, что «элз» означает «иначе» и что оператор IF... THEN... ELSE считается намного лучше, чем просто IF... THEN.

— Намного ли, не знаю, но лучше. Здесь после THEN нужно писать то, что делаешь, если логическое условие выполняется, а после ELSE то, что надо делать, если это условие





не выполняется. Притом после THEN можно писать даже несколько операторов, но только они должны поместиться в одной строке.

— А после ELSE?

— Бейсиков! Это же каждому понятно, что можно. Можно писать в той же строке программы и продолжить в следующей строке или в следующих строках. Кстати, можно и писать по-старому — без ELSE с одним только THEN.

— И по-видимому, можно писать после THEN один только номер строки вместо полного оператора с GOTO плюс номер строки.

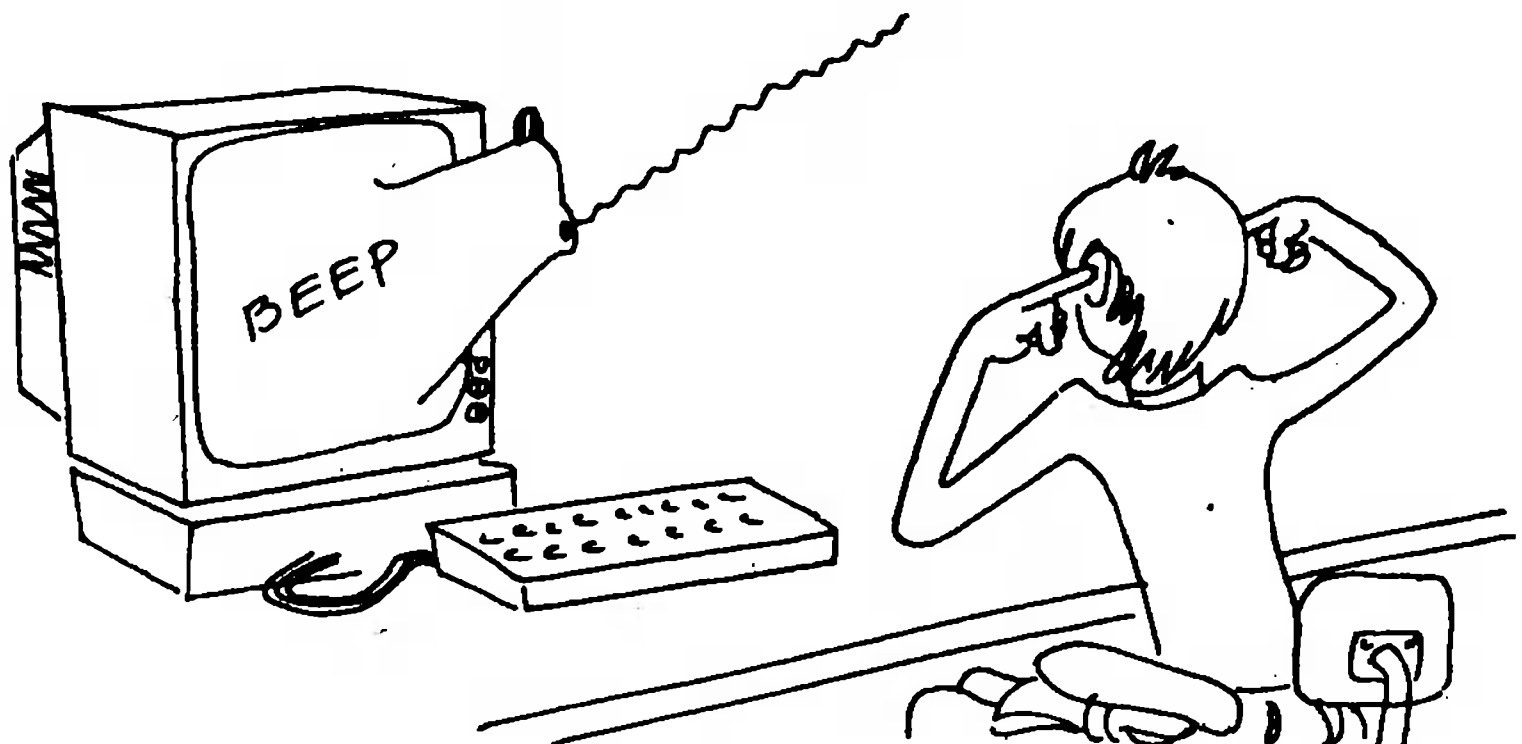
— Да. В строке 60 оператор ветвления передает управление следующей строке 70, если код нажатой клавиши не равен 16, т. е. если нажимали клавишу, отличную от центральной клавиши с цифрой 5 в дополнительной клавиатуре справа.

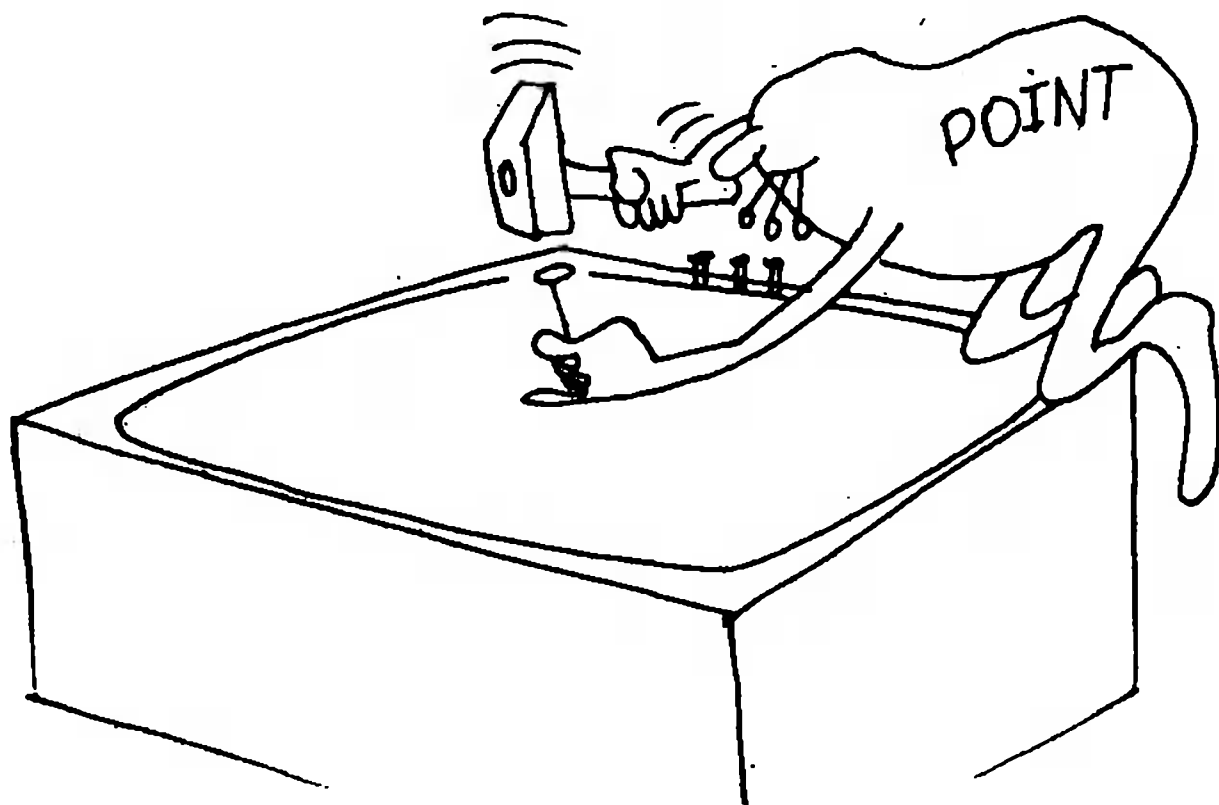
— Ты говорила, что эта клавиша отключает режим рисования и переводит работу на передвижение графического курсора без рисования.

— Я и сейчас от своих слов не отказываюсь. Дополнию только, что эта клавиша в режиме передвижения включает режим рисования. Это достигается тем, что я новое значение переменной C0 получаю, отнимая старое от семерки. Если старое было 7, получу 0, если старое было 0, получу 7. Хитро?

— Хитро. И потом новое значение C0 используешь в качестве номера нового цвета переднего фона. А что потом пишешь? Ведь оператор BEEP, кажется, заставлял машину пищать.

— Действительно, «бип» означает «пищать», и писк на самом деле происходит. А пишу я для того, чтобы пользователь знал, что произошла смена режима. И еще, если по неопытности пользователь будет эту клавишу держать нажатой слишком долго, то произойдет обратное переключение режима, о чем его известит еще один писк.





— Ну, строки с номерами от 70 до 140 все похожи, все передвигают курсор на один шаг в нужном направлении.

— Строка 150 тоже понятна. После нажатия клавиши стирания CTRN сверху справа дополнительной клавиатуры жизнь начинается сначала — графический экран гасится и управление передается на начало программы. А из строки 160 видно, что после нажатия клавиши ИЗ сверху слева дополнительной клавиатуры работа программы прекращается.

— А почему ты там после оператора STOP еще пишешь оператор GOTO?

— Так просто. Если после остановки программы пользователь опомнился, что останавливаться не надо было, он сможет ввести команду CONT и работа будет продолжена, как будто ничего не было.

— На строку 170 мы приходим, если пользователь нажал клавишу, которую программа не использует. Так я понимаю?

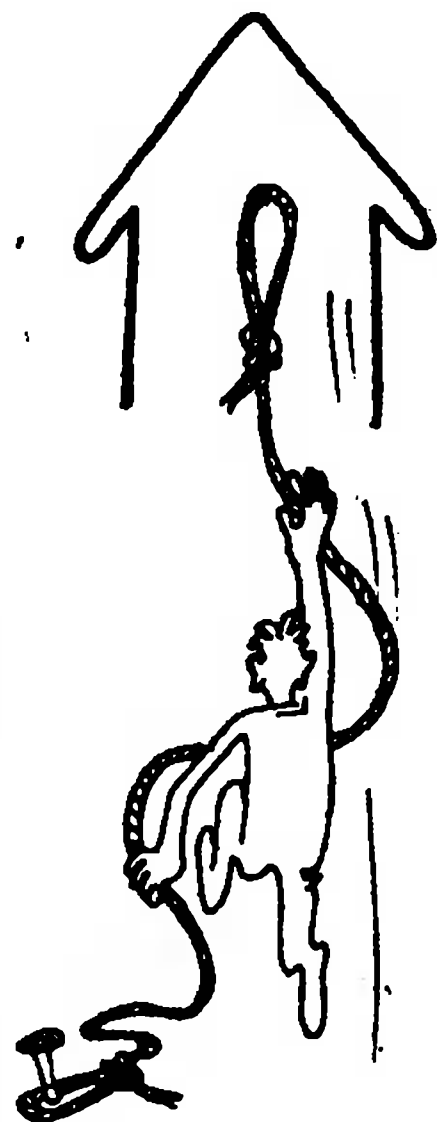
— Все верно. Тогда я спокойно ожидаю следующего нажатия клавиши.

— А как мы могли бы «проразмыслить» твою программу?

— Программа, как видишь, совсем маленькая, и делает она тоже немного. И должна самокритично заметить, имеет серьезный недостаток. В режиме передвижения графического курсора может получиться неприятная вещь. При передвижении стрелочки, когда она периодически рисуется и стирается, может стереться и основной рисунок, если я передвигаю стрелочку так, что она захватывает основной рисунок.

— А что тут можно сделать? . .

— Можно было бы попробовать использовать функцию POINT, которая дает цвет точки, если ее координаты задать в скобках после слова POINT. Но задавать можно только абсолютные координаты, в то время как мы их не имеем, поскольку все время двигаемся относительно той точки, в которой находимся.



— Подожди чуть-чуть! Ведь можно было бы двигаться и параллельно рассчитывать координаты. Сдвинулись на одну единицу вправо, добавляем к координате x единицу, сдвинулись влево, отнимаем.

— Конечно, так можно. Тогда пришлось бы перед рисованием стрелочки определить, какого цвета точки она пересечет, запомнить эти цвета и, стерев стрелочку, восстановить их.

— Много пришлось бы повозиться.

— Еще бы! Значительно быстрее можно добавить возможность закрашивания областей. Мы рисуем контур области, потом передвигаемся внутрь ее и нажимаем клавишу, которая вызывает закрашивание этой области.

— Надо использовать оператор. . . оператор. . . PAINT.

— Точно. Еще можно было бы сделать режимы ускоренного движения по нескольку шагов в указанном направлении, а то сейчас уснуть можно, пока нарисуешь, скажем, прямую вдоль всего экрана. Как видишь, работы программистам на «Корветах» хватит.

— Кстати, о работе. Я принес задачки по программированию. Давай посмотрим, может быть, найдешь интересную для себя проблему.

25. ЧТО ЕЩЕ МОЖНО СДЕЛАТЬ

— Задачи тут разные. Имеются и простые, и посложнее. Простые задачи, наверное, тебя не будут интересовать, но еще неизвестно, сможешь ли ты определить, какая задача простая, какая сложная.

1. И г р а с ч и с л а м и. Машина «задумывает» одно число от 1 до 1000. Игрок должен это число угадать максимум за десять попыток. Каждый раз, получив число, программа сообщает, больше оно или меньше «задуманного». Придумать алгоритм игры.

2. Та же игра, но только сейчас число задумывает игрок, а программа угадывает.

3. Те же игры, что и в предыдущих задачах, но сейчас можно «мухлевать» — один раз за всю игру можно дать неправильный ответ. Придумать алгоритм игры за угадывающего и задумывающего. Могут быть два варианта игры: когда игрок обязательно один раз за игру «мухлюет» и когда может «мухлевать», а может и не «мухлевать».

4. С п о р т л о т о. Составить программы для любителей

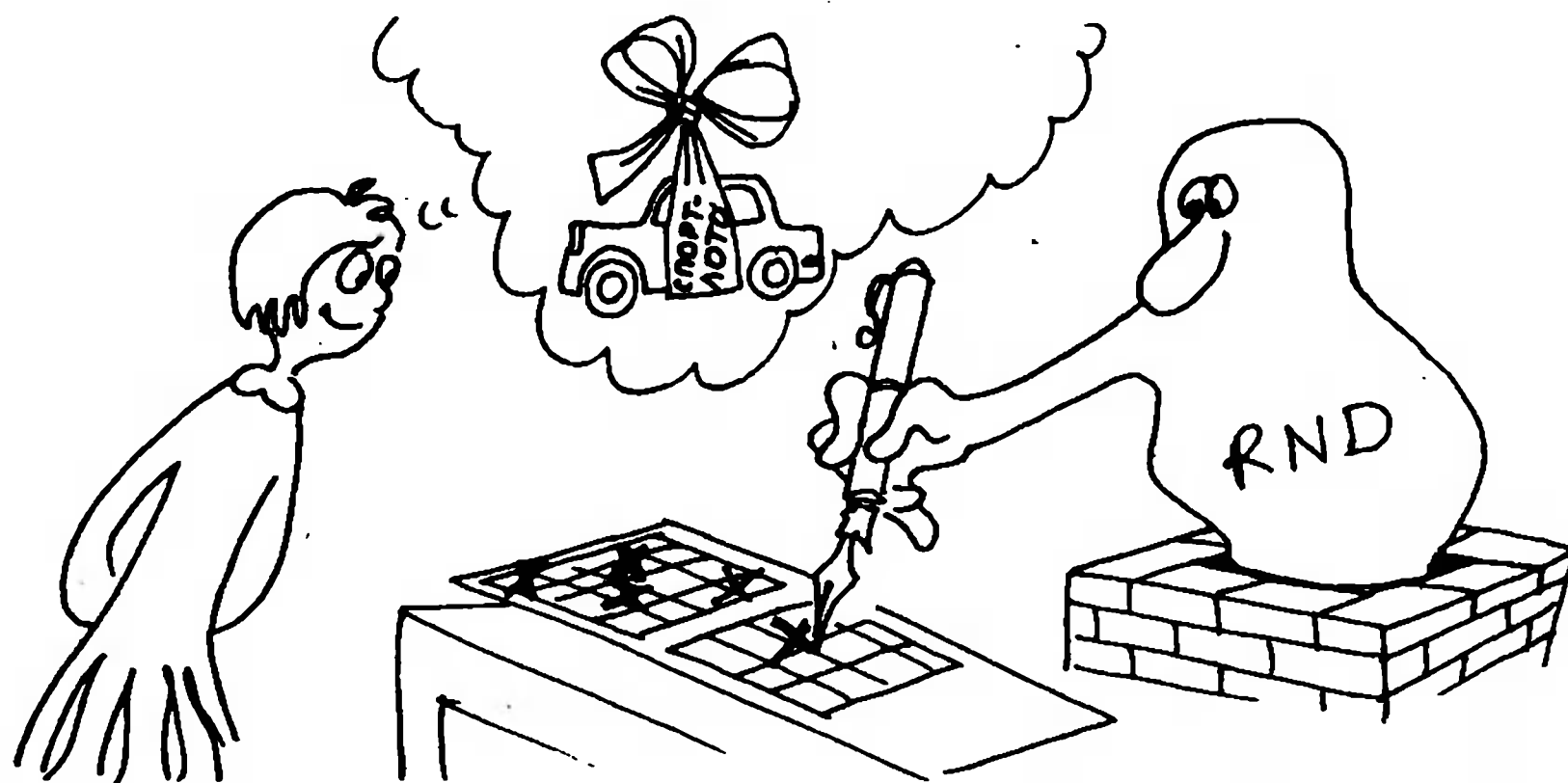
этой игры. Одну, которая печатает 6 из 45 и 5 из 36 случайных чисел. Вторую, которая на диске собирала бы информацию о всех розыгрышах — какие числа оказались счастливыми, искала бы закономерности в их появлении и потом прогнозировала бы выигрышные числа следующего розыгрыша.

5. Запрограммировать такую игру: первый игрок называет число в пределах от 1 до 10, второй прибавляет к нему число в пределах от 1 до 10 и сообщает сумму и т. д. Кто первый из игроков назовет 100, тот победитель. Программа может играть за первого и за второго игроков. Придумать алгоритм, обеспечивающий победу, и запрограммировать его. Составить еще одну программу, которая на победный алгоритм переходила бы не сразу, чтобы противник не догадался, каков алгоритм. А если программа «видит», что противник уже этот алгоритм знает, то переходит сразу же на победный алгоритм.

6. Болгарская народная задача. Один крестьянин имеет 25 буйволиц. Первая из них в день дает 1 л молока, вторая — 2 л, и т. д. до двадцать пятой. Крестьянин должен справедливо разделить буйволиц между своими пятью сыновьями, так чтобы каждый получил одинаковое количество буйволиц и получал от них одинаковое количество молока в день.

7. Составить программу, которая число от 1 до 10 000, введенное цифрами, выводила бы на русском, английском, немецком языках и наоборот. Например, вводим «25», выводит «двадцать пять», и, наоборот, вводим «двадцать пять», выводит «25».

8. Составить программу перевода чисел от 1 до 1000, записанных арабскими цифрами, в числа, записанные римскими цифрами, и наоборот.



9. В машину вводятся три числа. Программа должна определить, могут ли эти числа быть длинами сторон треугольника. Будет ли этот треугольник равносторонним или равнобедренным? Прямоугольным, тупоугольным или остроугольным? Какова его площадь? Найти высоты треугольника. Можно ли из этих высот составить новый треугольник? Если можно, то будет ли новый треугольник равносторонним или равнобедренным? (Квадратный корень находит функция $SQR(X)$.)

10. Игра «Три дракона». Программа выводит два целых случайных числа в пределах от 1 до 100. Играющий должен ввести третье число. Если все три числа могут быть длинами сторон треугольника, то играющий получает 3 очка, если не годятся, то теряет 33. Если треугольник окажется равнобедренным, то добавляются еще три очка, а если равносторонним, то 33. Если треугольник прямоугольный, то играющему присуждается победа с титулом «троекратный гротсмейстер». Для обычной победы надо собрать $33 \cdot 3 + 3$ очка. Игра осложняется тем, что могут появиться три дракона. Для того чтобы определить, появится ли какой-нибудь из драконов, выбирается случайное число 1, 2 или 3. Это делается три раза сразу после ввода играющим своего числа. Если в первый раз выпадет 3, то появляется первый дракон. Из основных трех чисел он выбирает то, которое делится на 3 (если такое есть), и уменьшает его на одну треть. Если во второй раз выпадает 3, то появляется второй дракон и увеличивает число, которое делится на 3 (если такое есть), на одну треть. Если же в третий раз выпадет 3, то третий дракон меньшее число умножает на 3.

Если периметр треугольника оказывается равным 33, то выбирается случайное число 1, 2 или 3. Дракон с этим номером проваливается в тартарары — больше появиться не может.

11. Быки и коровы. Программа выбирает четыре случайных числа от 1 до 9 и составляет из них строку символов. Играющий должен ее отгадать, вводя свои предположения. После каждого ввода программа сообщает, сколько цифр угадано точно — сколько «быков» и сколько цифр названо правильно, но помещено в неправильных местах — сколько «коров». Например, программой выбрано число 6834, тогда игра может проходить, например, так:

Догадки играющего	Ответы программы
3333	1 бык
3366	2 коровы
6633	2 быка
7633	1 бык, 1 корова
6833	3 быка
6843	2 быка, 2 коровы
6834	Угадано за 7 ходов

12. Соревнуются два игрока. Первый отворачивается, а второй вводит в машину какое-нибудь слово. Программа гасит экран, перепутывает в слове буквы случайным образом и выводит полученное слово на экран. Например, введено слово КОНКАТЕНАЦИЯ, выводится КАТОКНАЯИЦЕН. Первый игрок должен угадать первоначальное слово и ввести его. Потом игроки меняются местами. Программа находит время, затраченное каждым игроком на обдумывание, и количество ошибок.

13. Игра «Виселица». Соревнуются два игрока. Первый отворачивается, а второй вводит в машину какое-нибудь слово. Программа гасит экран и выводит это слово, зашифровав черточками все буквы, кроме совпадающих с первой и последней. Например, если введено слово ИНТЕЛЛИГЕНТ, то выводится И-Т---И---Т. Первый игрок угадывает буквы по одной, и программа определяет, годятся ли они в каком-то месте или нет. Если годятся, то соответствующая черточка заменяется правильной буквой. Фиксируется количество попыток.

14. «Искусственный интеллект». Программа выводит на экран предложение поговорить. Если в ответ вводится предложение, оканчивающееся восклицательным знаком, программа выводит: «Пожалуйста, поспокойнее!» Если в конце предложения вопросительный знак, выводится: «Нужно подумать». В других случаях программа с одинаковой вероятностью отвечает: «Да, я об этом слышала» или «Ну и что?» После каждого ответа следует просьба: «Поговорим еще, мне скучно».

15. Проверка памяти. Программа показывает случайно выбранную букву и через короткое время гасит экран. Играющий должен ввести ту же самую букву. Потом программа показывает две случайно выбранные буквы. И т. д., пока играющий не ошибется. В конце выводится, сколько букв он сумел запомнить.

16. Статистическое моделирование. Автомшины одинаковой длины останавливаются на обочине улицы длиной M в случайных местах. Сколько автомашин в среднем вместит такая стоянка, если одна автомашина занимает K метров? Составить программу для моделирования такой ситуации на ЭВМ: на отрезок $[0; 1]$ бросаются отрезки меньшей длины: K/M . Всякий раз, когда какой-нибудь из них падает на ранее упавший, бросание повторяется. Процесс прекращается, если не осталось сплошного свободного места длиной более K/M . В конце подсчитывается количество маленьких отрезков, вошедших в отрезок $[0; 1]$. Процесс повторяют 1000 (или более) раз и находят среднюю вместимость отрезка $[0; 1]$.

17. Составить программу для хранения на диске результатов игр футбольного или хоккейного чемпионата СССР или



мира. Программа по требованию должна выпечатывать текущее состояние турнира.

18. На языке эсперанто все глаголы в настоящем времени оканчиваются на *-as*, в прошедшем — на *-is*. Других слов с такими окончаниями нет. Составить программу, которая текст на языке эсперанто преобразовала бы из прошедшего в настоящее время. Учесть, что после слова может быть не только пробел, но и знаки препинания. Нужно ли знать язык эсперанто для решения этой задачи?

19. Разработать новый язык программирования БОРЯ (отображение Бейсика на русский язык). В этом языке слова в операторах пишутся не на английском, а на русском языке. Например, вместо *GO TO* пишется *ИДИ НА*, вместо *RETURN* пишется *ВОЗВРАТ* и т. д. Составить программу, которая переводила бы программы на *БОРЕ* в программы на Бейсике. Программа на *БОРЕ* записана в файле на диске — каждая строка в виде строки символов. И наоборот, на диске записана программа на Бейсике — считать ее, перевести в программу на *БОРЕ* и записать в файл на диске.

20. Составить 20 программ для обучения и повторения материала уроков текущей четверти. Можно делать и так, чтобы обучение проходило в виде какой-то игры, подобно игре «Три дракона» (см. задачу 10).

— 20 программ! Ничего не скажешь, остроумно! Больше нельзя было придумать?

— Почему бы и нет! Неужели ты бы не составила?

— Кто его знает... А вообще пора кончать.

Был погожий летний вечер. Над городом только что пролетела гроза, и солнце спешило до вечера высушить мокрые тротуары. Никто не обращал внимания на юношу и девушку, которые под липами бульвара не спеша шли в сторону вокзала. Внезапно девушка резко повернулась и воскликнула:

— Знаешь что! Жизнь прекрасна еще и потому, что человек может программировать!

И так как эту истину первой открыла не она, то ни Петя, ни плескающиеся в лужах воробьи не возражали.

26. ЧЕМ ВСЕ ЭТО КОНЧИЛОСЬ

Нет, не пришлось Тоне поработать на «Корветах». Действительно, школа «Корветы» получила, но шефы подарили и еще один компьютер, совместимый с компьютерами фирмы IBM (ай-би-эм), и Тоня вызвалась его осваивать. Она уже хорошо знала, что таких компьютеров в окружающем мире намного больше, чем «Корветов», и опыт работы на них может пригодиться после школы. Кроме того, работать на этой, более мощной машине, казалось, будет интереснее.

Слово «совместимый» здесь означает, что этот компьютер работает совершенно так же, как те, которые производит фирма IBM, хотя в действительности производитель может быть совсем другим. Другими словами, программы, которые разработаны на совместимых компьютерах, должны работать на них независимо от того, кто эти компьютеры сделал.

Конечно, опыт программирования, полученный на предыдущих занятиях, Тоне здорово пригодился, в особенности корветный, но пришлось освоить и много нового. Но это было совсем нетрудно.

Разве трудно понять, где у компьютера устройства ввода с дискеток (таких устройств обычно два) и как вставлять в них дискетки. Их видно невооруженным глазом. Чуть труднее с твердым диском, или винчестером: его не видно, потому что он спрятан внутри корпуса, но это не страшно, так как с ним ничего руками делать не надо, программисты работают только с его содержанием. Твердый диск ведь заменить нельзя, он остается одним и тем же на все время жизни компьютера.

Если же говорить о содержании твердого диска и дискеток, то это файлы, которые на них записаны. Первое, что хочется узнать, имея в руках дискетку,— это что на ней записано, сохранилось ли то, что записывали вчера. Но даже для этого вообще-то простого дела надо знать немножко операционную систему ДОС (Дисковая Операционная Система), которую применяют на персональных компьютерах.

ОПЕРАЦИОННАЯ СИСТЕМА ДОС

Операционная система начинает свою работу сразу после включения компьютера, как говорят программисты — происходит ее загрузка. Когда загрузка кончилась и компьютер готов к работе, последняя выведенная на экран строка будет скорее всего иметь вид:

```
C:\>_
```

Сначала про букву С. Каждое дисковое устройство на компьютере обозначается или кодируется буквой латинского алфавита (А, В, С и т. д.) и двоеточием после нее. Буквами А и В обозначаются устройства ввода с дискеток, а буквой С — твердый диск. Так вот, написанной выше строкой компьютер говорит пользователю: «Все нормально, можешь работать с твердым диском». Знаки «\>» в конце строки нужны только для того, чтобы выделить сообщения системы.

Тоне дали дискетку, на которой можно было хранить все, что угодно. Поэтому после загрузки системы она переходила на одно из устройств для ввода с дискеток. Для этого после системного сообщения надо написать обозначение этого устройства и двоеточие. Двоеточие обязательно! Получится так:

```
C:\>A:_
```

Спешить нажимать клавишу ввода не надо. Сначала надо вставить дискетку в то устройство, которое обозначается буквой А. Для этого, конечно, надо знать, какой буквой каждое из этих устройств обозначено. Если все правильно, то после нажатия клавиши ввода на следующей строке экрана появится:

```
A:\>_
```

Значит, все в порядке, можно работать с той дискеткой, которая в устройстве А. Конечно, таким же образом можно перейти к устройству В или вернуться назад к твердому диску С.

Чтобы вывести на экран содержание дискетки — перечень файлов, записанных на ней, необходимо ввести команду DIR или, если больше нравится строчными буквами, dir, что является сокращением английского слова «directory», означающего — каталог.

```
A:\>dir_
```

Сейчас же на экране появится каталог дискетки — аккуратный столбик с названиями файлов, их расширениями, длиной, измеряемой количеством символов, датой и временем

создания. Например, для одного файла это может выглядеть так.

GWBASIC EXE 80848 4—07—87 3.45p

В системе ДОС имеются расширения файлов с заранее оговоренным значением. Например, расширения EXE или COM означают, что эти файлы являются готовыми для работы программами и их можно запустить, написав так:

A:\>GWBASIC__

Можно даже не писать само расширение. После нажатия клавиши ввода программа GWBASIC будет считана с дискетки и начнет работу. Более того, в компьютере Тони было так сделано, что если на дискетке такого файла нет, то система начинает искать его на твердом диске.

Кроме расширения EXE еще есть расширение BAS, которое дается файлам с текстами программ на языке Бейсик, PAS — на Паскале, TXT — для файлов, содержащих тексты на русском, английском или другом естественном языке, и много других.

БЕЙСИК НА ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРАХ

Бейсик на персональных компьютерах имеется, и притом с весьма полными и богатыми возможностями. Порой кажется, что этих возможностей даже слишком много. К сожалению, интерпретаторы Бейсика бывают разные. Каждая фирма, производящая программы, считает своим долгом сделать и интерпретатор Бейсика.

Ближе всего к Бейсикам персональных компьютеров MSX Бейсик, который осваивала Тоня на «Корветах». Можно сказать, что фирма — производитель «Корветов» — пыталась сделать компьютер, совместимый с персональным компьютером — IBM. Довольно хорошо это удалось с учительским компьютером, к которому Тоню, к сожалению, не подпустили. Так что при переходе на персональный компьютер ей пришлось осваивать дополнительно запись на диск и вывод на печать. Но пришлось что-то и позабыть. Например, на «Корвете», были два отдельных экрана — алфавитно-цифровой и графический в одном лице. А на персональном компьютере один экран, но с двумя режимами — текстовым и графическим. Ничего страшного, но при переходе к графическому режиму для рисования весь экран гасится. Для Тони это создавало неудобства, когда при рисовании хотелось что-то на рисунке и написать.

ЭКРАННЫЙ РЕДАКТОР

Особенно интересным Тоне показался экранный редактор программ Бейсика. Вводишь,водишь программу в машину,

потом выведешь кусочек на экран директивой LIST, посмотришь — ой, ошибка! Ничего страшного. Курсор-то может гулять по всему экрану. Спокойно подводишь его к нужному месту, пользуясь клавишами со стрелками, исправляешь, что надо, и нажимаешь клавишу ввода. Строка, в которой в этот момент находился курсор, отправляется в память компьютера на свое место, определяемое номером. И даже номер можно поменять. Тогда после нажатия клавиши ввода возникнет в программе новая строка с этим новым номером, а со старым номером, конечно, останется неизменной. Таким образом можно получать похожие строки, сколько хочется.

Исправлять тоже удобно. Имеется на клавиатуре специальная клавиша удаления символа с надписью DEL на ней. Если нажать эту клавишу, то символ в том месте экрана, где находится курсор, исчезает, а остальные символы сдвигаются. Можно писать поверх неправильного текста, а можно и впихивать забытые символы внутрь строки. Для этого нужно только нажать клавишу INS, курсор станет больше, чтобы показать пользователю, что работа идет в режиме вставки, и теперь введенный символ будет сдвигать вправо все символы той же строки. Хочется опять писать поверх текста — пожалуйста, достаточно только еще раз нажать клавишу INS, курсор станет опять маленькой черточкой, и установится режим перезаписи.

Главная трудность у Тони была в том, что она забывала нажать клавишу ввода после исправления каждой строки. Бывало, после исправления всего экрана она обнаруживала повторным просмотром, что исправлена только одна последняя строка, в которой она нажала клавишу ввода, остальные же строки стоят, как стояли, ошибочными, и приходится повторить нудную процедуру исправлений.

Помощь при вводе оказывает и возможность ввода целого слова, нужного в Бейсике, при одновременном нажатии клавиши ALT и буквы или при нажатии функциональных клавиш F1, F2 и так далее до F10. Например, ALT и буква «P» дают сразу слово PRINT, а F1 дает LIST. Самое фантастичное, что пользователь может сам менять значения функциональных клавиш, если ему не нравятся придуманные создателями интерпретатора. Для этого существует специальный оператор KEY, о котором написано дальше.

Есть и другие полезные клавиши. Скажем, нажатие клавиши ESC очищает ту строку экрана, в которой находится курсор. Одновременное нажатие клавиши CTRL и клавиши HOME очищает весь экран. Клавиша END перебрасывает курсор в конец, а клавиша HOME — на начало той строки, в которой курсор сейчас находится.

Были еще и другие возможности, которые Тоня, усердно работая, хорошо освоила.

27. ИЗ ЗАПИСНОЙ ТЕТРАДКИ ТОНИ

Работая с персональным компьютером, Тоне однажды пришла в голову гениальная мысль — записать в тетрадке краткую информацию обо всех операторах, о которых что-то узнала. А то программируешь, программируешь... и вдруг засомневаешься — так надо писать этот оператор или не так. Тогда записи могут помочь.

Для того чтобы укоротить описание этих операторов, Тоня воспользовалась следующими обозначениями, которые, правда, придумала не она:

Слова и символы, заключенные

в квадратные скобки [], не обязательны — оператор будет работать даже, если в этом месте ничего не написать;

в угловые скобки <>, поясняют, что в этом месте писать при использовании этого оператора в программе.

Ни те, ни другие скобки не являются частью оператора и при программировании не должны в нем появляться.

НС означает произвольный номер строки программы,

ОП — произвольное обозначение переменной или элемента массива,

АВ — произвольное арифметическое выражение,

СВ — произвольное символьное выражение.

Хотя, записи в тетрадке совсем не полные, но для успешной работы Тоне их хватило.

Вначале она вписала два напоминания себе: Нельзя забывать, что все операторы, которые что-то рисуют, могут это делать только в графическом режиме! Нельзя забывать, что в графических операторах слово STEP определяет, что задаваемые координаты будут относительными! Потом шли операторы, расположенные в алфавитном порядке ключевых слов, а потом функции и директивы.

ОПЕРАТОРЫ

<ОП> = <АВ>

С этим оператором все ясно. Присвоение оно и есть присвоение.

BEEP

И с этим оператором все ясно — он издает небольшой писк.

CIRCLE [STEP] (x, y), <радиус>[, <цвет>]

Рисование окружности с координатами центра x и y, указанного радиуса и указанного цвета, который можно не задавать, если он будет цветом переднего фона. Этот оператор может быть еще сложнее, чтобы рисовать эллипсы и даже дуги эллипсов.

CLOSE [**<номер файла>**] [**<номер файла> . . .**]

Закрываются файлы с указанными логическими номерами. Если номеров нет, то закрываются все файлы, открытые в данный момент.

CLS

Весь экран очищается и закрашивается цветом заднего фона.

COLOR [**<передний фон>**] [**<задний фон>**]

В текстовом режиме задаются номера цвета выводимых символов (передний фон) и цвета заднего фона. Номер любого цвета может отсутствовать — тогда будет сохранено текущее значение. Если отсутствует номер цвета переднего фона, запятая после него должна оставаться. Например: **COLOR ,2** задаст цвет заднего фона номер 2, а цвет переднего фона останется каким был. Для цветного дисплея законны 16 цветов — с номерами от 0 до 15, а для черно-белого — цвета с номерами от 0 до 7, но номера от 0 до 6 фактически обозначают черный цвет и только номер 7 — белый.

COLOR [**<задний фон>**] [**<набор цветов>**]

В графическом режиме задаются номера цвета заднего фона и набора цветов, которыми можно рисовать на экране. Таких наборов для цветного дисплея обычно 2, и состоят они из 3 различных цветов.

DATA **<список значений>**

В списке значений хранилища данных надо записать данные в том порядке, в каком их будут считывать операторы **READ** в программе. Данные могут быть как числовые, так и символьные.

DEF FN**<имя>** [**<аргумент>**] [**<аргумент> . . .**]

= <выражение>
Для дальнейшего использования определяется функция от переменных программы, которую можно записать в одной строке. К ней потом можно будет обращаться, написав **FN** и ее имя, а потом в скобках те переменные (или выражения), со значениями которых потребуется проводить работу. «Имя» — допустимое обозначение числовой или символьной переменной, «аргумент» — тоже обозначение, а «выражение», естественно, допустимое выражение, которое должно быть арифметическим или символьным в зависимости от того, какое имя задавали самой функции.

DIM **<название массива>** (**<список размерностей>**) [**. . .**]

Объявляются массивы, и занимается для них место в памяти. Пишется название массива и после него, в скобках, максимально возможные значения его размерностей, отделяемые запятыми. Потом через запятые идут описания других нужных массивов.

END

Является сигналом интерпретатору, что нужно закончить работу программы окончательно и бесповоротно. Все открытые файлы закрываются.

FOR <переменная цикла> = <начало> **TO** <конец> [**STEP** <шаг>]

Последующие операторы до оператора конца цикла **NEXT** будут выполняться несколько раз для значений переменной цикла, начиная от значения арифметического выражения «начало», с приращением, равным значению выражения «шаг», до тех пор, пока переменная цикла не станет больше значения выражения «конец».

GET [**STEP**] (x1, y1) — (x2, y2), <название массива>

В указанном массиве чисел сохраняется рисунок, расположенный в прямоугольнике, координаты левого верхнего угла которого задаются числами x1 и y1, а координаты правого нижнего угла — числами x2 и y2. Массив должен быть достаточно большим. Потом этот рисунок можно будет восстановить оператором **PUT**, и притом даже в другом месте, что создаст эффект передвижения рисунка.

GOSUB <НС>

Передача управления подпрограмме, которая начинается с указанной строки. Возврат назад осуществляется оператором **RETURN**.

GOTO <НС>

Передача управления строке с указанным номером. Если такого номера нет, то выдается сообщение об ошибке.

IF <условие> **THEN** <операторы> [**ELSE** <операторы>]

Если «условие» истинно, то выполняются все те операторы, которые идут после **THEN**, а если оно ложно, то те, которые после **ELSE**. Например, после **THEN** можно написать **GOTO** и получить обычный оператор условного перехода. При использовании этого оператора важно то, что он весь должен уместиться в одной строке. Правда, следует помнить, что строке программы Бейсика разрешена длина до 255 символов, и, таким образом, для ее записи может не хватить даже трех строк экрана.

INPUT [;][<подсказка>;]<ОП>[,<ОП>...]

Ввод данных с экрана. Если после **INPUT** имеется точка с запятой, то после ввода данных курсор не перескочит на следующую строку. «Подсказка» — это символьная константа (именно константа, а не переменная!), значение которой появится на экране при запросе ввода перед вопросительным знаком, а если после этой константы будет запятая, а не точка с запятой, то вопросительного знака не будет, будет

только подсказка. В конце оператора список ввода — переменные, значения которых должны быть введены.

KEY <номер функциональной клавиши>, <текст>

Задается «текст», который будет выводиться на экран при нажатии клавиши с этим номером. Задавать можно в виде символьной константы (текста в кавычках) или переменной, если есть желание использовать этот оператор в программе. Нет смысла задавать таким образом слишком большой текст, потому что он будет обрублен до 15 знаков. **KEY LIST** выводит значения всех клавиш на экран.

KILL <описание файла>

Описанный файл стирается с диска (дискетки).

LINE [STEP] [(x1, y1)] — (x2, y2) [, <цвет>] [, B[F]]

Проводится прямая между точками с координатами (x1, y1) и (x2, y2). Если координаты первой точки не указаны, то прямая проводится с того места, где находится графический курсор, до указанной точки. Можно указать цвет, если он отличается от цвета переднего фона. Буква B в конце оператора все меняет: рисуется прямоугольник, координаты левого верхнего угла которого (x1, y1), а координаты правого нижнего угла (x2, y2). Оператор с буквами BF в конце нарисует прямоугольник и закрасит его указанным цветом.

LOCATE <номер строки>, <номер позиции>

В текстовом режиме ставит курсор в указанное место экрана.

MID\$(<символьная переменная>, <номер> [, <длина>]) = <CB>

В указанной переменной, начиная с символа с указанным номером, существующие символы заменяются на символы из значения правой части, начиная с первого. Если указана длина, то будет заменено только такое количество символов; если нет, то замена идет до тех пор, пока не закончатся символы в одной из строк.

OPEN <описание файла> **FOR** <режим> **AS** # <номер файла>

Открывает файл. Описание файла включает в себя код устройства, каталог, в котором находится файл, и название самого файла. Режим — это либо **OUTPUT** для вывода, либо **INPUT** для ввода, либо **APPEND** для добавления данных в конце существующего файла. Номер файла — логический номер файла.

PAINT [STEP] (x, y) [, <цвет окраски>] [, <цвет края>]]

Замкнутая область графического экрана, в которой находится точка с координатами x и y, окрашивается указанным цветом или цветом переднего фона, если цвет закрашки не указан. Можно также указать новый цвет края этой области.

[L]**PRINT** <список единиц вывода> [;]

Выводит значения переменных, указанных в списке. Если бук-

вы L нет, то вывод идет на экран, если она есть, то на печать. В списке названия переменных могут разделяться запятой или точкой с запятой. Если разделить запятой, то соответствующие значения выведутся в разных зонах вывода, а если точкой с запятой, то непосредственно друг за другом.

PRESET [STEP] (x, y) [, <цвет>]

Точка с координатами x и y закрашивается указанным цветом или цветом заднего фона, если цвет не указан. Есть еще оператор PSET, который отличается только тем, что точка закрашивается цветом переднего фона, если цвет не указан.

PUT [STEP] (x, y) <название массива>

Рисунок, сохраненный в указанном массиве чисел оператором GET, восстанавливается в прямоугольнике, координаты левого верхнего угла которого x и y. Повторный вывод того же рисунка в том же месте сотрет изображение, так что, проделывая такой двойной вывод в нужных местах, можно передвигать рисунок по экрану.

REM <текст>

Оператор сторожит «текст», чтобы не думали, что это программа.

RESTORE [<НС>]

Переход на чтение данных с начала хранилища. Следующий оператор READ начнет считывать данные с начала хранилища, т. е. с первого оператора DATA в программе. Если же указан номер строки, то считывание начинается с этой строки.

RETURN [<НС>]

После завершения работы подпрограммы возвращает управление назад на оператор, следующий за оператором GOSUB, который вызвал подпрограмму. Если же указан номер строки, то управление будет передано строке с этим номером.

SCREEN <режим>

Задается режим экрана как числовая константа либо переменная. Может быть разным для разных экранов, часто SCREEN 0 означает текстовый режим, SCREEN 1 — графический режим с 320 точками по горизонтали, а SCREEN 2 — графический режим с 640 точками по горизонтали. По вертикали в обоих случаях 200 точек.

SOUND <частота>, <длительность>

Раздается звук с указанной в герцах частотой, которая может быть в пределах от 37 до 32767, и указанной длительностью, которая задается в тиканьях внутренних часов компьютера. К сожалению, эти часы могут тикать по-разному у компьютеров разных фирм, поэтому значение для длительности зву-

чания приходится подбирать. Имеются и более сложные операторы управления звуком, которые позволяют воспроизводить вполне сносную музыку.

STOP

Программа прекращает работу, но может потом ее и продолжить, если выполнить директиву CONT. Продолжения не будет, если после прекращения работы хоть немножко изменить текст программы.

ФУНКЦИИ

ABS(<AB>) — абсолютное значение арифметического выражения

ASC(<строка символов>) — код первого знака символьной строки

CHR\$(<AB>) — нахождение символа по заданному коду

COS(<AB>) — косинус

EXP(<AB>) — экспонента

INT(<AB>) — целая часть

INSTR(<номер>, <1.строка>, <2.строка>) — место первого вхождения 2.строки в 1.строку, начиная счет с указанного номера

LEFT\$(<CB>, <количество>) — выделение указанного количества первых символов строки

LEN(<CB>) — длина строки символов

LOG(<AB>) — натуральный логарифм

MID\$(<CB>, <первый номер>, <длина>) — выделение из символьной строки подстроки указанной длины, начиная с указанного номера

PI — число 3.14159

RIGHT\$(<CB>, <количество>) — выделение указанного количества последних символов строки

RND — случайное число

SIN(<AB>) — синус

SPACE\$(<количество>) — символьная строка, состоящая из указанного количества пробелов

SQR(<AB>) — квадратный корень

STR\$(<AB>) — символьная строка, представляющая значение указанного арифметического выражения

VAL(<строка символов>) — числовое значение строки символов

ДИРЕКТИВЫ

На персональном компьютере разница между директивами и операторами небольшая, потому что здесь любую директиву можно писать в строках программы, и она будет выполняться.

CLEAR

Очистка памяти. Значения всех числовых переменных становятся равными нулю, а символьные переменные принимают пустое значение. Освобождается вся память, занятая значениями переменных, перестает звучать запущенный звук.

CONT

Запускает программу дальше после ее останова, скажем, оператором STOP.

DELETE [$\langle$ НС $\rangle$][—[$\langle$ НС $\rangle$]]

Удаление строк программы. Самый простой способ использования — написать только один номер. Тогда будет удалена только эта строка. Если после номера имеется тире (знак минуса), то удаляются все строки от указанного номера до конца программы. Если тире перед номером, то от начала программы до строки с указанным номером включительно. А если указать два номера с тире посередине, то будут удалены все строки с номерами в этом интервале. И, наконец, если номеров нет, то удаляется вся программа.

FILES

Выводит на экран содержание рабочего каталога.

[L]LIST [$\langle$ НС $\rangle$][—[$\langle$ НС $\rangle$]]

Вывод программы на экран или на печать, если использовать директиву LIST. В описании директивы DELETE надо слово «удаляется» заменить на слово «выводится», чтобы получить описание этих директив.

NEW

Уничтожает программу в памяти, весь ее текст, значения переменных и вообще всякое воспоминание о ней.

RENUM [$\langle$ новый номер $\rangle$][,[$\langle$ старый номер $\rangle$][, $\langle$ интервал $\rangle$]]

Строки программы перенумеруются, начиная с указанного старого номера, которому дается указанный новый номер, а для последующих строк номера вычисляются последовательно, добавляя указанный интервал. Если интервал не указан, берется 10, а если не указан старый номер, то нумерация начинается с начала программы.

RUN [$\langle$ НС $\rangle$]

Программа запускается с указанной строки или с начала, если строка не указана.

SAVE $\langle$ описание файла $\rangle$

Программа записывается на диск в указанном файле.

SYSTEM

Конец работы и выход из Бейсика в операционную систему.

Вот и все, что было в записной тетрадке Тони.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

- Алгоритм 16
- Алгоритм схемы 17
- Арифметическое выражение 42
- Ввода оператор 56
- Ветвления оператор 71
- Встроенная функция 103
- Вывода оператор 41, 51
- Граница цикла 87
- Директива 52
- Директива BYE 61
 - LIST 54
 - LISTNH 54
 - LOAD 216
 - NEW 52
 - OLD 70
 - RENUM 212
 - RESEQ 65
 - RUN 55
 - RUNNH 55
 - SAVE 58
- Дисплей 11
- Дисплей графический 219
- Закрытие файла 172
- Значение переменной 48
- Зона вывода 66
- Имя переменной 47
- Имя файла 59
- Интерпретатор 39
- Клавиша фиксации 208
- Клавиша функциональная 210
- Код устройства 59
- Команда 39
- Конкатенация 128
- Координаты экрана 220
- Курсор 13
- Логический номер файла 170
- Магнитные диски 9
- Магнитофон 225
- Максимальное количество элементов 82
- Непосредственный режим 40
- Номер строки программы 53
- Номер устройства 59
- Обозначение переменной 47
- Обозначение файла 59
- Оператор 46
 - CIRCLE 224
 - CLEAR 232
 - CLOSE 172
 - CLS 213
 - COLOR 222
 - DATA 160
 - DEF FN 145
 - END 54
 - FOR ... NEXT 89
 - GO TO 84
 - GOSUB 162
 - IF ... THEN 71, 233
 - INPUT 57
 - KILL 172
 - LINE 221
 - MOTOR ON 225
 - MOTOR OFF 225
 - ON 119
 - OPEN 169
 - PAINT 223
 - PCLS 213
 - POINT 235
 - PRESET 223
 - PRINT 41
 - READ 160
 - REM 97
 - RESTORE 161
 - RETURN 163
 - STOP 54
- Операционная система 39
- Открытие файла 169
- Отладка программы 152
- Память 9
- Переменная 47
- Переменная INKEY 229
- Переменная символьная 126
- Переменная цикла 87
- Подпрограмма 162
- Присвоения оператор 40, 47
- Программа 35

Псевдографические символы 230
Символьная переменная 126
Случайные числа 109
Сравнение по алфавиту 130
Строка программы 52
Строка символов 127
Счетчик 25
Условного перехода оператор 71
Файл 169
Функция ASC 230
 CLK ✕ 143
 DAT ✕ 141
 INSTR 216
 INT 104
 LEFT ✕ 215
 LEN 135
 MID ✕ 216

POS 136
RND 109
RIGHT ✕ 215
SEG ✕ 137
SPACE ✕ 217
STR ✕ 140
STRING ✕ 216
TAB 182
VAL 140
Функция встроенная 103
Функция программиста 145
Цикл 18
Цикла оператор 87
Цикла переменная 87
Экспоненциальная форма числа 43
Элемент массива 81
Язык программирования 36

СОДЕРЖАНИЕ

О чем рассуждали Тоня и Петя, прочитав все последующее	3	7. Первая программа Тони	46
1. Как все началось	5	Кирпичики программ — операторы	—
2. Первое знакомство	7	Все о переменных	47
Компьютеры бывают всякие	8	Бейсик можно переводить на русский	48
Из чего состоит компьютер	—	Как получить программу	51
Дисплей — компьютерное окно в мир	11	Как нумеровать строки	52
Умеет ли думать компьютер	—	Проверка ввода	54
Тоня уже начинает действовать	13	...И запуск программ	55
Что еще видно на клавиатуре	14	Ввод во время работы программы	56
Тоне захотелось программировать	15	Запись программы из памяти на диск	58
3. Тоня Соображалкина узнает, что такое алгоритм	16	Запись программы из памяти на бумагу	60
В основе программ лежат алгоритмы	—	8. Небольшая польза от программирования	62
Алгоритмы изображают в виде схем	17	Градусы Фаренгейта и Цельсия	—
Важное понятие — «цикл»	18	Программа должна быть понятной для всех!	63
Введение в робототехнику	22	Тоня еще улучшает вывод	65
4. Еще один алгоритм	23	Опять ошибка	67
Алгоритм для начинающих учиться	24	Программисты и их ошибки	68
Работа со счетчиком классов	25	9. Веток много-много, но не дерево	69
Тоня улучшает алгоритм	26	Сейчас об операторе ветвления	71
Ошибка!	27	У Тони получается диалог с машиной	73
5. Языки, языки, языки	29	Особенности перенумерации	75
Мир состоит из атомов и алгоритмов	—	Что Тоня уже умеет	76
Еще об алгоритмах	30	10. Тоня уже знает о программировании все	79
Кулинария, алгоритмы и компьютеры	32	О массивах чисел	80
Языки для общения с компьютерами	33	Какой нужен массив, догадайся сам	82
Входная и выходная информация	34	Маленькая программа для ввода оценок	83
Программы кто-то должен составлять	35	Оператор, всегда передающий управление	84
Языки — средство общения	36	Как закончить ввод?	86
6. Начинается работа	38	Оператор-счетчик	87
Все делает интерпретатор	—	И что все это означает	88
Вызов интерпретатора	39	Правильные и неправильные операторы цикла	89
Работа в непосредственном режиме	40	И еще раз та же самая программа	91
Результаты всегда нужно вывести	41	Программа суммирования чисел	92
Две формы числа в Бейсике	43	Быстро и плохо? Это не стиль программистов!	93
Сколько будет двенадцать в степени двенадцать?	44	11. Делай, как я, делай лучше меня!	95
А если ошибешься?	45	Программа решения самого простого уравнения	—
		Как пояснить свою программу	97

12. Небольшой экскурс в бухгалтерское дело	100	Как машина могла бы помочь в изучении географии	166
Нужные всем функции	103	...Языков.	167
Ошибки округления в компьютере неизбежны	104	...Истории.	—
13. Компьютер или учитель!	106	...И еще кое-чего	168
Наука обучать называется методикой	107	19. Тоня узнает о файлах	169
Генератор случайных чисел — функция RND	109	Файл — книга?	—
Зачем нужны случайные числа	110	Логический номер файла	170
Оценки — вещь нужная	112	Действие с файлами	172
14. Главное — это учет	113	Запись в файл	173
Чему еще можно учить детей?	114	Компьютер поздравляет с праздником	174
Магазин в памяти компьютера	116	О двумерных массивах	177
Еще один оператор перехода	119	Составить поздравление нелегко	179
Что делать в начале работы	120	Совсем простая подпрограмма	182
Что делать после каждой покупки	—	Еще одна подпрограмма	183
О данных и их обработке	122	Проверка правильности программы	185
Еще об обработке данных	124	Возможные ошибки	187
15. Вычислительная машина не только вычисляет	126	20. Тоня продолжает осваивать файлологию	188
Символьные переменные	—	Что еще можно напечатать	189
Вывод символов	127	Список учеников на диске	—
Единственное действие с символами	128	Работа со списком	193
Как сравнивают строки символов	129	Как дополнить список	194
Программа упорядочения двух слов	131	Для чего такой список	195
16. Работа со словами	133	Ввод характеристик учеников	196
Как стать хозяином своих слов	134	День рождения друга забыть нельзя!	198
Сколько символов в строке?	135	21. Небольшая экскурсия по улице Суворова	201
Что найдешь в символьной строке?	136	22. На abordаж «Корветов»!	204
Разбиение на части символьной строки	137	Это и есть корвет	205
Связь между числами и символами	139	Новая клавиатура	207
Календарь в компьютере	141	Общее между двумя Бейсиками	211
Часы в компьютере	143	Команды и функциональные клавиши	212
Функции программиста — самодельный инструмент	145	Работа с символьными переменными	214
17. Машина осваивает русский язык	147	23. Как рисовать компьютером	217
Программировать надо умеючи	148	24. Тоня учит Петю	227
Ошибка или сбой машины?	150	25. Что еще можно сделать	236
Горькая правда, а не мораль	151	26. Чем все это кончилось	241
Перед тестированием полезно подумать	153	27. Из записной тетрадки Тони	245
Языки и программирование	154	Предметный указатель	252
18. Еще о том, как компьютер учит детей	155		
Учим географию	156		
Хранение данных в программе	160		
Важное понятие — подпрограмма	162		
Автоматизация педагогики	163		
Машина — экзаменатор	164		

РЕБЯТА!

РОДИТЕЛИ!

УЧИТЕЛЯ!

Мы для вас подготовили отличную вещь! Дискету со всеми программами книги, с решениями всех ее задач и еще с парой десятков интересных программ на Бейсике — серьезных и игровых. И при том на всяких разных компьютерах. Давайте мы эту дискету вам вышлем! Расходы такие: за дискету не много, нам за старание того меньше и, конечно, за пересылку. Это расходы, кто будет работать один с этими программами, а если кто собирается их копировать или размножать, например, для школьного кабинета, тому надо будет заплатить немножко больше. Расходы можно оплатить и по перечислению.

**Пишите нам сегодня! Откладывать не надо! Цены растут!
Не забудьте указать доступный вам компьютер и Бейсик.**

**Адрес такой: 226250, Рига, ГСП, бульв. Райниса, 29.
Институт математики и информатики ЛУ
Мартузану Бруно Яновичу.
(он знает, что к чему)**

Тоня и Петя

Учебное издание

МАРТУЗАН БРУНО ЯНОВИЧ

ПОДРУЖИСЬ СО МНОЙ, КОМПЬЮТЕР!

Зав. редакцией Т. А. Бурмистрова
Редактор А. К. Компанец
Младший редактор О. В. Агапова
Художники Р. Р. Витковский, Л. М. Чернышев
Художественный редактор Ю. В. Пахомов
Технический редактор С. В. Китаева
Корректор М. Ю. Сергеева

ИБ № 12636

Сдано в набор 19.12.89. Подписано к печати 24.06.91. Формат 70 × 90¹/₁₆. Бум. офсетная № 2. Гарнит. журн. рубл. Печать офсетная. Усл. печ. л. 18,72 + 0,29 форз. Усл. кр.-отт. 38,38. Уч.-изд. л. 15,36 + 0,44 форз. Тираж 55 000 экз. Заказ 2357.

Ордена Трудового Красного Знамени издательство «Просвещение» Министерства печати и информации Российской Федерации. 127521, Москва, 3-й проезд Марьиной рощи, 41.

Печать с диапозитивов Саратовского ордена Трудового Красного Знамени полиграфического комбината Министерства печати и информации Российской Федерации. 410004, Саратов, ул. Чернышевского, 59, на Смоленском полиграфкомбинате Министерства печати и информации Российской Федерации. 214020, Смоленск, ул. Смольянинова, 1.



РАБОТАТЬ



УЧИТЬСЯ



СХЕМА АЛГОРИТМА

ПОСТУПИТЬ В ШКОЛУ. ПРИСВОИТЬ
ЗНАЧЕНИЕ 0 числу K ($K=0$)

ОКОНЧЕНА ЛИ ШКОЛА, Т.Е.
 K УЖЕ РАВНО 10?

ДА

НЕТ

УСПЕШНО ЗАНИМАТЬСЯ
ОДИН ГОД В $K+1$ -ОМ КЛАССЕ

НЕТ

ПЕРЕВЕЛИ ЛИ
В СЛЕДУЮЩИЙ КЛАСС?

ДА

УВЕЛИЧИТЬ ЧИСЛО K
НА ЕДИНИЦУ ($K=K+1$)

В БОЛЬШУЮ
ЖИЗНЬ!

